

Graph sketching-based Space-efficient Data Clustering

Anne Morvan^{*†‡}
 anne.morvan@cea.fr

Krzysztof Choromanski[§]
 kchoro@google.com

Cédric Gouy-Pailleur^{*}
 cedric.gouy-pailleur@cea.fr

Jamal Atif[†]
 jamal.atif@dauphine.fr

Abstract

In this paper, we address the problem of recovering arbitrary-shaped data clusters from datasets while facing *high space constraints*, as this is for instance the case in many real-world applications when analysis algorithms are directly deployed on resources-limited mobile devices collecting the data. We present DBMSTClu a new space-efficient density-based *non-parametric* method working on a Minimum Spanning Tree (MST) recovered from a limited number of linear measurements *i.e.* a *sketched* version of the dissimilarity graph \mathcal{G} between the N objects to cluster. Unlike k -means, k -medians or k -medoids algorithms, it does not fail at distinguishing clusters with particular forms thanks to the property of the MST for expressing the underlying structure of a graph. No input parameter is needed contrarily to DBSCAN or the Spectral Clustering method. An approximate MST is retrieved by following the dynamic *semi-streaming* model in handling the dissimilarity graph \mathcal{G} as a stream of edge weight updates which is sketched in one pass over the data into a compact structure requiring $O(N \text{ polylog}(N))$ space, far better than the theoretical memory cost $O(N^2)$ of \mathcal{G} . The recovered approximate MST \mathcal{T} as input, DBMSTClu then successfully detects the right number of nonconvex clusters by performing relevant cuts on \mathcal{T} in a time linear in N . We provide theoretical guarantees on the quality of the clustering partition and also demonstrate its advantage over the existing state-of-the-art on several datasets.

1 Introduction

Clustering is one of the principal data mining tasks consisting in grouping related objects in an unsupervised manner. It is expected that objects belonging to the same cluster are more similar to each other than to objects belonging to different clusters. There exists a variety of algorithms performing this task. Methods like k -

means [1], k -medians [2] or k -medoids [3] are useful unless the number and the shape of clusters are unknown which is unfortunately often the case in real-world applications. They are typically unable to find clusters with a nonconvex shape. Although DBSCAN [4] does not have these disadvantages, its resulting clustering still depends on the chosen parameter values.

One of the successful approaches relies on a graph representation of the data. Given a set of N data points $\{x_1, \dots, x_N\}$, a graph can be built based on the dissimilarity of data where points of the dataset are the vertices and weighted edges express distances between these objects. Besides, the dataset can be already a graph \mathcal{G} modeling a network in many fields, such as bioinformatics - where gene-activation dependencies are described through a network - or social, computer, information, transportation network analysis. The clustering task consequently aims at detecting clusters as groups of nodes that are densely connected with each other and sparsely connected to vertices of other groups. In this context, Spectral Clustering (SC) [9] is a popular tool to recover clusters with particular structures for which classical k -means algorithm fails. When dealing with large scale datasets, a main bottleneck of the technique is to perform the partial eigendecomposition of the associated graph Laplacian matrix, though. Another inherent difficulty is to handle the huge number of nodes and edges of the induced dissimilarity graph: storing all edges can cost up to $O(N^2)$ where N is the number of nodes. Over the last decade, it has been established that the dynamic streaming model [5] associated with linear sketching techniques [6] - also suitable for distributed processing -, is a good way for tackling this last issue. In this paper, the addressed problem falls within a framework of storage limits allowing $O(N \text{ polylog}(N))$ space complexity but not $O(N^2)$.

Contributions. The new clustering algorithm DBMSTClu presented in this paper brings a solution to these issues: 1) detecting arbitrary-shaped data clusters, 2) with no parameter, 3) in a time linear to the number of points, 4) in a space-efficient manner by

^{*}CEA, LIST, 91191 Gif-sur-Yvette, France.

[†]Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE, 75016 Paris, France.

[‡]Partly supported by the DGA (French Ministry of Defense).

[§]Google Brain Robotics, New York, USA.

working on a limited number of linear measurements, a *sketched* version of the streamed dissimilarity graph \mathcal{G} . DBMSTClu returns indeed a partition of the N points to cluster relying only on a Minimum Spanning Tree (MST) of the dissimilarity graph \mathcal{G} taking $O(N)$ space. This MST can be space-efficiently approximatively retrieved in the dynamic semi-streaming model by handling \mathcal{G} as a stream of edge weight updates sketched in only one pass over the data into a compact structure taking $O(N \text{ polylog}(N))$ space. DBMSTClu then automatically identifies the right number of nonconvex clusters by cutting suitable edges of the resulting approximate MST in $O(N)$ time.

The remaining of this paper is organized as follows. In §2 the related work about graph clustering and other space-efficient clustering algorithms is described. §3 gives fundamentals of the sketching technique that can be used to obtain an approximate MST of the dissimilarity graph. DBMSTClu (DB for Density-Based), the proposed MST-based algorithm for clustering is then explained in §4.1, its theoretical guarantees discussed in §4.2 and the implementation enabling its scalability detailed in §4.3. §5 presents the experimental results comparing the proposed clustering algorithm to other existing methods. Finally, §6 concludes the work and discusses future directions.

2 Related work

2.1 General graph clustering. The approach of graph representation of the data has led to an extensive literature over graph clustering related to graph partitioning [7]. From the clustering methods point of view, DenGraph [10] proposes a graph version of DBSCAN which is able to deal with noise while work from [11] focuses on the problem of recovering clusters with considerably dissimilar sizes. Recent works include also approaches from convex optimization using low-rank decomposition of the adjacency matrix [12, 13, 14, 15]. These methods bring theoretical guarantees about the exact recovery of the ground truth clustering for the Stochastic Block Model [16, 17, 18] but demand to compute the eigendecomposition of a $N \times N$ matrix (resp. $O(N^3)$ and $O(N^2)$ for time and space complexity). Moreover they are restricted to unweighted graphs (weights in the work from [15] refer to likeliness of existence of an edge, not a distance between points).

2.2 MST-based graph clustering. The MST is known to help recognizing clusters with arbitrary shapes. Clustering algorithms from this family identify clusters by performing suitable cuts among the MST edges. The first algorithm, called Standard Euclidean MST (SEMST) is from [19] and given a number of ex-

pected clusters, consists in deleting the heaviest edges from the Euclidean MST of the considered graph but this completely fails when the intra-cluster distance is lower than the inter-clusters one. For decades since MST-based clustering methods [20, 21] have been developed and can be classified into the group of density-based methods. MSDR [21] relies on the mean and the standard deviation of edge weights within clusters but will encourage clusters with points far from each other as soon as they are equally “far”. Moreover, it does not handle clusters with less than three points.

2.3 Space-efficient clustering algorithms.

Streaming k -means [23] is a one-pass streaming method for the k -means problem but still fails to detect clusters with nonconvex shapes since only the centroid point of each cluster is stored. This is not the case of CURE algorithm [24] which represents each cluster as a random sample of data points contained in it but this offline method has a prohibitive time complexity of $O(N^2 \log(N))$ not suitable for large datasets. More time-efficient, CluStream [25] and DenStream [26] create microclusters based on local densities in an online fashion and aggregate them later to build bigger clusters in offline steps. Though, only DenStream can capture non-spherical clusters but needs parameters like DBSCAN from which it is inspired.

3 Context, notations and graph sketching preprocessing step

Consider a dataset with N points. Either the underlying network already exists, or it is assumed that a dissimilarity graph \mathcal{G} between points can be built where points of the dataset are the vertices and weighted edges express distances between these objects. For instance, this can be the Euclidean distance. In both cases, the graphs considered here should follow this definition:

DEFINITION 3.1. (GRAPH $\mathcal{G} = (V, E)$) A graph $\mathcal{G} = (V, E)$ consists in a set of nodes V and a set of edges $E \subseteq V \times V$. The graph is undirected but weighted. The weight w on an edge between node i and j - if this edge exists - corresponds to the normalized predefined distance between i and j , s.t. $0 < w \leq 1$. $|V| = N$ and $|E| = M$ stand resp. for the cardinality of sets V and E . $E = \{e_1, \dots, e_M\}$ and for all edges e_i a weight w_i represents a distance between two vertices. In the sequel, $E(\mathcal{G})$ describes the set of edges of a graph \mathcal{G} .

