A Polynomial Algorithm for Balanced Clustering via Graph Partitioning*

Luis Evaristo Caraballo†[1], José-Miguel Díaz-Báñez[1], and Nadine Kroher[1]

1 **Department of Applied Mathematics II, University of Seville**
{lcaraballo,dbanez,nkroher}@us.es

Abstract

The objective of clustering is to discover natural groups in datasets and to identify geometrical structures which might reside there, without assuming any prior knowledge on the characteristics of the data.The problem can be seen as detecting the inherent separations between groups of a given point set in a metric space governed by a similarity function. The pairwise similarities between all data objects form a weighted graph adjacency matrix which contains all necessary information for the clustering process, which can consequently be formulated as a graph partitioning problem. In this context, we propose a new cluster quality measure which uses the maximum spanning tree and allows to compute the optimal clustering under the min-max principle in polynomial time. Our algorithm can be applied when a load-balanced clustering is required.

1 Introduction

The objective of clustering is to divide a given dataset into groups of similar objects in an unsupervised manner. Clustering techniques find frequent application in various areas, including computational biology, computer vision, data mining, gene expression analysis, text mining, social network analysis, VLSI design, and web indexing, to name just a few. Commonly, a metric is used to compute pair-wise similarities between all items and the clustering task is formulated as a graph partitioning problem, where a complete graph is generated from the similarity matrix. In fact, many graph-theoretical methods have been developed in the context of detecting and describing inherent cluster structures in arbitrary point sets using a distance function [2].

Here, we propose a novel clustering algorithm based on a quality measure that uses the maximum spanning tree of the underlying weighted graph and addresses a balanced grouping with the min-max principle. More specifically, we aim to detect clusters which are balanced with respect to their ratio of intra-cluster variance to their distance to other data instances. In other words, we allow clusters with weaker inner edges to be formed, if they are located at large distance of other clusters (Figure 1). We prove that an optimal clustering under this measure can be computed in polynomial time using dynamic programming.

Such cluster properties are typically desired when grouping sensors in Wireless Sensor Networks [1] and multi-robot task allocation in Cooperative Robotics where the goal is to allocate tasks to cooperative robots while minimizing costs [5]. Another application area arises from the field of Music Information Retrieval, where several applications rely on the

* This research has received funding from the projects COFLA2 (Junta de Andalucía, P12-TIC-1362) and GALGO (Spanish Ministry of Economy and Competitiveness and MTM2016-76272-R AEI/FEDER,UE) and from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922. The problems studied here were introduced and partially solved during a visit to Havana University, Cuba in October 2016.

† L.E. Caraballo is supported by the Spanish Government under the FPU grant agreement FPU14/04705.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

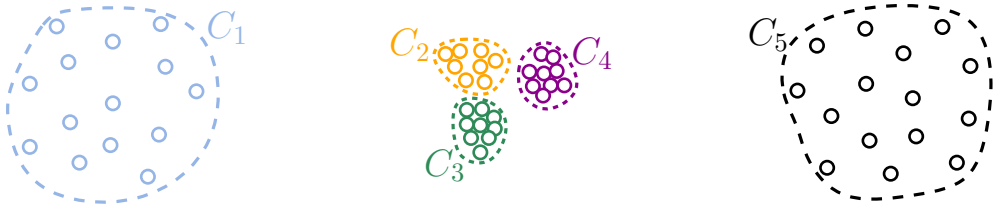

Figure 1 Illustration of the desired cluster properties: The ratios of inner variance to distance to other clusters is balanced among groups. Clusters C_1 and C_5 exhibit a higher variance but are also further apart from the other clusters.

unsupervised discovery of similar (but not identical) melodies or melodic fragments. In this context, clustering methods can be used to explore large music collections with respect to melodic similarity, or to detect repeated melodic patterns within a composition [4].

