Maximum Split Clustering Under Connectivity Constraints

Pierre HANSEN
GERAD and École des Hautes Études Commerciales
Département des Méthodes Quantitatives en Gestion

Brigitte JAUMARD*
CRT, GERAD and Université de Montréal
Département d'Informatique et de Recherche Opérationnelle

Christophe MEYER
Université de Montréal
Département d'Informatique et de Recherche Opérationnelle

Bruno SIMEONE
University of Rome "La Sapienza"
Department of Statistics, Probability and Applied Statistics

Valeria DORING
École Polytechnique de Montréal

July, 2003

*Corresponding author

Abstract

Consider N entities to be classified (e.g., geographical areas), a matrix of dissimilarity between pairs of entities, a graph H with vertices associated with these entities such that the edges join the vertices corresponding to contiguous entities. The split of a cluster is the smallest dissimilarity between an entity of this cluster and an entity outside of it. The single-linkage algorithm (ignoring contiguity between entities) provides partitions into M clusters for which the smallest split of the clusters, called split of the partition, is maximum. We study here the partitioning of the set of entities into M connected clusters for all M between $N-1$ and 2 (i.e., clusters such that the subgraphs of H induced by their corresponding sets of entities are connected) with maximum split subject to that condition. We first provide an exact algorithm with a $\Theta(N^2)$ complexity for the particular case in which H is a tree. This algorithm suggests in turn a first heuristic algorithm for the general problem. Several variants of this heuristic are also explored. We then present an exact algorithm for the general case based on iterative determination of cocycles of subtrees and on the solution of auxiliary set covering problems. As solution of the latter problems is time-consuming for large instances, we provide another heuristic in which the auxiliary set covering problems are solved approximately. Computational results obtained with the exact and heuristic algorithms are presented on test problems from the literature.

Keywords: Constrained clustering; Contiguity; Connectivity; Partition; Split.

Acknowledgements. The first two authors have been supported by FCAR (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) grants 2001-ER-70686 and grant N00014-92-J-1194 from the Office of Naval Research. Work of the first author has also been supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant GP0105574. It has also been supported in part by AFOSR grant F49620-93-1-0041 to Rutgers University. Work of the second author has been supported by NSERC Grant GP0036426, by FCAR 90NC0305, and by a NSF Professorship for Women in Science at Princeton University. Work of the third author was supported in part by the Spezialforschungsbereich F003 "Optimierung und Kontrolle", Projektbereich Diskrete Optimierung. Work of the fourth author was done in part during two visits to GERAD, Montréal.

1 Introduction

Many clustering algorithms (Hartigan 1975; Späth 1980; Gordon 1981; Kaufman and Rousseeuw 1989) partition the entities of a given set into homogeneous and/or well separated clusters. Homogeneity means that entities within the same cluster should be similar to each other and separation that entities in different clusters should be dissimilar from each other. Often, separation and homogeneity are conflicting goals. In this paper, we focus on separation. In some applications it is desirable for operational reasons to impose additional constraints on the clusters. Special purpose algorithms or modified versions of standard methods are then required. Constrained clustering has been reviewed by Gordon (1973, 1979, 1996), Lefkovitch (1980) and Murtagh (1985). The constraints most often considered are bounds on the cardinality or the weight of the clusters, and connectivity constraints, when areas in geographical space (usually the plane) are classified.

A simple criterion for separation of the clusters of a partition is the split, i.e., the smallest dissimilarity between entities in different clusters. Recall that the single-linkage algorithm maximizes the split of all partitions it leads to (Delattre and Hansen 1980). Strength and weaknesses of the single linkage algorithm, and hence implicitly of the split criterion, have been extensively discussed in the literature, see e.g., Jardine and Sibson (1971) for axiomatic arguments. Let us only recall here that, as noted by Hubert (1973), the partitions obtained by the single linkage algorithm are invariant to any monotone transformation of the dissimilarities. It is well known that the single linkage algorithm suffers from the chaining effect: very dissimilar entities at the endpoints of a chain of consecutive pairwise similar entities may be assigned to the same cluster. Adding contiguity constraints may alleviate this defect. As the split is a separation criterion, one should of course adopt a different criterion, e.g., diameter or variance for applications in which homogeneity is more important than separation. Maximum split partitions subject to constraints on the weight (or cardinality) of the clusters are studied in Hansen, Jaumard and Musitu (1990). It is shown there that finding such partitions is strongly NP-complete but can often be done in reasonable computing time even for large data sets (a few seconds for 500 entities on a SUN 3/50). Other algorithms for cardinality constrained clustering have been proposed by Lebart (1978) and Diday *et al.* (1979).

From now on, we adopt again the split criterion and consider connectivity constraints on the clusters. These constraints are usually expressed (e.g., Murtagh

1985) by introducing a contiguity graph H with vertices associated with the entities (usually areas in geographical space) and edges joining pairs of contiguous entities (usually areas with a common border). Connectivity constrained partitions are those such that the subgraphs of H induced by the vertices associated with the entities of each cluster are connected. The monotony invariance property cited above carries over to this type of constrained clustering. Many authors (e.g., Scott 1971; Lebart 1978; Ferligoj and Batagelj 1982, 1983; Perruchet 1983) have proposed heuristics for connectivity constrained clustering based on a modification of the general scheme for agglomerative hierarchical clustering: at each iteration, two clusters are merged only if they correspond to two contiguous regions. This rule implies that many feasible clusters are excluded from the set of partitions considered. Moreover, the computational complexity (see Garey and Johnson (1979) for definitions and notation, including the θ and O symbols used throughout this paper) is slightly increased over that of the best hierarchical agglomerative clustering algorithms. It goes from $\Theta(N^2)$ for the single-linkage algorithm to $O(N^2 \log N)$ if an efficient implementation with heaps, similar to that of Day and Edelsbrunner (1984) is used. Resorting to union-find data structures (see, e.g., Aho, Hopcroft and Ullman (1974) for basic definitions) would also lead to an implementation in $O(N^2 \log N)$. As shown below, other heuristics can be designed which have a slightly lower complexity and usually provide better solutions as estimated by the value of the split. It is also possible to maximize exactly the split subject to connectivity constraints. The two main algorithms of this paper do so, when H is a tree and a general graph respectively.

An efficient modification of the single-linkage algorithm has been proposed by Monestiez (1977) for a problem closely related to connectivity constrained maximum split clustering. Only dissimilarities associated with pairs of entities corresponding to the two endpoints of an edge of H are considered. Then a minimum spanning tree of H is computed (with edges weighted by the dissimilarities), its edges ranked in order of non-decreasing dissimilarity values, and added to an initially empty graph one at a time, as in the Gower-Ross (1969) algorithm. Other versions, also proposed by Monestiez (1977), use the variance criterion. Monestiez's algorithm has several advantages. First, it maximizes a specific criterion for all partitions it leads to. Let us indeed define the *border split* of a cluster as the smallest dissimilarity between an entity in that cluster and one outside it *in a contiguous area*. Let us further define the border split of a partition as the smallest border split of its clusters. Then Monestiez's (1977) algorithm maximizes the border split of the partitions into $N, N-1, \ldots, 2$

clusters it leads to. The proof follows from Theorem 1 and Corollary 2 of Delattre and Hansen (1980), after replacing all dissimilarities between pairs of entities not corresponding to edges of H by arbitrarily large values. Second, Monestiez's algorithm is often extremely quick. In the general case its time complexity is in $\Theta(N^2)$. If H is planar, which is the case when areas in the plane or pixels in a grid are to be clustered, this complexity becomes $O(N \log N)$, due to the ranking phase assuming Cheriton and Tarjan's (1976) $O(N)$ algorithm is used to determine the minimum spanning tree. Moreover, if dissimilarities are ordinal, bucket sort may be used and the complexity reduces to $\Theta(N)$. Maximizing the border split of a partition is of interest in many applications, such as finding contours in image processing or defining classes in cartography. However, it is different from maximizing the split of a partition. Indeed, two clusters may have very dissimilar entities across their borders and very similar entities inside both of them. If it is desired to have *all* entities in different clusters as dissimilar as possible in a threshold sense, one must use the split criterion and not the border split one. This is what is done in this paper.

Note that dynamic programming algorithms for clustering problems in which H is a path have been proposed by several authors (e.g., Fisher 1958; Rao 1971; Lucertini, Perl and Simeone 1993; see also Bellman and Dreyfus 1962 as a reference on dynamic programming). These algorithms yield connected clusters. They can be easily adapted to the split criterion. Note also that univariate clustering can be reduced to partitioning on a path.

The paper is organized as follows. Maximum split partitioning subject to connectivity constraints is formulated mathematically in the next section, where it is also shown to be strongly NP-complete. Properties of an exact solution are studied in Section 3. The particular case where the contiguity graph H is a tree is considered in Section 4. This corresponds to problems in biological classification in which affine taxa in a phylogenic tree must be grouped in such a way that the clusters are subtrees (Jardine and Sibson 1971). Maravalle and Simeone (1985) propose an $O(N^3)$ algorithm for this case. Using a result of Rosenstiehl (1967) and of Delattre and Hansen (1980), we obtain a $\Theta(N^2)$ algorithm. This suggests a heuristic for the general case, based on the selection of a spanning tree of H, which is discussed in Section 5. Several variants of this heuristic are also explored. Exact solution of maximum split clustering under connectivity constraints is studied in Section 6. A solution is proposed which iteratively determines a spanning subforest F of H (possibly with isolated vertices) whose trees correspond to clusters. This is done by

finding cocycles of subtrees (disconnected with respect to H) of a minimum spanning tree of G with edges again weighted by the dissimilarities, and solving set covering problems expressing that F must contain at least one edge of H belonging to each such cocycle. The exact solution of all set covering problems may be very time consuming for large problems. This suggests another heuristic, discussed in Section 7, in which these covering problems are solved approximately. Experimental results with the exact and heuristic algorithms are presented in Section 8. Conclusions are drawn in Section 9.