Freely of any parameter, DBMSTClu performs the nodes clustering from its unique input: an MST of \mathcal{G} . So an independent preprocessing is required for its recovery by any existing method. To respect our space restrictions though, the use of a graph sketching

technique is motivated here: from the stream of its edge weights, a sketch of \mathcal{G} is built and then an approximate MST is retrieved exclusively from it.

Streaming graph sketching and approximate MST recovery. Processing data in the dynamic streaming model [5] for graph sketching implies: 1) The graph should be handled as a stream s of edge weight updates: $s = (a_1, \dots, a_j, \dots)$ where a_j is the j -th update in the stream corresponding to the tuple $a_j = (i, w_{old,i}, \Delta w_i)$ with i denoting the index of the edge to update, $w_{old,i}$ its previous weight and Δw_i the update to perform. Thus, after reading a_j in the stream, the i -th edge is assigned the new weight $w_i = w_{old,i} + \Delta w_i \geq 0$. 2) The method should make only one pass over this stream. 3) Edges can be both inserted or deleted (*turnstile* model), i.e. weights can be increased or decreased (but have always to be nonnegative). So weights change regularly, as in social networks where individuals can be friends for some time then not anymore.

The algorithm in [6] satisfies these conditions and is used here to produce in an online fashion a limited number of linear measurements summarizing edge weights of \mathcal{G} , as new data a_j are read from the stream s of edge weight updates. Its general principle is briefly described here. For a given small ϵ_1 , \mathcal{G} is seen as a set of unweighted subgraphs \mathcal{G}_k containing all the edges with weight lower than $(1 + \epsilon_1)^k$, hence $\mathcal{G}_k \subset \mathcal{G}_{k+1}$. The \mathcal{G}_k are embodied as N virtual vectors $v^{(i)} \in \{-1, 0, 1\}^M$ for $i \in [N]^1$ expressing for each node the belonging to an existing edge: for $j \in [M]$, $v_j^{(i)}$ equals to 0 if node i is not in e_j , 1 (resp. -1) if e_j exists and i is its left (resp. right) node. All $v^{(i)}$ are described at L different “levels”, i.e. L virtual copies of the true vectors are made with some entries randomly set to zero s.t. the $v^{(i),l}$ get sparser as the corresponding level $l \in [L]$ increases. The $v^{(i),l}$ for each level are explicitly coded in memory by three counters: $\phi = \sum_{j=1}^M v_j^{(i),l}$; $\iota = \sum_{j=1}^M j \cdot v_j^{(i),l}$; $\tau = \sum_{j=1}^M v_j^{(i),l} z^j \bmod p$, with p a suitably large prime and $z \in \mathbb{Z}_p$. The resulting compact data structure further named a *sketch* enables to draw almost uniformly at random a nonzero weighted edge among \mathcal{G}_k at any time among the levels vectors $v^{(i),l}$ which are 1-sparse (with exactly one nonzero coefficient) thanks to ℓ_0 -sampling [27]:

DEFINITION 3.2. (ℓ_0 -SAMPLING) An (ϵ, δ) ℓ_0 -sampler for a nonzero vector $x \in \mathbb{R}^n$ fails with a probability at most δ or returns some $i \in [n]$ with probability $(1 \pm \epsilon) \frac{1}{|\text{supp } x|}$ where $\text{supp } x = \{i \in [n] \mid x_i \neq 0\}$.

The sketch requires $O(N \log^3(N))$ space. It follows that the sketching is technically *semi-streamed* but in

practice only one pass over the data is needed and the space cost is significantly lower than the theoretical $O(N^2)$ bound. The time cost for each update of the sketch is $\text{polylog}(N)$. The authors from [6] also proposed an algorithm to compute in a single-pass the approximate weight \tilde{W} of an MST \mathcal{T} - the sum of all its edge weights - by appropriate samplings from the sketch in $O(N \text{polylog}(N))$ time. They show that $W \leq \tilde{W} \leq (1 + \epsilon_1) W$ where W stands for the true weight and $\tilde{W} = N - (1 + \epsilon_1)^{r+1} cc(\mathcal{G}_r) + \sum_{k=0}^r \lambda_k cc(\mathcal{G}_k)$ with $\lambda_k = (1 + \epsilon_1)^{k+1} - (1 + \epsilon_1)^k$, $r = \lceil \log_{1+\epsilon_1}(w_{max}) \rceil$ s.t. w_{max} is the maximal weight of \mathcal{G} and cc denotes the number of connected components of the graph in parameter. Here an extended method is applied for obtaining rather an approximate MST - and not simply its weight - by registering edges as they are sampled. Referring to the proof of Lemma 3.4 in [6], the approach is simply justified by applying Kruskal’s algorithm where edges with lower weights are first sampled².

Note that the term MST is kept in the whole paper for the sake of simplicity, but the sketching technique and so does our algorithm enables to recover a Minimum Spanning Forest if the initial graph is disconnected.

4 The proposed MST-based graph clustering method: DBMSTClu

4.1 Principle. Let us consider a dataset with N points. After the sketching phase, an approximate MST further named \mathcal{T} has been obtained with $N - 1$ edges s.t. $\forall i \in [N - 1]$, $0 < w_i \leq 1$. Our density-based clustering method DBMSTClu exclusively relies on this object by performing some cuts among the edges of the tree s.t. $K - 1$ cuts result in K clusters. Note that independently of the technique used to obtain an MST (the sketching method is just a possible one), the space complexity of the algorithm is $O(N)$ which is better than the $O(N^2)$ of Spectral Clustering (SC). The time complexity of DBMSTClu is $O(NK)$ which is clearly less than the $O(N^3)$ one implied by SC. After a cut, obtained clusters can be seen as subtrees of the initial \mathcal{T} and the analysis of their qualities is only based on edges contained in those subtrees. In the sequel, all clusters C_i , $i \in [K]$, are assimilated to their associated subtree of \mathcal{T} and for instance the maximal edge of a cluster will refer to the edge with the maximum weight from the subtree associated to this cluster.

Our algorithm is a parameter-free divisive top-down procedure: it starts from one cluster containing the whole dataset and at each iteration, a cut which

²We would like to thank Mario Lucic for the fruitful private conversation and for coauthoring with Krzysztof Choromanski the MSE sketching extension during his internship at Google.

¹In the sequel, for a given integer a , $[a] = \{1, \dots, a\}$.

maximizes some criterion is performed. The criterion for identifying the best cut to do (if any should be made) at a given stage is a measure of the *validity* of the resulting clustering partition. This is a function of two positive quantities defined below: *Dispersion* and *Separation* of one cluster. The quality of a given cluster is then measured from Dispersion and Separation while the quality of the clustering partition results from the weighted average of all cluster validity indices. Finally, all those measures are based on the value of edge weights and the two latter ones lie between -1 and 1 .

