Recherche et tests d'algorithmes de clustering spatial pour webdataviz

Emna CHIKHAOUI
Etienne DUPERRON

M2 - SIGMA

Encadrants

Laurent JEGOU
Marion MAISONOBE

Février 2019

Sommaire

Contexte

Le projet NetScience, géré par les UMR CNRS LISST (Toulouse) et Géographie-Cités (Paris), s'intéresse à l'observation de la géographie mondiale des activités scientifiques. Dans le cadre de ce projet, une application en ligne appelée NetsCity a été mise en place afin de déterminer et d'analyser la répartition spatiale des publications et des collaborations scientifiques à l'échelle mondiale au niveau d'agglomérations. Une agglomération ou un "cluster" est définie par le regroupement de localités étant assez proches géographiquement. Ces dernières sont issues du géocodage des adresses des auteurs ayant rédigé et publiés des articles scientifiques entre 1999 et 2018. L'ensemble de ces publications scientifiques sont récupérées via des extractions de bibliothèques numériques telles que Web of Science.

Objectif

L'objectif de cet atelier est de trouver d'autres méthodes de segmentation spatiale - différentes de celle mise en place- qui permettent d'automatiser les traitements (nécessité d'une supervision ou non par un utilisateur) et d'intégrer d'autres types de données dans l'étape de clustering comme le nombre de publications par localités ou la facilité d'accès (en fonction des transports disponibles).

Méthodologie

Pour la recherche des nouvelles méthodes, nous avons adopté une approche basée sur le principe de l'entonnoir.

En effet, nous sommes partis à la recherche des méthodes automatiques de clustering en général. Puis au fur et à mesure, nous avons éliminé celles qui ne répondaient pas vraiment aux besoins, comme les méthodes qui ne peuvent pas manipuler de matrices de distances ou celles qui nécessite d'indiquer en paramètre d'entrée le nombre de cluster voulu (méthodes de partition de l'espace telle que celle de K-means par exemple). Nous avons également cherché des méthodes qui peuvent prendre en considération des contraintes (rivière, montagne...) et des facilités (autoroutes…) bien que nous n'avons pas pu les utiliser au final car elles pourraient particulièrement enrichir et rendre plus pertinents les résultats (voir partie "Autres méthodes")

Aux termes de ces recherches nous avons décidé de retenir trois grandes méthodes pour la phase de test :

1. Méthodes retenues

Toutes les méthodes retenues peuvent faire leur traitement à partir d'une matrice de distances et ne requièrent pas un nombre de clusters déjà prédéfini.

A. Classification hiérarchique agglomérative : Agglomerative Nesting (AGNES)

Cet algorithme est basé sur une approche *"ascendante"* : chaque élément à traiter est considéré initialement comme un cluster/groupe à part entière. Les deux éléments les plus proches selon une notion de distance ou dissimilarité sont regroupés en un cluster. Cette-dernière peut être aussi bien géographique (basée sur des coordonnées de longitude/latitude ou projetées) que mathématique (la distance euclidienne par exemple, ou dans le cas d'un espace à N dimensions, càd N coordonnées par élément). Le processus de regroupement par paire est réitéré jusqu'à ce que l'on obtienne un cluster comportant l'ensemble des éléments.

Il est donc nécessaire pour cette méthode de disposer initialement d'une matrice de distances ou de dissimilarité, c'est-à-dire de calculer pour chaque paire d'éléments 2 à 2 de l'ensemble traité la distance qui les sépare.

Exemple de matrice de distances pour un petit nuage de points :

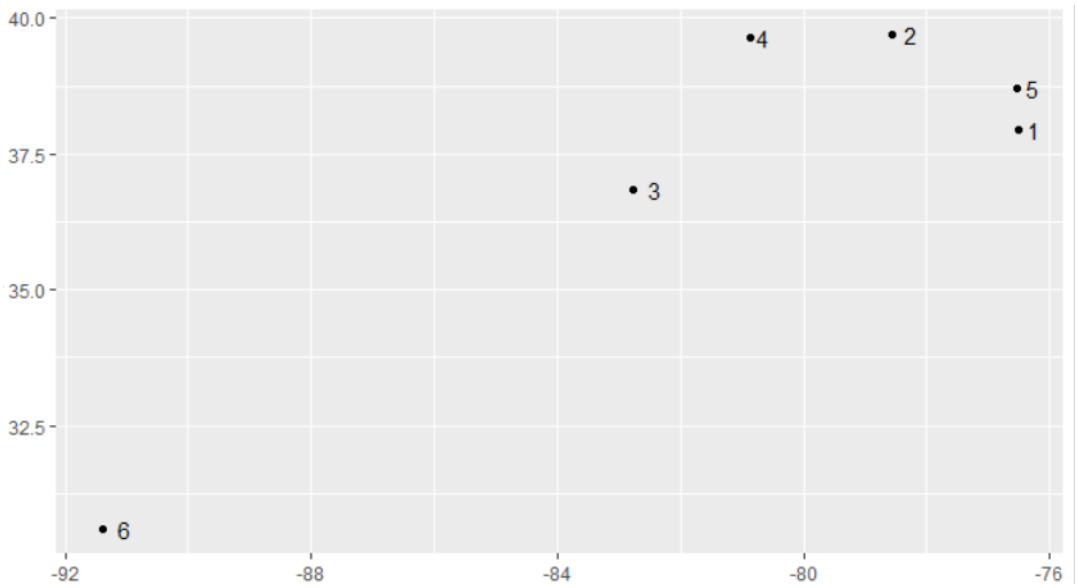


Fig1. Nuage de points

	1	2	3	4	5	6
1	0	262.9247	567.8107	420.6199	82.8491	1591.847
2	262.9247	0	485.305	196.3852	207.9427	1541.745
3	567.8107	485.305	0	352.1676	587.5749	1057.006
4	420.6199	196.3852	352.1676	0	388.0215	1387.102
5	82.8491	207.9427	587.5749	388.0215	0	1629.424
6	1591.8472	1541.7454	1057.0059	1387.1022	1629.4236	0

Fig2. Matrice des distances
(ici distances géographiques calculées à partir de longitude/latitude)

Les résultats de chaque étape du processus de classification sont conservés et synthétisés sous la forme d'un arbre de classification autrement appelé dendrogramme. Il rend compte de la hiérarchie, de la structure et de la proximité des éléments et des différents groupes.

Schéma du processus de classification hiérarchique agglomérative :

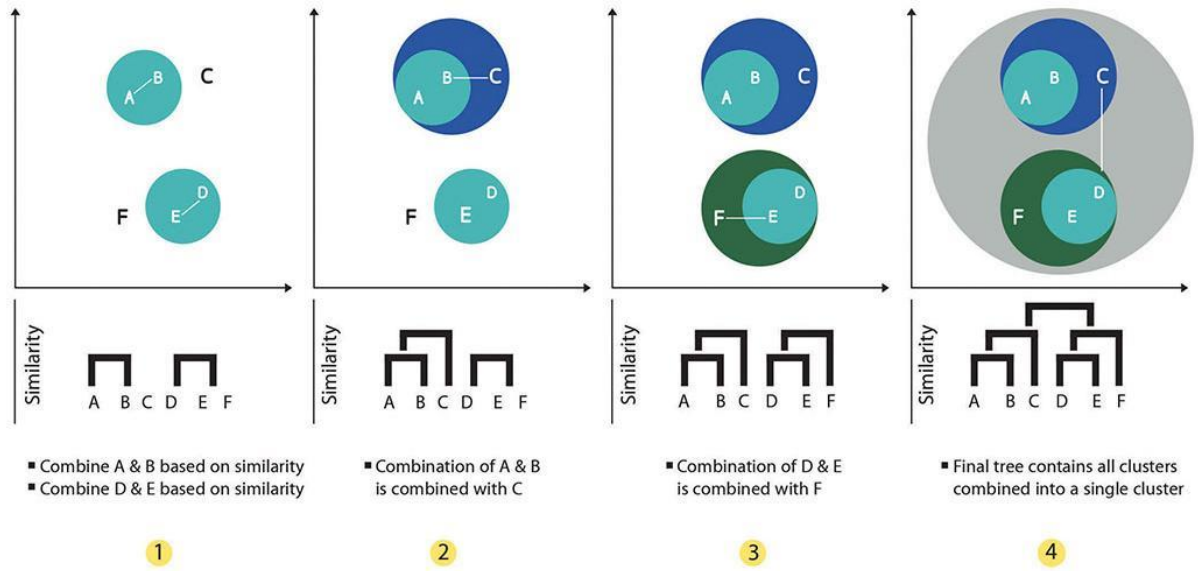
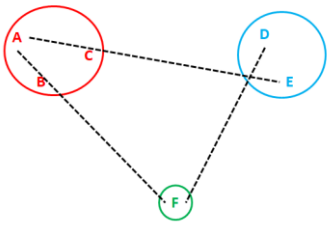


Fig3. Nuages de points avec les clusters et construction du dendrogramme

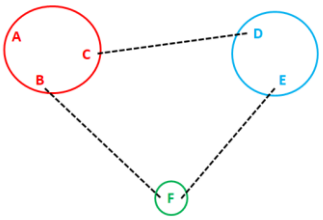
Source : https://www.brandidea.com/hierarchicalclustering.html

De plus, il existe plusieurs méthodes d'agglomération des groupes. En effet, une fois qu'un cluster est créé, il est nécessaire d'actualiser la matrice des distances, c'est-à-dire de calculer les distances entre le cluster créé et les autres éléments ou clusters restants (la matrice rétrécit à chaque itération). Il existe donc plusieurs manières déterminer les nouvelles distances :

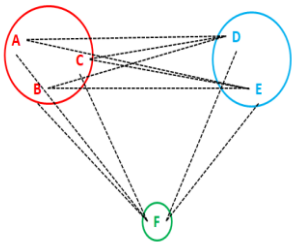
- méthode du saut maximum ou du diamètre ou complete linkage : la plus grande distance possible entre éléments des deux clusters (ou entre clusters et éléments) sera considérée comme la nouvelle distance.



- méthode du saut minimum ou single linkage : la plus petite distance possible entre éléments des deux clusters (ou entre clusters et éléments) sera considérée comme la nouvelle distance.



- méthode du saut moyen ou UPGMA (Unweighted Pair Group Method with Arithmetic mean) : la moyenne arithmétique de toutes les distances possibles entre éléments des deux clusters (ou entre clusters et éléments) sera considérée comme la nouvelle distance.



- méthode de Ward : elle découle des statistiques est basée sur un calcul d'inertie. Les éléments et cluster sont regroupés de manière à ce que l'augmentation de l'inertie intercluster soit maximum, autrement dit que l'augmentation de l'inertie intracluster soit minimale. Elle s'apparente à la méthode de Jenks en sémiologie.

  L'inertie est déterminée à partir des distances entre centres de gravité de clusters et le centre de gravité global de tous les éléments (inertie intercluster) ou entre centre de gravité de cluster et éléments de cluster (inertie intracluster).

Une fois la classification terminée, il reste cependant à définir/matérialiser les clusters que l'on va considérer. En effet, selon la hauteur à laquelle on se place sur le dendrogramme, il n'y aura pas le même nombre de clusters. Il est possible de fixer un nombre voulu de groupes mais cela ne nous intéresse pas dans le cadre de ce projet. Ainsi il est également possible de fixer une hauteur de coupe (qui peut être relative), ou bien de passer par des

indices de stabilité ou de dissimilarité entre les clusters selon si l'on passe de k à k+1 clusters.

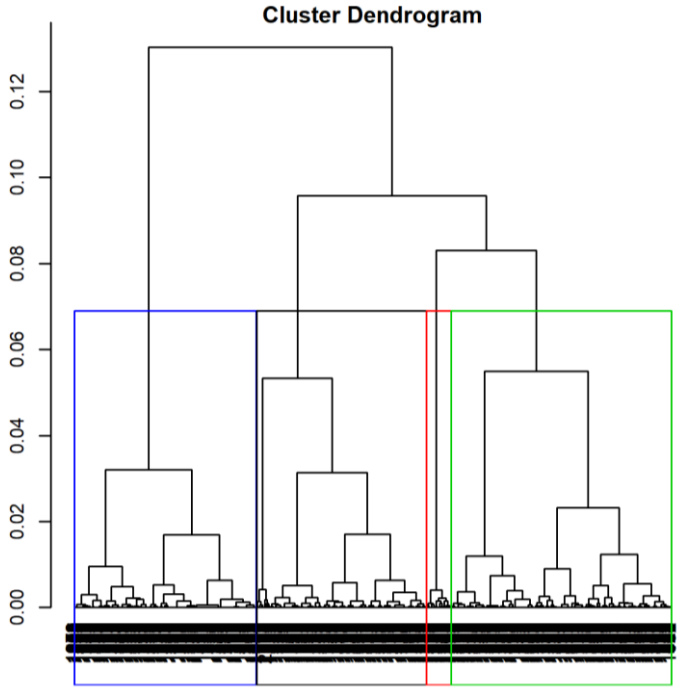


Fig4. Hauteur de coupe

Source : http://larmarange.github.io/analyse-R/classification-ascendante-hierarchique.html

Ici par exemple avec une hauteur de coupe à 0.07 4 clusters sont obtenus.

L'implémentation de l'algorithme AGNES que nous avons principalement utilisée s'appelle *fastcluster*. Celle-ci a l'avantage d'être optimisée ce qui réduit notoirement les temps de calcul, mais également d'être disponible sous R et Python.

Documentation : https://cran.rstudio.com/web/packages/fastcluster/vignettes/fastcluster.pdf

Nous avons également utilisé une autre implémentation sous R, le package ClustGeo, qui permet de pondérer les éléments. Il s'agit d'une classification hiérarchique agglomérative de type Ward (il n'est pas possible de choisir sa méthode d'agglomération).

Documentation : https://cran.r-project.org/web/packages/ClustGeo/ClustGeo.pdf

B. Classifications basées sur la densité:

Ces méthodes détectent les zones où les points sont concentrés et séparés par des espaces vides ou clairsemés. Les points qui ne font pas partie d'un cluster sont classés comme du bruit.

- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**

Le principe de cet algorithme est assez simple. Le voisinage d'un point arbitraire est récupéré. S'il contient suffisamment de points alors l'ensemble sera considéré comme appartenant à un cluster et ce voisinage sera ensuite parcouru de proche en proche afin de trouver tous les points du groupe. Autrement, le point sera considéré comme du bruit.

DBSCAN requiert donc 2 paramètres en entrée : le rayon de recherche du voisinage des points (Eps) ainsi que le nombre minimum de points qu'il doit y avoir dans ce voisinage pour que ceux-ci soient considérés comme un cluster (MinPts). Ces paramètres d'entrées correspondent donc à une estimation de la densité de points des clusters.

Et en effet, à cause de la façon dont fonctionne DBSCAN, il est sensible aux variations importantes de densités entre clusters. Autrement dit, lorsque le jeu de points traité présentent plusieurs clusters ayant des densités très différentes, DBSCAN aura du mal à les identifier en même temps (soit les paramètres seront optimaux pour les clusters à faible densité, soit pour les clusters à forte densité).

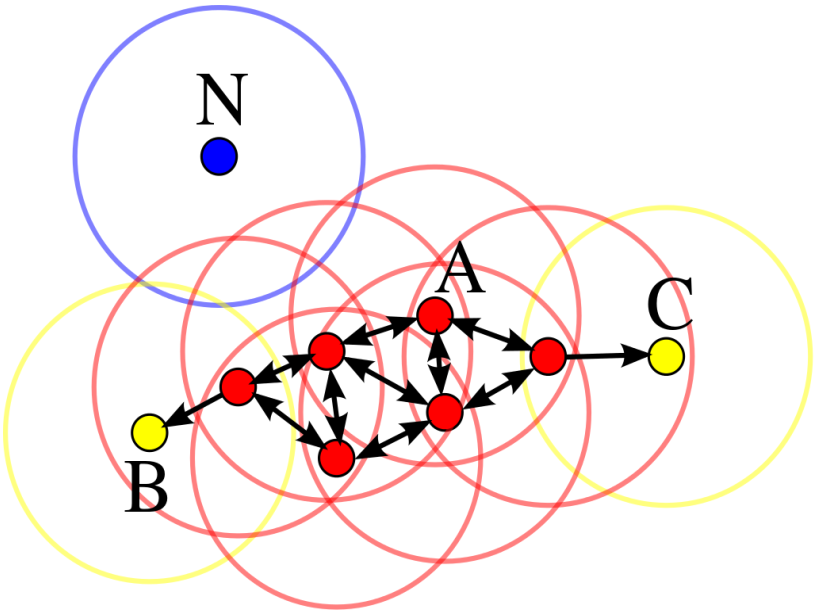


Fig5. Illustration du processus de DBSCAN

Source : https://fr.wikipedia.org/wiki/DBSCAN

Les points A sont les points déjà dans le cluster. Les points B et C sont atteignables depuis A et appartiennent donc au même cluster. Le point N est une donnée aberrante puisque son voisinage ne contient pas MinPts points ou plus.

Nous avons utilisé 2 implémentations de cet algorithme :

- Package *dbscan* sous R : https://cran.r-project.org/web/packages/dbscan/dbscan.pdf
- Fonction *DBSCAN* de la bibliothèque *scikitlearn* sous Python : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.dbscan.html

- **Hierarchical DBSCAN (HDBSCAN)**

Cet algorithme est une méthode hybride : il étend DBSCAN en le convertissant en un algorithme de classification hiérarchique, puis il utilise une technique basée sur la stabilité des clusters permettant de déterminer les regroupements les plus pertinents. Contrairement à DBSCAN, HDBSCAN n'est pas sensible à des variations de densité importante entre clusters, ce qui permet d'avoir des résultats potentiellement plus pertinents.

Il présente 5 étapes majeures :

1) Transformer l'espace en fonction de la densité : les distances pour les points dans les zones de faibles densités vont être artificiellement augmentées alors que celles-ci seront inchangées dans les zones à fortes densités. Cela permet de réduire l'effet des points pouvant potentiellement être du bruit et donc d'avoir une meilleure classification des structures les plus importantes.

2) Construire l'arbre couvrant de poids minimal ou ACM (Minimum Spanning Tree) à partir du graphe pondéré des distances entre les points : cela permet de déterminer la structure général de l'ensemble de points et les proximités les plus faibles.

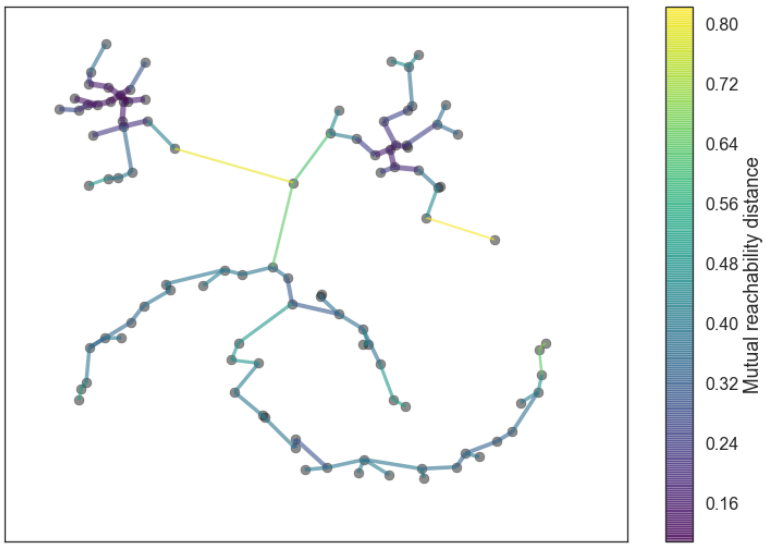


Fig5. Arbre Couvrant de poids Minimal (ACM)

3) Établir la classification hiérarchique à partir de l'ACM : il s'agit simplement de deux représentations différentes d'une structure ou hiérarchie, il suffit donc simplement de convertir l'ACM en un dendrogramme.

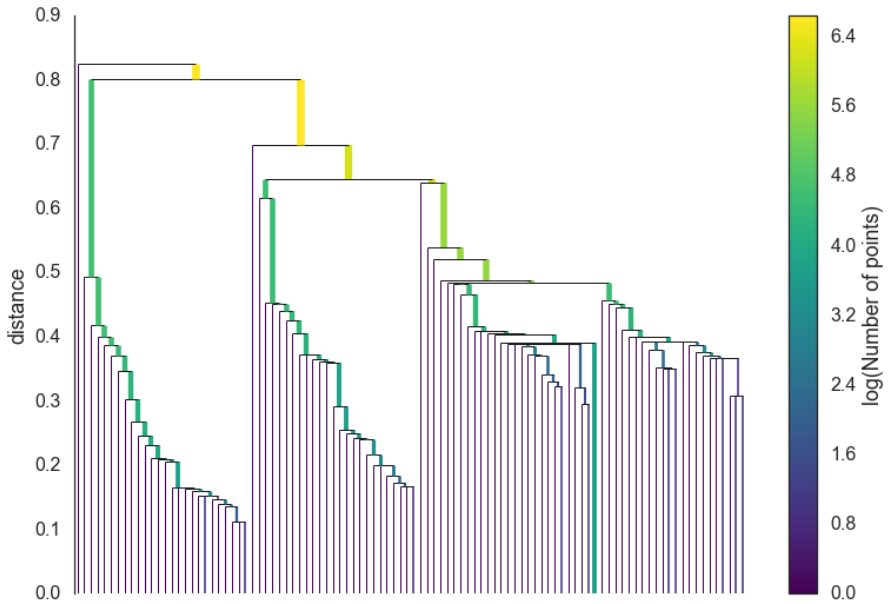


Fig6. Dendrogramme d'ACM

4) Condenser le dendrogramme par rapport à la taille minimale des clusters : cette étape permet d'utiliser par la suite un critère de stabilité pour déterminer les clusters les plus pertinents. Il s'agit de simplifier le dendrogramme en conservant seulement les clusters les plus importants (supérieurs à la taille minimale).

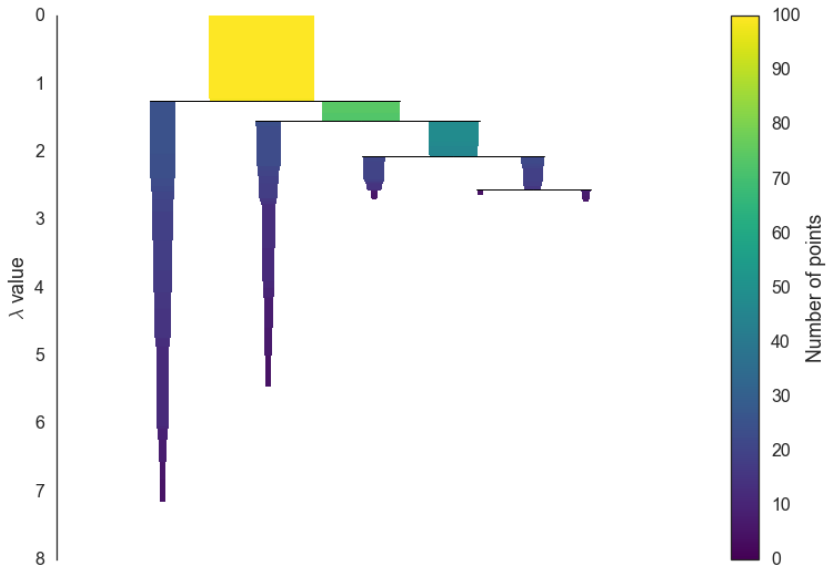


Fig7. Simplification du dendrogramme

5) Extraire les clusters stables du dendrogramme condensé : le dendrogramme est parcouru depuis la base vers le haut (c'est à dire de plus petits aux plus grands clusters). Pour chaque noeud, la stabilité du cluster parent est comparée aux stabilités des clusters descendants. Si celles-ci est supérieure à la somme des stabilité des descendants alors on poursuit. Autrement les clusters descendants sont considérés comme stables.

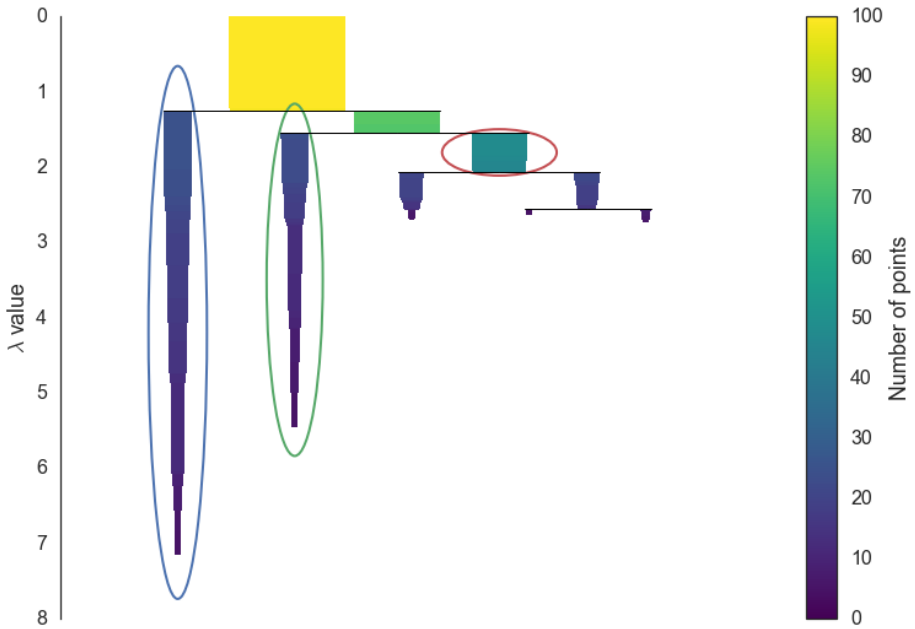


Fig8. Clusters stables

Les images ainsi que le détail des étapes de HDBSCAN ci-dessus constituent un résumé de l'information d'une documentation Python expliquant l'algorithme : https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

Cette-dernière explique très bien et de manière assez précise les différentes étapes. Il est donc recommandé de la parcourir pour avoir des informations plus complètes.