2 Problem statement

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of points or nodes in a metric space and suppose that there exists a function to estimate the similarity between two nodes. Let A be the matrix holding similarity values computed for every pair of elements in V. The value $A[i, j]$ is the similarity between the nodes v_i and v_j. If $A[i, j] > A[i, l]$ then the node v_i is more similar to v_j than to v_l. Our goal is to create groups such that similar nodes are located in the same cluster and dissimilar nodes are in separate clusters.

Let $G = (V, E, w)$ be a weighted and undirected graph induced by A. In this paper, such graphs are simply referred to as "graph". E is the set of edges and contains an edge for every unordered pair of nodes, and w is a weight function $w : E \longrightarrow (0, 1)$ such that $w(e)$ is the similarity between the nodes connected by e (i.e. if $e = \{v_i, v_j\}$, then $w(e) = A[i, j]$).

Let $C \subseteq V$ be a cluster. The *outgoing edge set* of C, denoted by $Out(C)$, is the set of edges connecting C with $V \setminus C$. Let $MST(C)$ be the maximum spanning tree of C. Let $\max(Out(C))$ and $\min(MST(C))$ be the weights of the heaviest and lightest edges of $Out(C)$ and $MST(C)$, respectively. We define the following function $\Phi(C)$ as the quality measure of a cluster C:

$$\Phi(C) = \begin{cases} 0 & \text{if} \quad C = V, \\ \max(Out(C)) & \text{if} \quad |C| = 1, \\ \dfrac{\max(Out(C))}{\min(MST(C))} & \text{otherwise} \end{cases}$$

Note that higher values of $\Phi(\cdot)$ correspond to worse clusters. Let $\Pi = \{C_1 \ldots, C_k\}$ be a k-clustering (clustering formed by k clusters) of G. To evaluate the quality of Π we use the quality of the worst cluster, $\Phi(\Pi) = \max_{i=1}^{k}\{\Phi(C_i)\}$. Denoting the set of all possible k-clusterings on G by $\mathcal{P}(k, G)$, we state the following optimization problems:

▶ Problem 2.1.

$$\min \quad \Phi(\Pi) \qquad \text{subject to: } \Pi \in \mathcal{P}(k, G).$$

When the value of k is unknown, the problem can be stated as follows:

▶ Problem 2.2.

$$\min \quad \Phi(\Pi) \qquad \text{subject to: } \Pi \in \bigcup_{k=2}^{n} \mathcal{P}(k, G).$$

3 Properties of the optimal clustering

Note that the problems stated above can be generalized to connected (not necessarily complete) graphs, by simply setting $\mathcal{P}(k, G)$ as the set of all the possible partitions of G in k connected components. In this extended abstract most of the proofs are omitted. A full version of the paper can be found in [3].

▶ **Lemma 3.1.** *Let G be a graph and let Π^* be an optimal clustering of G for Problem 2.1 in $\mathcal{P}(k, G)$. Then, $\Phi(\Pi^*) \leq 1$.*

▶ **Lemma 3.2.** *Let G be a graph and let Π^* be an optimal clustering of G for Problem 2.1. Let C be a cluster in Π^*. If $|C| > 1$, then every bipartition of C has a crossing edge in a maximum spanning tree of G.*

▶ **Theorem 3.3.** *Let G be a graph and let $\Pi^* \in \mathcal{P}(k, G)$ be an optimal clustering of G for Problem 2.1. For every cluster $C \in \Pi^*$, the maximum spanning tree of C is a subtree of a maximum spanning tree of G and the heaviest outgoing edge of C is in a maximum spanning tree of G.*

Proof. (Sketch) Let C be a cluster of Π^*. If $|C| = 1$ then, obviously, $MST(C) \subset MST(G)$. If $|C| > 1$ then $MST(C) \subseteq MST(G)$ by Lemma 3.2. The second part of the theorem, claiming that the heaviest edge in $Out(C)$ is in $MST(G)$, is deduced from the properties of the MST. ◀