DEFINITION 4.1. (CLUSTER DISPERSION) The Dispersion of a cluster C_i (DISP) is defined as the maximum edge weight of C_i . If the cluster is a singleton (i.e. contains only one node), the associated Dispersion is set to 0. More formally:

$$(4.1) \quad \forall i \in [K], \text{DISP}(C_i) = \begin{cases} \max_{j, e_j \in C_i} w_j & \text{if } |E(C_i)| \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

DEFINITION 4.2. (CLUSTER SEPARATION) The Separation of a cluster C_i (SEP) is defined as the minimum distance between the nodes of C_i and the ones of all other clusters $C_j, j \neq i, 1 \leq i, j \leq K, K \neq 1$ where K is the total number of clusters. In practice, it corresponds to the minimum weight among all already cut edges from \mathcal{T} comprising a node from C_i . If $K = 1$, the Separation is set to 1. More formally, with $\text{Cuts}(C_i)$ denoting cut edges incident to C_i ,

$$(4.2) \quad \forall i \in [K], \text{SEP}(C_i) = \begin{cases} \min_{j, e_j \in \text{Cuts}(C_i)} w_j & \text{if } K \neq 1 \\ 1 & \text{otherwise.} \end{cases}$$

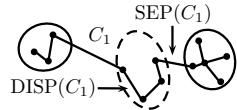


Figure 1: SEP and DISP definitions with $N = 12$, $K = 3$ for dashed cluster C_1 in the middle.

Fig. 1 sums up the introduced definitions. The higher the Separation, the farther is the cluster separated from the other clusters, while low values suggest that the cluster is close to the nearest one.

DEFINITION 4.3. (VALIDITY INDEX OF A CLUSTER) The Validity Index of a cluster C_i is defined as:

$$(4.3) \quad V_C(C_i) = \frac{\text{SEP}(C_i) - \text{DISP}(C_i)}{\max(\text{SEP}(C_i), \text{DISP}(C_i))}$$

The Validity Index of a Cluster (illustration in Fig. 2) is defined s.t. $-1 \leq V_C(C_i) \leq 1$ where 1 stands for the

best validity index and -1 for the worst one. No division by zero (i.e. $\max(\text{DISP}(C_i), \text{SEP}(C_i)) = 0$) happens because Separation is always strictly positive. When Dispersion is higher than Separation, $-1 < V_C(C_i) < 0$. Conversely, when Separation is higher than Dispersion, $0 < V_C(C_i) < 1$. So our clustering algorithm will naturally encourage clusters with a higher Separation over those with a higher Dispersion.

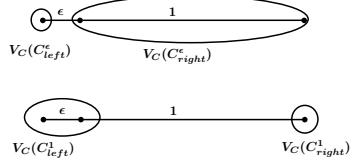


Figure 2: Validity Index of a Cluster's example with $N = 3$. For a small ϵ , cutting edge with weight ϵ or 1 gives resp. left and right partitions. (up) $V_C(C_{\epsilon}^{\epsilon}) = 1$; $V_C(C_{\epsilon}^{\epsilon}) = \epsilon - 1 < 0$. (bottom) $V_C(C_1^1) = 1 - \epsilon > 0$; $V_C(C_1^1) = 1$. The bottom partition, for which validity indices of each cluster are positive, is preferred.

DEFINITION 4.4. (VALIDITY INDEX OF A CLUSTERING PARTITION) The Density-Based Validity Index of a Clustering partition $\Pi = \{C_i\}, 1 \leq i \leq K$, $\text{DBCVI}(\Pi)$ is defined as the weighted average of the Validity Indices of all clusters in the partition where N is the number of points in the dataset.

$$(4.4) \quad \text{DBCVI}(\Pi) = \sum_{i=1}^K \frac{|C_i|}{N} V_C(C_i)$$

The Validity Index of Clustering lies also between -1 and 1 where 1 stands for an optimal density-based clustering partition while -1 stands for the worst one.

Our defined quantities are significantly distinct from the *separation* and *sparseness* defined in [28]. Indeed, firstly, their quantities are not well defined for special cases when clusters have less than four nodes or a partition containing a lonely cluster. Secondly, the way they differentiate internal and external nodes or edges does not properly recover easy clusters like convex blobs. Moreover, our DBCVI differs from the Silhouette Coefficient [29]. It does not perform well with nonconvex-shaped clusters and although this is based on close concepts like *tightness* and also *separation*, the global coefficient is based on the average values of Silhouette coefficients of each point, while our computation of DBCVI begins at the cluster level.

DBMSTClu is summarized in Algorithm 1. It starts from a partition with one cluster containing the whole dataset whereas the associated initial DBCVI is set to the worst possible value: -1 . As long as there exists a cut which makes the DBCVI greater from (or equal

to) the one of the current partition, a cut is greedily chosen by maximizing the obtained DBCVI among all the possible cuts. When no direct improvement is possible, the algorithm stops. It is guaranteed that the cut edge locally maximizes the DBCVI at each iteration since by construction, the algorithm will try each possible cut. In practice, the algorithm stops after a reasonable number of cuts, getting trapped in a local maximum corresponding to a meaningful cluster partition. This prevents from obtaining a partition where all points are in singleton clusters. Indeed, such a result ($K = N$) is not desirable, although it is optimal in the sense of the DBCVI, since in this case, $\forall i \in [K]$, $\text{DISP}(C_i) = 0$ and $V_C(C_i) = 1$. Moreover, the non-parametric characteristic helps achieving stable partitions. In Algorithm 1, `evaluateCut(.)` computes the DBCVI when the cut in parameter is applied to \mathcal{T} .

Algorithm 1 Clustering algorithm DBMSTClu

```

1: Input:  $\mathcal{T}$ , the MST
2:  $dbcvi \leftarrow -1.0$ ;  $clusters = []$ ;  $cut\_list \leftarrow [E(\mathcal{T})]$ 
3: while  $dbcvi < 1.0$  do
4:    $cut\_tp \leftarrow \text{None}$ ;  $dbcvi\_tp \leftarrow dbcvii$ 
5:   for each  $cut$  in  $cut\_list$  do
6:      $newDbcvii \leftarrow \text{evaluateCut}(\mathcal{T}, cut)$ 
7:     if  $newDbcvii \geq dbcvii\_tp$  then
8:        $cut\_tp \leftarrow cut$ ;  $dbcvi\_tp \leftarrow newDbcvii$ 
9:   if  $cut\_tp \neq \text{None}$  then
10:     $clusters \leftarrow \text{cut}(clusters, cut\_tp)$ 
11:     $dbcvi \leftarrow dbcvii\_tp$ ;  $\text{remove}(cut\_list, cut\_tp)$ 
12:   else
13:     break
14: return  $clusters, dbcvii$ 
```

4.2 Quality of clusters. An analysis of the algorithm and the quality of the recovered clusters is now given. The main results are: 1) DBMSTClu differs significantly from the naive approach of SEMST by preferring cuts which do not necessarily correspond to the heaviest edge (Prop. 4.1 and 4.2). 2) As long as the current partition contains at least one cluster with a negative validity index, DBMSTClu will find a cut improving the global index (Prop. 4.3). 3) Conditions are given to determine in advance if and which cut will be performed in a cluster with a positive validity index (Prop. 4.4 and 4.5). All are completely independent of the sketching phase. Prop. 4.1 and 4.2 rely on the two basic lemmas regarding the first cut in the MST:

LEMMA 4.1. HIGHEST WEIGHTED EDGE CASE *Let \mathcal{T} be an MST of the dissimilarity data graph. If the first cut from $E(\mathcal{T})$ made by DBMSTClu is the heaviest edge, then resulting DBCVI is nonnegative.*

Proof. For the first cut, both separations of obtained clusters C_1 and C_2 are equal to the weight of the considered edge for cut. Here, this is the one with the highest weight. Thus, for $i = 1, 2$, $\text{DISP}(C_i) \leq \text{SEP}(C_i) \implies V_C(C_i) \geq 0$. Finally, the DBCVI of the partition, as a convex sum of two nonnegative quantities, is clearly nonnegative.

LEMMA 4.2. LOWEST WEIGHTED EDGE CASE *Let \mathcal{T} be an MST of the dissimilarity data graph. If the first cut from $E(\mathcal{T})$ done by DBMSTClu is the one with the lowest weight, then resulting DBCVI is negative or zero.*

Proof. Same reasoning in the opposite case s.t. $\text{SEP}(C_i) - \text{DISP}(C_i) \leq 0$ for $i \in \{1, 2\}$.