2 Problem statement and complexity

Let $O = \{O_1, O_2, \ldots, O_N\}$ denote a set of $N = |O|$ entities and $D = (d_{k\ell})$ a $N \times N$ matrix of dissimilarities between pairs of these entities. A dissimilarity $d_{k\ell}$ is a real number which satisfies the conditions $d_{k\ell} \geq 0$, $d_{kk} = 0$ and $d_{k\ell} = d_{\ell k}$ for $k, \ell = 1, 2, \ldots, N$. The more the entities O_k and O_ℓ differ one from the other, the larger is the corresponding dissimilarity $d_{k\ell}$. A partition $P_M = \{C_1, C_2, \ldots, C_M\}$ of the entities of O into M clusters C_j ($j = 1, 2, \ldots, M$) is such that no cluster is empty, any two clusters have an empty intersection and the union of all clusters is equal to O. Let Π_M denote the set of partitions P_M of O into M clusters. Recall that the split $s(C_j)$ of a cluster C_j is the smallest dissimilarity between an entity in C_j and an entity outside C_j:

$$s(C_j) = \min_{k,\ell: O_k \in C_j,\, O_\ell \notin C_j} d_{k\ell}$$

and the split $s(P_M)$ of the partition P_M is the smallest split of its clusters:

$$s(P_M) = \min_{j=1,2,\ldots,M} s(C_j).$$

We recall here, for easier reference, some elementary concepts of graph theory (see, e.g., Berge (1973) for a complete reference). In general, we denote by $V(G)$ and $E(G)$ the set of vertices and the set of edges of a graph G, respectively. A weighted complete graph $G = (V, E(G))$ is associated with O and D in the usual way, i.e., vertices $v_j \in V$ correspond to entities $O_j \in O$, and edges $\{v_k, v_\ell\} \in E$ are weighted by the dissimilarities $d_{k\ell}$. Another graph $H = (V, E(H))$ with the same vertex set as G is used to express the connectivity constraints. We call H the *contiguity graph*. An edge $\{v_k, v_\ell\}$ belongs to $E(H)$ if and only if the entities O_k and O_ℓ are contiguous. A

path in H is a sequence of vertices $(v_{k_1}, v_{k_2}, \ldots, v_{k_p})$ such that $\{v_{k_\ell}, v_{k_{\ell+1}}\} \in E(H)$ for $\ell = 1, 2, \ldots, p-1$; v_{k_1} is the *initial* vertex and v_{k_p} the *terminal* vertex of the path, which is said to join v_{k_1} to v_{k_p}. If $v_{k_1} = v_{k_p}$, the path is a *cycle*. All paths and cycles considered below are *elementary*, i.e., all their vertices are distinct. A *connected component* of a graph H is a maximal subset C of vertices such that for any two vertices v_k and v_ℓ in C, there is a path in H joining v_k and v_ℓ. When $V = C$ the graph H is said to be *connected*. A *forest* is a graph without cycles. A *tree* is a connected forest. A *spanning forest* of H is a forest F such that $V(F) = V(H)$ and $E(F) \subseteq E(H)$. A spanning forest F is *maximal* if there is no spanning forest whose set of edges properly includes $E(F)$. A maximal spanning forest of a connected graph is a (spanning) tree. We assume below that the contiguity graph H is connected. If $S \subseteq V$, we denote by $w(S)$ the *cocycle* of S, i.e., the set of edges of H which have one endpoint in S and the other in $V \setminus S$; moreover we denote by $H(S)$ the *subgraph of H induced by S*, i.e., the graph whose set of vertices is S and whose set of edges is $\Big\{\{v_k, v_\ell\} \in E(H) : v_k, v_\ell \in S\Big\}$. A *connected M-partition* of O is a partition $P_M = \{C_1, C_2, \ldots, C_M\}$ of O into M clusters, such that $H(C_j)$ is connected for every $j = 1, 2, \ldots, M$. We denote by $\Pi_M^C(H)$ – or simply Π_M^C when the graph H is understood – the set of all connected M-partitions of O.

The *maximum split clustering problem under connectivity constraints* may be formulated:

$$\text{Determine } P_M \in \Pi_M^C$$

such that $s(P_M)$ is maximum for $M = 2, 3, \ldots, N-1$.

A recognition form of this problem is:

CONNECTED MAX SPLIT GRAPH CLUSTERING

Instance: Graph $H = (V, E)$; $N \times N$ symmetric matrix $(d_{k\ell})$ (with $N = |V|$) of nonnegative rationals with zero diagonal elements; rational nonnegative number s; integer M, $1 < M < N$.

Question: Is there a connected M-partition P_M of H such that $s(P_M) \geq s$?

The following theorem suggests that the above problem may be computationally hard to solve. For definitions of NP-completeness and related concepts, the reader is again referred to Garey and Johnson (1979) or for a brief introduction, to Hansen, Jaumard and Musitu (1990).

Theorem 1 CONNECTED MAX SPLIT GRAPH CLUSTERING *is strongly NP-complete.*

Proof: Clearly the problem belongs to the class NP. In order to prove its NP-completeness, we exhibit a reduction from the STEINER PROBLEM on graphs, which is known to be NP-complete (Garey and Johnson, 1979). The latter problem is defined as follows.

STEINER PROBLEM

Instance: Graph $H' = (V', E')$; subset $R \subseteq V'$; integer $b \leq N - 1$, where $N = |V'|$.

Question: Is there a tree T, which is a subgraph of H' that includes all vertices of R and has at most b edges ?

A reduction of STEINER PROBLEM to CONNECTED MAX SPLIT GRAPH CLUSTERING is readily obtained.

Take $H = H'$; set $d_{k\ell}$ equal to 0 if $v_k, v_\ell \in R$ or $v_k = v_\ell$, and $d_{k\ell}$ equal to 1 otherwise; set M to $N - b$ and s to 1. We now show that STEINER PROBLEM has a YES-answer if and only if CONNECTED MAX SPLIT GRAPH CLUSTERING has a YES-answer. Suppose that STEINER PROBLEM has a YES-answer. Define a partition P as follows. One cluster of P is the vertex-set $V(T)$ of T; the remaining clusters are the singletons $\{v\}$ with $v \in V \setminus V(T)$. P is connected, and since $|V(T)| \leq b + 1$, P consists of at least $N - b = M$ clusters. Finally, $s(P) = 1$. If P has more than M clusters, a partition with exactly M clusters and split ≥ 1 can be obtained by merging some of the clusters (see Proposition 4 in the next Section).
Conversely, let $P_M = \{C_1, C_2, ..., C_M\}$ be a connected partition with split at least 1 (and hence exactly 1) and $M = N - b$ clusters. Notice that R must be contained in some cluster C_j of P_M, else $s(P_M)$ would be 0. Let T^i be a spanning tree of the connected subgraph $H(C_i)$, $i = 1, \ldots, M$ and let F be the spanning forest of H whose connected components are $T^1, T^2, ..., T^M$. The total number of edges of F is $N - M = b$: hence, T^j has at most b edges. Since T^j includes all vertices of R, the answer to STEINER PROBLEM is YES.

Finally, to show that CONNECTED MAX SPLIT GRAPH CLUSTERING is strongly NP-complete (and so cannot be solved by a pseudo-polynomial algorithm unless $P = NP$) we observe that it may be expressed in such a way that it is not a number problem, i.e., that the length of the encoding of the largest coefficient is bounded. Indeed, it suffices to replace the dissimilarity values by their rank number. Since this

transformation is monotone and the split is a threshold-type criterion (i.e., depends on a single dissimilarity value), the optimal partitions are unchanged. ∎

Remark 1. Since STEINER PROBLEM remains NP-complete if H' is planar or bipartite, CONNECTED MAX SPLIT GRAPH CLUSTERING remains NP-complete for such classes of graphs. However, as shown in Section 4, the connected maximum split clustering problem can be solved in polynomial time when H is a tree.

3 A combinatorial formulation and some basic properties

In the present section we derive a combinatorial formulation of the maximum split clustering problem under connectivity constraints; furthermore, we collect some theoretical properties which provide a basis for the algorithms to be discussed in Sections 4 to 7. The reader who is primarily interested in clustering applications may skip all proofs in this section.

The *length* or *weight* of an edge $\{v_k, v_\ell\}$ of G is defined to be the dissimilarity $d_{k\ell}$ between entities O_k and O_ℓ. Let $T = (V, E(T))$ be a spanning tree of G with minimal total length (i.e., a minimum spanning tree).

Proposition 1 *The split of an arbitrary partition (connected or not) of V is always equal to the length of some edge of T.*

Proof: See Rosenstiehl (1967), Delattre and Hansen (1980). ∎

Notice that, while a minimum spanning tree T is not necessarily unique (unless all dissimilarity values are different), the list S of *lengths* of the edges of T is unique. We define $S \equiv [s_1, s_2, \ldots, s_{N-1}]$, where s_j is the length of the j^{th} shortest edge of T.

Remark 2. It is possible that, for some $s \in S$, there is no connected partition of H whose split is equal to s, as shown by the following example.

Example 1. Let D be the dissimilarity matrix of Table 1 and let H be the graph of Figure 1.

A minimum spanning tree T of G is shown in Figure 2. Here $S = [1, 2, 2, 3, 4]$, and one can check that there is no connected partition of H with split 3. Indeed,

	1	2	3	4	5	6
1	0	3	4	6	1	8
2	3	0	5	2	4	3
3	4	5	0	9	4	7
4	6	2	9	0	2	4
5	1	4	4	2	0	6
6	8	3	7	4	6	0

Table 1: Dissimilarities (associated with edges of G)

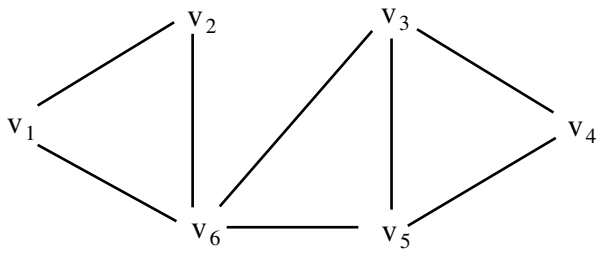

Figure 1: Contiguity graph H of Example 1

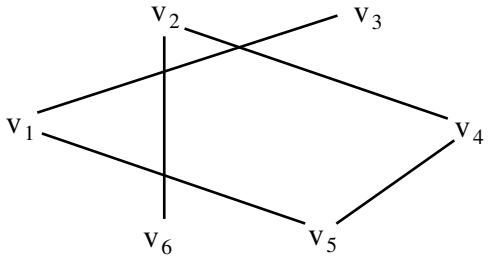

Figure 2: Minimum spanning tree T of G for Example 1

entities O_1 and O_5 must belong to the same cluster since $d_{15} = 1$. This entails that entity O_6 must belong to the same cluster than O_1 and O_5 to satisfy the connectivity constraints as vertex v_6 belongs to all paths linking v_1 to v_5 (note that it is an articulation point of graph H). Finally entities O_2, O_4 and O_5 must be in the same cluster since $d_{24} = 2 = d_{45}$. The resulting partition $\{\{v_1, v_2, v_4, v_5, v_6\}, \{v_3\}\}$ has a split equal to 4.

For a given $s \in S$, let G_s be the graph $(V, E(G_s))$ where $E(G_s) \equiv \Big\{\{v_k, v_\ell\} \in E(G) : d_{k\ell} < s\Big\}$. Starting from any given minimum spanning tree T of G, define T_s

in a similar way, i.e., $E(T_s) \equiv \Big\{\{v_k, v_\ell\} \in E(T) : d_{k\ell} < s\Big\}$. Observe that T_s is a spanning forest of G_s.