▶ **Corollary 3.4.** *Let G be a graph and let $\Pi^* \in \bigcup_{k=2}^{n} \mathcal{P}(k, G)$ be an optimal clustering of G for Problem 2.2. For every cluster $C \in \Pi^*$, the maximum spanning tree of C is a subtree of a maximum spanning tree of G and the heaviest outgoing edge of C is in a $MST(G)$.*

We introduce the following notion: Let T be a spanning tree of a graph G. Let $\Pi \in \mathcal{P}(k, T)$ be a clustering of T. The evaluation function $\Phi_T(\Pi)$ operates as usual, but it is restricted to the set of edges forming T. Therefore, the optimal solution for Problem 2.1 on T is $\Pi^\dagger \in \mathcal{P}(k, T)$ such that $\Phi_T(\Pi^\dagger) \leq \Phi_T(\Pi)$ for every other clustering $\Pi \in \mathcal{P}(k, T)$.

▶ **Theorem 3.5.** *Let G be a graph and let T be a maximum spanning tree of G. If $\Pi^* \in \mathcal{P}(k, G)$ and $\Pi^\dagger \in \mathcal{P}(k, T)$ are the optimal clusterings for Problem 2.1 on G and T, respectively; then $\Phi(\Pi^*) = \Phi_T(\Pi^\dagger)$.*

▶ **Corollary 3.6.** *Let G be a graph and let T be a maximum spanning tree of G. If $\Pi^* \in \bigcup_{k=2}^{n} \mathcal{P}(k, G)$ and $\Pi^\dagger \in \bigcup_{k=2}^{n} \mathcal{P}(k, T)$ are the optimal clusterings for Problem 2.2 on G and T, respectively; then $\Phi(\Pi^*) = \Phi_T(\Pi^\dagger)$.*

4 The algorithm

First, recall that Theorem 3.5 and Corollary 3.6 provide a nice property, which allows us to reduce Problems 2.1 and 2.2 from a graph to its maximum spanning tree. Consequently, given a similarity graph, we can operate on its maximum spanning tree $T = (V, E, w)$. From now on, we will use E to denote the set of edges in the maximum spanning tree. Observe that every cluster C in T determines only one subtree of T. Then, using $MST(C)$ to denote the maximum spanning tree in C, may be confusing or redundant. Therefore, instead of using $MST(C)$ we will use $E(C)$ (set of edges connecting nodes in C).

The proposed algorithm is based on *dynamic programming*. We show that the stated problems have an *optimal substructure* which allows us to build the optimal solution in T

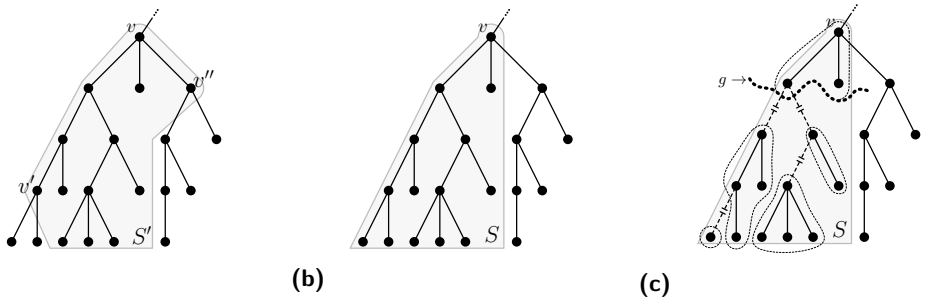

Figure 2 (a) A subtree S' which is not considered. (b) A considered subtree S. (c) Representation of a clustering Π of S. The *head cluster* is above the curve g. The edges of $Out_S(h(\Pi))$ are the edges in S stabbed by g. The clusters of Π that are below g constitute the headless clustering $\Pi \setminus \{h(\Pi)\}$.

from local solutions for subtrees of T. From here on, we consider that the tree T is rooted at $r \in V$. Let $p(v)$ be the parent of v and $c(v)$ be the set of children of v. Given a tree T, let S be a subtree of T and let v be the node with minimum depth in S. Then we say that S is rooted at v. In the rest of this paper, we only consider subtrees S rooted at v that contain all the descendants of vertices $v' \in S \setminus \{v\}$ (see Figures 2a and 2b). We say $S = T_v$ if S contains all the descendants of v.