PROPOSITION 4.1. WHEN THE FIRST CUT IS NOT THE HEAVIEST *Let \mathcal{T} be an MST of the dissimilarity data graph with N nodes. Let us consider this specific case: all edges have a weight equal to w except two edges e_1 and e_2 resp. with weight w_1 and w_2 s.t. $w_1 > w_2 > w > 0$. DBMSTClu does not cut any edge with weight w and cuts e_2 instead of e_1 as a first cut iff:*

$$w_2 > \frac{2n_2 w_1 - n_1 + \sqrt{n_1^2 + 4w_1(n_2^2 w_1 + N^2 - Nn_1 - n_2^2)}}{2(N - n_1 + n_2)}$$

where n_1 (resp. n_2) is the number of nodes in the first cluster resulting from the cut of e_1 (resp. e_2). Otherwise, e_1 gets cut.

Proof. See supplementary material.

This proposition emphasizes that the algorithm is cleverer than simply cutting the heaviest edge first. Indeed, although $w_2 < w_1$, cutting e_2 could be preferred over e_1 . Moreover, no edge with weight w can get cut at the first iteration as they have the minimal weight in the tree. Indeed it really happens since an approximate MST with discrete rounded weights is used when sketching is applied.

PROPOSITION 4.2. FIRST CUT ON THE HEAVIEST EDGE IN THE MIDDLE *Let \mathcal{T} be an MST of the dissimilarity data graph with N nodes. Let us consider this specific case: all edges have a weight equal to w except two edges e_1 and e_2 resp. with weight w_1 and w_2 s.t. $w_1 > w_2 > w > 0$. Denote n_1 (resp. n_2) the number of nodes in the first cluster resulting from the cut of e_1 (resp. e_2). In the particular case where edge e_1 with maximal weight w_1 stands between two subtrees with the same number of points, i.e. $n_1 = N/2$, e_1 is always preferred over e_2 as the first optimal cut.*

Proof. See supplementary material.

REMARK 4.1. Let us consider the MST in Fig. 3 with $N = 8$, $w_1 = 1$, $w_2 = w_3 = 1 - \epsilon$, and other weights set to ϵ . Clearly, it is not always preferred to cut e_1 in the middle since for $\epsilon = 0.1$, $DBCVI_2 \approx 0.27 > DBCVI_1 = \epsilon = 0.1$. So, it is a counter-example to a possible generalization of Prop. 4.2 where there would be more than three possible distinct weights in \mathcal{T} .

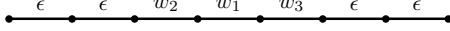


Figure 3: Counter-example for Remark 4.1

These last propositions hold for every iteration in the algorithm.

PROPOSITION 4.3. FATE OF NEGATIVE V_C CLUSTER
Let $K = t+1$ be the number of clusters in the clustering partition at iteration t . If for some $i \in [K]$, $V_C(C_i) < 0$, then DBMSTClu will cut an edge at this stage.

Proof. See supplementary material.

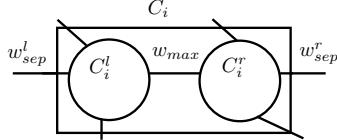


Figure 4: Generic example of Prop. 4.3 and 4.5's proofs.

Thus, at the end of the clustering algorithm, each cluster will have a nonnegative V_C so at each step of the algorithm, this bound holds for the final DBCVI: $DBCVI \geq \sum_{i=1}^K \frac{|C_i|}{N} \max(V_C(C_i), 0)$.

PROPOSITION 4.4. FATE OF POSITIVE V_C CLUSTER I
Let \mathcal{T} be an MST of the dissimilarity data graph and C a cluster s.t. $V_C(C) > 0$ and $SEP(C) = s$. DBMSTClu does not cut an edge e of C with weight $w < s$ if both resulting clusters have at least one edge with weight greater than w .

Proof. See supplementary material.

PROPOSITION 4.5. FATE OF POSITIVE V_C CLUSTER II

Consider a partition with K clusters s.t. some cluster C_i , $i \in [K]$ with $V_C(C_i) > 0$ is in the setting of Fig. 10 i.e. cutting the heaviest edge e with weight w_{max} results in two clusters: the left (resp. right) cluster C_i^l (resp. C_i^r) with n_1 points (resp. n_2) s.t. $DISP(C_i^l) = d_1$, $SEP(C_i^l) = w_{sep}^l$, $DISP(C_i^r) = d_2$ and $SEP(C_i^r) = w_{sep}^r$. Assuming w.l.o.g. $w_{sep}^l > w_{sep}^r$, e gets cut iff:

$$\frac{\left(\frac{n_1 d_1 + n_2 d_2}{n_1 + n_2}\right)}{w_{max}} \leq \frac{w_{max}}{w_{sep}^r}.$$

Proof. See supplementary material.

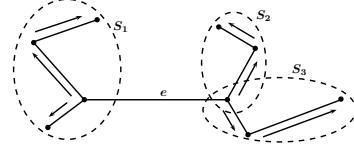


Figure 5: Illustration of the recursive relationship for left and right Dispersions resulting from the cut of edge e : $DISP_{left}(e) = \max(w(S_1))$, $DISP_{right}(e) = \max(w(S_2), w(S_3))$ where $w(.)$ returns the edge weights. Separation works analogically.

4.3 Implementation for linear time and space complexities.

Algorithm 1 previously described could lead to a naive implementation. We now briefly explain how to make the implementation efficient in order to achieve the linear time and space complexities in N (code on <https://github.com/annemorvan/DBMSTClu/>). The principle is based on two tricks. 1) As observed in §4.2: for a performed cut in cluster C_i , $V_C(C_j)$ for any $j \neq i$ remain unchanged. Hence, if $V_C(C_j)$ are stored for each untouched cluster after a given cut, only the edges of C_l and C_r resp. the left and right clusters induced by e 's cut need to be evaluated again to determine the DBCVI in case of cut. Thus the number of operations to find the optimal cut decreases drastically over time as the clusters become smaller through the cuts. 2) However finding the first cut already costs $O(N)$ time hence paying this price for each cut evaluation would lead to $O(N^2)$ operations. Fortunately, this can be avoided as SEP and DISP exhibit some recurrence relationship in \mathcal{T} : when knowing these values for a given cut, we can deduce the value for a neighboring cut (cf. Fig. 5). To determine the first cut, \mathcal{T} should be hence completely crossed following the iterative version of the Depth-First Search. The difficulty though is that the recursive relationship between the quantities to update is directional: left and right w.r.t. the edge to cut. So we develop here *double Depth-First search* (see principle in Algorithm 2): from any given edge of \mathcal{T} , edges left and right are all visited consecutively with a Depth-First search, and SEP and DISP are updated recursively thanks to a carefully defined order in the priority queue of edges to handle.

5 Experiments

Tight lower and upper bounds are given in §3 for the weight of the approximate MST retrieved by sketching. First experiments in §5.1 show that the clustering results do not suffer from the use of an approximate MST instead of a real one. Experiments from §5.2 prove then the scalability of DBMSTClu for large values of N .

5.1 Safety of the sketching. The results of DBMSTClu are first compared with DBSCAN [4] because it can compete with DBMSTClu as 1) nonconvex-shaped

Algorithm 2 Generic Double Depth-First Search

```

1: Input:  $\mathcal{T}$ , the MST;  $e$ , the edge of  $\mathcal{T}$  where the
   search starts;  $n\_src$ , source node of  $e$ 
2:  $Q = \text{dequeue}()$  //Empty priority double-ended queue
3: for incident edges to  $n\_src$  do
4:   pushBack( $Q, (incident\_e, n\_src, FALSE)$ )
5:   pushBack( $Q, (e, n\_src, TRUE)$ )
6: for incident edges to  $n\_trgt$  do
7:   pushBack( $Q, (incident\_e, n\_trgt, FALSE)$ )
8:   pushBack( $Q, (e, n\_trgt, TRUE)$ )
9: while  $Q$  is not empty do
10:    $e, node, marked = \text{popFront}(Q)$ 
11:    $opposite\_node = \text{getOtherNodeEdge}(e, node)$ 
12:   if not  $marked$  then
13:     for incident edges to  $node$  do
14:       pushBack( $Q, (incident\_e, node, F)$ )
15:       pushBack( $Q, (e, node, T)$ )
16:       pushFront( $Q, (e, opposite\_node, T)$ )
17:     else
18:       doTheJob( $e$ ) //Perform here the task
19: return

```

clusters are recognized, 2) it does not require explicitly the number of expected clusters, 3) it is both time and space-efficient (resp. $O(N \log N)$ and $O(N)$). Then, the results of another MST-based algorithm are shown for comparison. The latter called Standard Euclidean MST (SEMST) [19] cuts the $K - 1$ heaviest edges of a standard Euclidean MST given a targeted number of clusters K . For DBMSTClu, the dissimilarity graph is built from computing the Euclidean distance between data points and passed into the sketch phase to produce an approximate version of the exact MST.