L'utilisation de HDBSCAN lors des tests s'est fait grâce à la bibliothèque *hdbscan* sous Python : https://hdbscan.readthedocs.io/en/latest/index.html

Et le package *dbscan* sous R : https://cran.r-project.org/web/packages/dbscan/dbscan.pdf

2. Protocole de test des algorithmes

Afin de comparer les méthodes de clustering retenues, nous avions à notre disposition un jeu de données d'environ 32 000 points correspondant chacun à une localité de publications scientifiques au cours de la période 1999-2014. Le nombre de publications scientifiques issues de chaque localité sur cette période de temps était également indiqué en plus des informations géographiques (Ville - Province - Pays). Les résultats pour la méthode actuellement utilisée dans l'application NetsCity étaient également présents.

Nous avons décidé de d'abord tester l'influence du nombre de points traités et l'influence du changement de paramètre de chaque méthode sur les résultats.

De plus, nous voulions étudier l'effet de l'introduction d'un élément non spatial par pondération dans les paramètres de certaines méthodes.

Enfin, pour pouvoir comparer les différents résultats, nous avons défini des métriques générales communes à tous les algorithmes. Des zooms sur certaines parties des données, comme les zones à fortes densités de points ou situées au niveau de traits de côte tortueux, ont été réalisés afin de compléter l'étude du comportement de chaque méthode de clustering.

A. Données en entrée

L'ensemble des tests ci-dessous ont été réalisés à partir de matrices de distances calculées avec la méthode des Grands Cercles, autrement dit les distances géographiques entre chaque point du jeu de données à partir de leurs coordonnées de longitude/latitude. Celles-ci sont exprimées en km.

Ce type de distance est différent d'une distance euclidienne qui correspond plus à une distance géométrique/mathématique.

B. Influence du nombre de points traités

Dans le processus de clustering, la taille de l'ensemble de points traités peut potentiellement influencer les résultats. En effet, certaines méthodes étant basées sur des calculs de densité, il peut y avoir de fortes variations selon le nombre de points. Mais c'est également le cas pour les classements hiérarchiques étant donné qu'ils sont basés sur un critère de distance (géographique ou mathématique). Il nous a donc paru pertinent de tester l'influence de ce paramètre.

Quatre ensembles de points ont donc été constitués (voir tableau ci-dessous). Afin d'avoir une certaine unité géographique (autrement dit une dispersion des points limitée), ces derniers ont été définis par rapport à des zones géographiques du globe.
Nous avons fait en sorte d'avoir environ une différence d'à peu près 7500 points entre chaque ensemble (cela correspond à ¼ de la taille de l'échantillon total pour l'ensemble du monde).

Zones	Nombre de points
Etats-Unis (uniquement le pays)	7443
Europe de l'Ouest et Afrique	15765
Eurasie, Afrique et Océanie	22775
Monde entier	32864

Tab1. Ensembles de points choisis

Pour pouvoir détecter l'influence des points traités, nous avons fixé les paramètres employés pour chaque méthode. Ces choix sont arbitraires et découlent de tests préliminaires effectués afin d'appréhender les méthodes.

Fastcluster	- Méthode d'agglomération : Complete - Hauteur de coupe du dendrogramme : 0.5% de la hauteur maximale
DBSCAN	- MinPts : 2 - Eps : 20
HDBSCAN	- MinPts : 3

Tab2. Paramètres choisis

Remarque : les résultats pour cette partie des tests ont seulement été produits sous R par manque de temps pour les réaliser sous Python.

C. Influence des paramètres de chaque méthode

Les valeurs choisies pour les paramètres de chaque méthode peuvent énormément influencer les résultats. Nous avons donc souhaité étudier la sensibilité des algorithmes. Afin de ne pas avoir des temps de traitement trop longs, nous avons réalisé les tests sur un sous-échantillon de l'ensemble total de points : les Etats-Unis. Nous avons choisi ce dernier car il comporte le plus de points **(7443)** et présente une bonne variabilité de densité (côte est très dense alors que l'ouest est moins peuplé).

Les paramètres de chaque méthode étaient variés comme suit:

- **Méthode de classification hiérarchique agglomérative**

La méthode d'agglomération utilisée par défaut pour AGNES/fastcluster est "**complete**". Mais, d'autres existent comme "**single", "average" et "ward.D"** et permettent aussi de couper le dendrogramme à partir d'une valeur de hauteur que nous avons fixée à **0,5% de la hauteur max.**

- **DBSCAN**

Les facteurs à modifier pour cette méthode sont le nombre minimum de points pour constituer un cluster (MinPts) et le rayon de recherche du voisinage (Eps).
En une première phase nous avons **fixé Eps à 20 km et varié MinPts à 1, 2, 3, 4 et 5 points.**
En une deuxième phase, nous avons inversé en **fixant MinPts à 2 et en variant Eps à 5, 10, 20, 40 et 50 km.**

- **HDBSCAN**

Le seul paramètre à moduler avec HDBSCAN est le nombre minimum de points pour constituer un cluster (**MinPts**) **qui était varié de 2, 3, 4 et 5 points**. Nous n'avons pas commencé de 1 point car cela est impossible pour cette méthode.

Remarque: l'ensemble des résultats pour cette partie des tests ont été produits avec les scripst R et Python.

D. Introduction d'un paramètre non spatial par pondération

Nous avons choisi d'introduire comme poids des points une variable non spatial : le nombre de publications pour chaque localité sur la période 1999-2014. Ces tests ont également seulement été fait sur les Etats Unies. Cette pondération a seulement été possible pour une classification hiérarchique agglomérative et DBSCAN.

- **Méthode de classification hiérarchique agglomérative**

La pondération a été réalisée avec le package *ClustGeo* sous R pour une méthode de clustering hiérarchique agglomérative de type "ward.D" avec une hauteur de coupe du dendrogramme définie à **5e-6 %** (0.000005/100). Des hauteurs plus élevées ne fournissaient pas de bons résultats.

Cette pondération ne peut se faire que sous Windows car le package utilisé ne fonctionne pas sous Ubuntu.

- **DBSCAN**

La fonction *dbscan* du package *dbscan* sous R suggère dans ses paramètres la possibilité d'ajout de poids pour les données traitées. Nous avons donc réalisé ce clustering avec comme paramètres MinPts = 2 et Eps = 20 km afin de pouvoir le comparer aux autres résultats.

E. Caractérisation des résultats

Afin de pouvoir interpréter les résultats obtenus et évaluer les méthodes retenues, nous avons défini les critères/métriques suivants :

- Nombre de clusters obtenus pour chaque méthode : **autrement dit le nombre de regroupements présentant plus de 2 points**

- Nombre de points isolés : les clusters à 1 point pour AGNES ou les points classés comme du bruit pour DBSCAN et HDBSCAN

- Nombre de clusters et de points isolés au niveau de zooms sur certaines zones denses en points et qui comportent plusieurs agglomérations selon la méthode actuelle de NetsCity :

Zones	Nb de points
Pays-Bas - Belgique	1230
Côte Est des Etats-Unis (Washington - NYC)	Environ 2200

Tab3. Nombre de clusters et de points isolés au niveau de zooms

En effet les zones denses sont souvent difficiles à traiter pour les algorithmes et c'est là où le plus d'erreurs seront à priori présentes.

- Nombre et pourcentage de clusters par méthode traversant soit une frontière administrative et soit un trait de côte :

Afin de déterminer si beaucoup de clusters intersectent des frontières ou des traits de côte, nous avons réalisé à partir des ensembles de points résultats des

enveloppes convexes par cluster. Ensuite nous avons déterminé si ces enveloppes sont totalement incluses dans les limites des pays ou non, ce qui permet de déterminer les nombres de clusters traversant une frontière ou un trait de côte et par conséquent de caractériser les performances des algorithmes et la qualité des résultats.

- Nous avons également zoomé sur certaines zones susceptibles d'avoir des clusters traversant des frontières ou traits de côte. Les résultats seront seulement qualitatifs :

Zones	Nombre de points
Royaume-Uni + Irlande + Nord de la France	2375
Nord de l'Allemagne + Danemark + Sud de la Suède	1677

Tab4. Zones susceptibles d'avoir des clusters traversant des frontières

Résultats et discussion

1. Influence du nombre des points traités

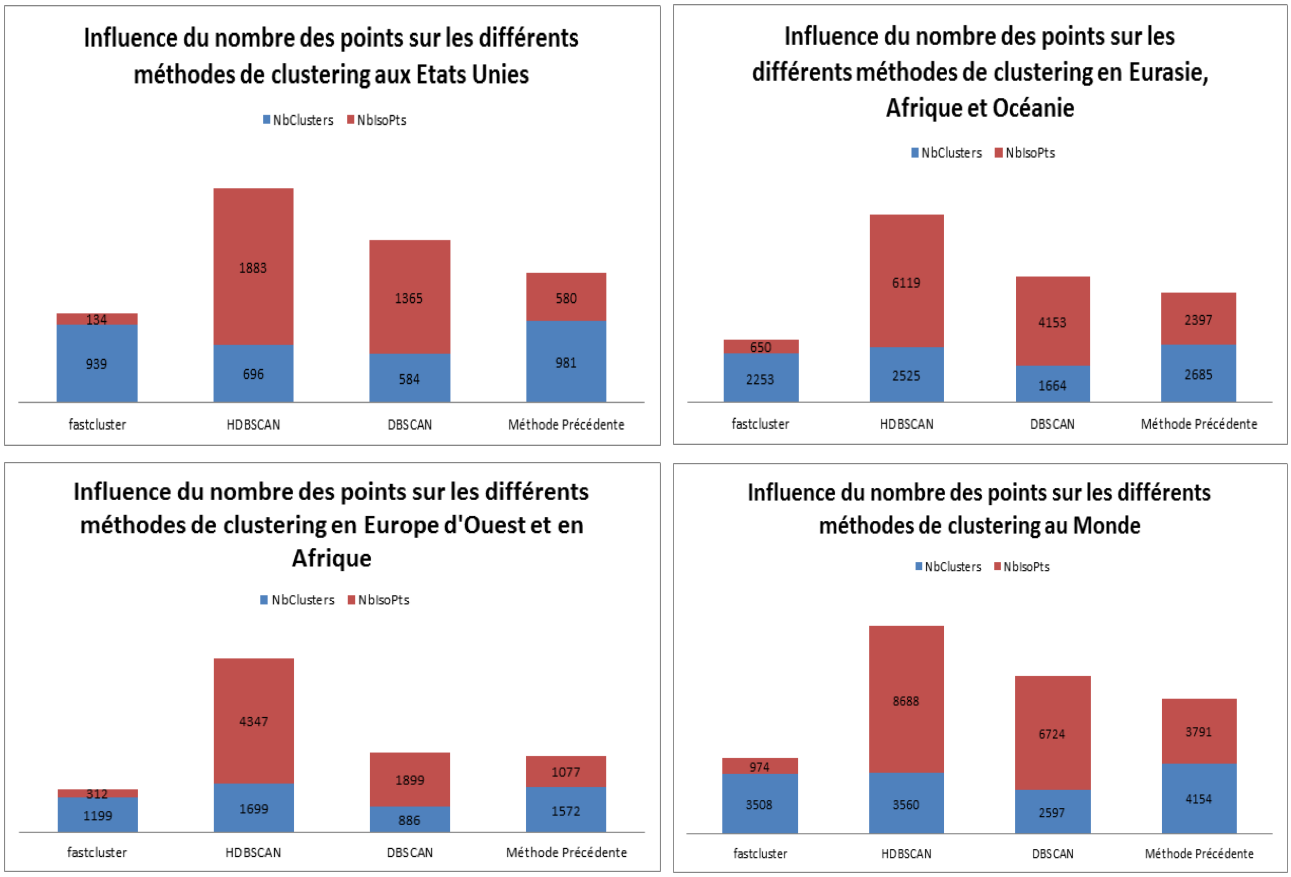


Fig9. Influence du nombre des points traités

Nous remarquons que chaque méthode de clustering fournit des résultats différents pour chaque région du monde. Le nombre de points affecte donc, évidemment, les résultats des algorithmes. A savoir, le nombre de clusters ainsi que le nombre de points isolés, augmente avec l'augmentation du nombre des points.

A première vue, c'est la méthode précédente qui produit le plus grand nombre de clusters par rapport aux nouvelles. En même temps elle génère un nombre de points isolés important.

Si nous regardons les résultats des nouvelles méthodes testées, nous notons que HDBSCAN génère à chaque fois le plus grand nombre de clusters sauf pour les Etats Unies où fastcluster l'a emporté et cela est peut-être dû à la concentration des points dans cette région qui peut influencer le fonctionnement de HDBSCAN puisqu'il est un algorithme basé sur la densité.

Parallèlement, HDBSCAN génère aussi le plus grand nombre de points isolés qui est égal à presque 2,5 fois le nombre de clusters.

A son tour, fastcluster produit un nombre de clusters important mais qui est proche de celui produit par HDBSCAN. L'avantage ici c'est que le nombre de points isolés est assez faible comparé aux autres méthodes.

Quant à DBSCAN, il paraît être l'algorithme le moins utile avec le plus faible nombre de clusters produits contre un nombre de points isolés très élevés. Cela est dû au fait que cette méthode est sensible aux grandes variations de densité.

Ce que nous pouvons donc retenir pour cette étape c'est que toutes les méthodes sont influencées par le nombre des points mais fastCluster semble se comporter plus efficacement.

2. Influence de la variation des paramètres de chaque méthode

A. Méthode agglomérative

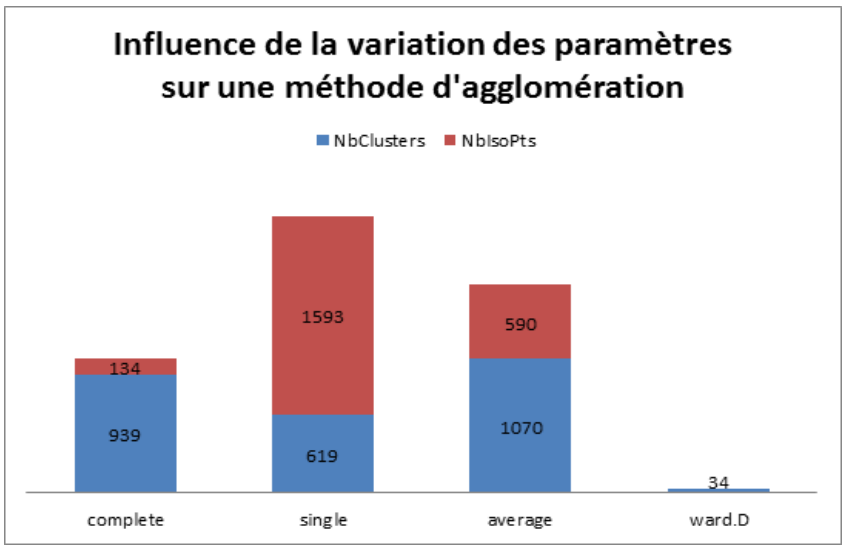


Fig10. Influence de la variation des paramètres sur la méthode agglomérative

Les résultats des différentes méthodes d'agglomération pour une classification hiérarchique sont très hétérogènes.

La méthode « complete » semble être la plus efficace avec un faible nombre de points isolés et un nombre de clusters important, suivie par la méthode « average » qui crée beaucoup plus de clusters mais aussi génère plus de points isolés.

« Single » occupe la troisième position dans l'échelle d'efficacité des méthodes avec un nombre de points isolés égal à 2,5 fois le nombre de clusters.

Et bien évidemment la méthode « Ward.D » n'est pas valable puisqu'elle génère de très gros clusters et pas de points isolés. Cela est expliqué par la hauteur de la coupe du dendrogramme qui ne convient pas entièrement pour cette méthode.

Utiliser un indice de stabilité ou de dissimilarité entre les clusters afin de déterminer automatiquement une hauteur de coupe appropriée pour chaque méthode d'agglomération permettrait à priori d'avoir des résultats fournissant plus d'information.

B. DBSCAN

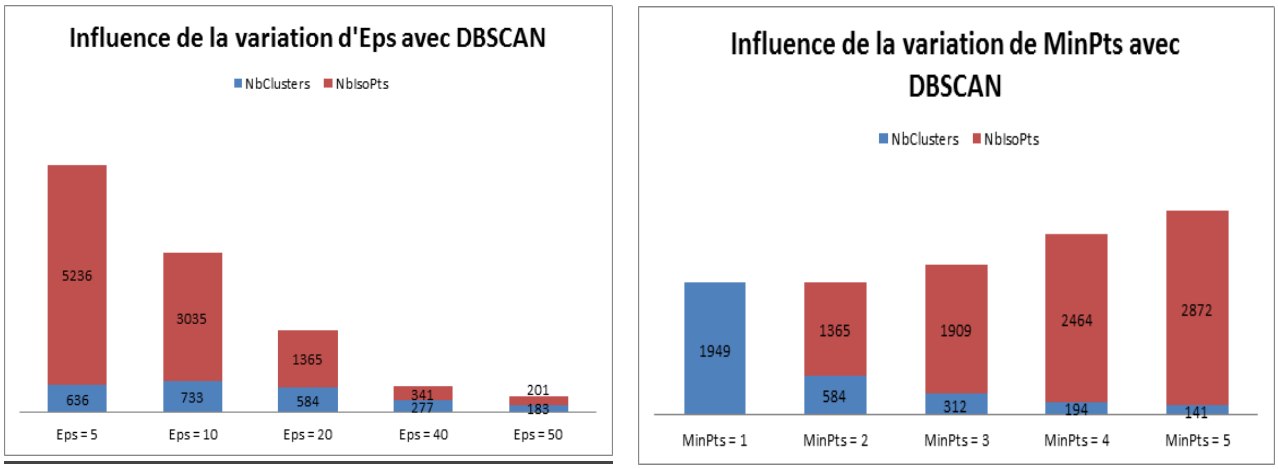


Fig11. Influence de la variation des paramètres sur DBSCAN

DBSCAN est très influencé par le changement de ses paramètres.

D'une part, pour un nombre minimum de points fixe (=2), la variation du rayon de recherche (Eps) agit énormément sur le nombre de clusters et points isolés formés. Ces derniers diminuent avec l'augmentation d'Eps, ce qui est cohérent.

Le nombre de points isolés générés est supérieur au nombre de clusters ce qui peut être expliqué par la sensibilité de cet algorithme à la densité étant donné la zone des Etats Unies comporte beaucoup de zones de densités variables.

Mais, nous pouvons noter qu'avec Eps = 20km un « optimum » de résultat de DBSCAN. En fait, le rapport nombre des clusters / nombre de points isolés peut être considéré comme acceptable en comparaison avec les résultats des autres valeurs d'Eps, car pour 7443 points (l'échantillon de points auquel nous avons appliqué DBSCAN) nous avons obtenu un bon nombre de clusters (584 clusters) pour un écart correct par rapport aux points isolés.

D'autre part, pour un Eps fixe à 20km, l'augmentation du nombre de points minimum formant un cluster contribue à l'augmentation du nombre des points isolés contre une diminution remarquable des clusters, ce qui est également cohérent avec la manière dont fonctionne DBSCAN.

C. HDBSCAN

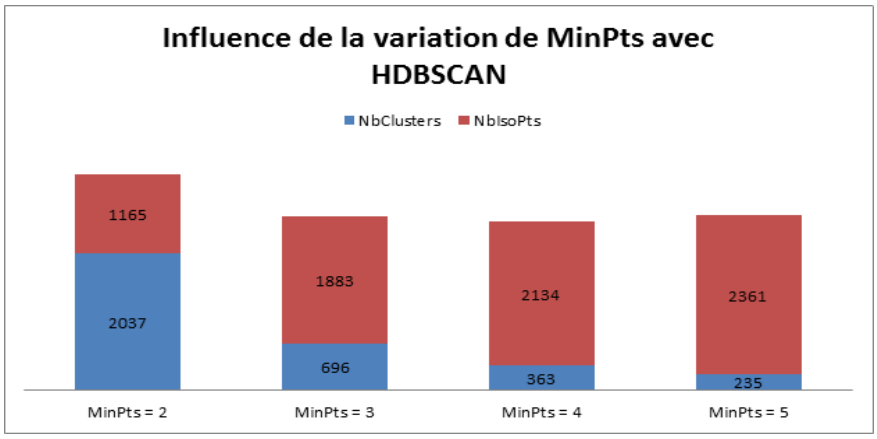


Fig12. Influence de la variation des paramètres sur HDBSCAN

Comme DBSCAN, l'augmentation du nombre de points minimum pour HDBSCAN engendre une augmentation des points isolés et une diminution des clusters.

L'optimum de fonctionnement de cette méthode dans une telle région dense est à MinPts= 2

3. Introduction d'un paramètre non spatial par pondération

A. Méthode de classification hiérarchique agglomérative

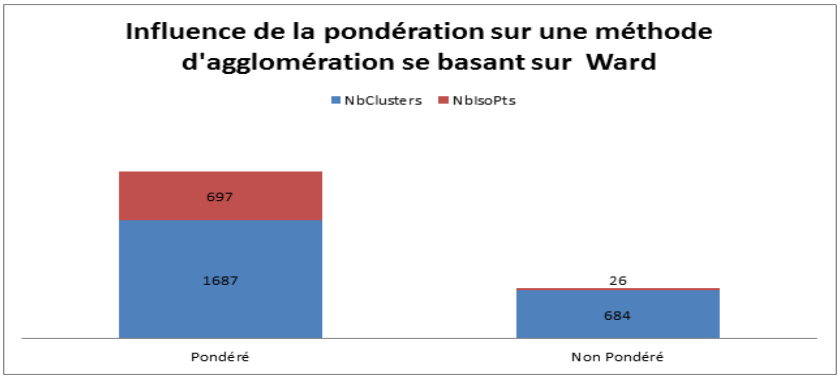


Fig13. Influence de l'introduction d'un paramètre non spatial par pondération sur une méthode d'agglomération se basant sur Ward

En baissant la hauteur de coupe au niveau du dendrogramme, nous constatons que la méthode Ward génère bien plus de clusters ainsi que quelques points isolés.

On observe que l'ajout du nombre de publications comme poids a beaucoup d'influence sur les résultats. Le nombre de clusters entre le traitement non pondéré et pondéré a augmenté de plus de 2,5. Cela peut s'expliquer par le fait que les points avec un poids élevé vont avoir par conséquent plus d'importance. Or si c'est le cas de beaucoup de points, l'unité de certains clusters peut être moins importante qu'avant et ces derniers vont être divisés en plusieurs clusters.

B. DBSCAN

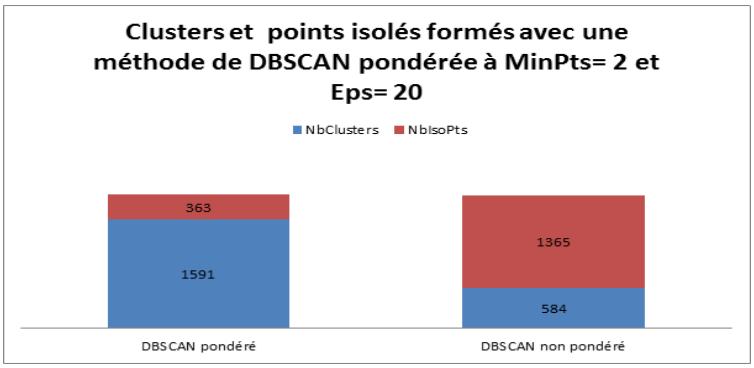


Fig14. Influence de l'introduction d'un paramètre non spatial par pondération sur DBSCAN

La méthode de DBSCAN pondérée crée 3 fois plus de clusters que celle non pondérée et diminue à l'un quart presque les points isolés.

Comme précédemment l'unité de certains clusters peut être moins importante qu'avant à cause de l'importance de points ayant un poids élevé, entraînant un nombre de cluster plus important.