The main idea of our algorithm is to operate on (local) clusterings of a subtree and perform a bottom-up dynamic programming strategy with two basic operations:

- `UpToParent`: knowing an optimal clustering of a subtree $S = T_v$ such that $T_v \neq T$, compute an optimal clustering of the subtree $\overline{S}$ formed by adding $p(v)$ to S.
- `AddChildTree`: knowing an optimal clustering of a subtree S rooted at $p(v)$, $v \notin S$; and knowing the optimal clustering of $Q = T_v$; compute an optimal clustering of the subtree resultant of joining S and Q.

Now, we elaborate on a (local) clustering Π of a subtree S rooted at v. We call the cluster containing the node v (root of the subtree) the *head cluster* of Π and we denote it $h(\Pi)$ (see Figure 2c). Let $Out_S(C)$ be the set of outgoing edges of C in S. In Figure 2c, $Out_S(h(\Pi))$ is formed by the edges stabbed by the curve g.

Given a clustering Π of a subtree S, let M denote the weight of the heaviest edge in $Out_S(h(\Pi))$, that is, $M = \max(Out_S(h(\Pi)))$. If $h(\Pi)$ contains all the nodes in S then we set $M = 0$. On the other hand, let μ denote the weight of the lightest edge in $E(h(\Pi))$, that is $\mu = \min(E(h(\Pi)))$. If $h(\Pi)$ is formed by single node we set $\mu = 1$. For convenience we introduce the functions $\Phi_S(\cdot)$ and $\Phi_S(\cdot)$ as restricted quality measures of a cluster and a clustering, respectively. They work as defined above, but are restricted to the edges of the subtree S, thus:

$$\Phi_S(h(\Pi)) = \frac{M}{\mu}. \tag{1}$$

Note that, if $S = T$, then $\Phi_S(h(\Pi)) = \Phi(h(\Pi))$. If $S = T_v \neq T$, then: $\Phi(h(\Pi)) = \frac{\max\{M, w(\{v,p(v)\})\}}{\mu}$. For every other cluster $C \in \Pi$, such that C is not the head cluster, the usual evaluation and the restricted one have the same value, $\Phi(C) = \Phi_S(C)$. Consequently, the restricted evaluation of the "*headless*" clustering $\Pi \setminus \{h(\Pi)\}$ is:

$$\Phi_S(\Pi \setminus \{h(\Pi)\}) = \Phi(\Pi \setminus \{h(\Pi)\}) = \max\{\, \Phi(C) \mid C \in \Pi \setminus \{h(\Pi)\} \,\}, \tag{2}$$

and the restricted evaluation of the clustering Π is:

$$\Phi_S(\Pi) = \max\{\Phi_S(h(\Pi)),\ \Phi(\Pi \setminus \{h(\Pi)\})\}. \tag{3}$$

Let S be a subtree of T, and let $\mathcal{H}(l, S, \mu)$ denote the set of l-clustering of S in which μ is the weight of the lightest edge in the head cluster. That is:

$$\mathcal{H}(l, S, \mu) = \{\ \Pi\ \mid\ \Pi \in \mathcal{P}(l, S) \quad \text{and} \quad \mu = \min(E(h(\Pi)))\ \}.$$

Now we are ready to state an encoding of a local solution and the invariant that allows us to apply dynamic programming:

▶ Notation 4.1. Suppose $\mathcal{H}(l, S, \mu)$ is not empty, then a clustering Π in $\mathcal{H}(l, S, \mu)$ is encoded by the ordered pair $O_S(l, \mu) = (M, b)$ if the following properties are fulfilled:

1. $b = \Phi_S(\Pi) = \min\{\ \Phi_S(\Pi')\ \mid\ \Pi' \in \mathcal{H}(l, S, \mu)\}$, and
2. $M = \max(Out_S(h(\Pi))) = \min\left\{\ \max(Out_S(h(\Pi')))\ \middle|\ \begin{array}{l}\Pi' \in \mathcal{H}(l, S, \mu)\ \text{ and}\\ \Phi_S(\Pi') = b\end{array}\ \right\}$.