Synthetic datasets. Experiments were performed on two classic synthetic datasets from the Euclidean space: noisy circles and noisy moons. Each dataset contains 1000 data points in 20 dimensions: the first two dimensions are randomly drawn from predefined 2D-clusters, as shown in Fig. 6 and 7, while the other 18 dimensions are random Gaussian noise.

Real dataset. DBMSTClu performances are also measured on the mushroom dataset (<https://archive.ics.uci.edu/ml/datasets/mushroom>). It contains 8124 records of 22 categorical attributes corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family. 118 binary attributes are created from the 22 categorical ones, then the complete graph (about 33 millions of edges) is built by computing the normalized Hamming distance (i.e. the number of distinct bits) between points.

Results. Fig. 6, 7 and 8 show the results for all previously defined datasets and aforementioned meth-

ods. The synthetic datasets were projected onto 2D spaces for visualization purposes. They were produced with a noise level such that SEMST fails and DBSCAN does not perform well without parameters optimization. In particular, for DBSCAN all the cross points correspond to noise. With the concentric circles, SEMST does not cut on the consistent edges, hence leads to an isolated singleton cluster. DBSCAN classifies the same point plus a near one as noise while recovering the two circles well. Finally, DBMSTClu finds the two main clusters and also creates five singleton clusters which can be legitimately considered as noise as well. With noisy moons, while DBSCAN considers three outliers, DBMSTClu detects the same as singleton clusters. As theoretically proved above, experiments emphasize the fact that our algorithm is more subtle than simply cutting the heaviest edges as the failure of SEMST shows. Moreover our algorithm exhibits an ability to detect outliers, which could be labeled as noise in a postprocessing phase. Another decisive advantage of our algorithm is the absence of any required parameters, contrarily to DBSCAN. For the mushroom dataset, if suitable parameters are given to SEMST and DBSCAN, the right 23 clusters get found while DBMSTClu retrieves them without tuning any parameter. Quantitative results for the synthetic datasets are shown in Table 3: the achieved silhouette coefficient (between -1 and 1), Adjusted Rand Index (ARI) (between 0 and 1) and DBCVI. For all the indices, the higher, the better. Further analysis can be read in supplementary material. For the mushroom dataset, the corresponding DBCVI and silhouette coefficient are resp. 0.75 and 0.47.

	Silhouette coeff.		ARI		DBCVI	
SEMST	0.16	-0.12	0	0	0.001	0.06
DBSCAN	0.02	0.26	0.99	0.99	-0.26	0.15
DBMSTClu	-0.26	0.26	0.99	0.99	0.18	0.15

Table 1: Silhouette coefficients, ARI and DBCVI for the noisy circles (left) and noisy moons (right) datasets.

5.2 Scalability of the clustering. For mushroom dataset, DBMSTClu's execution time (avg. on 5 runs) is 3.36s while DBSCAN requires 9.00s. This gives a first overview of its ability to deal with high number of clusters. Further experiments on execution time were conducted on large-scale random weighted graphs generated from the Stochastic Block Model with varying equal-sized number of clusters K and N . The scalability of DBMSTClu is shown in Fig. 9 and Table 2 by exhibiting the linear time complexity in N . Graphs with 1M of nodes and 100 clusters were easily clustered. In Table 2, the row with the execution time ratio between $K = 100$ and $K = 5$ illustrates the first trick from §4.3 as the observed time ratio is around 2/3 of the

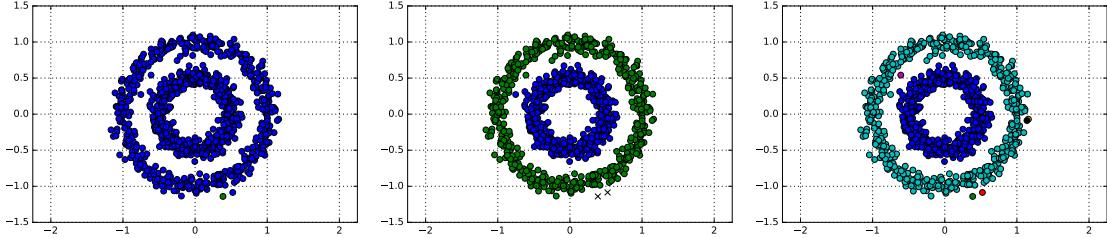


Figure 6: Noisy circles: SEMST, DBSCAN ($\epsilon = 0.15$, $minPts = 5$), DBMSTClu with an approximate MST.

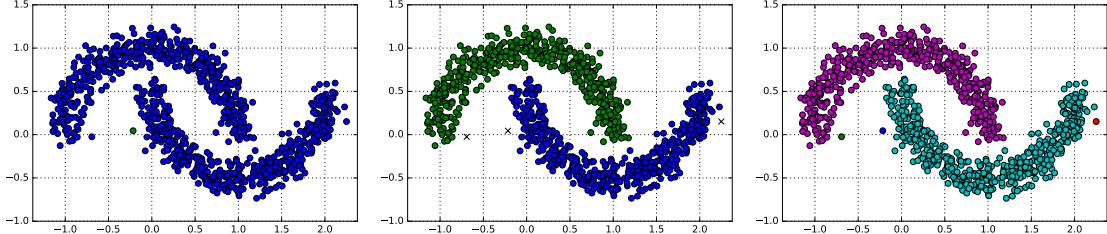


Figure 7: Noisy moons: SEMST, DBSCAN ($\epsilon = 0.16$, $minPts = 5$), DBMSTClu with an approximate MST.

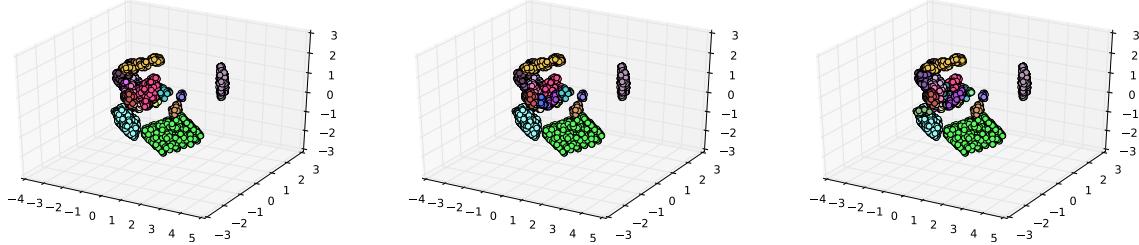


Figure 8: Mushroom dataset: SEMST, DBSCAN ($\epsilon = 1.5$, $minPts = 2$), DBMSTClu with an approximate MST (projection on the first three principal components).

theoretical one $100/5 = 20$.

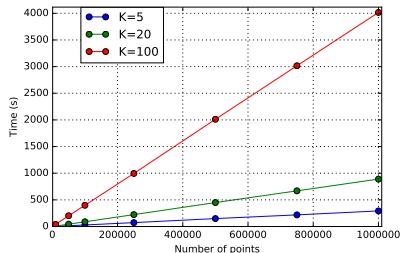


Figure 9: DBMSTClu's execution time with values of $N \in \{1K, 10K, 50K, 100K, 250K, 500K, 750K, 1M\}$.