4. Les zones denses

A. La Cote Est USA

- **Influence du nombre de points**

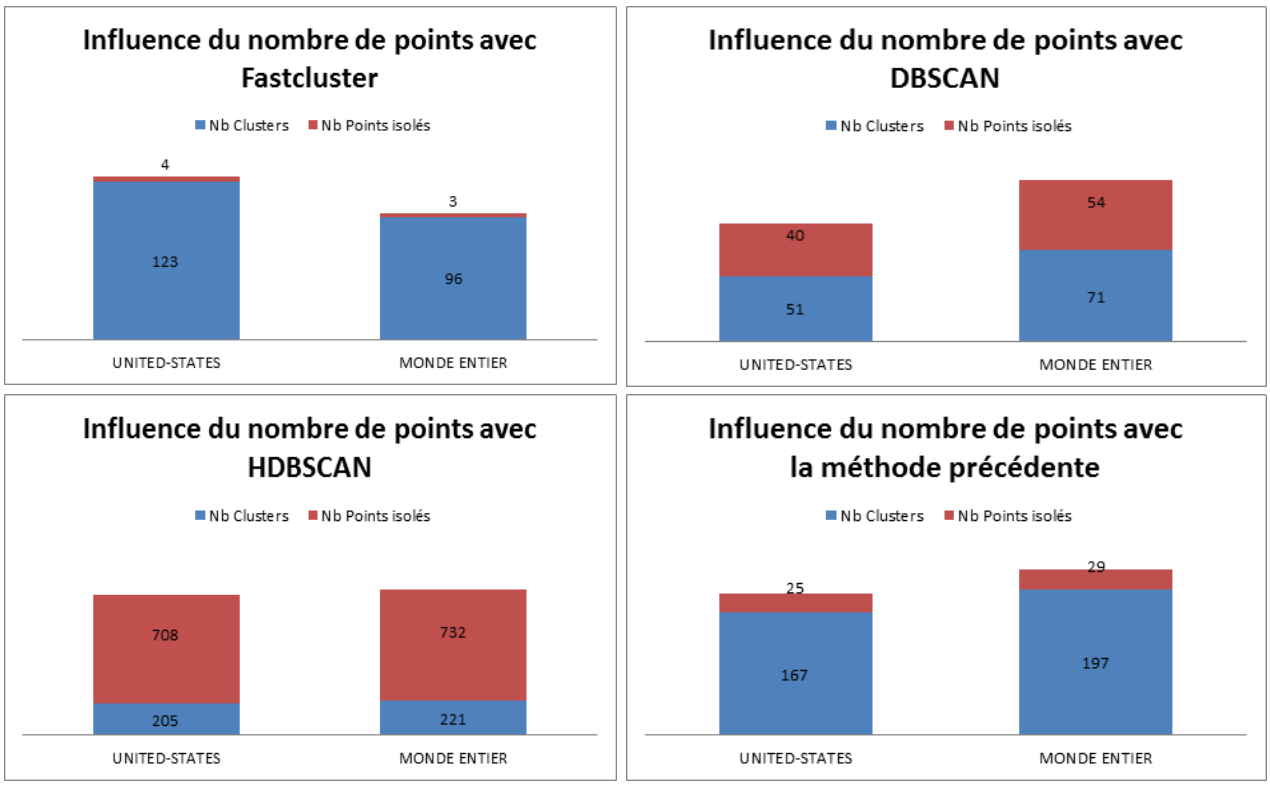


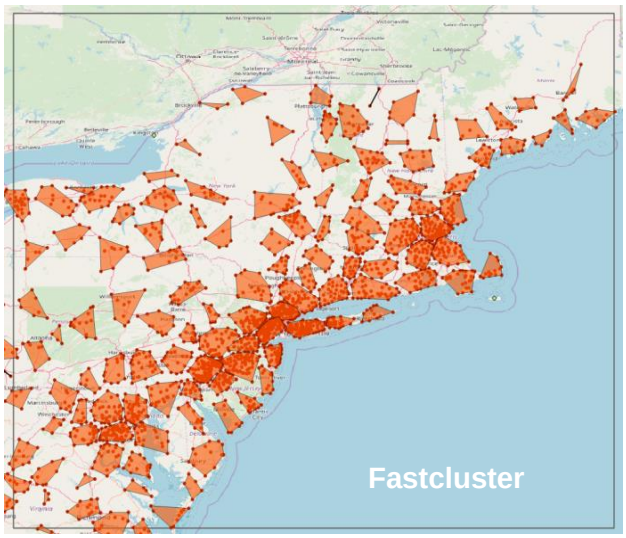
Fig15. Influence du nombre de points dans la côte Est des Etats Unies

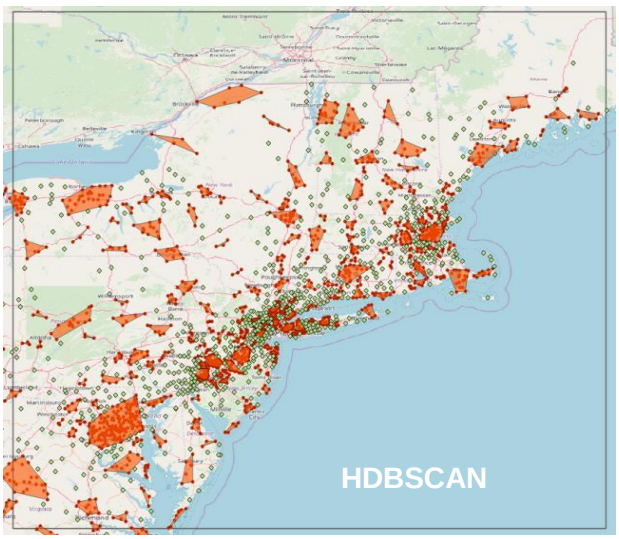
Ici le nombre de clusters obtenus au niveau de la côte Est des Etats-Unis est quasi identique que le jeu de points soit seulement les Etat-Unis ou bien le monde entier. Il est même plus élevé pour le jeu des Etats-Unis avec la méthode fastcluster.

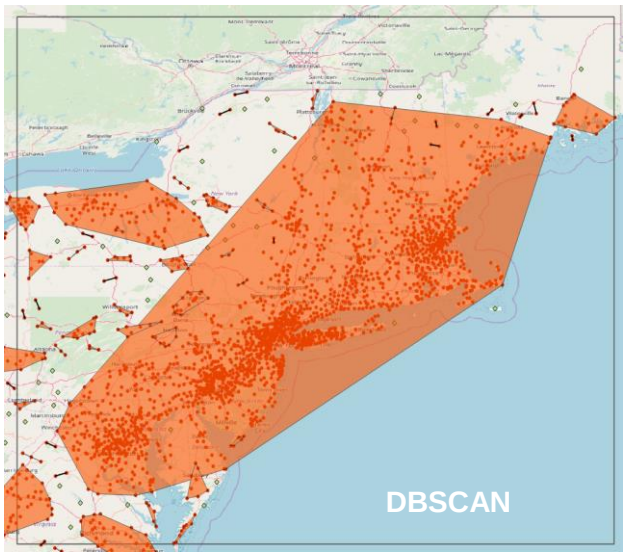
Cette dernière génère moins de points isolés que la méthode précédente.
Tandis que les 2 méthodes se basant sur la densité (HDBSCAN et DBSCAN) produisent beaucoup de points isolés aussi bien que de clusters (3 fois plus de points isolés que de clusters pour HDBSCAN) ce qui démontre encore une fois les tendances observées précédemment pour DBSCAN et HDBSCAN à classer énormément de point comme du bruit.

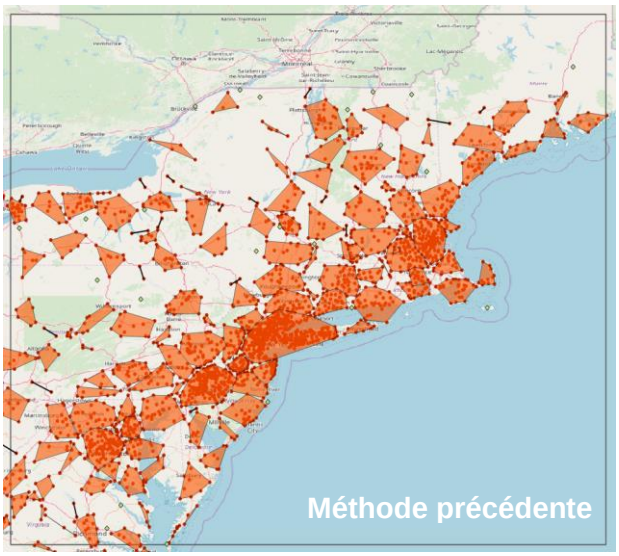
Visualisation des résultats

Pour le jeu de données limité aux Etats-Unis

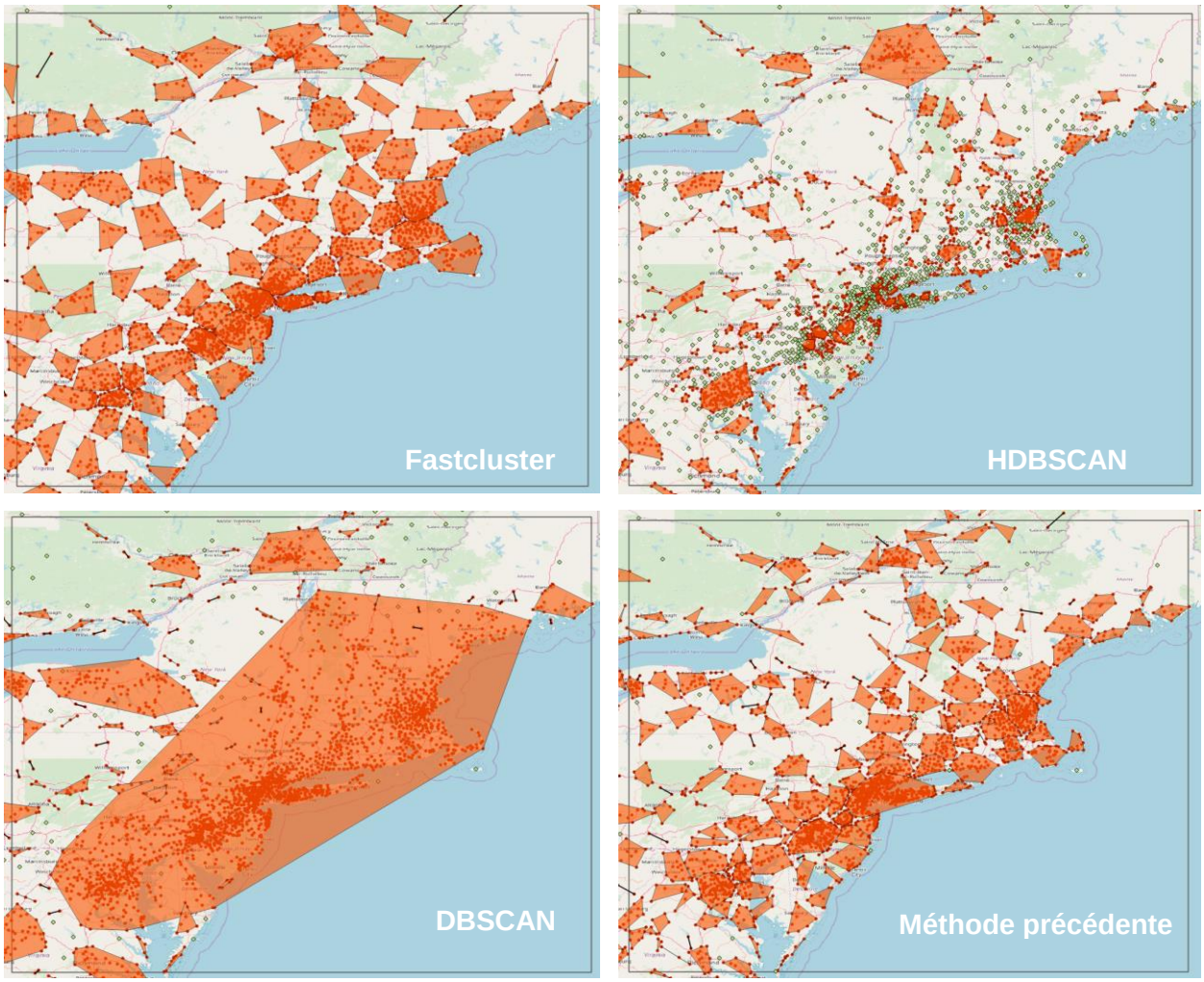








Pour le jeu de données du monde entier



Les résultats sont assez consistants selon que l'on constate.

- **Influence des paramètres**

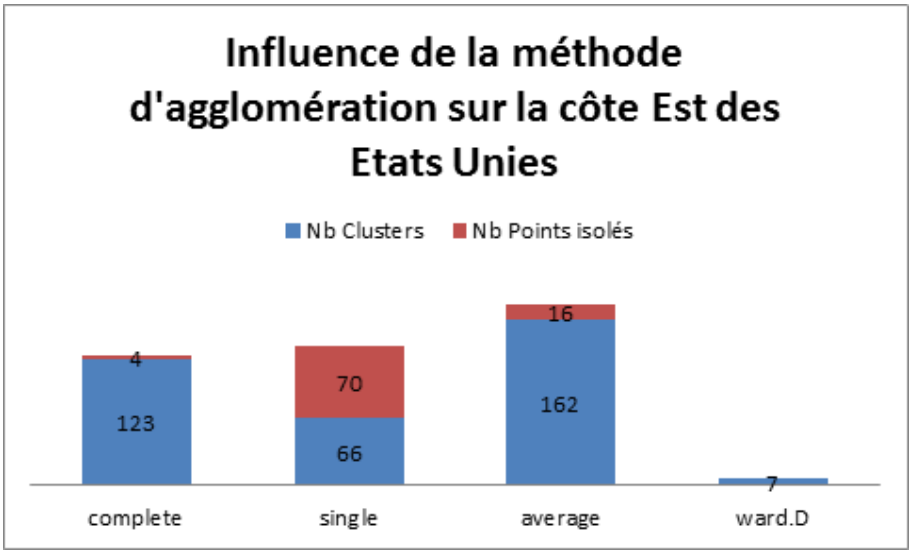
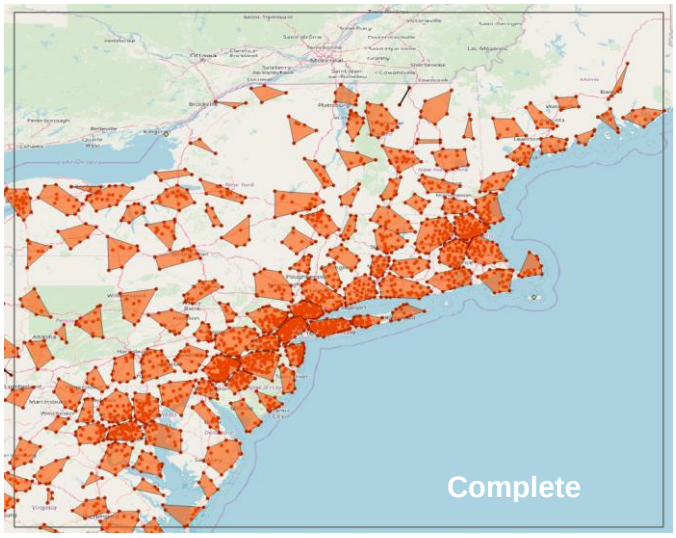
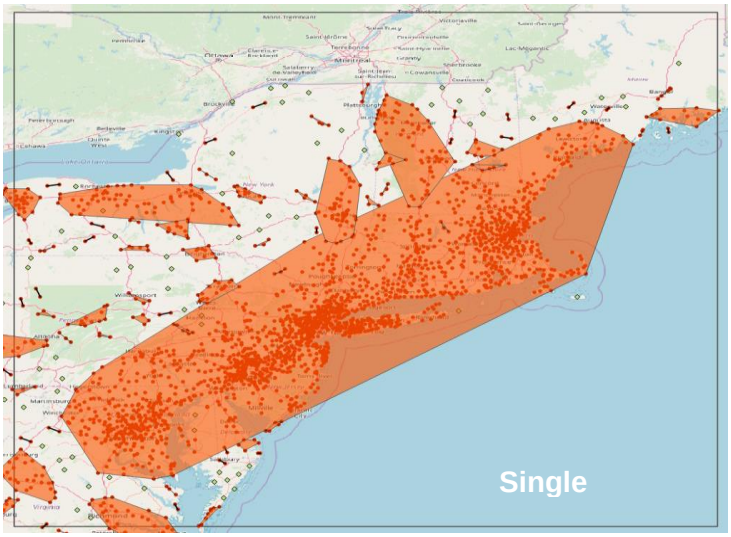
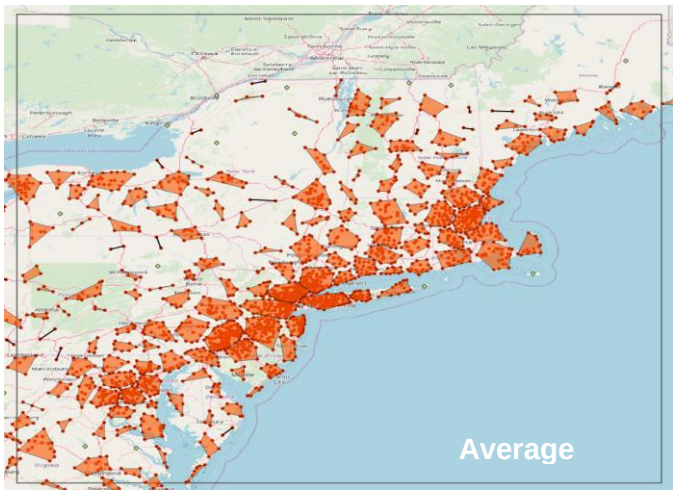


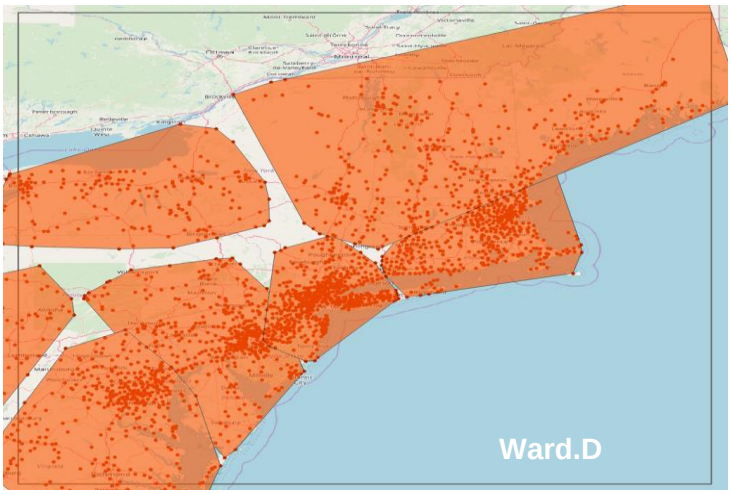
Fig16. Influence des paramètres sur la méthode d'agglo dans la côte Est des Etats Unies

Visualisation des résultats :









L'algorithme d'agglomération hiérarchique, dans la côte Est des Etats Unies, manifeste le même comportement (vu dans la partie précédente) face au changement de la méthode d'agglomération.

En effet, "complete" suivie par "average" forment les 2 méthodes les plus efficaces en clustering surtout dans les régions denses. Comme précédemment, les résultats pour la méthode de type Ward diffère grandement ce qui est à priori dû à la hauteur de coupe qui n'est pas forcément adaptée bien qu'elle soit relative à la hauteur maximale du dendrogramme.

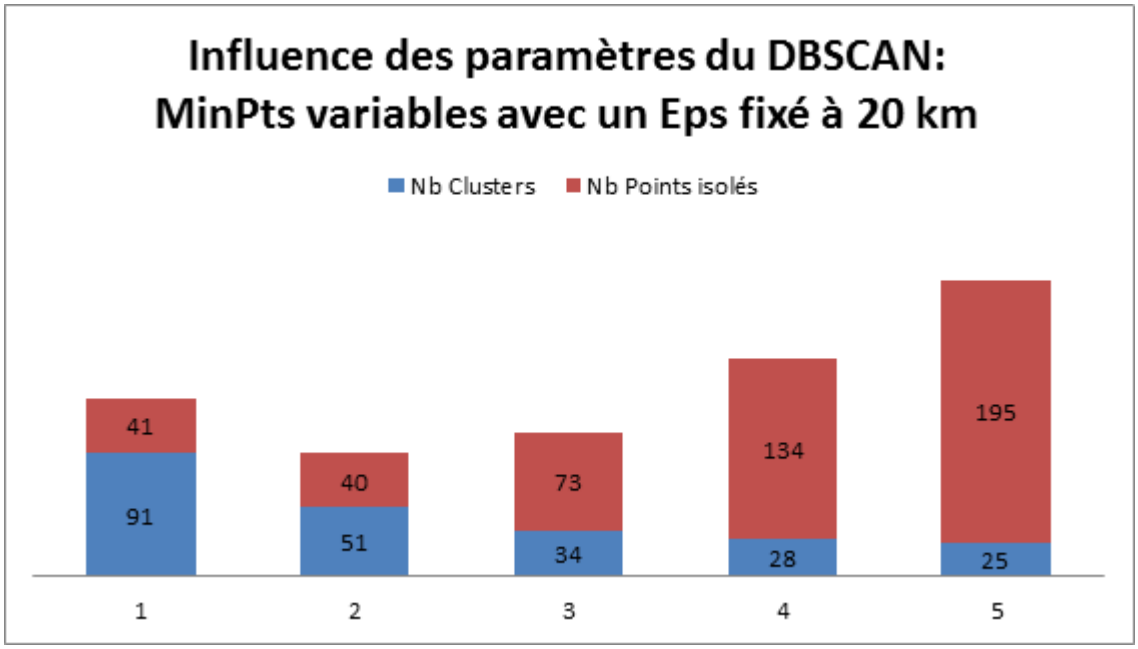
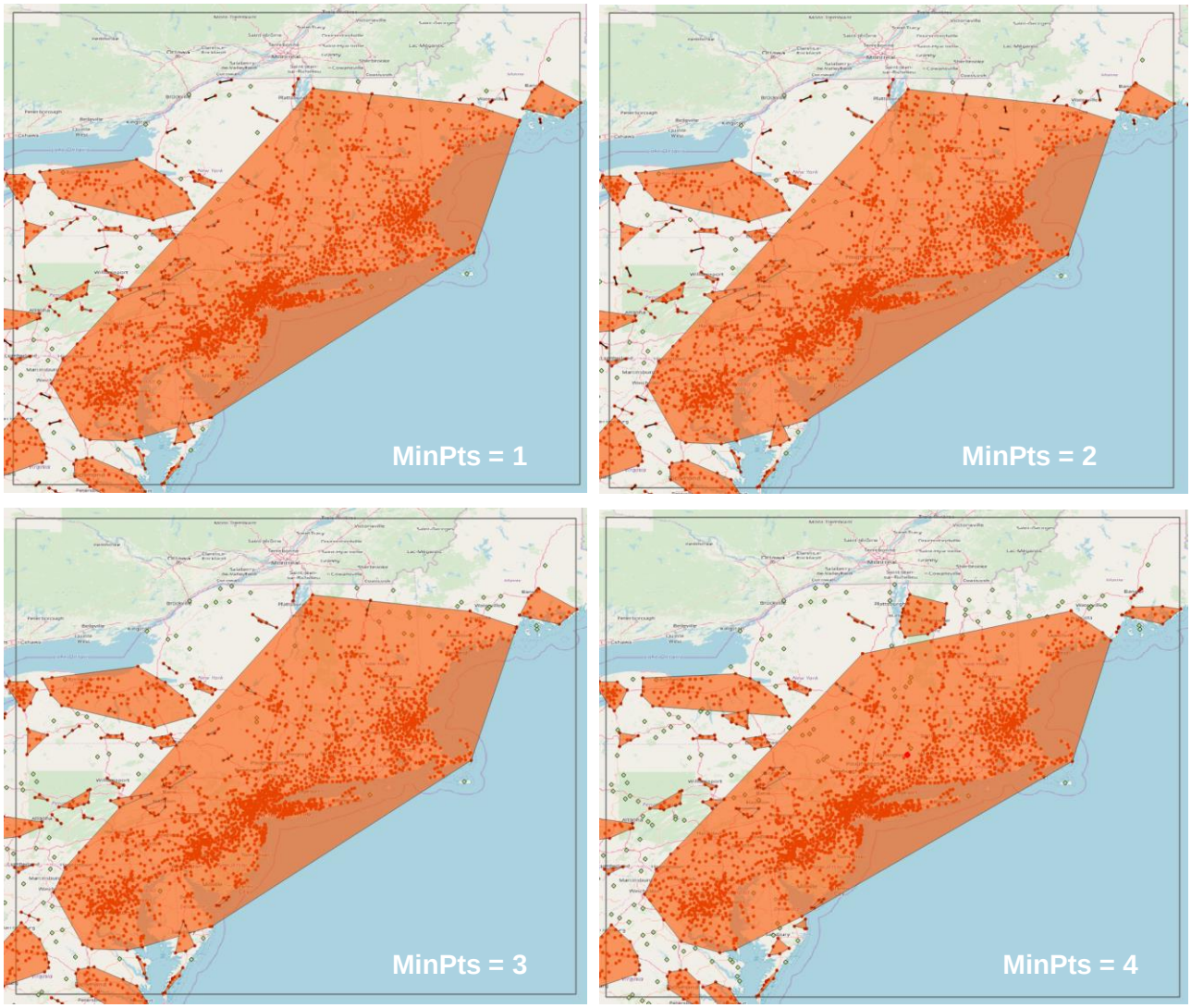
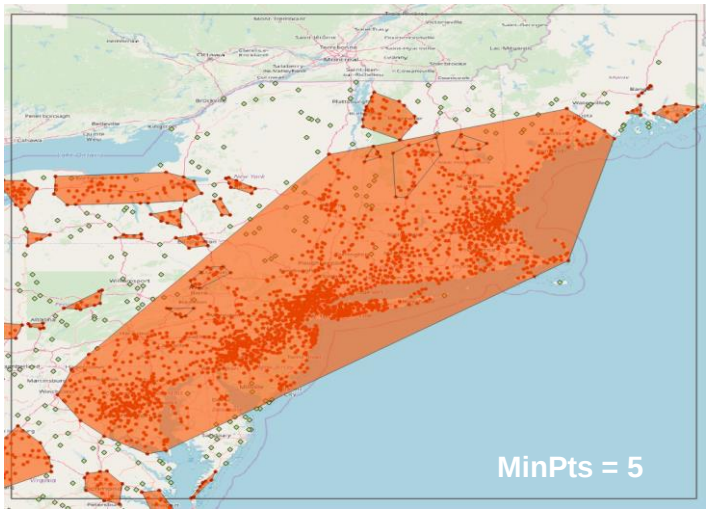