If $\mathcal{H}(l, S, \mu)$ is empty, then $O_S(l, \mu) = (\infty, \infty)$, where ∞ indicates the "infinity" value.

We set $O_S(l, \mu)$ as (∞, ∞) if $1 < \min\{\ \Phi_S(\Pi)\ \mid\ \Pi \in \mathcal{H}(l, S, \mu)\ \}$. Then, given a subtree S, $O_S(\cdot, \cdot)$ is a function whose domain is $\mathbb{N}_{[1,k]} \times (w(E) \cup \{1\})$ and image $\{(\infty, \infty)\} \cup (w(E) \cup \{0\}) \times \mathbb{R}_{[0,1]}$ where $w(E) = \{\ w(e)\ \mid\ e \in E\ \}$. If $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$, then, by using equations (1), (2) and (3), we obtain that $O_S(l, \mu)$ encodes a clustering Π (not necessarily unique) where:

$$\Phi_S(\Pi) = b = \max\left\{\frac{M}{\mu}, \Phi(\Pi \setminus \{h(\Pi)\})\right\}. \tag{4}$$

For the sake of simplicity, we use the following notation for $O_S(l, \mu) = (M, b)$: $O_S(l, \mu)[1] = M$ and $O_S(l, \mu)[2] = b$. Note that if we have the function O_T then the evaluation of the optimal clusterings for Problems 2.1 and 2.2 are $\min\{\ O_T(k, \mu)[2]\ \mid\ \mu \in w(E) \cup \{1\}\ \}$, and $\min\{\ O_T(k, \mu)[2]\ \mid\ k \in \mathbb{N}_{[2,n]}$ and $\mu \in w(E) \cup \{1\}\ \}$, respectively.

▶ **Lemma 4.2.** *Let S be a subtree rooted at v. Let Π and Π' be two different clusterings of S such that $\min(E(h(\Pi))) = \min(E(h(\Pi'))) = \mu$. If $\Phi_S(\Pi) < \Phi_S(\Pi') \leq 1$ then $\max(Out_S(h(\Pi))) \leq \max(Out_S(h(\Pi')))$.*

▶ **Corollary 4.3.** *Let S be a subtree. For a given value $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$, every l-clustering $\Pi \in \mathcal{H}(l, S, \mu)$ fulfills that: $\Phi_S(\Pi) \geq b$, and $\max(Out_S(h(\Pi))) \geq M$.*

The following lemma is the key of the proposed dynamic programming:

▶ **Lemma 4.4.** *Let S be a subtree rooted at v. Let $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$ and let Π be an l-clustering of S encoded by $O_S(l, \mu)$. Let Q be a subtree of S rooted at $v' \in c(v)$. By removing the edge $e = \{v', v\}$ from S, an l'-clustering $\mathcal{A}$ of Q is induced. Let $\mu' = \min(E(h(\mathcal{A})))$. By replacing $\mathcal{A}$ with a clustering $\mathcal{B}$ encoded by $O_Q(l', \mu')$ and restoring the edge e, a new clustering Π' of S is obtained, which is also encoded as $O_S(l, \mu) = (M, b)$ (Figure 3 depicts the case when e connects nodes in different clusters).*

Using above properties, the recurrence formulas for dynamic programming can be established. Let us mention here one of them, which corresponds to one case in the proof of the Theorem 4.5 (`UpToParent` operation). Let $\omega = w(\{v, p(v)\})$. For $\mu = 1$ we can prove that:

$$O_{\overline{S}}(l, 1) = \left(\omega, \min_{\mu'}\left\{\max\left\{\omega, O_S(l-1, \mu')[2], \frac{\max\{\omega, O_S(l-1, \mu')[1]\}}{\mu'}\right\}\right\}\right).$$