6 Conclusion

In this paper we introduced DBMSTClu a novel *space-efficient* Density-Based Clustering algorithm which only relies on a Minimum Spanning Tree (MST) of the dissimilarity graph \mathcal{G} : the spatial and time costs are resp. $O(N)$ and $O(NK)$ with N the number of data points and K the number of clusters. This enables to deal easily with graphs of million of nodes. Moreover, DBMST-

Clu is *non-parametric*, unlike most existing clustering methods: it automatically determines the right number of nonconvex clusters. Although the approach is fundamentally independent from the sketching phase, its robustness has been assessed by using as input an approximate MST of the sketched \mathcal{G} rather than an exact one. The graph sketch is computed dynamically on the fly as new edge weight updates are read in only one pass over the data. This brings a space-efficient solution for finding an MST of \mathcal{G} when the $O(N^2)$ edges cannot fit in memory. Hence, our algorithm adapts to the semi-streaming setting with $O(N \text{ polylog } N)$ space. Our approach shows promising results, as evidenced by the experimental part regarding time and space scalability and clustering performance even with sketching. Further work would consist in using this algorithm in privacy issues, as the lost information when sketching might ensure data privacy. Moreover, as it is already the case for the graph sketching, we could look for adapting both the MST recovery and DBMSTClu to the fully online setting, i.e. to be able to modify dynamically cur-

$K \setminus N$	1000	10000	50000	100000	250000	500000	750000	1000000
5	0.34	2.96	14.37	28.91	73.04	148.85	218.11	292.25
20	0.95	8.73	43.71	88.51	223.18	449.37	669.29	889.88
100	4.36	40.25	201.76	398.41	995.42	2011.79	3015.61	4016.13
“100/5”	12.82	13.60	14.04	13.78	13.63	13.52	13.83	13.74

Table 2: Numerical values for DBMSTClu’s execution time (in s) varying N and K (avg. on 5 runs). The last row shows the execution time ratio between $K = 100$ and $K = 5$.

rent MST and clustering partition as a new edge weight update from the stream is seen.

References

- [1] S. Lloyd, *Least Squares Quantization in PCM*, IEEE Trans. Inf. Theor., 28 (1982), pp. 129–137.
- [2] A. Jain and R. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Upper Saddle River, NJ, USA, 1988.
- [3] L. Kaufman and P. Rousseeuw, *Clustering by means of medoids*, Statistical Data Analysis Based on the L_1 Norm and Related Methods, pp. 405–416, 1987.
- [4] M. Ester, H. Kriegel, J. Sander, X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, AAAI Press (1996), pp. 226–231.
- [5] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, Found. Trends Theor. Comput. Sci., 1 (2005), pp. 117–236.
- [6] K. Ahn, S. Guha and A. McGregor, *Analyzing Graph Structure via Linear Measurements*, ACM-SIAM Symposium on Discrete Algorithms, 2012, Kyoto, Japan, pp. 459–467.
- [7] S. Schaeffer, *Survey: Graph Clustering*, Comput. Sci. Rev., 1 (2007), pp. 27–64.
- [8] M. Girvan and M. Newman, *Community Structure in Social and Biological Networks*, Proceedings of the National Academy of Sciences of the United States of America, 99 (2012), pp.7821–7826.
- [9] M. Nascimento and A. de Carvalho, *Spectral methods for graph clustering - A survey*, European Journal of Operational Research, 211 (2011), pp.221–231.
- [10] T. Falkowski, A. Barth and M. Spiliopoulou, *DENGRAPH: A Density-based Community Detection Algorithm*, IEEE/WIC/ACM International Conference on Web Intelligence, pp. 112–115, 2007.
- [11] N. Ailon, Y. Chen and H. Xu, *Breaking the Small Cluster Barrier of Graph Clustering*, CoRR, abs/1302.4549, 2013.
- [12] S. Oymak and B. Hassibi, *Finding Dense Clusters via “Low Rank + Sparse” Decomposition*, CoRR, abs/1104.5186, 2011.
- [13] Y. Chen, S. Sanghavi and H. Xu, *Clustering Sparse Graphs*, Advances in Neural Information Processing Systems, 25 (2012), pp. 2204–2212.
- [14] Y. Chen, A. Jalali, S. Sanghavi and H. Xu, *Clustering Partially Observed Graphs via Convex Optimization*, J. Mach. Learn. Res., 15 (2014), pp. 2213–2238.
- [15] Y. Chen, and S. Lim and H. Xu, *Weighted Graph Clustering with Non-uniform Uncertainties*, International Conference on Machine Learning, 32 (2014), Beijing, China, pp. II-1566–II-1574.
- [16] P. Holland, K. Laskey and S. Leinhardt, *Stochastic blockmodels: First steps*, Social Networks, 5 (1983), pp.109–137.
- [17] A. Condon and R. Karp, *Algorithms for Graph Partitioning on the Planted Partition Model*, Random Struct. Algorithms, 18 (2001), pp.116–140.
- [18] K. Rohe, S. Chatterjee and B. Yu, *Spectral clustering and the high-dimensional stochastic blockmodel*, Ann. Statist., 39 (2011), pp. 1878–1915.
- [19] C. Zahn, *Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters*, IEEE Trans. Comput., 20, (1971), pp. 68–86.
- [20] T. Asano, B. Bhattacharya, M. Keil, F. Yao, *Clustering Algorithms Based on Minimum and Maximum Spanning Trees*, Symposium on Computational Geometry, Urbana-Champaign, IL, USA, pp. 252–257, 1988.
- [21] O. Grygorash, Y. Zhou and Z. Jorgensen, *Minimum Spanning Tree Based Clustering Algorithms*, IEEE Int.Conference on Tools with Artificial Intelligence, pp. 73–81, 2006.
- [22] K. Ahn, S. Guha and A. McGregor, *Graph Sketches: Sparsification, Spanners, and Subgraphs*, ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Scottsdale, AZ, USA, pp. 5–14, 2012.
- [23] N. Ailon, R. Jaiswal, C. Monteleoni, *Streaming k-means approximation*, Advances in Neural Information Processing Systems, 22 (2009), pp. 10–18.
- [24] S. Guha, R. Rastogi and K. Shim, *Cure: An Efficient Clustering Algorithm for Large Databases*, Inf. Syst., 26 (2001), pp. 35–58.
- [25] C. Aggarwal, J. Han, J. Wang, and P. Yu, *A Framework for Clustering Evolving Data Streams*, International Conference on Very Large Data Bases, 29 (2003), Berlin, Germany, pp.81–92.
- [26] F. Cao, M. Ester, W. Qian and A. Zhou, *Density-based clustering over an evolving data stream with noise*, SIAM Conference on Data Mining (2006), pp. 328–339.
- [27] G. Cormode and D. Firmani, *A unifying framework for ℓ_0 -sampling algorithms*, Distributed and Parallel Databases, 32 (2014), pp. 315–335.
- [28] D. Moulavi, P. Jaskowiak, R. Campello, A. Zimek and J. Sander, *Density-Based Clustering Validation*, SIAM ICDM, Philadelphia, PA, USA, pp. 839–847, 2014.
- [29] P. Rousseeuw, *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*, J. Comput. and Appl. Math., 20 (1987), pp. 53–65.

7 Proofs.

7.1 Proof of Prop. 4.1.

PROPOSITION 4.1. WHEN THE FIRST CUT IS NOT THE HEAVIEST *Let \mathcal{T} be an MST of the dissimilarity data graph with N nodes. Let us consider this specific case: all edges have a weight equal to w except two edges e_1 and e_2 resp. with weight w_1 and w_2 s.t. $w_1 > w_2 > w > 0$. DBMSTClu does not cut any edge with weight w and cuts e_2 instead of e_1 as a first cut iff:*

$$w_2 > \frac{2n_2w_1 - n_1 + \sqrt{n_1^2 + 4w_1(n_2^2w_1 + N^2 - Nn_1 - n_2^2)}}{2(N - n_1 + n_2)}$$

where n_1 (resp. n_2) is the number of nodes in the first cluster resulting from the cut of e_1 (resp. e_2). Otherwise, e_1 gets cut.