Fig17. Influence des paramètres sur DBSCAN dans la côte Est des Etats Unies





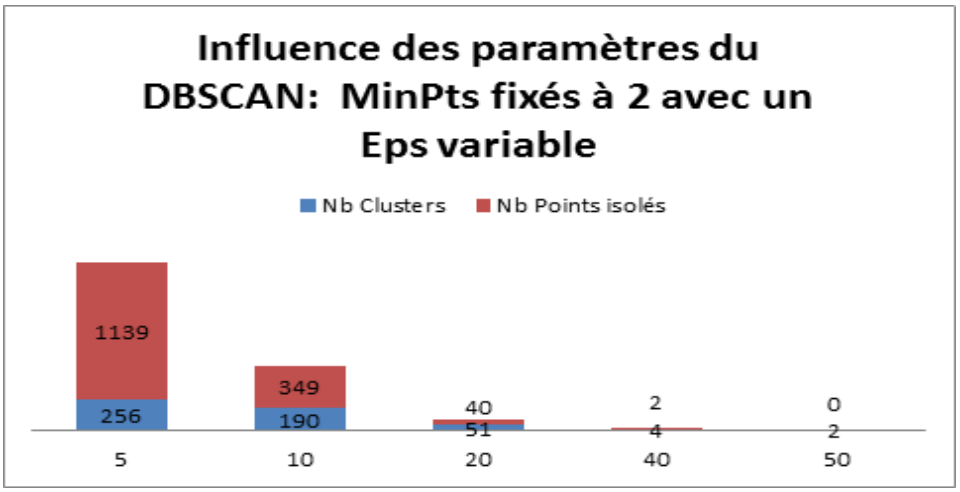
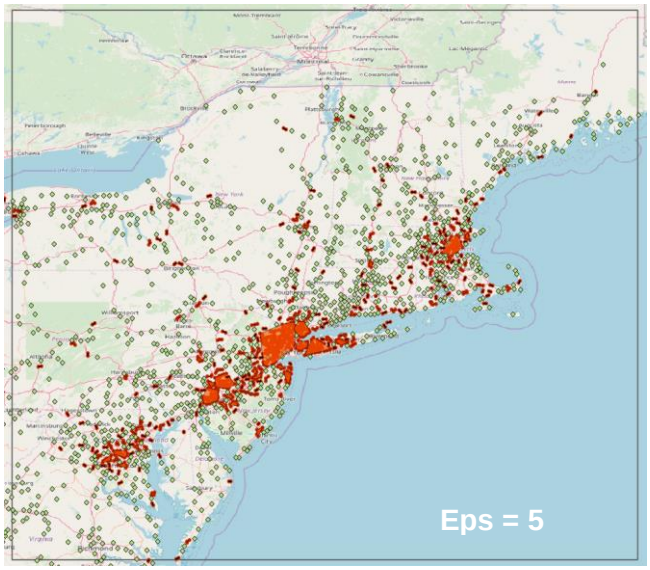
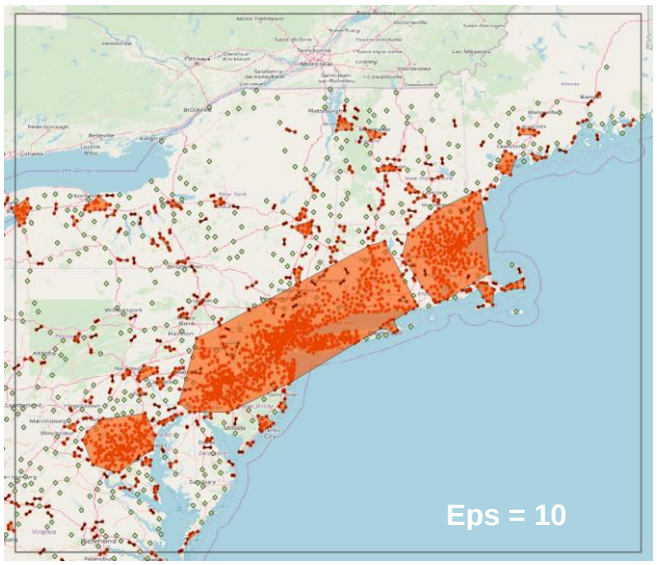
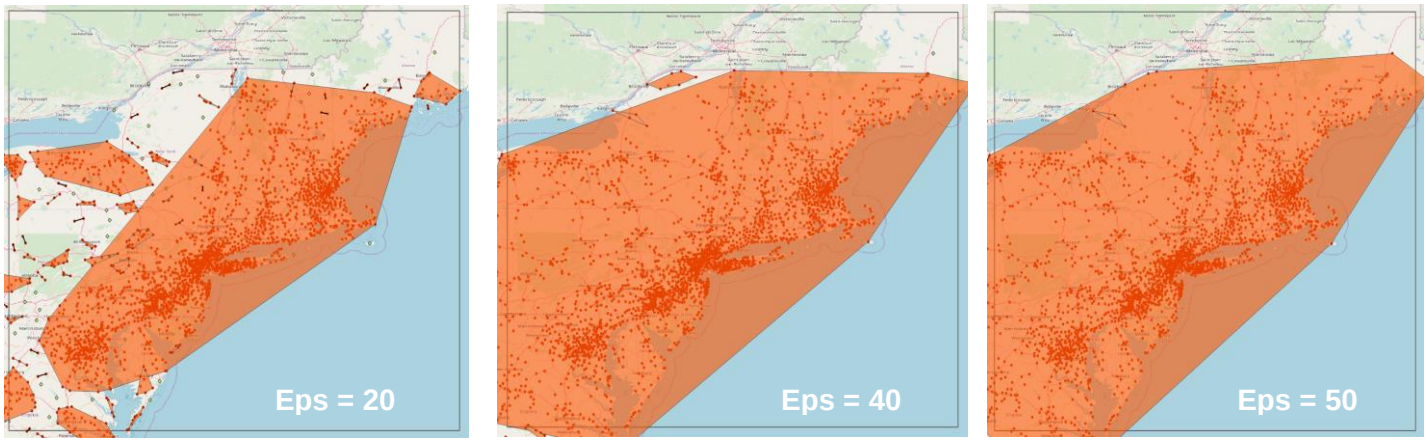


Fig17. Influence des paramètres sur DBSCAN dans la côte Est des Etats Unies







Pareil, DBSCAN montre les mêmes résultats vus dans la partie précédente avec, éventuellement, une diminution de nombre de clusters avec l'augmentation des MinPts et aussi d'Eps et une variation proportionnelle du nombre des points isolés.
Le paramétrage « idéal » est remarqué à un Eps= 20km avec 2 points minimum.

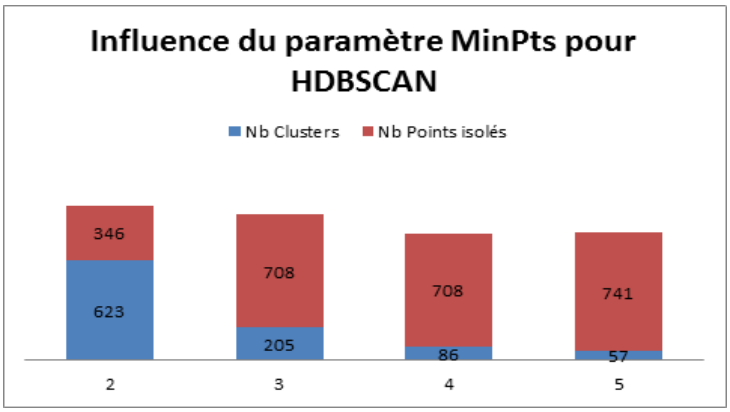
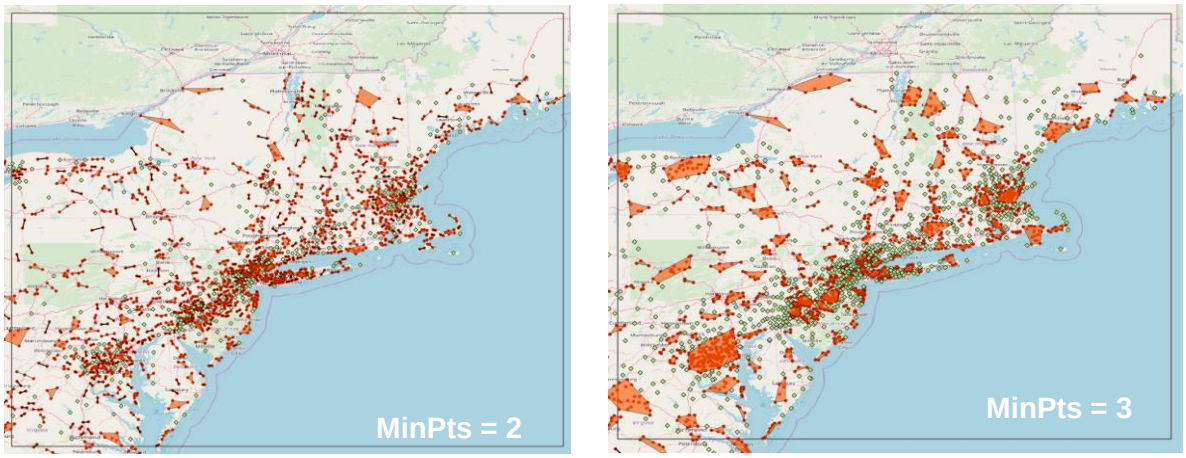
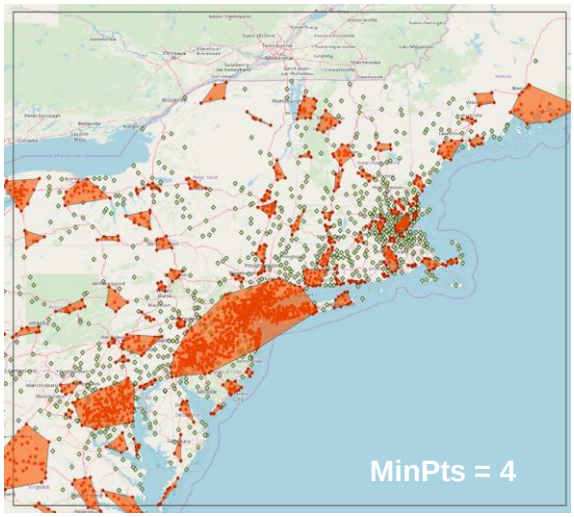
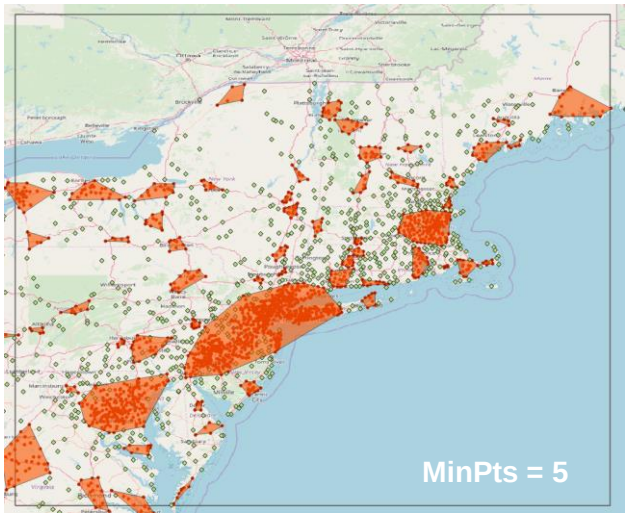


Fig18. Influence des paramètres sur HDBSCAN dans la côte Est des Etats Unies

HDBSCAN témoigne une augmentation des points isolés et une diminution des clusters face à l'augmentation du nombre de points minimum. Identiquement à DBSCAN, l'optimum de fonctionnement de HDBSCAN dans une telle zone dense est à MinPts = 2.







- **Influence de la pondération**

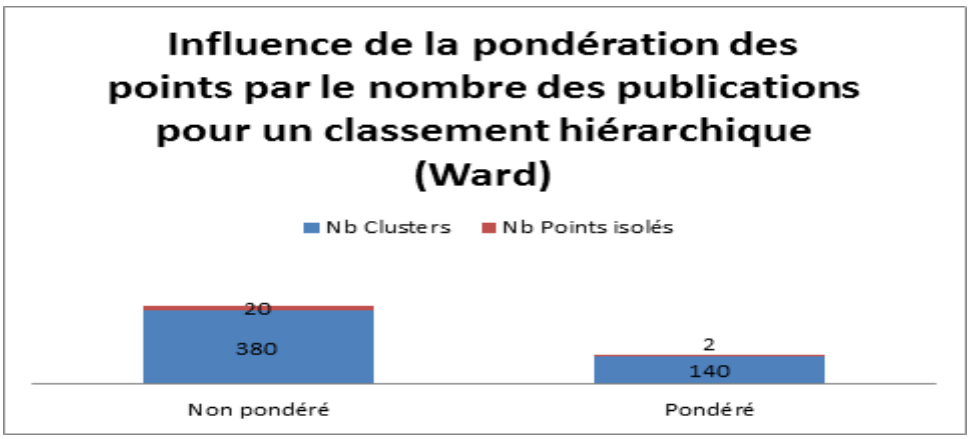
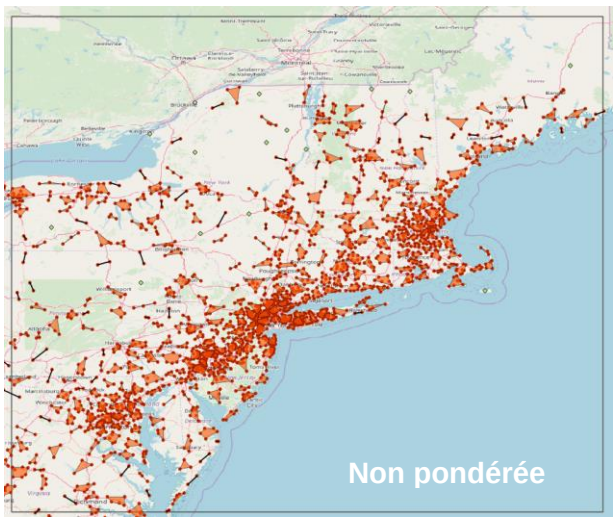
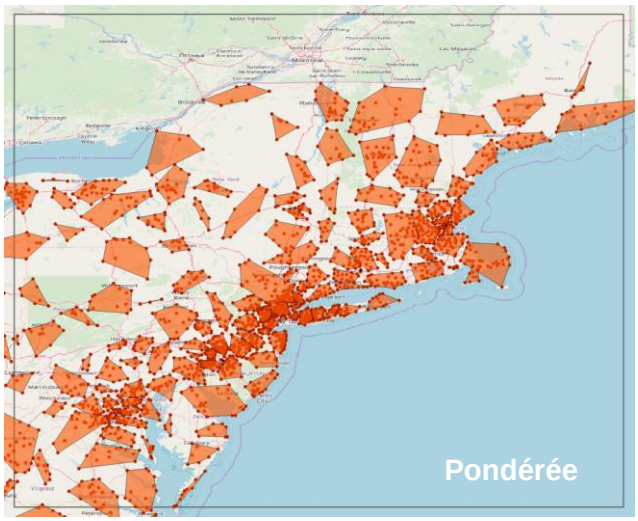


Fig19. Influence de la pondération sur Ward dans la côte Est des Etats Unies

Contrairement à ce qui est vu précédemment, la pondération avec le nombre de publications cette fois a diminué le nombre de clusters et le nombre des points isolés. Cela montre la pondération a toujours un effet important même dans les zones denses.





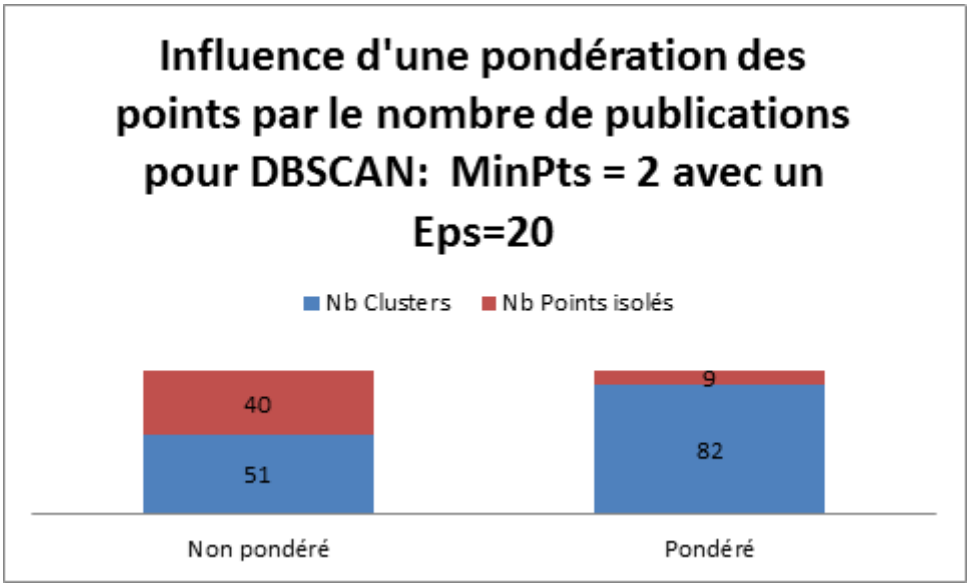
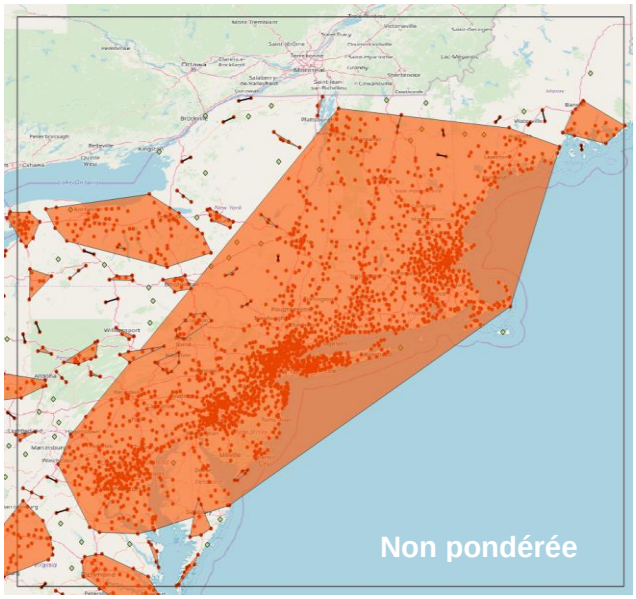
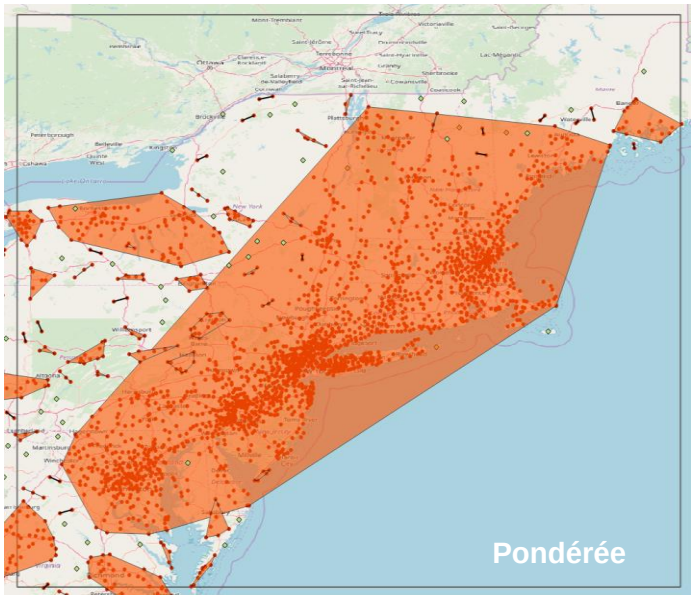


Fig20. Influence de la pondération sur DBSCAN dans la côte Est des Etats Unies

La méthode de DBSCAN non pondérée produit moins de clusters que celle pondérée et diminue énormément les points isolés, ce qui affirme l'importance du poids élevé de certains points qui engendre l'augmentation du nombre de cluster.





B. Pays-Bas - Belgique

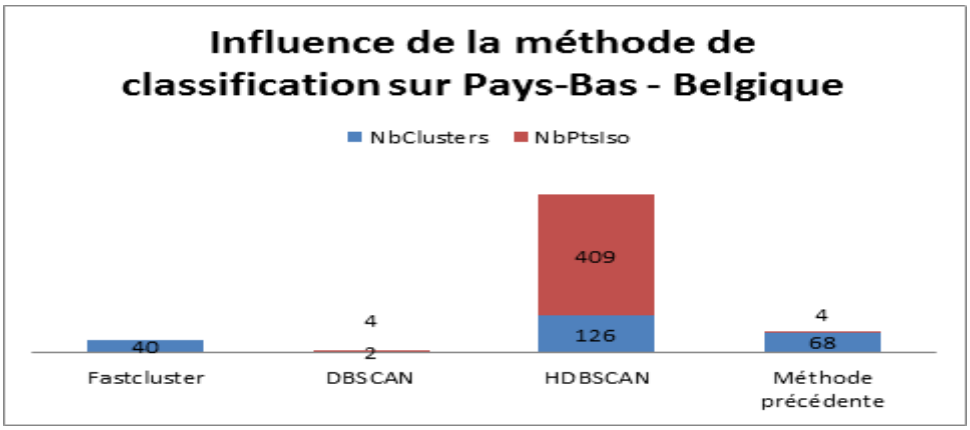
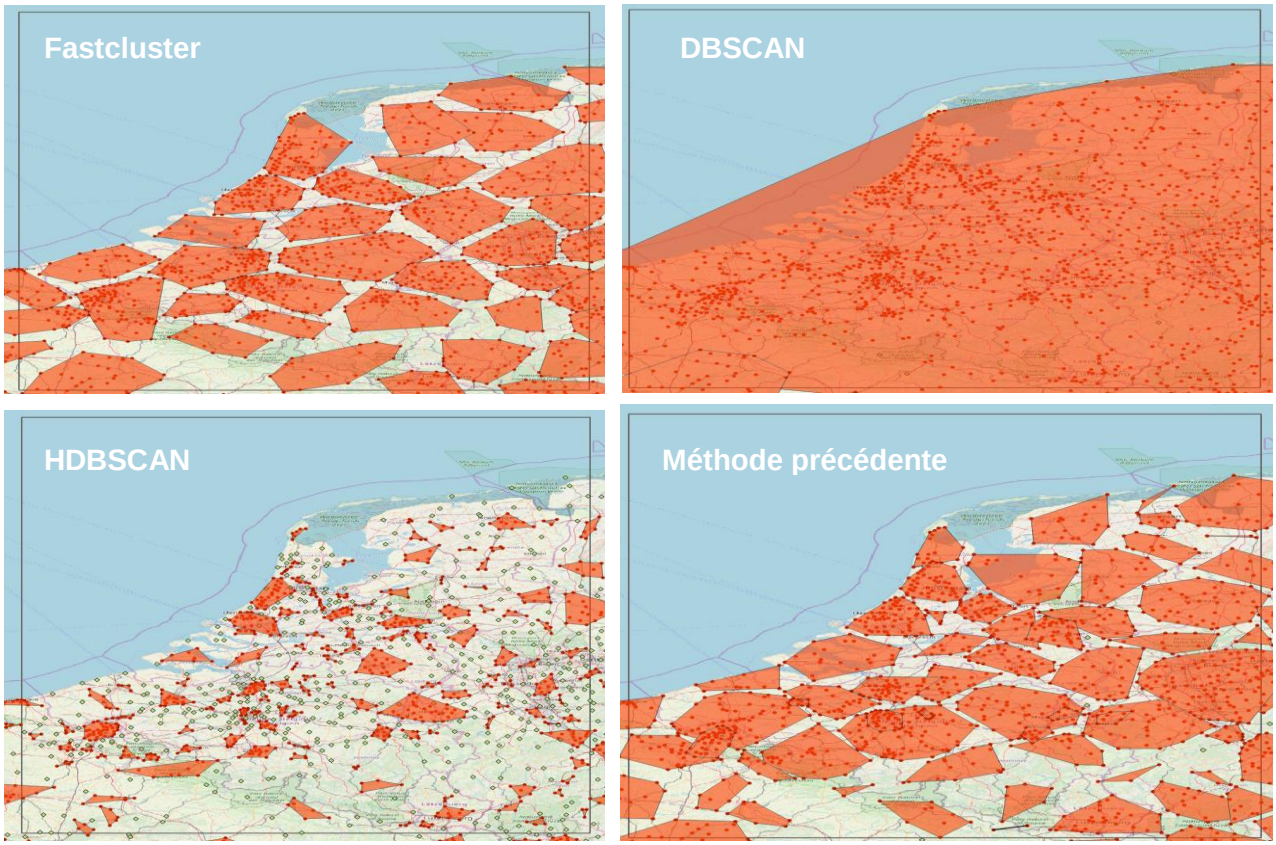


Fig21. Influence de la méthode de classification sur les Pays-Bas - Belgique

Pour cette zone dense, fastCluster semble être le plus adéquat puisqu'il a réussi à créer des clusters sans laisser de points isolés.
Quant à DBSAN et HDBSAN, ils ont montré des résultats inadaptés en créant, respectivement, 2 gros clusters qui couvrent toute la région et 126 clusters avec 3 fois de plus de points bruit.