Proposition 2 *A partition P_M has split at least s if and only if for every edge $\{v_k, v_\ell\}$ of T_s, both v_k and v_ℓ belong to the same cluster of P_M.*

Proof: Clearly P_M has split at least s if and only if each connected component of G_s is contained in some cluster of P_M. Thus, in order to establish the proposition, it will be enough to prove that G_s and T_s have the same connected components.

Since $E(T_s) \subseteq E(G_s)$, every connected component of T_s is contained in some connected component of G_s. Conversely, let γ be a path in G_s joining v_k and v_ℓ. We shall exploit the fact that one can always build the minimum spanning tree T_s via Kruskal's (1956) greedy algorithm. This algorithm, starting from $E = \emptyset$, selects, at each iteration, the shortest edge e which has not been previously examined and then inserts e into E, unless e forms a cycle together with the other edges of E. At the end, one has $E = E(T)$. Right after all edges $\{v_{k'}, v_{\ell'}\}$ such that $d_{k'\ell'} < s$ have been examined, one has $E = E(T_s)$. If e is any edge of γ, then either $e \in E(T_s)$ or e joins two vertices in a same connected component of T_s. It follows that v_k and v_ℓ belong to the same connected component of T_s. ∎

The next definition leads to a purely combinatorial reformulation of CONNECTED MAX SPLIT GRAPH CLUSTERING.

Definition 1 *If F and F' are two forests having the same set of vertices, F* wraps *F' if each connected component of F' is (vertex wise) contained in some connected component of F.*

Next, we introduce the following combinatorial problem:

FOREST WRAPPING

Given a (connected) graph H, a forest B having the same vertex set as H and an integer M, $1 < M < N$, is there a spanning forest F of H having at least M connected components and such that F wraps B ?

Proposition 3 CONNECTED MAX SPLIT GRAPH CLUSTERING *is reducible to* FOREST WRAPPING.

Proof: Given a YES-instance of CONNECTED MAX SPLIT GRAPH CLUSTERING, define B to be T_s. Let $P_M = \{C_1, \ldots, C_M\} \in \Pi_M^C(H)$. Let T^j be a spanning tree of the connected graph $H(C_j)$, $j = 1, \ldots, M$, and let F be the spanning forest whose set of edges is $E(T^1) \cup E(T^2) \cup \ldots \cup E(T^M)$. By Proposition 2, any two vertices in the same connected component of T_s belong to the same cluster C_j of P_M, and hence to the same tree T^j. Hence, F wraps B. Conversely, assume that F is a spanning forest of H which wraps B and has at least M connected components $V_1, \ldots, V_{M'}$. Then $P_{M'} = \{V_1, \ldots V_{M'}\}$ is a connected partition and its split is at least s. ∎

Notice that, when B has only one non-singleton connected component, FOREST WRAPPING essentially becomes the Steiner problem on a graph.

For $s \in S$, define $m(s)$ to be the maximum number of clusters in a connected partition whose split is at least s, i.e., the maximum number of connected components in a spanning forest F of H which wraps T_s. Note that F has $m(s)$ connected components if and only if F has the smallest number of edges among the spanning forests wrapping T_s.

Proposition 4 *For every M, $1 \leq M \leq m(s)$, there is a $P_M \in \Pi_M^C(H)$ whose split is at least s.*

Proof: Let $P_{m(s)} \in \Pi_{m(s)}^C(H)$. Since H is connected, there must be, in $P_{m(s)}$, two clusters C_i and C_j which are adjacent (i.e., there is an edge of H having one endpoint in C_i and the other one in C_j). Merging C_i and C_j into a single cluster, one obtains a connected partition with $m(s)-1$ components and with split at least s. Iterating this procedure, one obtains the desired result. Notice that the connectivity of H plays an crucial role here. ∎

We shall denote by $F(s)$ a spanning forest of H which wraps T_s and has the smallest number of edges (briefly, an *optimal forest*).
For a given $s \in S$, let $K_1, \ldots, K_q$ be the connected components of T_s: $K_1, \ldots, K_q$ define subsets whose vertices must be in a same connected component of $F(s)$. The next result allows us to merge some of these subsets:

Proposition 5 *Let K_i and K_j be two connected components of T_s. If there exist 2 vertices $u, v \in K_i$ that are disconnected in the subgraph of H induced by the set of vertices $V \setminus V(K_j)$, then the vertices of K_i and K_j must be in a same connected component of $F(s)$.*

Proof: Since u and v belong to the same connected component of T_s, they must be in the same cluster in P_M. Let C be the connected component of $F(s)$ that contains u and v. By definition, there exists a path γ between u and v, with all vertices of γ in C. By assumption, this path contains a vertex of K_j. Since all vertices of K_j must be in the same component, it follows that C contains K_j. ∎

Each time we merge two subsets K_i and K_j, we add an edge in T_s between one arbitrary vertex of K_i and one arbitrary vertex of K_j, so that the resulting merged subset is still a connected component of a forest. We denote this forest by T'_s. Observe that Proposition 5 remains valid for T'_s.

The next result allows one to pinpoint a set Q of edges which must certainly be contained in some $F(s)$, thereby reducing the computational effort required to find $F(s)$.

Let $K_1, \ldots, K_q$ be the connected components of T'_s. Let F^i be a maximal spanning forest of $H(K_i)$. Let $Q = \bigcup_{i=1}^{q} E(F^i)$.

Proposition 6 *There always exists an optimal forest $F(s)$ whose set of edges contains Q.*

Proof: Let $F(s)$ be an arbitrary optimal forest. Assume that $\Big|Q \cap E\Big(F(s)\Big)\Big| = t$, where $0 \leq t < |Q|$. Consider an arbitrary edge $e = \{v_k, v_\ell\} \in Q \setminus E\Big(F(s)\Big)$. Since v_k, v_ℓ belong to the same connected component of T'_s and since $F(s)$ wraps T'_s, there must be a path γ in $F(s)$ joining v_k to v_ℓ. At least one of the edges e' along γ must belong to $E\Big(F(s)\Big) \setminus Q$, otherwise the edges of γ together with e would form a cycle contained in Q, which is impossible. By replacing e' by e one obtains a spanning forest $F'(s)$ which has the same number of edges as $F(s)$ and still wraps T'_s. Furthermore, one has $\Big|Q \cap E\Big(F'(s)\Big)\Big| = t + 1$. Iterating this procedure one obtains the desired result. ∎

We next mention a corollary of a result of Simeone (1978) which specifies a family of subgraphs of H containing or giving an optimal solution.

Proposition 7 *There exists a spanning tree $\hat{H}$ of H such that the optimal solutions of* CONNECTED MAX SPLIT GRAPH CLUSTERING *for H and $\hat{H}$ coincide.*

Proof: See Maravalle and Simeone (1995). ∎

In fact the result of Maravalle and Simeone (1995) is much more general than Proposition 7, as it applies to connectivity constrained clustering with *any* criterion which does not depend of the graph H. However, the proof of this result is not constructive.

Finally, let us discuss some consequences of the results of this section.

Since the objective function of the maximum split clustering problem under connectivity constraints is of the bottleneck type (max-min), it is natural to think of a threshold algorithm for its solution: namely, given a certain threshold parameter s, such an algorithm would try to find a connected partition whose split is at least s and having a prescribed (or a maximum) number of clusters. Proposition 1 says that it is always sufficient to consider at most $N-1$ special values of the threshold s, which are easily obtainable via a minimum spanning tree computation. Proposition 2 provides an efficient way to check that the split of a given partition is at least s. Proposition 3 provides a purely combinatorial formulation, i.e., FOREST WRAPPING, of CONNECTED MAX SPLIT GRAPH CLUSTERING. In Section 6 we shall investigate this formulation to develop an exact algorithm for the latter problem. Proposition 4 implies that, once a connected partition with split at least s and having the maximum number of clusters has been found, it is quite easy to find a connected partition whose split is at least s and having any prescribed smaller number of clusters. Finally, Propositions 5 and 6 enable us to simplify the problem by showing that certain edges must be present in some or in all optimal forests. The above considerations lead to the algorithms described in the next sections.

4 Maximum split clustering on trees

As mentioned in the introduction, the particular case of CONNECTED MAX SPLIT GRAPH CLUSTERING where the connectivity graph H is a tree has been studied by Maravalle and Simeone (1985), Maravalle, Simeone and Naldini (1997). They provide an $O(N^3)$ algorithm to solve it. We next present an algorithm whose worst case complexity is $\Theta(N^2)$.

When H is a tree, by deleting (or "cutting") $M-1$ edges of H but not their endpoints, one obtains a forest with M components, and these components are the

clusters of a connected M-partition. Conversely, all connected M-partitions arise in this way.

The algorithm CTREE described below scans the edges of a minimum spanning tree T of G in non-decreasing order of lengths, $s_1 \leq s_2 \leq \cdots \leq s_{N-1}$ (ties are broken arbitrarily). Let $\{v_k, v_\ell\}$ be the current edge and let $s_i = d_{k\ell}$. If we want to obtain a connected partition with split $> s_i$, then v_k and v_ℓ must belong to the same cluster C_j. This implies that no edge along the unique path γ of H joining v_k and v_ℓ must be cut. Hence, all vertices along γ must belong to C_j. The algorithm shrinks all these vertices into a single vertex in order to prevent edges along γ from being cut. We denote by $C(v_k)$ the cluster of the current partition which contains O_k.

Algorithm CTREE

Step 1. Initialization

Compute a minimum spanning tree T of $G = (V, E(G))$
Rank the edges $\{v_k, v_\ell\}$ of T in non-decreasing order of $d_{k\ell}$;
$C_k = C(v_k) \leftarrow \{O_k\}$ for $k = 1, 2, ..., N$;
$P_N = \{C_1, C_2, ..., C_N\}$ (P_N is the granular partition in which all clusters are singletons); $\hat{H} \leftarrow H$; $M \leftarrow N$;

Step 2. Path shrinking

For $i = 1, 2, \ldots, N-1$ **do**
    Let $\{v_k, v_\ell\}$ be the i^{th} edge of T;
    If $C(v_k) \neq C(v_\ell)$ **then**
        Let γ be the (unique) path joining $C(v_k)$ and $C(v_\ell)$ in $\hat{H}$;
        Let M' be the number of vertices of γ (v_k and v_ℓ excluded);
        $M \leftarrow M - M' - 1$;
        For each vertex v of γ, merge $C(v)$ with $C(v_k)$;
        Shrink all vertices of γ into a single vertex;
        Update $\hat{H}$ accordingly;
    EndIf
    Output the (connected) partition P_M of H
EndFor.

Theorem 2 *Let $s \in S$. Right after all edges of T with length smaller than s have been processed, the current partition P is a connected partition of H with split at least s and with a maximum number of clusters.*

Proof: The result is a consequence of the following two remarks:

(i) For every pair of vertices v_k, v_ℓ such that $\{v_k, v_\ell\} \in E(T)$ and $d_{k\ell} < s$, no edge along the path γ of T joining v_k and v_ℓ is cut in P. This follows from the fact that, when edge $\{v_k, v_\ell\}$ is processed, all vertices of γ are shrunk into a single vertex. Notice that some of them might have been shrunk during previous iterations.