Proof. Let $DBCVI_1$ (resp. $DBCVI_2$) be the DBCVI after cut of e_1 (resp. e_2). As w (resp. w_1) is the minimum (resp. maximal) weight, the algorithm does not cut e since the resulting DBCVI would be negative (cf. Lemma 4.2) while $DBCVI_1$ is guaranteed to be positive (cf. Lemma 4.1). So, the choice will be between e_1 and e_2 but e_2 gets cut iff $DBCVI_2 > DBCVI_1$. $DBCVI_1$ and $DBCVI_2$ expressions are simplified w.l.o.g. by scaling the weights by w s.t. $w \leftarrow 1$, $w_1 \leftarrow w_1/w$, $w_2 \leftarrow w_2/w$, hence $w_1 > w_2 > 1$. Then,

$$\begin{aligned} DBCVI_2 &> DBCVI_1 > 0 \\ \iff & \frac{n_2}{N} \left(\frac{w_2}{w_1} - 1 \right) + \left(1 - \frac{n_2}{N} \right) \left(1 - \frac{1}{w_2} \right) \\ & - \frac{n_1}{N} \left(1 - \frac{1}{w_1} \right) + \left(1 - \frac{n_1}{N} \right) \left(1 - \frac{w_2}{w_1} \right) > 0 \\ \iff & w_2^2 \underbrace{(N + n_2 - n_1)}_a + w_2 \underbrace{(n_1 - 2n_2w_1)}_b \\ & + \underbrace{(n_2 - N)w_1}_{c<0} > 0. \end{aligned}$$

Clearly, $\Delta = b^2 - 4ac$ is positive and c/a is negative. But $w_2 > 0$, then $w_2 > \frac{-b+\sqrt{b^2-4ac}}{2a}$ which gives the final result after some simplifications.

7.2 Proof of Prop. 4.2.

PROPOSITION 4.2. FIRST CUT ON THE HEAVIEST EDGE IN THE MIDDLE *Let \mathcal{T} be an MST of the dissimilarity data graph with N nodes. Let us consider this specific case: all edges have a weight equal to w except two edges e_1 and e_2 resp. with weight w_1 and w_2 s.t. $w_1 > w_2 > w > 0$. Denote n_1 (resp. n_2) the number of nodes in the first cluster resulting from the cut of e_1 (resp. e_2). In the particular case where edge e_1 with*

maximal weight w_1 stands between two subtrees with the same number of points, i.e. $n_1 = N/2$, e_1 is always preferred over e_2 as the first optimal cut.

Proof. A reductio ad absurdum is made by showing that cutting edge e_2 i.e. $DBCVI_2 > DBCVI_1$ leads to the contradiction $w_1/w < 1$. With the scaling process from Prop. 4.1' proof:

$$\begin{aligned} DBCVI_1 &= \frac{1}{2} \left(1 - \frac{1}{w_1} \right) + \frac{1}{2} \left(1 - \frac{w_2}{w_1} \right) = 1 - \frac{1}{2w_1} - \frac{w_2}{2w_1} \\ DBCVI_2 &= \frac{n_2}{N} \left(\frac{w_2}{w_1} - 1 \right) + \left(1 - \frac{n_2}{N} \right) \left(1 - \frac{1}{w_2} \right) \\ &= 1 - \frac{1}{w_2} + \frac{n_2}{N} \underbrace{\left(\frac{w_2}{w_1} + \frac{1}{w_2} - 2 \right)}_{=A} \end{aligned}$$

There is $w_2 > w = 1$, so $\frac{1}{w_2} < 1$. Besides $w_2 < w_1$ so $\frac{w_2}{w_1} < 1$ thus, $A < 0$. Let now consider w.l.o.g. that edge e_2 is on the "right side" (right cluster/subtree) of e_1 (similar proof if e_2 is on the left side of e_1). Hence, it is clear that for maximizing $DBCVI_2$ as a function of n_2 , we need $n_2 = n_1 + 1$. Then,

$$\begin{aligned} DBCVI_2 &> DBCVI_1 \\ \iff & -\frac{1}{w_2} + \left(\frac{1}{2} + \frac{1}{N} \right) \left(\frac{w_2}{w_1} - 2 + \frac{1}{w_2} \right) > -\frac{1}{w_1} - \frac{w_2}{w_1} \\ \iff & \left(\frac{1}{2w_1} + \frac{1}{Nw_1} + \frac{1}{2w_1} \right) w_2 - 1 - \frac{2}{N} + \frac{1}{2w_1} \\ & + \left(-1 + \frac{1}{2} + \frac{1}{N} \right) \frac{1}{w_2} > 0 \\ \iff & \underbrace{\left(1 + \frac{1}{N} \right) w_2^2}_{{a>0}} + w_2 \underbrace{\left(\frac{1}{2} - w_1 \left(1 + \frac{2}{N} \right) \right)}_{{b<0}} \\ & + w_1 \underbrace{\left(\frac{1}{N} - \frac{1}{2} \right)}_{{c<0}} > 0 \end{aligned}$$

As $c/a < 0$ and $w_2 > 0$, $w_2 > \frac{N}{2(N+1)} [w_1 \left(1 + \frac{2}{N} \right) - \frac{1}{2} + \sqrt{\Delta}]$ with $\Delta = (w_1 \left(1 + \frac{2}{N} \right) - \frac{1}{2})^2 + 4(1 + \frac{1}{N})(\frac{1}{2} - \frac{1}{N})w_1$. This inequality is incompatible with $w_1 > w_2$ since:

$$\begin{aligned} w_1 > w_2 &\iff w_1 > \frac{N}{2(N+1)} [w_1 \left(1 + \frac{2}{N} \right) - \frac{1}{2} + \sqrt{\Delta}] \\ \iff w_1 + \frac{1}{2} &> \sqrt{\Delta} \\ \iff \frac{4}{N} w_1^2 \left(1 + \frac{1}{N} \right) &+ \frac{4}{N} w_1 \left(-1 - \frac{1}{N} \right) < 0 \\ \iff w_1 < 1 &: ILLICIT \end{aligned}$$

Indeed, after the scaling process, $w_1 < 1 = w$ is not possible since by hypothesis, $w_1 > w$. Finally, it is not allowed to cut e_2 , the only remaining possible edge to cut is e_1 .

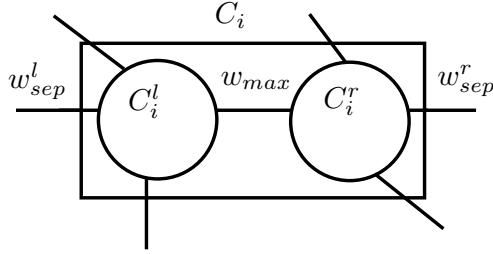


Figure 10: Generic example for proof of Prop. 4.3 and 4.5.

7.3 Proof of Prop. 4.3.

PROPOSITION 4.3. FATE OF NEGATIVE V_C CLUSTER
Let $K = t+1$ be the number of clusters in the clustering partition at iteration t . If for some $i \in [K]$, $V_C(C_i) < 0$, then DBMSTClu will cut an edge at this stage.

Proof. Let $i \in [K]$ s.t. $V_C(C_i) < 0$ i.e. $\text{SEP}(C_i) < \text{DISP}(C_i)$. We denote w_{sep}^l the minimal weight outing cluster C_i and w_{max} the maximal weight in subtree S_i of C_i i.e. $\text{SEP}(C_i) \stackrel{\text{def}}{=} w_{sep}^l$ and $\text{DISP}(C_i) \stackrel{\text{def}}{=} w_{max}$. Hence, $w_{sep}^l < w_{max}$. By cutting the cluster C_i on the edge with weight w_{max} , we define C_i^l and C_i^r resp. the left and right resulting clusters.