▶ **Theorem 4.5** (UpToParent operation). *Let $S = T_v$ such that $T_v \neq T$, and let $\overline{S}$ be the subtree formed by adding $p(v)$ to S. If we know the function O_S, then the function $O_{\overline{S}}$ can be computed in $O(kn)$.*

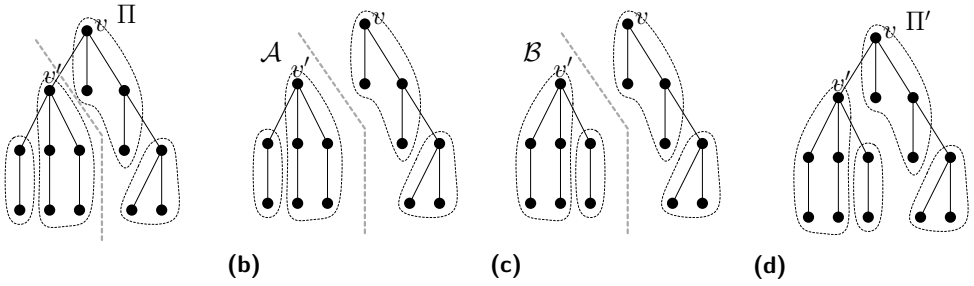

Figure 3 Removing the edge $e = \{v, v'\}$ when e connects nodes in different clusters. (a) Initial situation. (b) Induced clustering $\mathcal{A}$ when e is removed. (c) Replacing $\mathcal{A}$ with another clustering $\mathcal{B}$. (d) Restoring the edge e and obtaining a new clustering Π'.

Finally, for the second operation we have:

▶ **Theorem 4.6** (AddChildTree operation). *Let $Q = T_v$ such that $T_v \neq T$, and let S be a subtree rooted at $p(v)$ such that v is not in S. Let P denote the subtree formed by joining S and Q. If we know the functions O_S and O_Q, then the function O_P can be computed in $O(k^2n^2)$.*

4.1 Complexity of the algorithm

Given a tree T and a value k we can calculate O_T by computing O_{T_v} for every node v in T in a bottom-up (from the leaves to the root) procedure using the mentioned operations. Note that, if v is leaf, then $O_{T_v}(1,1) = (0,0)$ and $O_{T_v}(l,\mu) = (\infty,\infty)$ if $l > 1$ or $\mu < 1$. To compute the function O_{T_v} of an inner node v, we proceed as follows: Let $\{v_1, \ldots, v_m\}$ be the set of children of v. First, considering $S = T_{v_1}$, compute $O_{\overline{S}}$ from O_S using the `UpToParent` operation. Subsequently, we proceed with joining the subtrees T_{v_i} one by one using the `AddChildTree` operation. When all the children have been added, the resultant subtree corresponds to T_v. Note that we apply a single operation per edge. Consequently, this algorithm takes $O(k^2n^3)$ time. Note, that with this algorithm, we obtain the evaluation of the optimal clustering; the clusters of an optimal solution can be computed "*navigating backwards*" through the computed functions.

Problem 2.2 can be solved using the same idea with a slightly more complex approach. We can use a similar algorithm based on functions $O_S(\mu)$, saving the parameter l, (which corresponds to the number of clusters) and then the spent time is $O\left(n^3\right)$ time.

References

1 I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

2 T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the fourth annual symposium on Computational geometry*, 1988.

3 L. E. Caraballo, J. M. Díaz-Báñez, and N. Kroher. *A Polynomial Algorithm for Balanced Clustering via Graph Partitioning*. arXiv:1801.03347, 2018.

4 N. Kroher, J.-M. Díaz-Báñez, and A. Pikrakis. Discovery of repeated melodic phrases in folk singing recordings. *IEEE Transactions on Multimedia*, 2017.

5 A. Ollero and I. Maza. *Multiple heterogeneous unmanned aerial vehicles*. Springer Publishing Company, Incorporated, 2007.