(ii) All edges of H are cut in P, except for those specified in (i).

In view of Proposition 2, **(i)** implies that $s(P) \geq s$. Moreover, the partition P is connected by construction. Finally, **(ii)** implies that P has a maximum number of clusters among all connected partitions with split at least s, since all such partitions must satisfy **(i)** anyhow. ∎

Algorithm CTREE can be implemented so as to run in $O(N^2)$ time. Actually, computing a minimum spanning tree of the (complete) graph G already takes $O(N^2)$ time. Reading the dissimilarity matrix requires $\Omega(N^2)$ time: thus the algorithm has a complexity in $\Theta(N^2)$, i.e., it is the best possible.

Note that the partition output by the algorithm has a split at least s, which means that the split may be actually greater. It follows that the algorithm may produce several partitions with the same split but with a different number of clusters. In that case, the partition that is generally considered the most interesting is the one with the largest number of clusters.

5 Some heuristics for general graphs

Algorithm CTREE iteratively merges the vertices of the unique path γ of the tree $\hat{H}$ between the vertices of $\hat{H}$ corresponding to the endpoints of the edge (v_k, v_ℓ) of T under consideration. Such a path is of course a shortest one in $\hat{H}$. This observation suggests a simple modification to be given to CTREE, in order to obtain a heuristic, called HTREE, for the general case. It suffices to specify that at each iteration one seeks a shortest path γ between the vertices corresponding to v_k and v_ℓ in $\hat{H}$. Otherwise, the rules are the same. Such an algorithm is called a greedy one because it optimizes some criterion (here minimization of the reduction in number of clusters) at each

iteration in a myopic way. So after a few iterations one may obtain partitions whose split is at least s for which the number of clusters is not the largest possible. The complexity of HTREE is in $O(N^3)$.

Example 2. Consider the graph H of Figure 3 and any dissimilarity matrix where $d_{26} = 3$, $d_{68} = d_{79} = 5$, $d_{19} = 8$, and all remaining dissimilarities are greater than or equal to 8. Then $S = [3, 5, 5, 8, \ldots]$. Choose $s = 8$. The edges of T_8 are $\{v_2, v_6\}$, $\{v_6, v_8\}$, $\{v_7, v_9\}$.

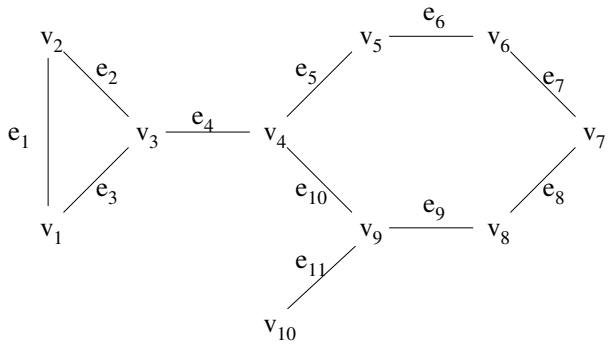

Figure 3: Contiguity graph of Example 2.

The heuristic HTREE finds the connected partition with 3 clusters shown in Figure 4, while a connected partition with a maximum number of clusters (*i.e.,* 4) is shown in Figure 5.

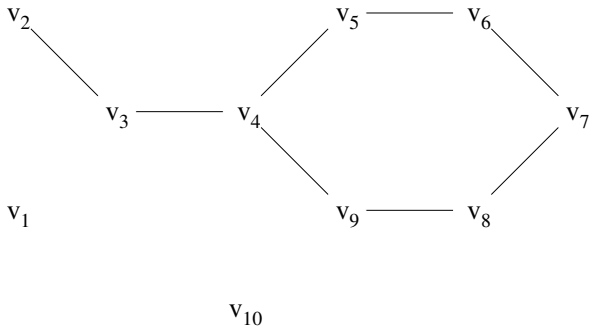

Figure 4: Partition found by HTREE ($s \geq 8$)

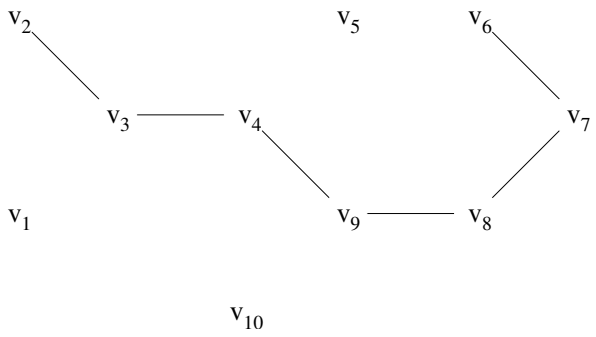

Figure 5: Optimal partition with maximum number of clusters ($s \geq 8$)

As mentioned above, algorithm HTREE proceeds in a myopic way; moreover, there may be arbitrary choices to make if, at a given iteration, there are several shortest paths between the vertices v_k and v_ℓ in H. Another way to obtain a heuristic, based on CTREE for the case where H is a general graph, which avoids the second drawback, is to select some spanning tree T_H of H and then apply CTREE to it. Proposition 7 provides motivation to do so. One criterion for selecting edges to belong to this tree is that they belong to shortest paths between endpoints v_k and v_ℓ of edges of T. This idea can be implemented as follows:

Heuristic PATHTREE

Step 1. Minimum spanning tree
Compute a minimum spanning tree T of $G = (V, E(G))$

Step 2. Weighting edges
Weight the edges $\{v_i, v_j\}$ of H by the number of edges $\{v_k, v_\ell\}$ of T such that $\{v_i, v_j\}$ belongs to a shortest path between v_k and v_ℓ;

Step 3. Maximum spanning tree
Find a maximum spanning tree T_H of H with the weights so defined;

Step 4. Path shrinking
Apply CTREE to T_H.

The complexity of heuristic PATHTREE is in $O(N^3)$, due to Step 2.

Focusing on the dissimilarity matrix $D = (d_{k\ell})$ suggests another, simpler way to obtain a spanning tree T_H:

Heuristic DISTREE

Step 1. Weighting edges
Weight the edges of H by the corresponding dissimilarities;

Step 2. Minimum spanning tree
Find a minimum spanning tree T_H of H with the weights so defined;

Step 3. Shrinking edges
Apply CTREE to T_H.

This heuristic has a complexity in $\Theta(N^2)$. Algorithm DISTREE is similar to the first steps of the MIDAS algorithm of Maravalle and Simeone (1995) for connected partitioning with the minimum inertia criterion (the latter algorithm also includes rules for improvement by tree-modification). The first two steps of DISTREE are also the same as those of the algorithm of Monestiez (1977) discussed above; the third one is different as CTREE, which exploits all dissimilarities, is then used instead of Gower and Ross' (1969) single linkage algorithm.

The three heuristics described in this section, as well as another one (derived from the exact algorithm) presented in Section 7 are compared experimentally in Section 8.

6 An exact algorithm for general graphs

In the present section we describe an iterative exact algorithm for solving the CONNECTED MAX SPLIT GRAPH CLUSTERING problem on a general graph. An informal description is given in Section 6.1. A formal description follows in Section 6.2. Some discussion on the number of iterations can be found in Section 6.3.

6.1 Informal description

As seen in Section 3, the problem can be reformulated as follows:

$$(P) \quad \begin{cases} \text{Given } s \in S, \text{ find a spanning forest } F \text{ of } H \\ \text{which wraps } T'_s \text{ and has the smallest number of edges.} \end{cases}$$

The algorithm proposed below finds a solution to (P) by solving a sequence of increasingly constrained set covering problems. The basic idea is the following. Let

$K_1, \ldots, K_q$ be the connected components of T'_s. Let K_i be one such component with cardinality ≥ 2 and let $v_k, v_\ell \in K_i, v_k \neq v_\ell$. Consider now an arbitrary subset $V_{k\ell}$ of V which contains v_k but not v_ℓ. We denote by $\mathcal{V}_{k\ell}$ the collection of all such subsets $V_{k\ell}$.

Let $\omega(V_{k\ell}, v_\ell)$ be the subset of edges $\{u, v\}$ of H with $u \in V_{k\ell}$ and $v \notin V_{k\ell}$ such that there exists a path from v to v_ℓ not going through any vertex from $V_{k\ell}$. Since an optimal forest F wraps T'_s, there must be a path γ in F joining v_k to v_ℓ. At least one of the edges of γ must belong to $\omega(V_{k\ell}, v_\ell)$. To enforce this we write down the constraint

$$\sum_{e \in \omega(V_{k\ell}, v_\ell)} x_e \geq 1,$$

where

$$x_e = \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, we can write down the following set covering problem, later called the master set covering problem

$$\text{(MSC)} \quad \begin{cases} \min \sum\limits_{e \in E(H)} x_e \\ \quad \text{subject to:} \\ \quad\quad \sum\limits_{e \in \omega(V_{k\ell}, v_\ell)} x_e \geq 1, \qquad i = 1, \ldots, q;\ v_k, v_\ell \in K_i;\ V_{k\ell} \in \mathcal{V}_{k\ell} \\ \quad\quad x_e \in \{0, 1\}, \qquad e \in E(H). \end{cases}$$

Any optimal solution to (MSC) must be a spanning forest of H wrapping T'_s and having a minimum number of edges.

The set covering problem (MSC) has only $m = |E(H)|$ variables but a huge number of constraints. Thus it is usually computationally infeasible to solve (MSC). Hence, we shall take a constraint generation (or cutting plane) approach: instead of solving (MSC) directly, we solve a nested sequence of set covering problems, the r^{th} of which involves a (usually small) subset L_r of constraints of (MSC), with $L_1 \subseteq L_2 \subseteq \cdots$.

An optimal solution to L_r yields a spanning forest F_r. If F_r wraps T'_s then F_r is an optimal solution to problem (P); if not, for each pair of vertices (v_k, v_ℓ) such that v_k and v_ℓ belong to the same connected component of T'_s but to two different connected components $C_i = V_{k\ell}$ and $C_j = V_{\ell k}$ of F_r, add to L_r the constraints corresponding to $\omega(C_i, v_\ell)$ and $\omega(C_j, v_k)$. Observe that these constraints are present in (MSC) but are absent from L_r, since they are violated by the optimal solution of the r^{th} set covering

problem. Call L_{r+1} the resulting set of constraints and iterate. Note that a same set covering constraint can be obtained for different $\omega(C, v)$: in that case of course, we generate the constraint only once.

In order to initialize the procedure, one takes F_0 to be the forest whose connected components are the single vertices of V. Clearly the algorithm is finite since new constraints are added at each iteration and the total number of constraints is bounded from above by $N2^N$.