Let us look at $V_C(C_i^l)$. If $\text{SEP}(C_i^l) \geq \text{DISP}(C_i^l)$ then $V_C(C_i^l) \geq 0 \geq V_C(C_i)$ else $V_C(C_i^l) = \frac{\text{SEP}(C_i^l)}{\text{DISP}(C_i^l)} - 1$. The definition of the Separation as a minimum and our cut imply that

$$\text{SEP}(C_i^l) \geq \min(\text{SEP}(C_i), w_{max}) \geq \text{SEP}(C_i).$$

Also the definition of the Dispersion as a maximum implies that $\text{DISP}(C_i^l) \leq \text{DISP}(C_i)$. Hence we get that $\frac{\text{SEP}(C_i^l)}{\text{DISP}(C_i^l)} - 1 \geq \frac{\text{SEP}(C_i)}{\text{DISP}(C_i)} - 1$ i.e. $V_C(C_i^l) \geq V_C(C_i)$ in this case too. The same reasoning holds for C_i^r showing that $V_C(C_i^r) \geq V_C(C_i)$. Finally,

$$\begin{aligned} DBCVI_{\text{aftercut}} &= \sum_{j \neq i} \frac{n_j}{N} V_C(C_j) + \frac{n_i^l}{N} V_C(C_i^l) + \frac{n_i^r}{N} V_C(C_i^r) \\ &\geq \sum_{j \neq i} \frac{n_j}{N} V_C(C_j) + \frac{n_i^l}{N} V_C(C_i) + \frac{n_i^r}{N} V_C(C_i) \\ &= DBCVI_{\text{beforecut}}. \end{aligned}$$

Hence cutting the edge with maximal weight in C_i improves the resulting DBCVI.

7.4 Proof of Prop. 4.4.

PROPOSITION 4.4. FATE OF POSITIVE V_C CLUSTER I
Let \mathcal{T} be an MST of the dissimilarity data graph and C a cluster s.t. $V_C(C) > 0$ and $\text{SEP}(C) = s$. DBMSTClu does not cut an edge e of C with weight $w < s$ if both resulting clusters have at least one edge with weight greater than w .

Proof. Let us consider clusters C_1 and C_2 resulting from the cut of edge e . Assume that in the associated subtree of C_1 (resp. C_2), there is an edge e_1 (resp. e_2) with a weight w_1 (resp. w_2) higher than w s.t. without loss of generality, $w_1 > w_2$. Since $V_C(C) > 0$, $s > w_1 > w_2 > w$. But cutting edge e implies that for $i \in \{1, 2\}$, $\text{DISP}(C_i) > \text{SEP}(C_i) = w$, and thus $V_C(C_i) < 0$. Cutting edge e would therefore mean to replace a cluster C s.t. $V_C(C) > 0$ by two clusters s.t. for $i \in \{1, 2\}$, $V_C(C_i) < 0$ which obviously decreases the current DBCVI. Thus, e does not get cut at this step of the algorithm.

7.5 Proof of Prop. 4.5.

PROPOSITION 4.5. FATE OF POSITIVE V_C CLUSTER II
Consider a partition with K clusters s.t. some cluster C_i , $i \in [K]$ with $V_C(C_i) > 0$ is in the setting of Fig. 10 i.e. cutting the heaviest edge e with weight w_{max} results in two clusters: the left (resp. right) cluster C_i^l (resp. C_i^r) with n_1 points (resp. n_2) s.t. $\text{DISP}(C_i^l) = d_1$, $\text{SEP}(C_i^l) = w_{sep}^l$, $\text{DISP}(C_i^r) = d_2$ and $\text{SEP}(C_i^r) = w_{sep}^r$. Assuming w.l.o.g. $w_{sep}^l > w_{sep}^r$, e gets cut iff:

$$\frac{\left(\frac{n_1 d_1 + n_2 d_2}{n_1 + n_2}\right)}{w_{max}} \leq \frac{w_{max}}{w_{sep}^r}.$$

Proof. As $V_C(C_i) > 0$, there is $\text{SEP}(C_i) = w_{sep}^r > w_{max}$. Then, the DBCVI before (K clusters) and after cut of w_{max} ($K+1$ clusters) are:

$$\begin{aligned} DBCVI_K &= \sum_{j \neq i}^K V_C(C_j) + \frac{n_1 + n_2}{N} \left(1 - \frac{w_{max}}{w_{sep}^r}\right) \\ DBCVI_{K+1} &= \sum_{j \neq i}^K V_C(C_j) + \frac{n_1}{N} \left(1 - \frac{d_1}{w_{max}}\right) \\ &\quad + \frac{n_2}{N} \left(1 - \frac{d_2}{w_{max}}\right) \end{aligned}$$

DBMSTClu cuts w_{max} iff $DBCVI_{K+1} \geq DBCVI_K$. So the result after simplification.

8 Complements on experiments.

Experiments were conducted using Python and scikit-learn library [1] on a single-thread process on an intel processor based node.

8.1 Safety of the sketching. Fig. 11 shows another result on a synthetic dataset: three blobs generated from three Gaussian distributions. With the three blobs, each method SEMST, DBSCAN and DBMSTClu performs well: they all manage to retrieve three clusters.

Quantitative results for the three synthetic datasets are shown in Table 3: the achieved silhouette coefficient, Adjusted Rand Index (ARI) and DBCVI. For all the indices, the higher, the better. Silhouette coefficient (between -1 and 1) is used to measure a clustering partition without any external information. For DBSCAN it is computed by considering noise points as singletons. We see that this measure is not very suitable for nonconvex clusters like noisy circles or moons. The ARI (between 0 and 1) measures the similarity between the experimental clustering partition and the known groundtruth. DBSCAN and DBMSTClu give similar almost optimal results. Finally, the obtained DBCVIs are consistent, since the best ones are reached for DBMSTClu.

References

- [1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

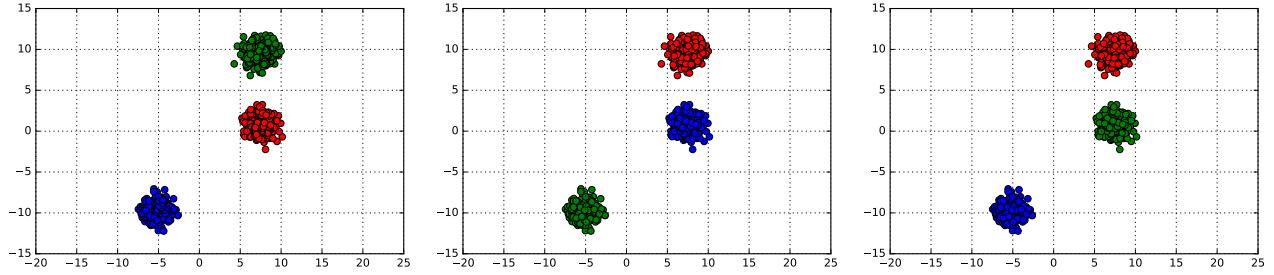


Figure 11: Three blobs: SEMST, DBSCAN ($\epsilon = 1.4$, $minPts = 5$), DBMSTClu with an approximate MST.

	Silhouette coeff.			Adjusted Rand Index			DBCVI		
SEMST	0.84	0.16	-0.12	1	0	0	0.84	0.001	0.06
DBSCAN	0.84	0.02	0.26	1	0.99	0.99	0.84	-0.26	0.15
DBMSTClu	0.84	-0.26	0.26	1	0.99	0.99	0.84	0.18	0.15

Table 3: Silhouette coefficients, Adjusted Rand Index and DBCVI for the blobs, noisy circles and noisy moons datasets with SEMST, DBSCAN and DBMSTClu.