5. Effet des frontières administratives et des traits de côte

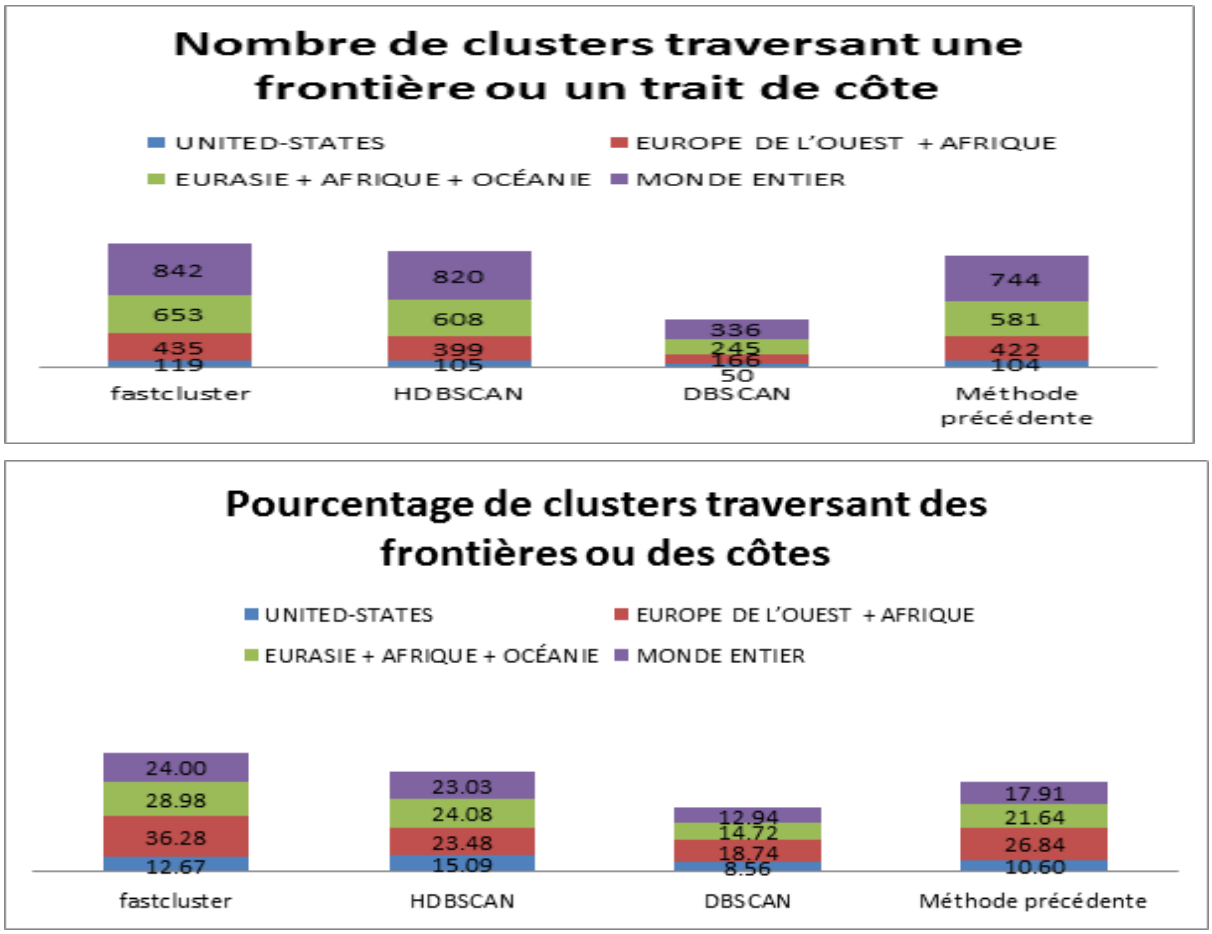


Fig22. Nombre et pourcentage des clusters traversant des frontières ou des côtes

D'après les résultats ci-dessus, DBSCAN est la méthode présentant le moins de clusters traversant des frontières ou des traits de côtes, aussi bien en valeur absolue qu'en pourcentage. Cela peut être potentiellement dû au fait que DBSCAN semble en général générer moins de clusters mais que ceux-ci ont une aire plus importante que pour les autres méthodes. Ainsi la part de clusters traversant une frontière ou une côte est mécaniquement réduite.

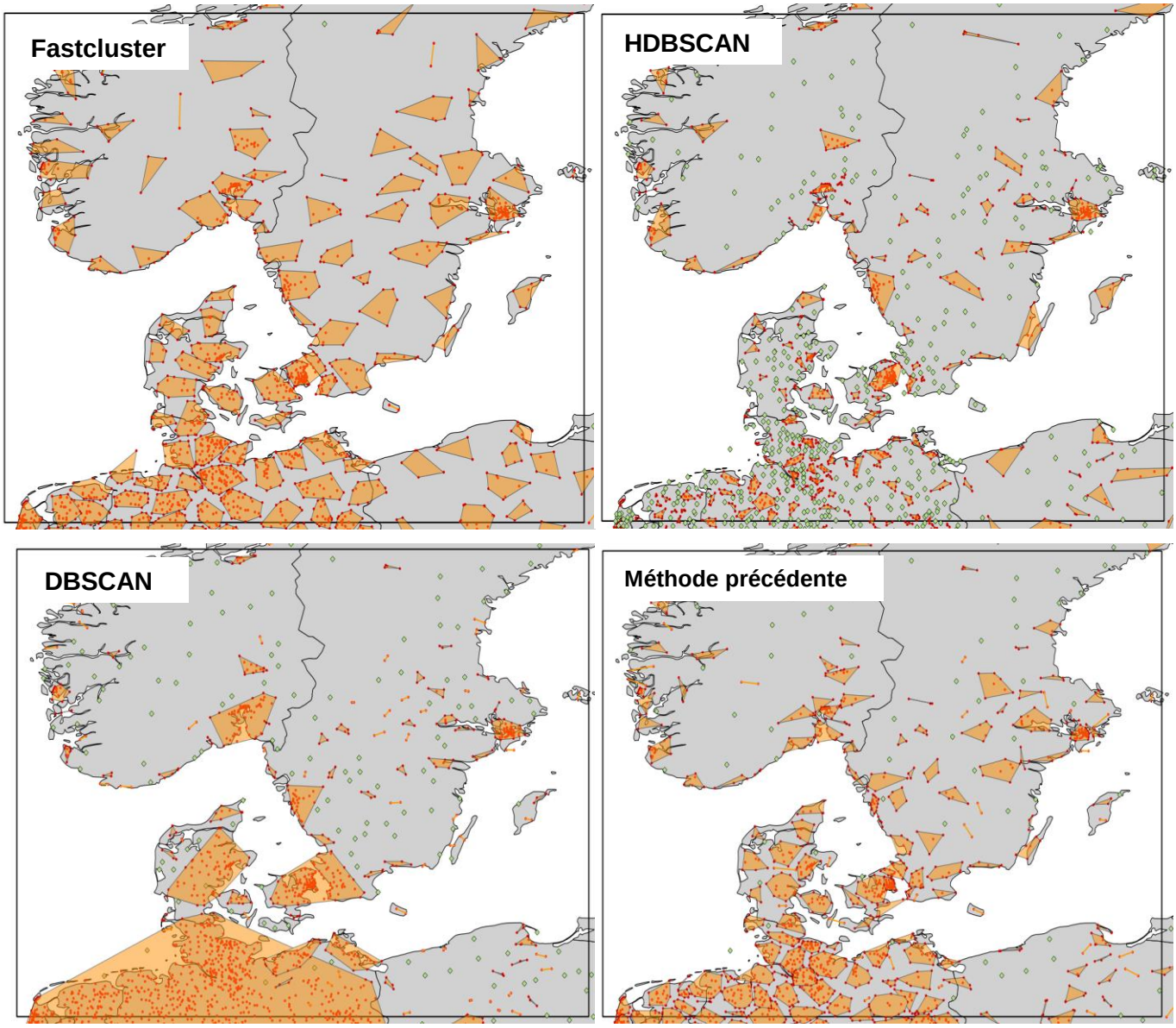
Autrement la méthode utilisée précédemment pour NetsCity présente à chaque fois les meilleurs résultats même si les écarts ne sont pas énormes (entre 4% et 8% selon l'ensemble de points considéré).

De manière générale, le pourcentage de clusters traversant une limite se situe entre 10 et 25% (hormis pour la méthode fastcluster avec le jeu de points d'Europe de l'Ouest et d'Afrique qui dont 36% des clusters sont concernés). Il y a donc encore des améliorations à faire sur ce plan. Un traitement par pays n'est pas envisageable car cela biaiserait trop les résultats. Néanmoins il serait possible par exemple de traiter cela manuellement après l'étape de clustering ou d'utiliser une méthode permettant l'indication d'obstacles.

Zooms sur les zones présentant des traits de côte tortueux

Le nombre de points de l'ensemble considéré à pas ou peu d'influence sur l'aspect spatial des résultats pour ces deux zones de zooms. Ainsi seulement les résultats pour un ensemble de points seront présentés.

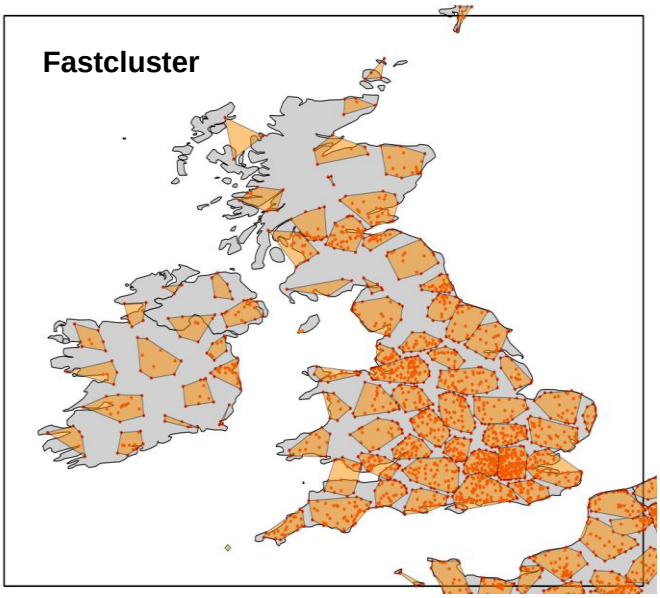
A. Nord de l'Allemagne - Danemark - Sud de la Scandinavie (Mer Baltique)

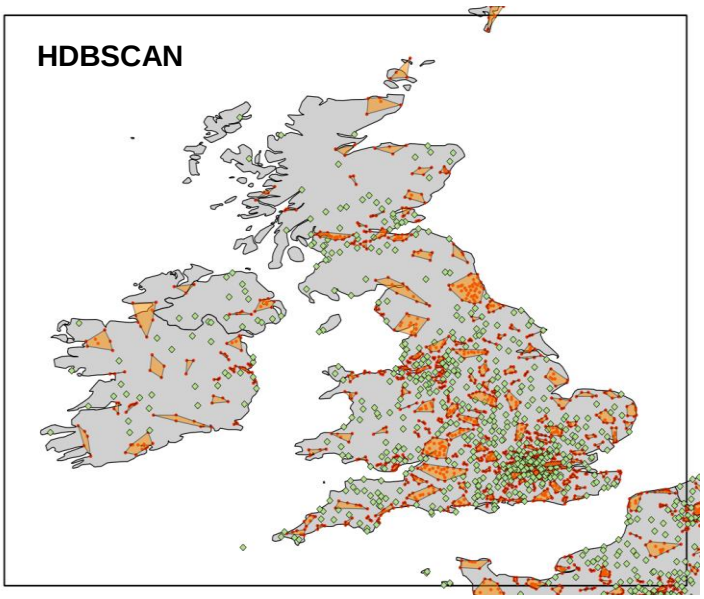


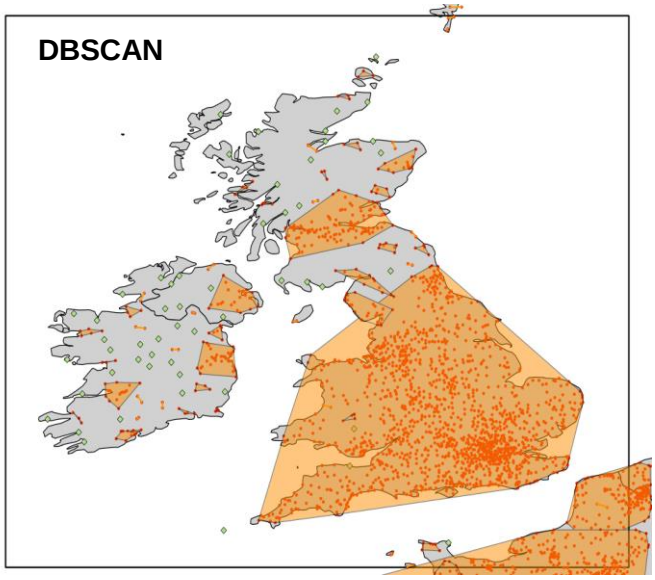
De manière générale, beaucoup de traits de côte sont franchis. La méthode HDBSCAN semble présenter le moins de clusters traversant une limite du fait de leur nombre et taille moins importants que dans les autres méthodes.

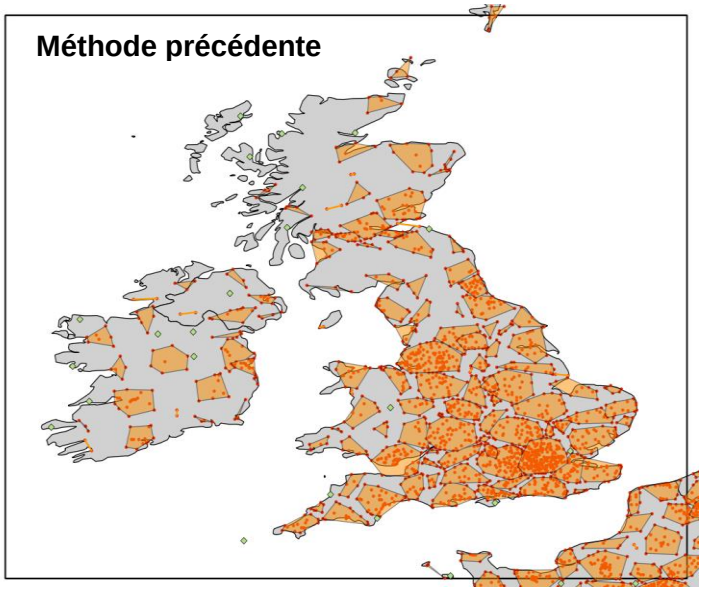
A chaque fois des points au Danemark et en Suède sont associés. Bien qu'il existe un pont entre le Danemark et la Suède la liaison reste partielle.

B. Royaume-Uni - Irlande









De même que précédemment beaucoup de clusters traversent un trait de côte ou bien un frontière administrative. Particulièrement en Ecosse des localités sur des petites îles sont regroupées avec des localités de l'île principale (au Nord-Ouest de la carte pour fastcluster par exemple) ou bien au Sud-Ouest de l'Angleterre.

Une nouvelle fois HDBSCAN semble présenter moins de clusters traversant des limites par son nombre plus restreint de clusters et de leur taille.

Autres méthodes

Comme a été mentionné dans la partie "Méthodologie", il existe évidemment d'autres méthodes de clustering.

1. Classification hiérarchique

A. DIANA (Divisive Analysis)

A l'inverse des méthodes agglomérative, cette technique construit la hiérarchie dans l'ordre inverse. Elle considère, en première étape, que tous les objets font partie d'un grand cluster. A chaque étape suivante, elle divise ce plus grand cluster en 2 petits groupes et ainsi de suite.
L'inconvénient de cette méthode c'est qu'elle ne prend généralement en considération les attributs des objets spatiaux

2. Classification basée sur la densité

Les clusters sont considérés comme des régions (pouvant avoir des formes arbitraires) des objets dans l'espace des données dans lequel les objets sont denses et séparés par des régions de faible densité d'objets (bruit).

A. DENCLUE (DENsity based CLUstEring)

Cet algorithme modélise la densité globale des points en utilisant la somme des fonctions d'influence des points. D'ailleurs, la détermination des attracteurs de densité provoque l'identification des clusters, même s'ils ont une forme arbitraire.
L'approche de DENCLUE est basée sur la concept que l'influence de chaque point de données sur son le voisinage peut être modélisée mathématiquement.
L'avantage de cet algorithme c'est qu'il arrive à gérer les clusters dans des ensembles de données ayant beaucoup de bruits.

3. Classification hiérarchique basée sur un modèle dynamique

A. CHAMELEON

Cette méthode de clustering fonctionne sur 2 étapes. Une première où elle utilise un algorithme de partitionnement graphique pour regrouper les données en un grand nombre de petits sous-groupes. Une deuxième qui utilise un algorithme de classification hiérarchique agglomératif pour trouver les véritables clusters en combinant ensemble - à plusieurs reprises- ces sous-groupes.

Son défaut c'est qu'elle ne traite pas un grand nombre de jeu de données. Le traitement est limité à un maximum de 10 000 points.

4. Méthode de partition

Cette méthode classe les données en groupes selon 2 critères: chaque cluster doit au moins contenir un point et un point peut appartenir à plus qu'un cluster.

A. K-means

La procédure suit un moyen simple et facile de classer ensemble un jeu de données via un certain nombre de clusters (fixes a priori).
L'idée principale est de définir des centroïdes (un centroïde pour chaque cluster) placés les plus loins possibles, les uns aux autres. Puis, il faut assigner chaque point au groupe qui a le centroïde le plus proche. Une fois que tous les points sont assignés à un cluster.
L'algorithme recalcule la position des centroïdes et fait plusieurs itérations jusqu'à ce que les centroïdes ne bougent plus.

B. K-medoids

Contrairement à K-means, k-medoids choisit les points de données comme centres (médoïdes) et travaille avec une arbitraire matrice de distances.
Un médoïde peut être défini comme l'objet d'un cluster dont la dissimilarité moyenne par rapport à tous les objets du cluster est minimal, c'est-à-dire qu'il s'agit d'un point très central dans le cluster.

- Le principal inconvénient de K-means et K-Medoids c'est que ces méthodes sont sensibles au bruit et au point central et ils peuvent détecter uniquement les clusters sphériques.

5. Classification basée sur une grille

Les méthodes basées sur la grille quantifient l'espace d'un objet en un nombre fini de cellules et puis effectuent les opérations requises sur l'espace quantifié.
Le principal avantage de cette méthode c'est son temps de traitement rapide qui dépend du nombre de cellules dans chaque dimension de l'espace quantifié.

A. STING (STatistical INformation Grid based method)

Le principe de cette méthode est de diviser la zone spatiale en cellules rectangulaires (en fonction de la latitude et de la longitude) à différents niveaux de résolution (structure hiérarchique).
Chaque cellule d'un niveau supérieur est partitionnée en un nombre de cellules à son niveau inférieur immédiat et ainsi de suite.

Les paramètres statistiques (les valeurs minimale et maximale, la moyenne et l'écart-type des valeurs dans une cellule…) sont calculés directement à partir des données chargées. En fait, les paramètres des cellules du niveau supérieur peuvent être facilement calculés à partir des paramètres des cellules du niveau inférieur.

L'avantage de cette méthode c'est que lorsque les données sont mises à jour, on n'a pas besoin de recalculer toutes les informations dans la hiérarchie des cellules. On peut juste effectuer une mise à jour incrémentielle.

B. CLIQUE (CLustering In QUEst)

Contrairement aux autres algorithmes décrits précédemment, CLIQUE intègre la méthode de clustering basée sur la densité et celle basée sur la grille.

L'espace de données est partitionné en unités rectangulaires non superposées par espaces égaux de partition le long de chaque dimension. Une unité est dense si la fraction du total des points de données qu'elle contienne dépasse un paramètre de modèle d'entrée.

CLIQUE est utile pour regrouper des données de grandes dimensions qui sont généralement très clairsemés et ne forment pas de clusters dans l'espace dimensionnel complet. Il évolue linéairement avec la taille de l'entrée et a une bonne évolutivité si le nombre des dimensions dans les données est augmenté. Cependant, la précision des résultats du clustering peuvent être dégradés au détriment de la simplicité de la méthode.

6. Classification basée sur des contraintes

Les algorithmes de classification basée sur des contraintes prennent en considération la présence des obstacles puisqu'ils affectent la proximité spatiale entre les entités et des facilitateurs puisqu'ils, leur tour, affectent l'accessibilité (ou la connectivité) entre les entités.

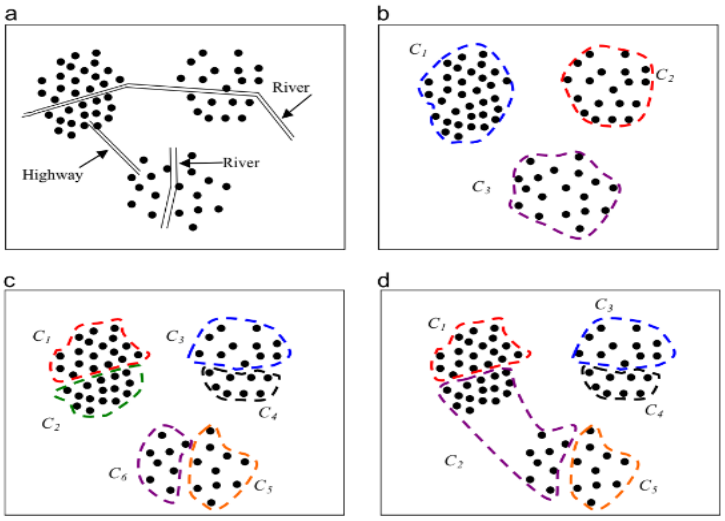


Fig23. Les clusters en présence d'obstacles et de facilitateurs. (a) Obstacle (rivière) et facilitateur (autoroute) ; (b) cluster résultant de l'ignorance des obstacles et des facilitateurs ; (c) cluster résultant de la considération des obstacles ; (d) cluster résultant de la considération des obstacles et des facilitateurs

A. COD-CLARANS

Cet algorithme permet de calculer la "distance dégagée" entre des points en se basant sur un graphe de visibilité. Il se sert d'une méthode de pré-clustering pour obtenir des micro-clusters à partir desquels il construit des clusters spatiaux.

Il est sensible à la variation de densité et les valeurs aberrantes. Aussi, il ne peut pas découvrir les clusters aux formes arbitraires (non sphérique comme les formes en ligne ou en cercle) et il ne considère pas les facilitateurs.

B. DBCluC

C'est une forme d'extension de de l'algorithme basé sur la densité : DBSCAN.

Pour modéliser les obstacles, DBCluC utilise des bords internes pour remplir l'espace visible d'obstacles (lignes d'obstruction).

Pour une entité, son voisin doit lui être visible par la ligne d'obstruction. En fait, si 2 points ne sont pas liés par la densité mais il existe un facilitateur qui traverse leurs voisinages, ces 2 points seront considérés comme liés par la densité.

La faiblesse de cet algorithme c'est qu'il est incapable de découvrir des clusters de différentes densités. De même, les lignes d'obstruction peuvent parfois conduire à des erreurs.

C. DBRS_O et DBRS+

DBRS_O est une extension de DBRS qui, à son tour, emploie une stratégie d'échantillonnage au hasard pour améliorer l'efficacité de DBSCAN.
DBRS + considère plusieurs obstacles dans un voisinage et calcule la distance dégagée, comme un entier, avec les obstacles intersectés. Aussi, il examine la longueur du facilitateur.
DBRS_O et DBRS+ sont sensibles aux distances inégales et le processus de clustering devient plus compliqué si on prend en compte l'existence des obstacles et des facilitateurs.