Example 3. Consider again Example 2, with $s = 8$. F_0 is the forest consisting of the 10 isolated vertices. The initial set L_1 of constraints is

$$\begin{array}{lcll} x_1 + x_2 & \geq & 1 & (\omega(\{v_2\}, v_6)) \\ x_6 + x_7 & \geq & 1 & (\omega(\{v_6\}, v_2) \text{ and } \omega(\{v_6\}, v_8)) \\ x_7 + x_8 & \geq & 1 & (\omega(\{v_7\}, v_9)) \\ x_8 + x_9 & \geq & 1 & (\omega(\{v_8\}, v_6)) \\ x_9 + x_{10} & \geq & 1 & (\omega(\{v_9\}, v_7)) \end{array}$$

(note that in the last constraint, the variable x_{11} does not appear because the only path from v_{10} to v_7 passes through v_9).
One optimal solution to the first set covering problem corresponds to the forest F_1 whose set of edges is $\{e_1, e_7, e_9\}$. The connected components of F_1 are $\{v_1, v_2\}$, $\{v_3\}$, $\{v_4\}$, $\{v_5\}$, $\{v_6, v_7\}$, $\{v_8, v_9\}$, $\{v_{10}\}$. Notice that v_2 and v_6 belong to different components of F_1, and so do v_6 and v_8, and v_7 and v_9. Hence, one obtains L_2 by adding to L_1 the constraints

$$\begin{array}{lcll} x_2 + x_3 & \geq & 1 & (\omega(\{v_1, v_2\}, v_6)) \\ x_6 + x_8 & \geq & 1 & (\omega(\{v_6, v_7\}, v_i) \text{ for } i = 2, 8, 9) \\ x_8 + x_{10} & \geq & 1 & (\omega(\{v_8, v_9\}, v_6) \text{ and } \omega(\{v_8, v_9\}, v_7)). \end{array}$$

An optimal solution F_2 to the second set covering problem corresponds to the set of edges $\{e_2, e_6, e_8, e_{10}\}$. One obtains L_3 by adding to L_2 the constraints

$$\begin{array}{rcl} x_4 & \geq & 1 \\ x_5 + x_7 & \geq & 1 \\ x_7 + x_9 & \geq & 1 \\ x_5 + x_9 & \geq & 1. \end{array}$$

An optimal solution F_3 corresponds to $\{e_2, e_4, e_7, e_8, e_9\}$. One obtains L_4 by adding the constraints

$$\begin{array}{rcl} x_5 + x_{10} & \geq & 1 \\ x_6 + x_{10} & \geq & 1. \end{array}$$

An optimal solution F_4 corresponds to $\{e_2, e_4, e_7, e_8, e_9, e_{10}\}$. This time F_4 wraps T_8. Thus, F_4 yields an optimal forest for $s = 8$. (One sees that F_4 coincides with the forest shown in Figure 5.)

One possible refinement of the algorithm is to use Propositions 5 and 6 to detect edges which must be present in an optimal forest $F(s)$. This is done by replacing T_s by T'_s.

Example 4. Consider again Example 2, with $s = 8$. At the beginning, the connected components of $T'_8 = T_8$ are $\{v_1\}$, $\{v_2, v_6, v_8\}$, $\{v_3\}$, $\{v_4\}$, $\{v_5\}$, $\{v_7, v_9\}$ and $\{v_{10}\}$. Applying two times Proposition 5 shows that v_3 and v_4 must be in the same cluster than v_2, which we can express by adding to T_8 the edges $\{v_2, v_3\}$ and $\{v_2, v_4\}$, obtaining the new forest T'_8. The connected components of T'_8 are $K_1 = \{v_1\}$, $K_2 = \{v_2, v_3, v_4, v_6, v_8\}$, $K_3 = \{v_7, v_9\}$, $K_4 = \{v_5\}$, $K_5 = \{v_{10}\}$. Since all paths in H from v_7 to v_9 pass through a vertex from K_2, we can merge K_2 and K_3 by Proposition 5. The edges of a maximal spanning forest of the subgraph $H(K_2 \cup K_3)$ induced by $K_2 \cup K_3$ are e_3, e_4, e_7, e_8, e_9. All these edges must be present in an optimal forest $F(8)$ by Proposition 6. The set of edges of the initial spanning forest F_0 is $E^1 = \{e_3, e_4, e_7, e_8, e_9\}$. Hence the connected components of F_0 are $\{v_2, v_3, v_4\}$, $\{v_5\}$, $\{v_6, v_7, v_8, v_9\}$ and $\{v_{10}\}$. The initial set L_1 of constraints is

$$\begin{aligned} x_5 + x_{10} &\geq 1 \qquad \text{(e.g., } \omega(\{v_2, v_3, v_4\}, v_6)) \\ x_6 + x_{10} &\geq 1 \qquad \text{(e.g., } \omega(\{v_6, v_7, v_8, v_9\}, v_2)). \end{aligned}$$

The optimal solution of the set covering problem consists of the edge e_{10}. The forest F_1 consisting of e_{10} and of the edges of E_1 wraps T_8 and hence it is an optimal $F(8)$. Therefore $P_4 = \Big\{\{v_1\},\ \{v_2, v_3, v_4, v_6, v_7, v_8, v_9\},\ \{v_5\},\ \{v_{10}\}\Big\}$ is a connected partition with split at least 8 and having the maximum number of clusters.

A further refinement of the exact algorithm consists in finding suboptimal — rather than optimal — solutions to the set covering problems by means of some fast heuristic, as long as new constraints are generated. As soon as a suboptimal solution of some set covering problem yields a wrapping forest (possibly not having a minimum number of edges), this problem is solved again, but this time up to optimality i.e., by an exact algorithm. Notice, however, that the optimal solution of the covering problem so obtained may not correspond to a wrapping forest. In this case the algorithm goes on, solving again heuristically and then exactly set covering problems until the exact algorithm yields a solution which corresponds to a wrapping forest.

6.2 Formal description

We now give a formal description of the full version of the algorithm.

Algorithm CCMSC

Step 1: (Initialization)
Compute a minimum spanning tree T of G with respect
to lengths of the edges given by the dissimilarities;
$S \leftarrow$ list of lengths of the edges of T, sorted in non-decreasing order;
$s \leftarrow$ first value of S; $T_s \leftarrow G(V, \emptyset)$

Step 2: (Preprocessing)
Use Proposition 5 to compute the forest T'_s;
Use Proposition 6 to compute a set of edges E^1 that must be present
in an optimal forest $F(s)$;
Let F be the forest (V, E^1);
heur $\leftarrow$ false;
Let SC be the trivial set covering problem $\begin{cases} \min \displaystyle\sum_{e \in E(H) \setminus E^1} x_e \\ \text{s.t.} \quad x_e \in \{0,1\}, \quad e \in E(H) \setminus E^1; \end{cases}$

Step 3 (Wrapping test and constraint generation)
For all $\{v_k, v_\ell\}$ in a same connected component of T'_s **do**
If v_k, v_ℓ belong to 2 different connected components C', C'' of F **then**
Update SC by adding the constraints $\displaystyle\sum_{e \in \omega(C', v_\ell)} x_e \geq 1$ and $\displaystyle\sum_{e \in \omega(C'', v_k)} x_e \geq 1$;
EndIf
EndFor;
If no constraint has been added then **go to** 5;

Step 4: (Heuristic solution of set covering subproblem)
heur $\leftarrow$ true;
Find a suboptimal solution $\tilde{x}$ to SC by a heuristic;
Let F be the forest whose set of edges is $E^1 \cup \{e \in E(H) : \tilde{x}_e = 1\}$;
Go to 3;

Step 5: (Optimality test)
If heur = false **then** go to 7 {an optimal forest $F(s)$ has been found};

Step 6: (Exact solution of set covering subproblem)
heur $\leftarrow$ false

Find an optimal solution x^* to SC;
Let F be the forest whose set of edges is $E^1 \cup \{e \in E(H) : x_e^* = 1\}$;
Go to 3;

Step 7: (Output optimal partition)
$M \leftarrow N - |E(F)|$;
Let P_M be the connected partition whose M clusters are the connected components of F; { in general $s(P_M) \geq s$};
Output $s, M, P_M, s(P_M)$;

Step 8: Next split value
Let s be the smallest value in S satisfying $s > s(P_M)$;
If s does not exist then **stop**;
Let T_s be the forest $\{V, E(T_s)\}$ with $E(T_s) = \Big\{\{v_k, v_\ell\} \in E(T) \mid d_{k\ell} < s\Big\}$;
Go to 2.

Note that in Step 8, we can start from the tree $T_{s''}$ obtained for the value s'' of the previous iteration and add the edges $\{v_k, v_\ell\}$ such that $s'' \leq d_{k\ell} < s$. Similarly, in Step 2, the preprocessing can be somewhat sped up by considering only the changes brought by the new edges.

Observe that the CCMSC algorithm usually provides optimal partitions for some values of M but not for all; in other words, only some of the values of the split $s \in S$ correspond to optimal partitions under connectivity constraints with a maximum number of clusters. For some applications, it may be desirable to find partitions with a *given* number M of connected clusters and the largest possible split under such conditions. One may proceed as follows. First, compute the split s of the partition with M clusters using the single linkage algorithm (s is equal to the $(M-1)^{th}$ value of S). Next, apply the CCMSC algorithm with $S = \{s\}$. The variant of the CCMSC algorithm so obtained will be denoted CCMSC(s). Let M' denote the number of clusters obtained. If $M' < M$, apply CCMSC(s') algorithm where s' is equal to the largest value of S smaller than s. Repeat the process until $M' \geq M$. If $M' > M$, use Proposition 4 to merge clusters until $M' = M$ (note that this merging step cannot result in an increase of the split).

6.3 Remark on the complexity of the algorithm

The CCMSC algorithm has to solve repeatedly set covering problems, which are NP-hard. In this section, we show that it is even not possible in general to bound the

number of set covering problems to solve by a polynomial in N (or equivalently the number of constraints in the last solved set covering problem).

Let us consider the graph H drawn in Figure 6 with $N = p^2 + 2$. Assume

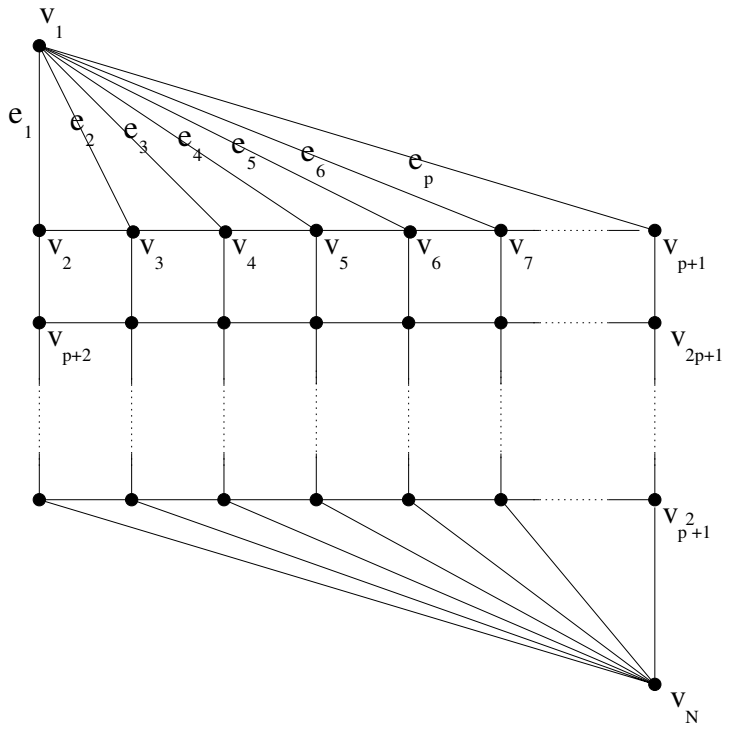

Figure 6: Contiguity graph H

that the dissimilarity between vertices v_1 and v_N is 1, that the dissimilarity between vertices v_1 and v_2 is 2 and that the other dissimilarities are ≥ 2. For simplicity we assume that all set covering problems are solved exactly (this is for example the case if the heuristic is very good).

For the first threshold split value $s = 1$, the algorithm finds the granular partition with N clusters, with split equal to 1. The next value in S is $s = 2$. The forest $T'_s = T_s$ is reduced to the edge $\{v_1, v_N\}$, F is the forest composed of the N isolated vertices and $E^1 = \emptyset$. Since $\{v_1, v_N\}$ belongs to T'_s but v_1 and v_N are in different connected components of F, we add the following constraints to the set covering problem in Step 3:

$$x_1 + \ldots + x_p \geq 1$$
$$x_{|E(H)|-p+1} + \ldots + x_{|E(H)|} \geq 1.$$

Observe that since we are minimizing $\sum_{i=1}^{|E(H)|} x_i$, any variable not appearing in the constraints will be at 0 in any optimal solution. Observe furthermore that when we add set covering constraints, new variables can be introduced only when the corresponding edge is incident to an edge whose variable was at 1 in the optimal solution of the previously solved set covering problem. This implies that at least during a certain number of iterations, the constraints of the set covering problems can be partitioned in two groups: one involving variables corresponding to edges that are close to v_1, and the other one involving variables corresponding to edges close to v_N. The set covering problems themselves can be decomposed accordingly. From now on, we focus on the partial set covering problems associated to v_1.

A subgraph of H is said to be a v_1-rooted subtree if it is a tree containing the vertex v_1. Given a graph G, the v_1-connected component of G is the connected component of G containing v_1. Note that a given set of vertices can be the v_1-connected component vertex set of more than one v_1-rooted subtree.
We first need the following Lemma:

Lemma 1
As long as the set covering constraints associated to v_1 and v_N do not include common variables, the CCMSC *algorithm generates a sequence of blocks $B_{v_1}^{(1)}, B_{v_1}^{(2)}, \ldots, B_{v_1}^{(k)}, \ldots$ of subgraphs of H, where:*

a) each block $B_{v_1}^{(k)}$ contains v_1-rooted subtrees of H with exactly k edges;

b) all v_1-rooted trees of $B_{v_1}^{(k)}$ have a different v_1-connected component vertex set;

c) for each possible v_1-connected component vertex set with $k+1$ vertices, there exists in $B_{v_1}^{(k)}$ a v_1-rooted tree with this v_1-connected component vertex set.

The order in which the trees appear in a particular block is not relevant.

Proof: Since the set covering constraints are added at each iteration, the sequence of optimal values is non-decreasing, hence the sequence of solutions generated has the block-structure indicated in the Lemma. Note also that since the set covering constraints are defined with respect to the connected component vertex set of the current v_1-rooted subtree, we cannot generate two subtrees with the same connected component vertex set: this shows the point b). It remains to show that 1) no subgraph of H other than trees containing v_1 can be generated and that 2) for all v_1-connected component vertex set of size $k+1$, a v_1-rooted subtree is generated. We proceed by induction on k. The property is true for $k = 1$: at the beginning, the set covering problem (recall that we consider only the part corresponding to v_1) contains

the unique constraint

$$x_1 + \ldots + x_p \geq 1 \tag{1}$$

expressing that at least one edge leaving v_1 must be chosen. An optimal solution of this set covering problem is obtained by fixing an arbitrary x_i to 1. The new set covering constraint that is added is then of the form

$$\sum_{j=1, j\neq i}^{p} x_j + \sum_{e \in E(H): e=\{v_{i+1}, v\}, v \neq v_i} x_e \geq 1.$$

Again an optimal solution is obtained by fixing a variable x_ℓ to 1 (note however that we must have $\ell \neq i$, otherwise the just introduced constraint would be violated). Using the same reasoning, we see that the algorithm enumerates the p solutions of value 1. Each of these solutions corresponds to a v_1-rooted subtree of H of size 1. No other solution of value 1 can be obtained, since it would violate the constraint (1). Moreover with each v_1-connected component of 2 vertices we can associate a v_1-subtree.

Now assume that the property is true up to k. We want to show that it is still true up to $k+1$. Assume first that a subgraph G of H that is not a v_1-rooted tree is generated. By the nature of the set covering constraints, G must be a forest, otherwise we could strictly improve the solution by removing one of the edges of a cycle. Consider now the connected component G' of G containing v_1: G' is a v_1-rooted subtree of H of size $\leq k$. By the induction assumption, the set covering problem contains a constraint relative to G' (or more precisely, to its v_1-connected component vertex set). Clearly this constraint is violated by G, which shows that only v_1-rooted subtrees of H can be generated.

Now assume that there exists a v_1-connected component vertex set S of H, with $k+2$ vertices, that violates the set covering constraint associated with a v_1-connected component vertex set S' of size $\leq k+2$ previously generated. We show that this leads to a contradiction. Since $|S| \geq |S'|$ and $S' \neq S$, we cannot have $S \subseteq S'$, hence there exists a vertex $u \in S \setminus S'$. Since S is v_1-connected, there exists a path in H joining v_1 to u. But this path necessarily contains an edge joining a vertex of S' to a vertex of $S \setminus S'$, hence the set covering constraint corresponding to S' is satisfied, a contradiction. Therefore all v_1-connected component vertex sets of size $k+2$ will be generated, which completes the proof. ∎

Proposition 8 *The* CCMSC *algorithm can take a super-polynomial number of iterations on some instances.*

Proof: Consider again the graph H of Figure 6. By Lemma 1, the algorithm enumerates all v_1-connected component vertex sets of the graph H in increasing order of size. The shortest path between v_1 and v_N has length $p+1$, hence there is still no interference between the partial set covering problems corresponding to v_1 and those corresponding to v_N when the algorithm enumerates the v_1-connected component vertex sets with $\lfloor \frac{p}{2} \rfloor + 1$ vertices. This gives a lower bound of $\binom{p}{\lfloor \frac{p}{2} \rfloor}$ on the number of iterations, which corresponds to the number of the v_1-connected component vertex sets with vertices in $\{v_2, \ldots, v_{p+1}\}$ (in addition of the vertex v_1). Since $p = O(\sqrt{N})$, the result follows. ∎

Remark 3. The above result also holds for more general graphs and for more general dissimilarity distributions. In particular, all what is needed in the proof of Proposition 8 is that the graph H satisfies the following properties:

- v_1 has at least p neighbours $v_2, \ldots, v_{p+1}$, with $p = O(\sqrt{N})$, such that from each of these vertices there exists a path to v_N not passing through v_1;
- There exists at least 2 vertex-independent paths between v_1 and v_N (this ensures that no preprocessing due to Proposition 5 occurs in Step 2);
- The shortest path between v_1 and v_N has length $\geq p+1$.

Note also that Proposition 8 is still true if instead of considering v_1-connected component vertex sets with $\lfloor \frac{p}{2} \rfloor + 1$ vertices with $p = O(\sqrt{N})$, we consider v_1-connected component vertex sets with $\lfloor \frac{p}{c} \rfloor + 1$ vertices with $p = O\left(N^{\frac{1}{c'}}\right)$ where c and c' are integer constants ≥ 2.
Finally there may be more than one dissimilarity with value 1. The simplest way to obtain such an instance is to consider c'' copies of H, where c'' is a constant, and merge the vertices v_1 of each copy together in order to obtain a connected graph.

7 Another heuristic for general graphs

Exact solution of the set covering subproblems is by far the most time-consuming part of the CCMSC algorithm. Quite often the heuristic of Step 4 is sufficient to find an optimal solution and Step 6 only aims at proving optimality of this solution. When large instances or dense contiguity graphs H are considered, computing time necessary for exact solution becomes prohibitive. Then another heuristic for CONNECTED MAX

SPLIT GRAPH CLUSTERING is obtained by only solving heuristically the set covering problems. Modifications to be made to the CCMSC algorithm are: *(i)* deletion of Steps 5 and 6 and *(ii)* move to Step 7 instead of Step 5 when Step 3 shows a wrapping has been obtained. We call HCOVER the heuristic obtained in this way.

The set covering algorithm used in CCMSC and HCOVER is close to that of Fisher and Kedia (1990). We used the primal heuristic and the first and third dual heuristics that they proposed.

8 Computational experience

All heuristic algorithms (HTREE, PATHTREE, DISTREE, HCOVER) as well as the exact CCMSC algorithm for general graphs proposed in this paper have been tested on a SUN ENTERPRISE-10000 computer (400 Mhz, 64 G). Seven test problems have been considered. The first five of them (TUSCANY, ROME, CAMPANY, LATIUM, UPPER LATIUM) are regional clustering problems from Maravalle and Simeone (1995). Their characteristics are recalled in Table 2. The graph H expresses contiguity between

Test problem	N	$\|E(H)\|$	density $= \frac{\|E(H)\|}{N}$	# characteristics
TUSCANY	287	773	2.69	1
ROME	121	302	2.50	1
CAMPANY	81	165	2.04	6
LATIUM	412	566	1.37	1
UPPER LATIUM	210	331	1.58	5
PARIS	90	444	4.93	-
USA	48	212	4.42	-

Table 2: Characteristics of the test problems

districts in the city for the ROME problem. For the other four problems, the vertices represent towns and the edges represent major roads between towns. Thus all graphs are planar or nearly planar. Dissimilarities are Euclidean distances between measurements of 1 to 6 socio-economic characteristics. Two additional test problems (PARIS, USA) have been constructed. For the PARIS problem, the entities correspond

to the districts of Paris and its surroundings. The graph H represents the contiguity between districts. For the USA problem, the entities correspond to the states of USA and the graph H represents the contiguity between states (the states Hawai and Alaska were removed in order to have a connected graph). For these four problems, the dissimilarity values are drawn at random in a uniform distribution on the interval $[0, 1]$. The dissimilarity matrices and the contiguity graphs for all these 7 instances are available on the web site *www.crt.umontreal.ca/∼sphinx1*.