D. AUTOCLUST+

C'est une extension d'AUTOCLUST qui est un algorithm basé sur les graphes.
AUTOCLUST+ utilise la triangulation de Delaunay pour modéliser la proximité spatiale entre les entités (si une arête de la triangulation de Delaunay croise des obstacles, cette arête sera supprimée).
Son point fort par rapport aux autres algorithmes déjà mentionnés, c'est qu'il est capable de découvrir des clusters de différentes densités et / ou de formes arbitraires sans avoir besoin de paramètres spécifiés par l'utilisateur.
Son défaut se présente dans l'apparition probable de petits clusters sans signification puisqu'il se sert des indicateurs statistiques dérivés du réseau de triangulation de Delaunay d'origine, même si certaines arêtes ont été supprimées après. De même, il est non robuste aux ensembles de données spatiales complexes et ne prend pas en compte les facilitateurs.

E. ASCDT+

Cet algorithme considère les obstacles dans le processus de construction de la proximité spatiale en utilisant l'opération d'intersection et la triangulation de Delaunay.
En effet, les contraintes à plusieurs niveaux sont utilisées pour segmenter le réseau de triangulation de Delaunay en une série de sous-graphiques ce qui donne une nouvelle définition de la région de clustering.
En plus, les facilitateurs sont mis en œuvre par une opération combinée.

.

Malgré qu'ASCDT+ semble être l'algorithme parfait qui peut considérer, à la fois, les obstacles et les facilitateurs, quelques reproches peuvent être notées. En fait, les résultats de clustering ne dépendent que de la longueur des bords. En conséquence, les régions de faibles densités inégales peuvent être segmentées en plusieurs clusters après la suppression des bords longs du réseau de triangulation de Delaunay. Aussi, il est difficile de distinguer les chaînes et les clusters simples en forme de ligne.

Conclusion

D'après l'ensemble des tests menés, la méthode de classification hiérarchique agglomérative AGNES semble être la meilleure option des trois étudiées. Elle se rapproche à chaque fois le plus des résultats obtenus avec la méthode actuelle de NetsCity sans avoir à réaliser de supervision contrairement à cette dernière.

La méthode d'agglomération "complete" semble en effet être la plus appropriée bien que "average" soit également intéressante même si elle produit plus de points isolés. Il faut également noter que la méthode de "ward" peut donner des résultats intéressant lorsqu'une hauteur de coupe à priori plus adéquate est utilisée, ou bien qu'une pondération par le nombre de publications est incluse. Cette différence notable entre la méthode de "ward" et les autres repose simplement sur sa nature : "complete" , "single" et "average" sont toutes proche et basées sur la géométrie alors que "ward" découle des statistiques. Il est donc important de rappeler à nouveau que le choix arbitraire d'une hauteur de coupe du dendrogramme à 0,5% de la hauteur maximale à potentiellement influencé les résultats.

HDBSCAN est une méthode intéressante mais pas forcément adaptée aux besoins de NetsCity. Cela se remarque particulièrement dans les zones denses ou énormément de points, classés dans différents petits clusters de tailles similaires par AGNES ou la méthode précédente, sont considérés comme du bruit lorsque MinPts est élevé ou alors sont regroupés en clusters très réduit lorsque MinPts est faible. Il ne semble pas y avoir de paramétrage optimal pour les zones denses. de toute manière HDBSCAN est la méthode qui induit de manière générale le plus de points isolés à chaque fois. Néanmoins l'implémentation sous Python permet plus de possibilité que sous R donc il faut tout de même garder cette piste à l'esprit.

DBSCAN quant à lui a pour seul avantage de générer peu de clusters traversant des limites de pays comparé aux autres méthodes. Néanmoins, cela est à priori du au fait que ses clusters sont moins nombreux mais plus étendus. Sur les autres aspects, il semble peu pertinent par rapport aux exigences : dans les zones de fortes densités les clusters sont beaucoup trop grand. Un moyen d'y remédier serait d'augmenter considérablement la valeur de MinPts ou de réduire Eps. Néanmoins, il y a aurait alors la majorité des points classés comme isolés. Cela est bien entendu dû à la sensibilité de cet algorithme aux grandes variations densités. Comme c'est assurément le cas des données traitées par NetsCity, il semble peu intéressant d'utiliser c'est méthode.

DBSCAN et HDBSCAN sont particulièrement sensible à la variation de leurs paramètres en entrée, moins que AGNES ou c'est la détermination de la hauteur de coupe qui prime vraiment au final.

Perspectives

Bien que la quantité de résultats produits soit assez importante, il aurait été intéressant d'étudier d'autres aspects ou possibilités.

L'introduction de variables autres l'information géographique dans le calcul des distances de la matrice de dissimilarité pourrait être une autre option pour intégrer des variables autres que spatiales. La distance calculée ne serait néanmoins plus géographique mais mathématique (distance euclidienne à priori). Même si des tests préliminaires ont donné des résultats très peu pertinents, il pourrait être intéressant d'explorer cette piste car les variables non spatiales ont été utilisées de manière brute. En le manipulant et les transformant un peu, il serait peut être possible d'obtenir des résultats intéressants. D'autant plus qu'il est également possible d'utiliser 2 matrices de distances simultanément dans certaines implémentations de AGNES (package R ClustGeo). Un ratio de représentant l'importance ou la part d'utilisation de chaque matrice est fixable par l'utilisateur. On pourrait donc imaginer mélanger de cette manière une matrice de distances spatiales et non spatiales.

La considération d'obstacles (limites des pays, traits de côtes, montagnes) et de facilitations (ponts, réseaux de transports) serait bien évidemment également intéressante. Cela pourrait se faire soit par l'introduction d'une variable non spatiale (accessibilité, rugosité par exemple), soit par l'utilisation d'une méthode de clustering le permettant (voir la partie bibliographique sur les autres méthodes recensées), ou encore par la création d'un algorithme personnalisé incluant une telle opération et pouvant simplement être une séquence de traitements faisant appel à d'autres algorithmes déjà implémentés. Il existe déjà beaucoup d'algorithmes permettant la prise en compte d'obstacles et de facilitations. Néanmoins nous n'avons trouvé aucune implémentation utilisable.

Il serait également essentiel de s'intéresser à l'effet du choix de la hauteur de coupe du dendrogramme sur les résultats des méthodes de classement hiérarchique. Plus particulièrement de déterminer comment utiliser des indices de stabilité ou de dissimilarité

pour définir la hauteur de coupe. Cela permettrait d'avoir une méthode automatique potentiellement plus fiable que simplement prendre un certain pourcentage de la hauteur maximale.

Enfin l'utilisation de méthodes issues des statistiques comme les indices locaux d'autocorrélation spatiale (travaux de Luc Anselin) pourrait également être intéressant. En effet il serait peut être possible de baser le regroupement des points sur les valeurs d'autocorrélation spatiale obtenues pour une variable comme le nombre de publications par exemple.

Ressources bibliographiques

Adaptive spatial clustering in the presence of obstacles and facilitators | Request PDF [WWW Document], n.d. URL https://www.researchgate.net/publication/256939048_Adaptive_spatial_clustering_in_the_presence_of_obstacles_and_facilitators (accessed 2.21.19).

Anselin, L., 1995. Local Indicators of Spatial Association—LISA. Geographical Analysis 27, 93–115. https://doi.org/10.1111/j.1538-4632.1995.tb00338.x

Asian Journal of Computer Science and Information Technology [WWW Document], n.d. URL http://www.innovativejournal.in/index.php/ajcsit (accessed 2.21.19).

Bailey, B., 2017. Lightning Talk: Clustering with HDBScan [WWW Document]. Towards Data Science. URL https://towardsdatascience.com/lightning-talk-clustering-with-hdbscan-d47b83d1b03a (accessed 2.21.19).

Choosing the Best Clustering Algorithms, n.d. . Datanovia. URL https://www.datanovia.com/en/lessons/choosing-the-best-clustering-algorithms/ (accessed 2.21.19).

Comparing different clustering algorithms on toy datasets — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html (accessed 2.21.19).

DBSCAN, 2018. . Wikipédia.

Gower, J.C., 1971. A General Coefficient of Similarity and Some of Its Properties. Biometrics 27, 857. https://doi.org/10.2307/2528823

Gruen, F.L. and B., 2019. CRAN Task View: Cluster Analysis & Finite Mixture Models.

HCPC Using FactoMineR: Video - Articles - STHDA [WWW Document], n.d. URL http://www.sthda.com/english/articles/22-principal-component-methods-videos/74-hcpc-using-factominer-video/ (accessed 2.21.19).

hdbscan function | R Documentation [WWW Document], n.d. URL https://www.rdocumentation.org/packages/dbscan/versions/1.1-3/topics/hdbscan (accessed 2.21.19).

HDBSCAN with the dbscan package [WWW Document], n.d. URL https://cran.r-project.org/web/packages/dbscan/vignettes/hdbscan.html (accessed 2.21.19).

Hierarchical Clustering [WWW Document], n.d. URL https://www.brandidea.com/hierarchicalclustering.html (accessed 2.21.19).

Hierarchical K-Means Clustering: Optimize Clusters, n.d. . Datanovia. URL https://www.datanovia.com/en/lessons/hierarchical-k-means-clustering-optimize-clusters/ (accessed 2.21.19).

Hinneburg, A., Gabriel, H.-H., 2007. DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation, in: R. Berthold, M., Shawe-Taylor, J., Lavrač, N. (Eds.), Advances in Intelligent Data Analysis VII. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 70–80. https://doi.org/10.1007/978-3-540-74825-0_7

How HDBSCAN Works — hdbscan 0.8.1 documentation [WWW Document], n.d. URL https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html (accessed 2.21.19a).

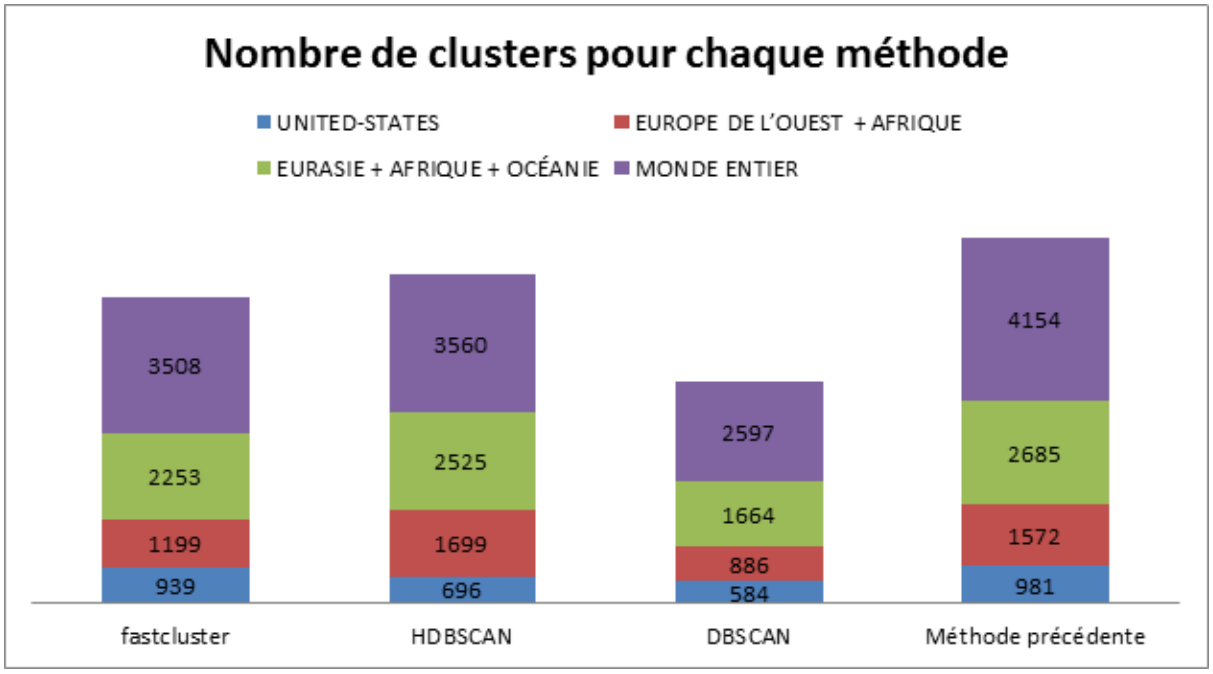
Karypis, G., Kumar, and V., 1999. Chameleon: hierarchical clustering using dynamic modeling. Computer 32, 68–75. https://doi.org/10.1109/2.781637

Maisonobe, M., Jégou, L., Eckert, D., 2018. Delineating urban agglomerations across the world: a dataset for studying the spatial distribution of academic research at city level. Cybergeo : European Journal of Geography. https://doi.org/10.4000/cybergeo.29637

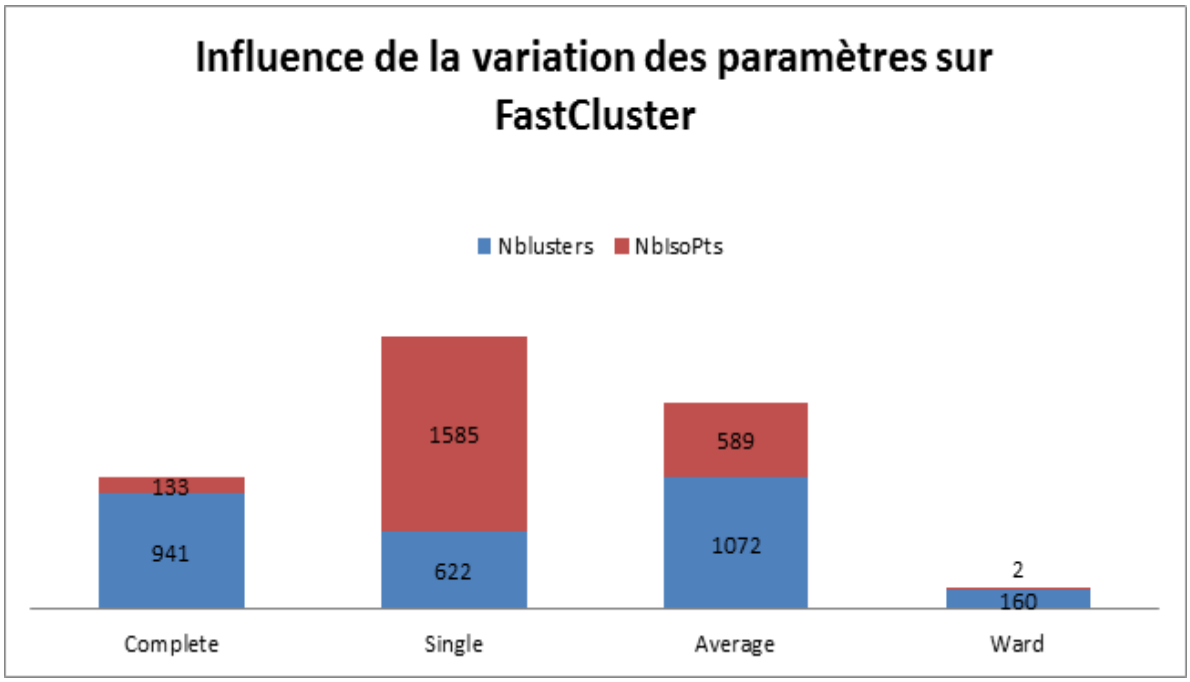
Müllner, D., 2013. fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. Journal of Statistical Software 53, 1–18. https://doi.org/10.18637/jss.v053.i09

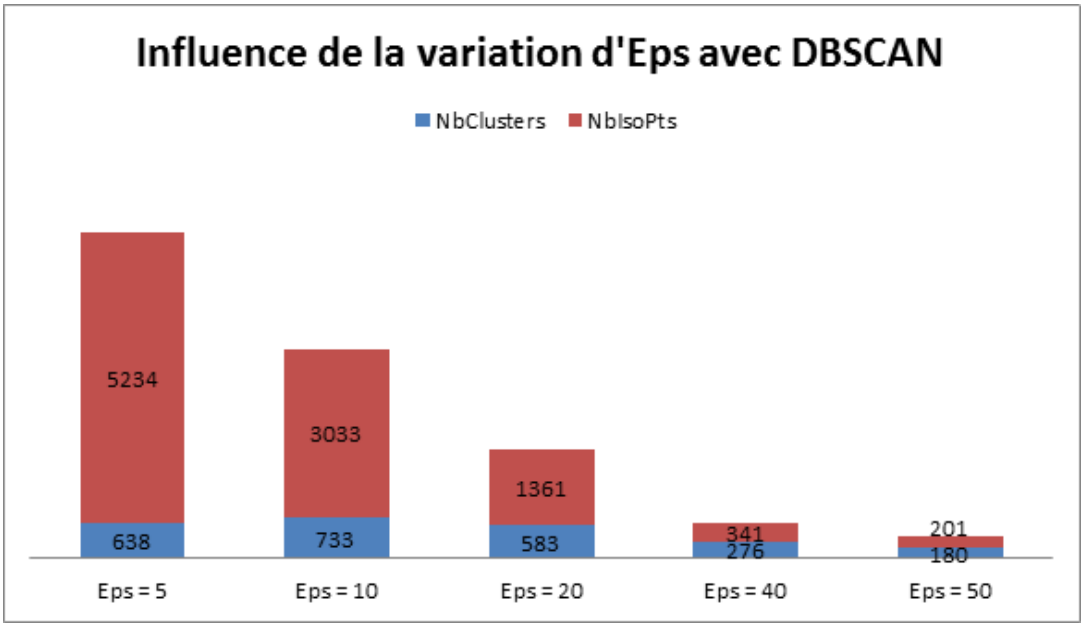
Sadler, J., n.d. Great Circles with R [WWW Document]. Jesse Sadler. URL https://jessesadler.com/post/great-circles-sp-sf/ (accessed 2.21.19).
sklearn.cluster.dbscan — scikit-learn 0.20.2 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/modules/generated/sklearn.cluster.dbscan.html (accessed 2.21.19).
ST_ClusterDBSCAN [WWW Document], n.d. URL http://postgis.net/docs/manual-dev/ST_ClusterDBSCAN.html (accessed 2.21.19).
The hdbscan Clustering Library — hdbscan 0.8.1 documentation [WWW Document], n.d. URL https://hdbscan.readthedocs.io/en/latest/index.html (accessed 2.21.19).
thematicmapping.org [WWW Document], n.d. URL http://thematicmapping.org/downloads/world_borders.php (accessed 2.21.19).
Wang, W., Yang, J., Muntz, R., n.d. STING: A Statistical Information Grid Approach to Spatial Data Mining 10.
Ward, J.H., 1963. Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association 58, 236. https://doi.org/10.2307/2282967
Zaïane, O.R., Lee, C.-H., 2002. Clustering Spatial Data in the Presence of Obstacles and Crossings: a Density-Based Approach 13.

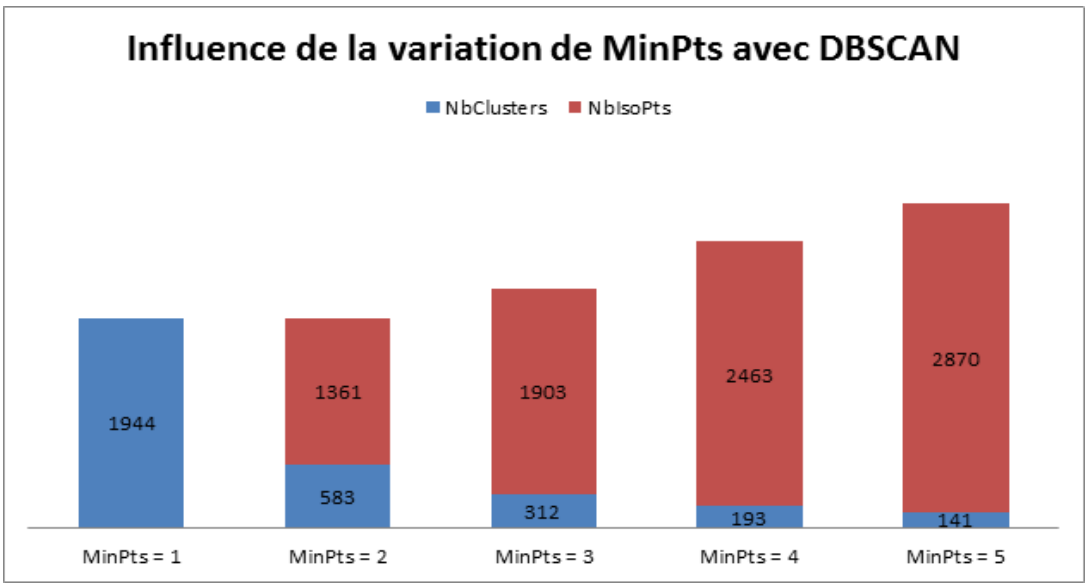
Annexe

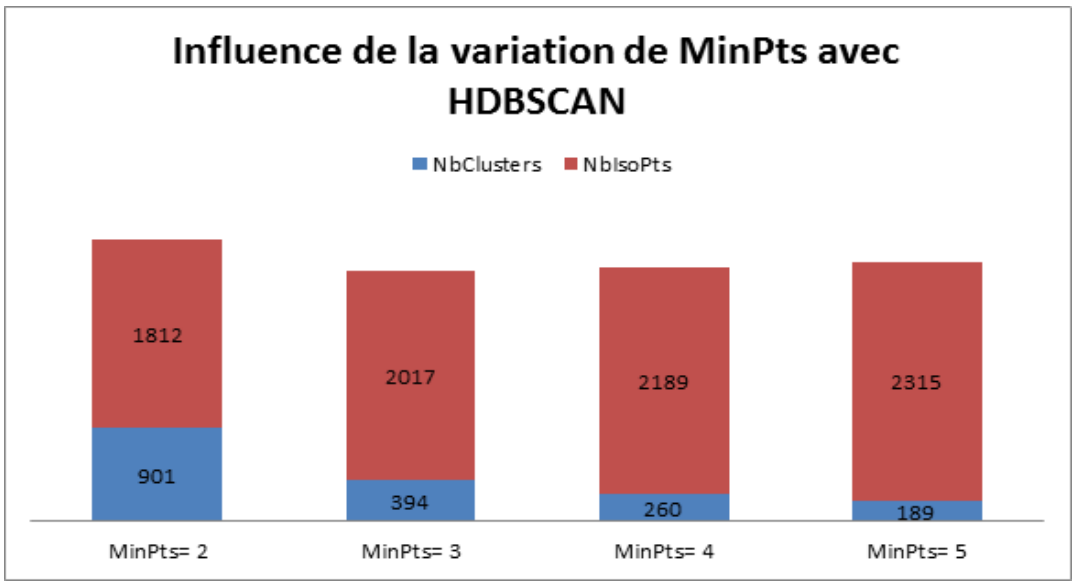


Les résultats d'influence du nombre de points obtenus par python :