The remaining of this section is organized as follows. In Section 8.1, we compare the performance of the four heuristics for all values of the threshold split on the CAMPANY instance. In Section 8.2, we do a similar analysis for the other instances but for a small subset of the threshold values. Finally in Section 8.3, we consider the variant of the exact CCMSC algorithm that gives a partition with the largest possible split for a specified number of clusters.

8.1 Heuristic solution of CAMPANY for all values of the threshold

We first solved the CAMPANY instance for all values of the threshold split by the four heuristics (recall that the threshold values are considered in increasing order). The computing times (in seconds) were the following: HTREE: 0.05s, PATHTREE: 0.14s, DISTREE: 0.04s, HCOVER: 1107182s (12 days 19 hours 33 minutes). The distribution of the time for HCOVER is given in Figure 7. We observe that the computing time is reasonable for roughly the 17 smallest threshold values and the 27 greatest values, but explodes for intermediate values.

Since solving exactly CAMPANY was not possible, we evaluate the quality of the heuristics with respect to the best value found by the 4 heuristics. Figure 8 represents the difference with respect to the best value (i.e., the number of clusters), for each heuristic and for each value of the threshold. The best results are obtained by the HTREE and HCOVER heuristics. Heuristic DISTREE performs particularly badly for the first half of the threshold values: this is due to the fact that the 2 clusters that have to be merged at the very beginning are connected together by a long path, and hence all clusters that are on this path must also be merged with the 2 clusters.

Another way to look at the results of Figure 8 is given in Table 3, which gives for each heuristic the number of times the solution found was the best one, the second

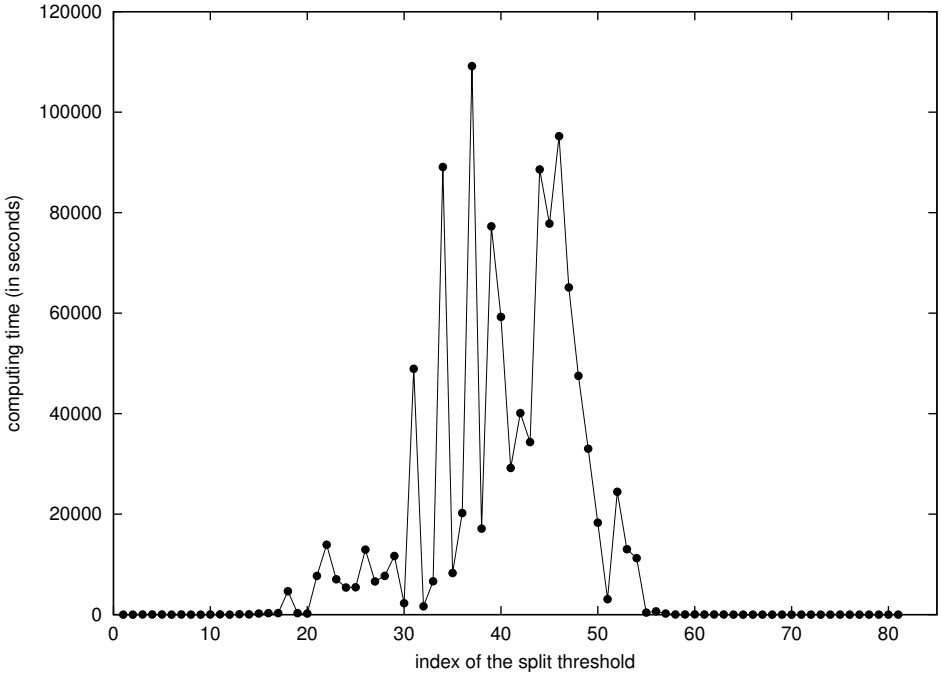

Figure 7: Repartition of the computing time for HCOVER

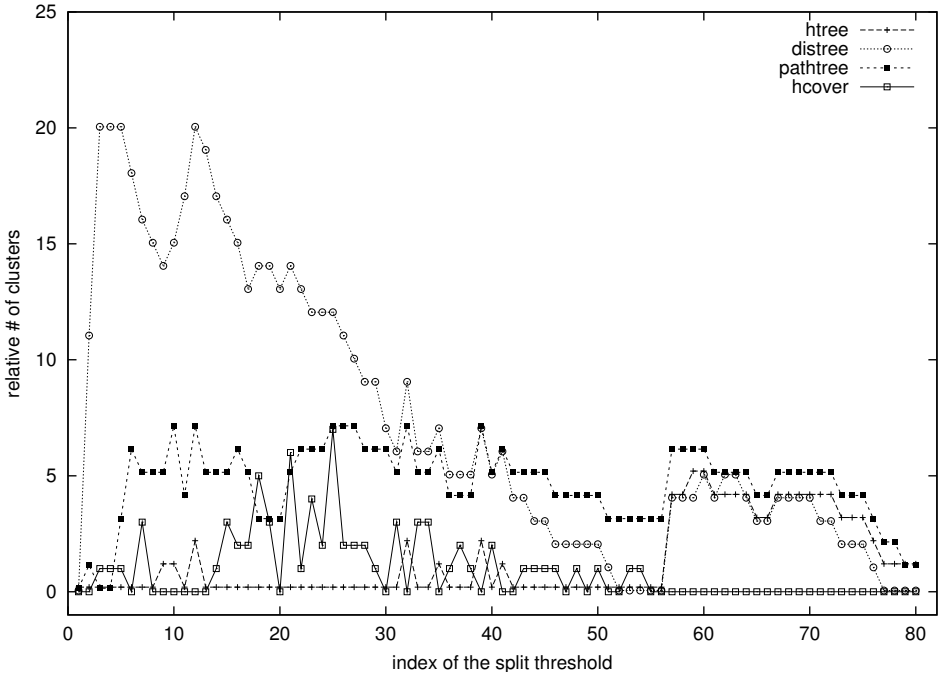

Figure 8: Difference to the best value for the 4 heuristics

best one, the third best one, etc... Note that HTREE finds the best solution slightly more often than HCOVER does. However if we consider rank 1 and 2 together, HCOVER wins clearly, at least with respect to the quality of the solution found (HCOVER takes considerably much more time through). DISTREE and PATHTREE are clearly outclassed.

	htree	distree	pathtree	hcover
1	47	8	3	45
2	19	22	13	33
3	14	23	41	2
4	0	27	23	0

Table 3: Number of times each rank is obtained for CAMPANY

Table 4 gives some more additional indicators for the first 10% smallest threshold values, the last 10%, and some intermediate values. For HCOVER, these indicators are defined as follows: *merge1* is the number of merges starting from the canonical partition (i.e., the partition where each entity defines a cluster) in order to obtain T_s; *merge2* is the number of additional merges done using the preprocessing of Proposition 5 to obtain T'_s; *iter* is the number of iterations through Step 4 of the algorithm. This number also corresponds to the number of set covering problems solved. The indicators *ncon* and *nvar* represent respectively the number of constraints and vari-

		HTREE		PATHTREE		DISTREE		HCOVER							
i	s_i	M	split	M	split	M	split	M	split	merge1	merge2	iter	ncon	nvar	tcpu
2	8.01	79	8.01	78	8.01	68	8.01	79	8.01	1	0	3	6	27	0.00
3	8.22	73	8.22	73	8.22	53	8.22	72	8.22	2	0	60	135	94	17.89
4	10.52	72	10.52	72	10.52	52	10.52	71	10.52	3	0	60	135	94	18.02
5	13.56	71	13.56	68	13.56	51	13.80	70	13.56	4	0	60	135	94	17.70
6	13.80	69	13.80	63	13.80	51	13.80	69	13.80	5	0	27	89	78	4.48
7	14.10	65	14.10	60	14.10	49	15.24	62	14.33	6	0	44	175	121	18.47
8	14.33	64	14.33	59	14.33	49	15.24	64	14.33	7	0	21	93	90	3.75
9	15.24	62	15.24	58	15.24	49	15.24	63	15.24	8	0	31	136	89	9.26
17	17.28	43	17.28	38	17.28	30	17.28	41	17.28	16	0	123	831	164	282.60
25	19.95	34	19.95	27	19.95	22	19.95	27	19.95	24	0	533	2896	168	5229.28
33	22.67	25	22.67	20	22.67	19	22.67	22	23.25	32	0	572	3218	168	6501.25
40	23.86	20	23.86	15	23.86	16	23.86	23	23.86	41	4	102	570	132	185.40
50	25.69	14	25.69	10	25.86	12	26.59	17	25.69	49	8	35	135	90	15.98
58	27.45	11	27.45	9	28.99	11	27.67	15	27.45	57	5	18	47	59	2.35
66	29.92	6	31.78	5	31.78	6	37.94	10	31.78	65	5	1	2	13	0.00
74	38.07	4	38.07	3	38.07	5	38.07	7	38.07	73	1	0	0	0	0.00
75	39.62	3	41.00	2	41.00	4	41.00	6	39.62	74	1	0	0	0	0.01
76	41.00	3	41.00	2	41.00	4	41.00	5	41.00	75	1	0	0	0	0.01
77	41.03	2	44.57	1	∞	3	44.57	3	44.57	76	2	0	0	0	0.01
78	44.57	2	44.57	1	∞	3	44.57	3	44.57	77	1	0	0	0	0.01
79	50.49	1	∞	1	∞	2	51.56	2	51.56	78	1	0	0	0	0.00
80	51.56	1	∞	1	∞	2	51.56	2	51.56	79	0	0	0	0	0.01
	tcpu	0.05		0.14		0.04									

Table 4: Results of the heuristics for Campany ($N = 81$)

ables in the last set covering problem solved. Finally *tcpu* is the total computing time in seconds.

We observe that for the largest values of the threshold, the preprocessing step is sufficient to solve the problem with HCOVER and the computing time is very small. For intermediate values of the threshold, when the preprocessing does not allow any reduction of the problem, HCOVER can take much more time than the other heuristics while giving a solution of poorer quality.

8.2 Heuristic solution of the other instances

In most practical applications, the partitions of interest are those with a relatively small number of clusters. Such partitions are obtained for the largest threshold values. Table 5 presents the results obtained by the heuristics on the other instances when seeking a partition with split at least equal to the i^{th} threshold value s_i, for large i. A limit of 10 hours was set for each run. The sign $>$ in the *tcpu* column means that the problem was not solved within that time. Since for the HTREE, PATHTREE and DISTREE heuristics, the program gives in one run the results for all values of the threshold, the overall computing time is indicated in the last line for these heuristics. HCOVER almost always provide the best results, within a reasonable computing time. The most notable exception is for the ROME instance: for $i = 98$, HCOVER finds a partition with 12 clusters after more than 16000s, whereas HTREE was able to find a partition with 14 clusters in less than 1s. For $i = 86$ and $i = 110$, HCOVER is not even able to finish within the allowed computing time. HCOVER is also not able to solve the PARIS2 instance for $i = 64$. Among the tree-based heuristics, HTREE generally gives the best results, except for LATIUM and UPPER LATIUM where it is clearly beaten by DISTREE.