Script python

```
#Importation des bibliothèques
import pandas as pd #lecture du fichier csv
import fastcluster #Hiérarchique agglomératif
from sklearn.cluster import DBSCAN #DBSCAN
import hdbscan #HDBSCAN
import numpy as np #Array
import geopy.distance #Calcul de la distance des grands cercles
from scipy.cluster.hierarchy import fcluster #Création des flat clusters
from scipy.spatial.distance import pdist, squareform
from shapely.geometry import Point
import geopandas
#------------------------------------------------------------------------
-

"""
Données
"""

#Importation des données
data =
pd.read_csv("/home/sigma/Bureau/NEtscity/localites_r18_arl_avecdisc.csv
",sep = ';', decimal = '.', header = 0, encoding = 'cp1252') #importer
les données
data = data.values #transformation en array numpy

#Récupération des coordonnées géographiques et des infos importantes :
col21 = data[:,0:21] #on récupère seulement les 21 1ères colonnes
col247 = data[:,245:249] #récupérer les 4 dernières colonnes =>
data_filt[:,21] =indice des pts
data_filt=np.hstack((col21,col247[:,:])) #concaténation horizontale

#Boucle for pour récupérer la liste des numéros de ligne ne présentant
pas de valeurs manquantes pour les coordonnées (soit lat = NaN, soit
lon = NaN soit les deux)
lstIndex = [] #initialisation
for rangPoint in range(len(data_filt)):
    #Si lat = colonne 4 ET lon = colonne 5 ne sont pas des valeurs NaN,
alors on stocke l'indice de ligne
    if np.isnan(data_filt[rangPoint,3]) == False and
np.isnan(data_filt[rangPoint,4]) == False:
        lstIndex += [rangPoint]

#Création de l'array filtrée : lignes avec des NaN omises
data_filt_na = data_filt[lstIndex]

np.where(np.isnan(np.array(data_filt_na[:,3:5], dtype = 'float64')))
#vérifier qu'il n'y a plus de valeurs NaN


#Subset du jeu de données initiale par pays
USA = np.where(data_filt_na[:,2] == "UNITED-STATES") #Ici on récupère
les indices des lignes ayant comme pays les Etats-Unis
data_filt_na = data_filt_na[USA] #Filtrage des lignes
```

```
"""
Matrice des distances
"""

#Création de la matrice des distances
distmatrix = np.zeros(shape = (len(data_filt_na),len(data_filt_na)))

#calcul de matrice des distances avec la fonction pdist
distmatrix = pdist(data_filt_na[:,3:5],lambda u, v:
geopy.distance.great_circle(u, v).km) #u: lat // v: long
#On obtient une matrice condensée donc il est nécessaire pour DBSCAN et
HDBSCAN
#de la transformer en matrice redondante carrée : fonction squareform
distmatrix = squareform(distmatrix)

np.savetxt('matrice_dist2', distmatrix, delimiter=';') #enregistrer la
matrice sous format txt


#importer la matrice de distance si elle existe déjà
distmatrix =
pd.read_csv("/home/sigma/Bureau/NEtscity/matrice_distance.csv",sep=';',
header=None,decimal = '.',encoding='utf-8')


"""
FASTCLUSTER
"""

#tableau des résultats des différentes méthodes (la position de chaque
pt dans les clusters selon les différentes méthodes)
PtsClusters= pd.DataFrame(np.zeros((len(data_filt_na),13),
dtype='float64'),
columns=['IdPoint','Latitude','Longitude','Ville','Province','Pays','id
Agglo','Agglomeration','Nbre publi','Average',
'Complete','Single','Ward'])

#tableau qui montre le nombre total des clusters  et les pts bruit
générés par chaque cluster
ClTot_Bruit= pd.DataFrame(np.zeros((2,4),dtype=int),
columns=['Average','Complete','Single','Ward'],index=['Nbre de
clusters','Pts Bruit'])

#liste avec les différentes méthodes du fastclsuter
method=['average','complete','single', 'ward']

for col in range(PtsClusters.shape[1]):

    if col==0: #la première colonne du tableau va recevoir les indices
des villes existants dans data_filt_na
        PtsClusters.iloc[:,0] = data_filt_na[:,21]
    elif col==1: #récupérer les lat
        PtsClusters.iloc[:,1] = data_filt_na[:,3]
    elif col==2:#récupérer les long
        PtsClusters.iloc[:,2] = data_filt_na[:,4]
    elif col==3:#récupérer les villes
        PtsClusters.iloc[:,3] = data_filt_na[:,0]
```

```
    elif col==4:#récupérer les provinces
        PtsClusters.iloc[:,4] = data_filt_na[:,1]
    elif col==5:#récupérer le pays
        PtsClusters.iloc[:,5] = data_filt_na[:,2]
    elif col==6:#récupérer les idComposite
        PtsClusters.iloc[:,6] = data_filt_na[:,23]
    elif col==7:#récupérer les nom d'agglo
        PtsClusters.iloc[:,7] = data_filt_na[:,24]
    elif col==8:#récupérer la somme des publications durant 15 ans (de
1999 à 2014)
        for row in range (len(data_filt_na)):
            PtsClusters.iloc[row,8]=sum(data_filt_na[row,5:20])

    else:
        fast=fastcluster.linkage(distmatrix,method[col-9]) #fascluster
algorithm
        cutheight = max(fast[:,2])*(0.5/100) #hauteur de découpage au
niveau du dendrogram 0.5%
        clusters = fcluster(fast, cutheight, criterion='distance')
#génération des clusters

        for row in range(PtsClusters.shape[0]):
            PtsClusters.iloc[row,col] = clusters[row] #remplir par
ligne les numéros de clusters correspondant à chaque pt selon la
méthode indiquée

        ClTot_Bruit.iloc[0,col-9] = max(clusters) #nbre total de
clusters par méthode

        Occ=PtsClusters.iloc[:,col].value_counts().values.tolist()
#liste avec la fréquence d'apparition de chaque nombre de clusters dans
la liste
        ClTot_Bruit.iloc[1,col-9] = np.count_nonzero(np.array(Occ)==1)
#nbre de pts bruit par méthode (compter le nombre de fois où 1 apparait
dans la liste Occ)



PtsClusters.to_csv('methodagglo_results2.csv',sep=';',encoding='utf-
8',decimal='.')  #enregistrer sous csv
ClTot_Bruit.to_csv('methodagglo_ClustersIsoPts2.csv',sep=';',encoding='
utf-8',decimal='.')  #enregistrer sous csv le nbre de clusters et pts
bruit par méthode



"""
DBSCAN
"""

#-----------------------
#EPS fixe à 20km (EF)
#-----------------------


tabDBSCAN_EF= pd.DataFrame(np.zeros((len(data_filt_na),8),
dtype='float64'), columns=['IdPoint','Latitude','Longitude','1 Pt', '2
```

```
Pts','3 Pts','4 Pts','5 Pts']) #tableau pour stocker les résultats de
DBSCAN pour un eps fixe à 20

DBSCAN_Tot_Bruit_EF= pd.DataFrame(np.zeros((2,5),dtype=int),
columns=['1 Pt', '2 Pts','3 Pts','4 Pts','5 Pts'],index=['Nbre de
clusters','Pts Bruit']) #tableau qui montre le nombre total des
clusters  et les pts bruit générés par chaque cluster

min_samples=[1,2,3,4,5] #nbre min de pts pour former un cluster
eps=20 #rayon auquel faut chercher des voisins

for col in range(tabDBSCAN_EF.shape[1]):

    if col==0: #la première colonne du tableau représente les indices
des points
        tabDBSCAN_EF.iloc[:,0] = data_filt_na[:,21]
    elif col==1: #récupérer les lat
        tabDBSCAN_EF.iloc[:,1] = data_filt_na[:,3]
    elif col==2:#récupérer les long
        tabDBSCAN_EF.iloc[:,2] = data_filt_na[:,4]

    else:
        db = DBSCAN(min_samples = min_samples[col-3], eps = eps, metric
= "precomputed", algorithm = "auto").fit(distmatrix) #DBSCAN algorithm
        clustersDB = db.labels_

        for row in range(tabDBSCAN_EF.shape[0]): #remplir la table avec
les résultats de l'algo
            tabDBSCAN_EF.iloc[row,col] = clustersDB[row]

            if max(clustersDB)==-1: #s'il n'existe pas de clusters,
DBSCAN renvoie -1 pour les pts non classés (bruit)
                DBSCAN_Tot_Bruit_EF.iloc[0,col-3] = 0
            else:
                DBSCAN_Tot_Bruit_EF.iloc[0,col-3] = max(clustersDB)
#nbre total de clusters par méthode

            DBSCAN_Tot_Bruit_EF.iloc[1,col-3] = len(np.where(clustersDB
== -1)[0]) #nbre de pts bruit par méthode

tabDBSCAN_EF.to_csv('DBSCANMinPts_results.csv',sep=';',encoding='utf-
8',decimal='.')  #enregistrer sous csv la table des résultats des
différents clusters
DBSCAN_Tot_Bruit_EF.to_csv('DBSCANMinPts_ClustersIsoPts.csv',sep=';',en
coding='utf-8',decimal='.') #enregistrer sous csv le nbre de clusters
et pts bruit par méthode


#----------------------
#MinPts fixe à 2 pts (PF)
#----------------------

tabDBSCAN_PF= pd.DataFrame(np.zeros((len(data_filt_na),8),
dtype='float64'), columns=['IdPoint','Latitude','Longitude','1 Pt', '2
Pts','3 Pts','4 Pts','5 Pts']) #tableau pour stocker les résultats de
DBSCAN pour un min de pt = 2
```

```
DBSCAN_Tot_Bruit_PF= pd.DataFrame(np.zeros((2,5),dtype=int),
columns=['5 km', '10 km','20 km','40 km','50 km'],index=['Nbre de
clusters','Pts Bruit']) #tableau qui montre le nombre total des
clusters  et les pts bruit générés par chaque cluster


eps2=[5,10,20,40,50] #rayon auquel faut chercher des voisins
min_samples2=2 #nbre min de pts pour former un cluster

for col in range(tabDBSCAN_PF.shape[1]):

    if col==0: #la première colonne du tableau représente les indices
des points
        tabDBSCAN_PF.iloc[:,0] = data_filt_na[:,21]
    elif col==1: #récupérer les lat
        tabDBSCAN_PF.iloc[:,1] = data_filt_na[:,3]
    elif col==2:#récupérer les long
        tabDBSCAN_PF.iloc[:,2] = data_filt_na[:,4]

    else:
        db2 = DBSCAN(min_samples=min_samples2, eps=eps2[col-3], metric
= "precomputed", algorithm = "auto").fit(distmatrix)
        clustersDB2 = db2.labels_
        for row in range(tabDBSCAN_PF.shape[0]):
            tabDBSCAN_PF.iloc[row,col] = clustersDB2[row]

            if max(clustersDB2)==-1: #s'il n'existe pas de clusters,
DBSCAN renvoie -1 pour les pts non classés (bruit)
                DBSCAN_Tot_Bruit_PF.iloc[0,col-3] = 0
            else:
                DBSCAN_Tot_Bruit_PF.iloc[0,col-3] = max(clustersDB2)
#nbre total de clusters par méthode

            DBSCAN_Tot_Bruit_PF.iloc[1,col-3] =
len(np.where(clustersDB2 == -1)[0]) #nbre de pts bruit par méthode


tabDBSCAN_PF.to_csv('DBSCANEps_results.csv',sep=';',encoding='utf-
8',decimal='.')  #enregistrer sous csv la table des résultats des
différents clusters
DBSCAN_Tot_Bruit_PF.to_csv('DBSCANEps_ClustersIsoPts.csv',sep=';',encod
ing='utf-8',decimal='.')  #enregistrer sous csv le nbre de clusters et
pts bruit par méthode


"""
HDBSCAN
"""

tabHDBSCAN= pd.DataFrame(np.zeros((len(data_filt_na),7),
dtype='float64'), columns=['IdPoint','Latitude','Longitude','2 Pts','3
Pts','4 Pts','5 Pts']) #tableau pour stocker les résultats de HDBSCAN
pour un min de pt = 2

HDBSCAN_Tot_Bruit= pd.DataFrame(np.zeros((2,4),dtype=int), columns=['2
Pts','3 Pts','4 Pts','5 Pts'],index=['Nbre de clusters','Pts Bruit'])
```

```
#tableau qui montre le nombre total des clusters  et les pts bruit
générés par chaque cluster


min_cluster_size=[2,3,4,5] #nbre de pts min pour former un cluster

for col in range(tabHDBSCAN.shape[1]):

    if col==0: #la première colonne du tableau représente les indices
des points
        tabHDBSCAN.iloc[:,0] = data_filt_na[:,21]
    elif col==1: #récupérer les lat
        tabHDBSCAN.iloc[:,1] = data_filt_na[:,3]
    elif col==2:#récupérer les long
        tabHDBSCAN.iloc[:,2] = data_filt_na[:,4]

    else:
        hdb = hdbscan.HDBSCAN(min_cluster_size[col-3], metric =
"precomputed").fit(distmatrix) #hdbscan algorithm
        clustersHDB = hdb.labels_
        for row in range(tabHDBSCAN.shape[0]):
            tabHDBSCAN.iloc[row,col] = clustersHDB[row]

            if max(clustersHDB)==-1: #s'il n'existe pas de clusters,
HDBSCAN renvoie -1 pour les pts non classés (bruit)
                HDBSCAN_Tot_Bruit.iloc[0,col-3] = 0
            else:
                HDBSCAN_Tot_Bruit.iloc[0,col-3] = max(clustersHDB)
#nbre total de clusters par méthode

            HDBSCAN_Tot_Bruit.iloc[1,col-3] = len(np.where(clustersHDB
== -1)[0]) #nbre de pts bruit par méthode


tabHDBSCAN.to_csv('HDBSCANMinPts_results.csv',sep=';',encoding='utf-
8',decimal='.')  #enregistrer sous csv la table des résultats des
différents clusters
HDBSCAN_Tot_Bruit.to_csv('HDBSCANMinPts_ClustersIsoPts.csv',sep=';',enc
oding='utf-8',decimal='.')  #enregistrer sous csv le nbre de clusters
et pts bruit par méthode



""" Shapefile """

###########
#FASCLUSTER
###########

# combine lat and lon column to a shapely Point() object
PtsClusters['geometry'] = PtsClusters.apply(lambda x:
Point((x.Longitude), (x.Latitude)), axis=1)

Fastcluster= geopandas.GeoDataFrame(PtsClusters, geometry='geometry')

# proj WGS84
Fastcluster.crs= "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
```

```
#export sous format shpape (.shp)
Fastcluster.to_file('Fastcluster.shp', driver='ESRI Shapefile')


###########
#DBSCAN_MinPts
###########
tabDBSCAN_EF['geometry'] = tabDBSCAN_EF.apply(lambda x:
Point((x.Longitude), (x.Latitude)), axis=1)

DBSCANMinPts= geopandas.GeoDataFrame(tabDBSCAN_EF, geometry='geometry')

# proj WGS84
DBSCANMinPts.crs= "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

#export sous format shpape (.shp)
DBSCANMinPts.to_file('DBSCANMinPts.shp', driver='ESRI Shapefile')



###########
#DBSCAN_EPS
###########
tabDBSCAN_PF['geometry'] = tabDBSCAN_PF.apply(lambda x:
Point((x.Longitude), (x.Latitude)), axis=1)

DBSCANEps= geopandas.GeoDataFrame(tabDBSCAN_PF, geometry='geometry')

# proj WGS84
DBSCANEps.crs= "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

#export sous format shpape (.shp)
DBSCANEps.to_file('DBSCANEps.shp', driver='ESRI Shapefile')



###########
#HDBSCAN
###########

tabHDBSCAN['geometry'] = tabHDBSCAN.apply(lambda x: Point((x.Longitude),
(x.Latitude)), axis=1)

HDBSCAN= geopandas.GeoDataFrame(tabHDBSCAN, geometry='geometry')

# proj WGS84
HDBSCAN.crs= "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

#export sous format shpape (.shp)
HDBSCAN.to_file('HDBSCAN.shp', driver='ESRI Shapefile')
```

Script R :

- **Influence des paramètres**

```
setwd("/home/sigma/Bureau/Scripts") #Indiquer le rÃ©pertoire de
travail

#Installation des packages
#install.packages("fastcluster")
#install.packages("dbscan")

#Chargement des packages
library("dbscan")
library("fastcluster")
library("sp")


### CHARGEMENT DU JEU DE DONNEES : ENSEMBLE DE 30000 PTS ###
#Le fichier csv doit Ãªtre prÃ©sent dans le rÃ©pertoire de travail
bigwos <- read.csv2("localites_r18_arl_avecdisc.csv", header = T, sep
= ";", dec = ".")
#On supprime les points ne poss?dant pas de coordonn?es correctes ou
tout court
bigwos <- bigwos[which(is.na(bigwos$lng) == F & is.na(bigwos$lat) ==
F),]
#View(bigwos)
#RÃ©cupÃ©ration des points ayant comme pays les USA
bigwos_filt <- bigwos[which(bigwos$pays == "UNITED-STATES"),]


### RÃ©cupÃ©ration des coordonnÃ©es et calcul du nombre de
publications ###
#n?cessaire de mettre dans le bon ordre : long / lat ; convertir en
numeric (factor de base) ; na.omit comme pr?c?demment
coords <- na.omit(matrix(c(bigwos_filt$lng,bigwos_filt$lat),ncol =
2, dimnames = list(NULL,c("lng","lat")))) #lon lat
coords <- as.data.frame(coords)
#on r?cup?re le nb de publi par an pour tous les points qui poss?dent
des coordonn?es
data = bigwos_filt[,6:21]
#somme du nb de publi
nbpubli <- c(sum(data[1,], na.rm = T))
for (i in 2:length(coords$lng)) {
  nbpubli <- c(nbpubli, sum(data[i,], na.rm = T))
}
nbpubli <- as.data.frame(nbpubli)


### Cr?ation d'un objet spatial ###
dtest <- sp::SpatialPointsDataFrame(coords = coords, data = nbpubli,
proj4string = sp::CRS("+proj=longlat +ellps=WGS84")) #pas projet?
#plot(dtest@coords)


### Calcul de la matrice des distances ###
```

```
#Matrice des distances g?ographiques (calcul?e avec une fonction du
package sp)
distmatrix <- sp::spDists(x = dtest) #quand on est en non projet?,
pDists renvoie des valeurs en km avec la m?thode des grands cercles !
#distmatrix <- sp::spDists(x = dtest@coords[1:15000,], longlat = T)
#traitement d'une sous partie de dtest
distmatrix <- stats::as.dist(distmatrix) #conversion de la matrice en
objet dist. Sinon pas utilisable par hclust


### INFLUENCE DE LA METHODE D'AGGLOMERATION POUR FASTCLUSTER ###
#Liste des mÃ©thodes testÃ©es
Lstmethodes = list("complete", "single", "average", "ward.D")

#CrÃ©ation d'une matrice stockant les valeurs de nbre de clusters et
nbre de points isolÃ©s
ClustersPtsIso <- matrix(nrow = 4, ncol = 2,
                         dimnames = list(c("complete", "single",
"average", "ward.D"),

c("NbClusters","NbIsoPts")))

#CrÃ©ation d'un dataframe stockant les NÂ° de clusters pour chaque
point selon diffÃ©rentes mÃ©thodes
#Tableau ou chaque ligne reprÃ©sente un point et ayant comme colonnes
:
#ville / province / pays / latitude  / longitude /
#Nb publi sommÃ©s / MÃ©thode complete / Single / Average / Ward.D
#Id d'agglomÃ©ration (mÃ©thode prÃ©cÃ©dente) / Nom agglomÃ©ration
(mÃ©thode prÃ©cÃ©dente)
results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es

ClusteringData <- as.data.frame(matrix(nrow =
length(bigwos_filt$ville), ncol = 4,
                     dimnames = list(NULL, c("complete", "single",
"average", "ward.D"))))

#Boucle for
for (methode in (1:length(Lstmethodes))) {
  #clustering
  fast <- fastcluster::hclust(distmatrix, method =
Lstmethodes[[methode]])
  #m?thode de clustering conditionne le fait de pouvoir utiliser
height comme argument pour couper l'arbre
  #Marche avec complete / single (mais pas g?nial) / average /
ward.D / ward.D2
  #fast #dÃ©tail du clustering
  #plot(fast) #afficher le dendrogramme

  #CrÃ©ation des clusters : coupe du dendrogramme Ã  une hauteur
relative
```

```
  fastcl <- stats::cutree(fast, h = max(fast$height)*0.5/100)
#Hauteur ? laquelle on coupe l'arbre = 0.5% de la hauteur max pour
matrice distances g?ographiques

  #Stockage des rÃ©sultats : NÂ° de cluster pour chaque point
  ClusteringData[,methode] <- fastcl

  NbClusters <- max(fastcl) #nb de clusters
  ClusterSize <- as.data.frame(table(fastcl)) #compte le nombre de
points par cluster
  #Pour fastcluster il faut repÃ©rer les clusters de 1 points
  IsolatedPts <- ClusterSize[which(ClusterSize$Freq <= 1),] #Clusters
de 1 pts = Points isolÃ©s
  NbIsolatedPts <- length(IsolatedPts$Freq) #Nb de points isolÃ©s
  #Stockage dans la matrice du nbre de clusters et de points isolÃ©s
  ClustersPtsIso[methode,1] = NbClusters #1ere colonne = Nb de
clusters
  ClustersPtsIso[methode,2] = NbIsolatedPts #2e colonne = Nb de
points isolÃ©s
}

#RÃ©cupÃ©ration des derniÃ¨res colonnnes
results <- cbind(results, ClusteringData) #Stockage des rÃ©sultats des
clustering
results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom d'agglo

#Export sous csv des rÃ©sultats
#Fichier stockant les coordonnÃ©es et NÂ° de clusters + autres infos
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/MethodeAggloFastclus
ter/methodeagglo_results.csv"
write.csv2(results,filename, sep = ";", dec = ".")
#Fichier stockant le nbre de clusters et de points isolÃ©s otenus
pour chaque mÃ©thode
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/MethodeAggloFastclus
ter/methodeagglo_ClustersIsoPts.csv"
write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")


### INFLUENCE DE LA PONDERATION PAR LE NBRE DE PUBLI POUR UNE
METHODE HIERARCHIQUE DE WARD ###

#### A FAIRE SUR MON PC / PAS POSSIBLE DE CHARGER LE PACKAGE
CLUSTGEO SOUS UBUNTU ###


### INFLUENCE DES PARAMETRES DE DBSCAN ###
##Influence de MinPts
Eps = 20 #eps fixe Ã  20 km
#Lite des valeurs de MinPts Ã  tester
MinPts = list(1,2,3,4,5)
#GÃ©nÃ©ration de matrices et dataframe similaires aux prÃ©cÃ©dents
#CrÃ©ation d'une matrice stockant les valeurs de nbre de clusters et
nbre de points isolÃ©s
```

```
ClustersPtsIso <- matrix(nrow = length(MinPts), ncol = 2,
                dimnames = list(c("MinPts = 1", "MinPts = 2", "MinPts =
3", "MinPts = 4","MinPts = 5"),

c("NbClusters","NbIsoPts")))
#CrÃ©ation d'un dataframe stockant les NÂ° de clusters pour chaque
point selon diffÃ©rentes mÃ©thodes
results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es

ClusteringData <- as.data.frame(matrix(nrow =
length(bigwos_filt$ville), ncol = length(MinPts),
                       dimnames = list(NULL, c("MinPts = 1", "MinPts =
2", "MinPts = 3", "MinPts = 4","MinPts = 5"))))

#Boucle for
for (i in (1:length(MinPts))) {
  db <- dbscan::dbscan(distmatrix, minPts = MinPts[[i]], eps = Eps)
#clustering
  #Stockage des rÃ©sultats : NÂ° de cluster pour chaque point
  ClusteringData[,i] <- db$cluster
  #Nb de clusters
  NbClusters = max(db$cluster)
  #Nb de points isolÃ©s
  ClusterSize <- as.data.frame(table(db$cluster)) #compte le nombre de
points par cluster
  if (ClusterSize$Var1[1] == 0) { #Si le premier indice trouvÃ© est 0
alors il y a des points isolÃ©s
    NbIsolatedPts <- ClusterSize$Freq[1]
  } else { #Sinon pas de points bruits
    NbIsolatedPts <- 0
  }
  #Stockage dans la matrice du nbre de clusters et de points isolÃ©s
  ClustersPtsIso[i,1] = NbClusters #1ere colonne = Nb de clusters
  ClustersPtsIso[i,2] = NbIsolatedPts #2e colonne = Nb de points
isolÃ©s
}

#RÃ©cupÃ©ration des derniÃ¨res colonnnes
results <- cbind(results, ClusteringData) #Stockage des rÃ©sultats des
clustering
results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom d'agglo

#Export sous csv des rÃ©sultats
#Fichier stockant les coordonnÃ©es et NÂ° de clusters + autres infos
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANMinPts_
results.csv"
write.csv2(results,filename, sep = ";", dec = ".")
#Fichier stockant le nbre de clusters et de points isolÃ©s otenus
pour chaque mÃ©thode
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANMinPts_
ClustersIsoPts.csv"
```

```
write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")


##Influence de Eps
MinPts = 2 #MinPts fixe Ã  2
#Lite des valeurs de Eps Ã  tester
Eps = list(5,10,20,40,50)
#GÃ©nÃ©ration de matrices et dataframe similaires aux prÃ©cÃ©dents
#CrÃ©ation d'une matrice stockant les valeurs de nbre de clusters et
nbre de points isolÃ©s
ClustersPtsIso <- matrix(nrow = length(Eps), ncol = 2,
                         dimnames = list(c("Eps = 5", "Eps = 10", "Eps
= 20", "Eps = 40","Eps = 50"),

c("NbClusters","NbIsoPts")))
#CrÃ©ation d'un dataframe stockant les NÂ° de clusters pour chaque
point selon diffÃ©rentes mÃ©thodes
results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es

ClusteringData <- as.data.frame(matrix(nrow =
length(bigwos_filt$ville), ncol = length(Eps),
                                       dimnames = list(NULL, c("Eps =
5", "Eps = 10", "Eps = 20", "Eps = 40","Eps = 50"))))

#Boucle for
for (i in (1:length(Eps))) {
  db <- dbscan::dbscan(distmatrix, minPts = MinPts, eps = Eps[[i]])
#clustering
  #Stockage des rÃ©sultats : NÂ° de cluster pour chaque point
  ClusteringData[,i] <- db$cluster
  #Nb de clusters
  NbClusters = max(db$cluster)
  #Nb de points isolÃ©s
  ClusterSize <- as.data.frame(table(db$cluster)) #compte le nombre de
points par cluster
  if (ClusterSize$Var1[1] == 0) { #Si le premier indice trouvÃ© est 0
alors il y a des points isolÃ©s
    NbIsolatedPts <- ClusterSize$Freq[1]
  } else { #Sinon pas de points bruits
    NbIsolatedPts <- 0
  }
  #Stockage dans la matrice du nbre de clusters et de points isolÃ©s
  ClustersPtsIso[i,1] = NbClusters #1ere colonne = Nb de clusters
  ClustersPtsIso[i,2] = NbIsolatedPts #2e colonne = Nb de points
isolÃ©s
}

#RÃ©cupÃ©ration des derniÃ¨res colonnnes
results <- cbind(results, ClusteringData) #Stockage des rÃ©sultats des
clustering
results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom d'agglo

#Export sous csv des rÃ©sultats
```

```
#Fichier stockant les coordonnÃ©es et NÂ° de clusters + autres infos
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANEps_res
ults.csv"
write.csv2(results,filename, sep = ";", dec = ".")
#Fichier stockant le nbre de clusters et de points isolÃ©s otenus
pour chaque mÃ©thode
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANEps_Clu
stersIsoPts.csv"
write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")


##Influence d'une pondÃ©ration par le nbre de publication
MinPts = 2 #MinPts fixe
Eps = 20 #Eps fixe
#CrÃ©ation d'un dataframe stockant les NÂ° de clusters pour chaque
point selon diffÃ©rentes mÃ©thodes
results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es
#CrÃ©ation d'une matrice stockant les valeurs de nbre de clusters et
nbre de points isolÃ©s
ClustersPtsIso <- matrix(nrow = 1, ncol = 2,
          dimnames =
list(c("DBSCAN_2_20_weighted"),c("NbClusters","NbIsoPts")))

#Clustering pondÃ©rÃ©
db <- dbscan::dbscan(distmatrix, minPts = MinPts, eps = Eps, weights
= unlist(nbpubli))
#Nb de clusters
NbClusters = max(db$cluster)
#Nb de points isolÃ©s
ClusterSize <- as.data.frame(table(db$cluster)) #compte le nombre de
points par cluster
if (ClusterSize$Var1[1] == 0) { #Si le premier indice trouvÃ© est 0
alors il y a des points isolÃ©s
  NbIsolatedPts <- ClusterSize$Freq[1]
} else { #Sinon pas de points bruits
  NbIsolatedPts <- 0
}
#Stockage dans la matrice du nbre de clusters et de points isolÃ©s
ClustersPtsIso[1,1] = NbClusters #1ere colonne = Nb de clusters
ClustersPtsIso[1,2] = NbIsolatedPts #2e colonne = Nb de points
isolÃ©s
#Stockage des donnÃ©es
results$DBSCAN_2_20_weighted <- db$cluster
results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom d'agglo

#Export sous csv des rÃ©sultats
#Fichier stockant les coordonnÃ©es et NÂ° de clusters + autres infos
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANweighte
d_results.csv"
write.csv2(results,filename, sep = ";", dec = ".")
```

```
#Fichier stockant le nbre de clusters et de points isolÃ©s otenus
pour chaque mÃ©thode
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/DBSCAN/DBSCANweighte
d_ClustersIsoPts.csv"
write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")


### INFLUENCE DES PARAMETRES DE HDBSCAN : MinPts ###
#Lite des valeurs de MinPts Ã  tester
MinPts = list(2,3,4,5)
#GÃ©nÃ©ration de matrices et dataframe similaires aux prÃ©cÃ©dents
#CrÃ©ation d'une matrice stockant les valeurs de nbre de clusters et
nbre de points isolÃ©s
ClustersPtsIso <- matrix(nrow = length(MinPts), ncol = 2,
                         dimnames = list(c("MinPts = 2", "MinPts = 3",
"MinPts = 4", "MinPts = 5"),

c("NbClusters","NbIsoPts")))
#CrÃ©ation d'un dataframe stockant les NÂ° de clusters pour chaque
point selon diffÃ©rentes mÃ©thodes
results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es

ClusteringData <- as.data.frame(matrix(nrow =
length(bigwos_filt$ville), ncol = length(MinPts),
                                       dimnames = list(NULL, c("MinPts
= 2", "MinPts = 3", "MinPts = 4", "MinPts = 5"))))

#Boucle for
for (i in (1:length(MinPts))) {
  hdb <- dbscan::hdbscan(distmatrix, minPts = MinPts[[i]])
#clustering
  #Stockage des rÃ©sultats : NÂ° de cluster pour chaque point
  ClusteringData[,i] <- hdb$cluster
  #Nb de clusters
  NbClusters = max(hdb$cluster)
  #Nb de points isolÃ©s
  ClusterSize <- as.data.frame(table(hdb$cluster)) #compte le nombre de
points par cluster
  if (ClusterSize$Var1[1] == 0) { #Si le premier indice trouvÃ© est 0
alors il y a des points isolÃ©s
    NbIsolatedPts <- ClusterSize$Freq[1]
  } else { #Sinon pas de points bruits
    NbIsolatedPts <- 0
  }
  #Stockage dans la matrice du nbre de clusters et de points isolÃ©s
  ClustersPtsIso[i,1] = NbClusters #1ere colonne = Nb de clusters
  ClustersPtsIso[i,2] = NbIsolatedPts #2e colonne = Nb de points
isolÃ©s
}

#RÃ©cupÃ©ration des derniÃ¨res colonnnes
```

```
results <- cbind(results, ClusteringData) #Stockage des rÃ©sultats des
clustering
results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom d'agglo

#Export sous csv des rÃ©sultats
#Fichier stockant les coordonnÃ©es et NÂ° de clusters + autres infos
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/HDBSCAN/HDBSCANMinPt
s_results.csv"
write.csv2(results,filename, sep = ";", dec = ".")
#Fichier stockant le nbre de clusters et de points isolÃ©s otenus
pour chaque mÃ©thode
filename =
"/home/sigma/Bureau/Scripts/InfluenceParametres/HDBSCAN/HDBSCANMinPt
s_ClustersIsoPts.csv"
write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")
```

- **Influence nombre du nombre des points**

```
setwd("/home/sigma/Bureau/Scripts") #Indiquer le rÃ©pertoire de
travail

#Installation des packages
#install.packages("fastcluster")
#install.packages("dbscan")

#Chargement des packages
library("dbscan")
library("fastcluster")
library("sp")

### CHARGEMENT DU JEU DE DONNEES : ENSEMBLE DE 30000 PTS ###
#Le fichier csv doit Ãªtre prÃ©sent dans le rÃ©pertoire de travail
bigwos <- read.csv2("localites_r18_arl_avecdisc.csv", header = T, sep
= ";", dec = ".")
#On supprime les points ne poss?dant pas de coordonn?es correctes ou
tout court
bigwos <- bigwos[which(is.na(bigwos$lng) == F & is.na(bigwos$lat) ==
F),]
#View(bigwos)

#Nom des ensembles de points Ã  traiter : correspond Ã©galement au
nom des dossiers stockant les sorties
#On a choisi de ne pas garder en mÃ©moire chaque ensemble de points
NomsTranches <-
list("world","eurasie_afrique_oceanie","europeouest_afrique","usa")

#BOUCLE FOR REALISANT L'ENSEMBLE DES OPERATIONS PAR TRANCHE DE
POINTS
for (tranche in (1:length(NomsTranches))) {

  ### Cr?ation des ensembles de points ###
  ensemble = NomsTranches[[tranche]]

  #1) Traitement de TOUS LES POINTS : 32864 pts
```

```
  if (ensemble == "world") {
    bigwos_filt <- bigwos
  }

  #2) Traitement de EURASIE + AFRIQUE + OCÃ□ANIE : 22775 pts
  if (ensemble == "eurasie_afrique_oceanie") {
    #Liste des pays du continent am?ricain pr?sents dans le jeu de
donn?es
    America = list("ANTIGUA-AND-BARBUDA", "ARGENTINA", "BAHAMAS" ,
                   "BARBADOS", "BELIZE", "BOLIVIA", "BRAZIL",
                   "CANADA", "CHILE", "COLOMBIA", "COSTA-RICA",
                   "CUBA", "DOMINICAN-REPUBLIC", "ECUADOR",
"GRENADA",
                   "GUATEMALA", "GUYANA", "HAITI","HONDURAS",
"JAMAICA",
                   "MEXICO", "NICARAGUA", "PANAMA", "PARAGUAY",
"PERU",
                   "ST-KITTS-AND-NEVIS", "ST-LUCIA",
                   "ST-VINCENT-AND-THE-GRENADINES", "SURINAME",
                   "TRINIDAD-AND-TOBAGO", "UNITED-STATES",
                   "URUGUAY", "VENEZUELA")
    #Filtrage : on retire les points du continent amÃ©ricain
    bigwos_filt <- as.data.frame(bigwos[which(bigwos$pays !=
America[[1]]),])
    for (i in (2:length(America))) {
      bigwos_filt <- bigwos_filt[which(bigwos_filt$pays !=
America[[i]]),]
    }
  }

  #3) Traitement de EUROPE DE Lâ□□OUEST + AFRIQUE : 15764 pts
  if (ensemble == "europeouest_afrique") {
    #Liste des pays Ã  retirer
    AmericaAsiaEE = list("ANTIGUA-AND-BARBUDA", "ARGENTINA", "BAHAMAS"
,
                         "BARBADOS", "BELIZE", "BOLIVIA", "BRAZIL",
                         "CANADA", "CHILE", "COLOMBIA", "COSTA-
RICA",
                         "CUBA", "DOMINICAN-REPUBLIC", "ECUADOR",
"GRENADA",
                         "GUATEMALA", "GUYANA", "HAITI","HONDURAS",
"JAMAICA",
                         "MEXICO", "NICARAGUA", "PANAMA",
"PARAGUAY", "PERU",
                         "ST-KITTS-AND-NEVIS", "ST-LUCIA",
                         "ST-VINCENT-AND-THE-GRENADINES",
"SURINAME",
                         "TRINIDAD-AND-TOBAGO", "UNITED-STATES",
                         "URUGUAY", "VENEZUELA",

                         "AUSTRALIA", "NEW-ZEALAND","PAPUA-NEW-
GUINEA", "FIJI",
                         "VANUATU", "SOLOMON-ISLANDS", "KIRIBATI",
"MICRONESIA",
                         "INDONESIA",
```

```
                          "CHINA","INDIA","JAPAN",
"RUSSIA","PHILIPPINES",
                          "MALAYSIA", "TAIWAN", "LAOS", "THAILAND",
"CAMBODIA","MYANMAR", "VIETNAM",
                          "SRI-LANKA", "SOUTH-KOREA","NORTH-
KOREA","PAKISTAN",
                          "NEPAL", "AFGHANISTAN", "IRAN",
"KAZAKHSTAN", "UZBEKISTAN", "TAJIKISTAN",

                          "TURKEY", "SAOUDI-ARABIA")
    #Filtrage : on retire les points hors de la zone d'Ã©tude
    bigwos_filt <- as.data.frame(bigwos[which(bigwos$pays !=
AmericaAsiaEE[[1]]),])
    for (i in (2:length(AmericaAsiaEE))) {
      bigwos_filt <- bigwos_filt[which(bigwos_filt$pays !=
AmericaAsiaEE[[i]]),]
    }
  }

  #4) Traitement des USA : 7443 pts
  if (ensemble == "usa") {
    bigwos_filt <- bigwos[which(bigwos$pays == "UNITED-STATES"),]
  }


  ### RÃ©cupÃ©ration des coordonnÃ©es et calcul du nombre de
publications ###
  #n?cessaire de mettre dans le bon ordre : long / lat ; convertir en
numeric (factor de base) ; na.omit comme pr?c?demment
  coords <- na.omit(matrix(c(bigwos_filt$lng,bigwos_filt$lat),ncol =
2, dimnames = list(NULL,c("lng","lat")))) #lon lat
  coords <- as.data.frame(coords)
  #on r?cup?re le nb de publi par an pour tous les points qui
poss?dent des coordonn?es
  data = bigwos_filt[,6:21]
  #somme du nb de publi
  nbpubli <- c(sum(data[1,], na.rm = T))
  for (i in 2:length(coords$lng)) {
    nbpubli <- c(nbpubli, sum(data[i,], na.rm = T))
  }
  nbpubli <- as.data.frame(nbpubli)


  ### Cr?ation d'un objet spatial ###
  dtest <- sp::SpatialPointsDataFrame(coords = coords, data =
nbpubli, proj4string = sp::CRS("+proj=longlat +ellps=WGS84")) #pas
projet?
  #plot(dtest@coords)


  ### Calcul de la matrice des distances ###
  #Matrice des distances g?ographiques (calcul?e avec une fonction du
package sp)
  distmatrix <- sp::spDists(x = dtest) #quand on est en non projet?,
pDists renvoie des valeurs en km avec la m?thode des grands cercles !
```

```
  #distmatrix <- sp::spDists(x = dtest@coords[1:15000,], longlat =
T) #traitement d'une sous partie de dtest
  distmatrix <- stats::as.dist(distmatrix) #conversion de la matrice
en objet dist. Sinon pas utilisable par hclust

  ### CLUSTERING ###
  ### AGNES : fastcluster ###
  fast <- fastcluster::hclust(distmatrix, method = "complete")
#clustering
  #m?thode de clustering conditionne le fait de pouvoir utiliser
height comme argument pour couper l'arbre
  #Marche avec complete / single (mais pas g?nial) / average /
ward.D / ward.D2
  fast
  #plot(fast) #afficher le dendrogramme
  #CrÃ©ation des clusters : coupe du dendrogramme Ã  une hauteur
relative
  fastcl <- stats::cutree(fast, h = max(fast$height)*0.5/100)
#Hauteur ? laquelle on coupe l'arbre = 0.5% de la hauteur max pour
matrice distances g?ographiques

  ### HDBSCAN ###
  hdb <- dbscan::hdbscan(distmatrix, minPts = 3, gen_hdbscan_tree =
T, gen_simplified_tree = T) #g?n?rer les arbres

  ### DBSCAN ###
  db <- dbscan::dbscan(distmatrix, minPts = 2, eps = 20) #distmatrix
au lieu de coords. Ici eps = 20 km

  #Stockage des clusters : regroupement des listes du NÂ° de clusters
pour chaque point selon chaque mÃ©thode
  clusters <- list(fastcl, hdb$cluster, db$cluster)


  ### CrÃ©ation du csv stockant le nb de clusters et de points
isolÃ©s par mÃ©thode ###
  #CrÃ©ation d'une matrice stockant les valeurs
  ClustersPtsIso <- matrix(nrow = 3, ncol = 2,
                           dimnames =
list(c("fastcluster","HDBSCAN","DBSCAN"),
                                       c("NbClusters","NbIsoPts")))
  #Remplissage de la matrice
  for (row in (1:length(clusters))) {
    NbClusters = max(clusters[[row]])     #Nb de clusters
    ClusterSize <- as.data.frame(table(clusters[row])) #compte le
nombre de points par cluster
    #Pour fastcluster il faut repÃ©rer les clusters de 1 points
    if (row == 1) {
      IsolatedPts <- ClusterSize[which(ClusterSize$Freq <= 1),]
#Clusters de 1 pts = Points isolÃ©s
      NbIsolatedPts <- length(IsolatedPts$Freq) #Nb de points isolÃ©s
    }
    #Pour HDBSCAN et DBSCAN, il suffit simplement rÃ©cupÃ©rer le nb
de points du cluster d'indice 0
    else {
```

```
      if (ClusterSize$Var1[1] == 0) { #Si le premier indice trouvÃ©
est 0 alors il y a des points isolÃ©s
        NbIsolatedPts <- ClusterSize$Freq[1]
      } else { #Sinon pas de points bruits
        NbIsolatedPts <- 0
      }
    }

    ClustersPtsIso[row,1] = NbClusters #1ere colonne = Nb de clusters
    ClustersPtsIso[row,2] = NbIsolatedPts #2e colonne = Nb de points
isolÃ©s
  }
  #Ecriture du csv
  #La variable ensemble sert Ã  la fois d'indication de rÃ©pertoire
et de nom de fichier
  filename =
paste("/home/sigma/Bureau/Scripts/InfluenceNbPts/",ensemble,"/",ense
mble,"_ClustersIsoPts.csv", sep = "")
  write.csv2(ClustersPtsIso,filename, sep = ";", dec = ".")


  ### Export des rÃ©sultats sous csv ###
  #Tableau ou chaque ligne reprÃ©sente un point et ayant comme
colonnes :
  #ville / province / pays / latitude  / longitude /
  #Nb publi sommÃ©s / NÂ° de cluster pour fastcluster / NÂ° pour
HDBSCAN / NÂ° DBSCAN
  #Id d'agglomÃ©ration (mÃ©thode prÃ©cÃ©dente) / Nom agglomÃ©ration
(mÃ©thode prÃ©cÃ©dente)
  results = data.frame(bigwos_filt[,1:5]) #Stockage de
ville/province/pays/lat/lng
  results$nbpubli = nbpubli$nbpubli #stockage nb de publi sommÃ©es
  results$fastcluster = fastcl
  results$HDBSCAN = hdb$cluster
  results$DBSCAN = db$cluster
  results <- cbind(results, bigwos_filt[,248:249]) #Id et Nom
d'agglo

  filename =
paste("/home/sigma/Bureau/Scripts/InfluenceNbPts/",ensemble,"/",ense
mble,"_results.csv", sep = "")
  write.csv2(results, filename, sep = ";", dec = ".")


} #fin de la grosse boucle for
```

Constraint-based spatial clustering

Algorithm	Description	Limitations
COD-CLARANS (Tung et al. 2001)	• calculate 'unobstructed distance' between points based on a visibility graph • use the pre-clustering method to obtain micro-clusters • construct spatial clusters	• sensitive to density variation and outliers • cannot discover clusters with arbitrary shapes (non-spherical e.g. line-shaped, circle-shaped...) • does not consider facilitators
DBCluC (Zaiane and Lee, 2002)	extends the concepts of density-reachable and density-connected of DBSCAN (an extended constraint-based DBSCAN) • To model the obstacles, it uses internal edges to fill the visible space of obstacles (obstruction lines) • For an entity, its neighbor should be visible to this entity itself by the obstruction line (if two points are not density-connected but there is a facilitator crossing their neighborhoods, they will be regarded as density-connected)	• unable to discover clusters of different densities (clusters with different densities in same dataset and/or clusters with uneven internal density) • obstruction lines may also lead to some additional errors (Wang et al. (2011))
DBRS_O (Wang and Hamilton, 2005) DBRS+ (Wang et al., 2011)	extension of DBRS (DBRS employs a strategy of random sampling to improve the efficiency of the DBSCAN / the concept of purity is developed to consider the non-spatial attribute of an entity) consider the continuity in a neighborhood (unobstructed distance is used to calculate the density if there is an obstacle in the neighborhood of an entity) DBRS+ considers multiple obstacles in a neighborhood & the unobstructed distance is calculated with intersected obstacles as being an integer. The length of the facilitator is further considered by DBRS+	• uneven density • the clustering process becomes more complicated with the consideration of obstacles and facilitator

AUTOCLUST+	an extension of the graph-based AUTOCLUST (Estivill-Castro and Lee, 2004) uses Delaunay triangulation to model spatial proximity among entities (If an edge in the Delaunay triangulation intersects with obstacles, this edge will be removed) able to discover clusters of different densities and/or arbitrary shapes without the need for user-specified parameters	some meaningless small clusters may occur in the clustering results (the statistical indicators are still derived from original Delaunay triangulation network, even if some edges in the original Delaunay triangulation network have been removed) not robust to complicated spatial datasets does not take facilitators into account
ASCDT+	• Obstacles are considered in the process of spatial proximity construction by using of intersection operation and Delaunay triangulation • Multi-level edge constraints (derived from ASCDT (Deng et al., 2011) are used to segment the Delaunay triangulation network into a series of sub-graphs • a new definition of clustering region is proposed • facilitators are implemented by combination operation.	the clustering results only depend on **the length of edge**s. As a result, region with low and uneven density may be segmented into several clusters after removing long edges from the Delaunay triangulation network,and it is difficult to distinguish chains and single line-shaped clusters.

Comparaison des solutions de cluster

Méthodes		Faisabilité technique (puissance de calcul nécessaire, temps d'attente = revient à la complexité = temps de calcul = f(nb de points/éléments), quantité de données gérée, type de logiciel pour l'utilisation)	Paramétrage / Supervision	Performance (= arriver à bien identifier les clusters) et Adaptation des résultats aux besoins
Hierarchical clustering	AGNES (fusion des objets similaires) => à la base chaque objet forme un cluster à part et on les agglomère après	Au moins O(n^2*log(n)) ou n= nombre de points test sur R: 100000 variables avec 20 entrées => qq secondes pour afficher le résultat R package: cluster * fonctions: agnes / diana fastcluster : implémentation optimisée et performante de AGNES : sous Python avec la bibliothèque fastcluster ou sous R avec le package *fastcluster*	Pas de paramétrage mais par contre il est nécessaire de choisir ou est-ce que l'on "coupe" l'arbre de classement ou dendogramme obtenu. Cette étape est en général réalisée de manière arbitraire par l'utilisateur. Néanmoins, il existe des moyens de déterminer automatiquement le niveau auquel le dendogramme sera coupé : indice de stabilité ou de dissimilarité entre les clusters selon si l'on passe de k à k+1 clusters	Performant sauf dans le cas ou il y a du bruit (clusters étalés ou qui se superposent) et assez coûteux en ressources Avantage : Possibilité d'introduire des variables en plus des coordonnées géographiques qui seront considérées dans le calcul des "distances" entre les points (passage de 2 dimensions = lon/lat à N dimensions). Autrement dit les aspects spatial et non-spatial sont traités en même temps. Tous les points sont affectés à un cluster (contrairement à DBSCAN par exemple)
	DIANA => à la base tous les objets forment un seul cluster et on les divise après			

Density-based clustering	DBSCAN => on choisit un pt et on regarde ses voisins pour former un cluster/noise	Implémentation sous PostGIS avec la fonction *ST_ClusterDBSCAN,* Python avec la bibliothèque scikit-learn (*sklearn.cluster.DBSCAN*) ou sous R avec le package *dbscan*. Complexity : O(n*log(n)) au mieux, O(n^2) si le epsilon choisi n'est pas optimal ou si les données comportent beaucoup de dimensions (> 10)	MinPts : le minimum de points pour former un cluster Epsilon : rayon auquel il faut chercher des voisins autour du point, si il y en a autant ou plus que MinPts alors le point est rattaché au cluster ou un cluster est créé	Performant et plus rapide que les méthodes hiérarchiques mise à part fastclust (=classement hiérarchique aggloméré optimisé) ? (moins coûteux). Avantage : Se base sur la densité des points, ce qui permet de réduire l'effet du bruit. Néanmoins l'algorithme fonctionne bien lorsque les clusters ont des densités du même ordre de grandeur. Autrement, les résultats sont moins bon et il est plus difficile de trouver les paramètres optimaux. De plus, seul la densité est considérée ici, il n'est pas possible d'introduire directement d'autres variables dans la détermination des clusters (rugosité par ex). Cependant il est possible de réaliser une analyse de points chauds (Getis-Ord) à partir des clusters obtenus afin de les confronter à d'autres variables. Donne potentiellement moins de cluster car les points peuvent être considéré comme du bruit

	HDBSCAN : Hierarchical DBSCAN	Implémentation sous Python avec la bibliothèque *hdbscan* ou sous R avec le package *dbscan*. Pas besoin d'avoir une puissance de calcul importante. Test de la fonction hdbscan du package R dbscan sur un jeu de donnée de dimension 2 sur 8000 points. Calcul de quelques secondes à peine. Donc à priori pas de soucis pour calculer des gros jeux de données. Complexity : O(n*log(n)) au mieux, O(n^2) si le epsilon choisi n'est pas optimal ou si les données comportent beaucoup de dimensions (> 10)	MinPts : nb minimum de points pour constituer un cluster MinSamples peut également être demandé (pour l'implémentation sous Python) : permet de fixer à quel point l'algorithme est conservatif. Plus cette valeur est grande et plus l'algo est conservatif, càd que plus de points seront déclaré comme étant du bruit et les clusters seront de densité croissante Alpha peut être aussi utilisé mais assez déconseillé (pour l'implémentation sous Python) : détermine également si le cluster et + ou - conservatif	Un peu plus performant et rapide que DBSCAN. Aussi basé sur la densité mais ne fonctionne pas de la même manière. Inclus une étape de classement hiérarchique ainsi qu'un critère de stabilité pour déterminer le nombre de clusters. Comme DBSCAN, certains points ne seront dans aucun cluster (= bruit) Peu ou pas sensibles à des écarts de densité important entre clusters. Paramétrage a peu d'effet sur les résultats.

Hierarchical Clustering Algorithm Using Dynamic Modeling	CHAMELEON	Traitement au maximum de 10000 points dans les exemples d'un article. Néanmoins nécessité à priori d'un très grande puissance de calcul pour utiliser cette méthode (utilisation d'un super computer). Implémentation seulement sous Python avec la bibliothèque PyCHAMELEON. Complexity with n = nb of points and m = nb of subclusters generated by the graph partitioning algorithm based on the k-nearest neighbor graph. m << n : 1) k-nearest neighbor : O(n*log(n)) 2) Two-phase clustering algorithm : O(n*m + n*log(n) + m²*log(m)) Complexity graph : https://www.desmos.com/calculator/a9cx4mppfw	k : Nombre de voisin à considérer pour la création du k-nearest neighbor graph MinSize : nb de points minimum que l'on considérera comme formant un cluster Alpha : user specified parameter. If α>1, then CHAMELEON gives a higher importance to the relative closeness (closeness between 2 subclusters normalised by the inner closeness of their elements). When α<1, it gives a higher importance on the relative inter-connectivity (same as closeness but with inter-connectivity).	A priori l'algorithme le plus lent car la méthode est assez lourde mais le plus robuste.

Statistical methods : spatial autocorrelation	Global scale : Moran's I G and G* (Getis-Ord) Local scale : LISA (= local Moran's I) Local G and G*	Implémentation sous R avec le package *spdep* et sous Python avec la bibli *PySAL*. N'est pas une méthode de clustering en soit, simplement une analyse statistique spatiale => nécessaire de réaliser un clustering après mais que l'on peut baser sur les valeurs d'autocorrélation spatiale ! On réalisera à priori un clustering hiérarchique dessus	Paramétrage et supervision dépendent de la méthode de clustering utilisée sur les valeurs d'autocorrélation spatiale	A voir dans les tests si cela est adapté !