		HTREE		PATHTREE		DISTREE		HCOVER							
i	s_i	M	split	M	split	M	split	M	split	merge1	merge2	iter	ncon	nvar	tcpu
							TUSCANY								
231	1.20	32	1.20	31	1.20	31	1.20	35	1.20	230	20	1	4	20	0.08
259	2.10	13	2.50	13	2.50	12	2.50	15	2.50	258	14	0	0	0	0.05
278	10.10	4	12.90	4	12.90	4	12.90	5	10.50	277	5	0	0	0	0.05
279	10.50	4	12.90	4	12.90	4	12.90	5	10.50	278	4	0	0	0	0.05
280	12.90	4	12.90	4	12.90	4	12.90	4	12.90	279	4	0	0	0	0.05
281	13.30	3	14.30	3	14.30	3	14.30	3	14.30	280	4	0	0	0	0.06
282	14.20	3	14.30	3	14.30	3	14.30	3	14.30	281	3	0	0	0	0.05
283	14.30	3	14.30	3	14.30	3	14.30	3	14.30	282	2	0	0	0	0.05
284	25.80	2	29.30	2	29.30	2	29.30	2	29.30	283	2	0	0	0	0.05
285	29.30	2	29.30	2	29.30	2	29.30	2	29.30	284	1	0	0	0	0.05
286	62.70	1	∞	1	∞	1	∞	1	∞	285	1	0	0	0	0.04
	tcpu	0.66		1.84		0.47									
							ROME								
86	2.63	23	2.63	13	2.63	12	2.63	-	-	-	-	-	-	-	>
98	5.12	14	5.12	7	5.12	5	6.19	12	5.12	97	1	956	3260	132	16865.16
110	9.20	4	12.91	3	12.91	4	12.91	-	-	-	-	-	-	-	>
112	12.56	4	12.91	3	12.91	4	12.91	4	12.91	111	0	273	957	121	1509.11
113	12.91	4	12.91	3	12.91	4	12.91	4	12.91	112	0	104	347	113	219.67
114	15.59	3	16.98	2	43.66	3	16.98	3	16.98	113	0	152	489	113	457.50
115	16.98	3	16.98	2	43.66	3	16.98	3	16.98	114	4	0	0	0	0.01
116	17.29	2	43.66	2	43.66	2	43.66	2	43.66	115	4	0	0	0	0.01
117	20.78	2	43.66	2	43.66	2	43.66	2	43.66	116	3	0	0	0	0.01
118	23.38	2	43.66	2	43.66	2	43.66	2	43.66	117	2	0	0	0	0.00
119	43.66	2	43.66	2	43.66	2	43.66	2	43.66	118	1	0	0	0	0.01
120	70.28	1	∞	1	∞	1	∞	1	∞	119	1	0	0	0	0.01
	tcpu	0.11		0.30		0.07									
							LATIUM								
331	1.70	21	1.70	21	1.70	25	1.70	27	1.70	325	57	4	15	21	0.24
372	2.80	14	3.10	14	3.10	18	3.10	19	3.10	371	21	3	6	13	0.16
403	17.30	2	243.00	3	26.60	6	26.60	6	26.60	402	4	0	0	0	0.11
404	25.50	2	243.00	3	26.60	6	26.60	6	26.60	403	3	0	0	0	0.11
405	26.60	2	243.00	3	26.60	6	26.60	6	26.60	404	2	0	0	0	0.11
406	29.50	2	243.00	2	243.00	5	29.50	5	29.50	405	2	0	0	0	0.11
407	48.60	2	243.00	2	243.00	4	80.80	4	80.80	406	2	0	0	0	0.10
408	80.80	2	243.00	2	243.00	4	80.80	4	80.80	407	1	0	0	0	0.11
409	130.10	2	243.00	2	243.00	3	243.00	3	243.00	408	1	0	0	0	0.09
410	243.00	2	243.00	2	243.00	3	243.00	3	243.00	409	0	0	0	0	0.06
411	529.90	1	∞	1	∞	2	529.90	2	529.90	410	0	0	0	0	0.06
	tcpu	1.00		2.66		0.99									
							UPPER LATIUM								
169	4.84	19	4.84	15	4.84	16	4.84	29	4.84	168	10	11	42	86	0.88
190	14.01	8	14.01	4	14.01	10	14.01	16	14.00	189	4	3	6	34	0.05
201	33.99	4	48.14	2	56.97	7	33.99	9	33.99	200	1	0	0	0	0.04
202	48.14	4	48.14	2	56.97	6	48.14	8	48.14	201	1	0	0	0	0.03
203	54.26	3	54.26	2	56.97	4	54.26	6	54.25	202	1	1	2	7	0.04
204	56.97	2	56.97	2	56.97	3	56.97	4	56.96	203	3	0	0	0	0.04
205	57.93	1	∞	1	∞	2	239.85	2	239.84	204	4	0	0	0	0.03
206	103.73	1	∞	1	∞	2	239.85	2	239.84	205	3	0	0	0	0.03
207	239.85	1	∞	1	∞	2	239.85	2	239.84	206	2	0	0	0	0.02
208	246.92	1	∞	1	∞	1	∞	1	∞	207	2	0	0	0	0.02
209	780.07	1	∞	1	∞	1	∞	1	∞	208	1	0	0	0	0.02
	tcpu	0.26		0.72		0.21									
							PARIS								
64	1.49	13	1.49	10	1.49	9	1.49	18	1.49	63	7	3	6	16	0.02
73	1.87	11	1.87	7	1.97	6	1.87	15	1.87	72	3	0	0	0	0.00
82	3.02	9	3.29	4	3.29	1	∞	8	3.29	81	1	0	0	0	0.00
83	3.29	8	3.29	4	3.29	1	∞	8	3.29	82	0	0	0	0	0.00
84	3.47	7	3.47	3	3.47	1	∞	7	3.47	83	0	0	0	0	0.00
85	4.49	6	4.49	2	4.67	1	∞	6	4.48	84	0	0	0	0	0.01
86	4.67	5	4.67	2	4.67	1	∞	5	4.67	85	0	0	0	0	0.00
87	4.75	4	4.78	1	∞	1	∞	4	4.78	86	0	0	0	0	0.00
88	5.39	3	5.39	1	∞	1	∞	3	5.39	87	0	0	0	0	0.01
89	6.28	2	6.28	1	∞	1	∞	2	6.27	88	0	0	0	0	0.01
	tcpu	0.08		0.19		0.06									
							USA								
35	3.74	12	3.82	8	3.82	9	3.82	13	3.82	34	1	0	0	0	0.01
39	3.97	9	3.97	6	3.97	5	3.97	10	3.96	38	0	0	0	0	0.00
44	5.95	5	5.95	4	5.95	3	5.95	5	5.94	43	0	0	0	0	0.01
45	6.14	4	6.14	3	6.14	2	6.70	4	6.14	44	0	0	0	0	0.00
46	6.30	3	6.30	2	6.70	2	6.70	3	6.30	45	0	0	0	0	0.00
47	6.70	2	6.70	2	6.70	2	6.70	2	6.70	46	0	0	0	0	0.00
	tcpu	0.02		0.06		0.02									

Table 5: Comparison of the heuristics

8.3 Performance of the exact algorithm for the search of partitions with specified number of clusters and largest possible split

Finally results with the exact CCMSC algorithm are reported in Table 6.

M	nbcalls	M'	iter-heur	iter-exact	mmax	split	cpu_heur	cpu_exact	tcpu
					Tuscany				
7	7	8	0	0	0	6.70	0	0	0.29
8	6	8	0	0	0	6.70	0	0	0.25
9	6	10	0	0	0	5.40	0	0	0.23
10	5	10	0	0	0	5.40	0	0	0.21
29	8	32	9	5	7	1.30	0.01	0.00	0.46
					Rome				
3	5	3	0	0	0	16.983	0	0	0.04
4	-	-	-	-	-	-	-	-	>
5	-	-	-	-	-	-	-	-	>
					Campany				
7	2	7	0	0	0	38.07	0	0	0.01
8	2	8	0	0	0	37.94	0.00	0.00	0.01
16	8	16	37	18	17	27.67	1.43	0.17	1.69
24	≥ 9	≥ 19	≥ 533	≥ 117	≥ 373	≥ 25.69	≥ 503.70	≥ 17188.05	>
65	-	-	-	-	-	-	-	-	>
73	7	73	430	37	288	8.22	191.69	421.22	613.33
					Latium				
8	4	8	0	0	0	15.70	0	0	0.38
9	5	9	3	1	6	9.80	0	0	0.53
10	10	10	21	7	6	7.10	0	0	1.33
41	17	45	87	33	23	1.20	1.14	0.05	4.66
					Upper Latium				
7	3	8	1	1	2	48.14	0	0	0.10
8	2	8	1	1	2	48.14	0	0	0.06
9	2	9	0	0	0	33.99	0	0	0.06
10	4	10	0	0	0	20.08	0	0	0.12
21	11	21	36	14	10	7.4189	0.02	0.00	0.61
					Paris				
7	1	7	0	0	0	3.47	0	0	0.00
8	1	8	0	0	0	3.29	0	0	0.00
9	3	9	0	0	0	2.80	0	0	0.02
18	9	18	12	4	6	1.51	0	0	0.07
					USA				
4	1	4	0	0	0	6.14	0	0	0.01
5	1	5	0	0	0	5.95	0	0	0.01
10	1	10	0	0	0	3.96	0	0	0.00

Table 6: Performance of the exact algorithm

We use the variant of the CCMSC algorithm which provides the optimal partition for a fixed value of M. As explained at the end of Section 6.2, this may require several applications of algorithm CCMSC(s): the parameter *nbcalls* indicates the number of calls to CCMSC(s). M' is the number of clusters of the partition found after the last call of CCMSC(s). The parameters *iter-heur* and *iter-exact* are, respectively, the total numbers of heuristic and exact applications of the set covering algorithm in the overall process; *cpu-heur* and *cpu-exact* are the corresponding computing times. The parameter *mmax* corresponds to the maximum number of constraints in all set covering problems solved. Finally *tcpu* gives the total computing time.
The computing times for the Rome problem was prohibitive, even for $M = 4$ (the algorithm get stuck when trying to solve exactly a set covering problem with 302 variables and 1266 constraints). However the other problems, including large ones, could be solved optimally for values of M going up to 9 to 16, depending on the problem. As in most applications, the user is only interested in small number of clusters, the exact CCMSC algorithm is therefore quite efficient and competitive with the heuristics as for the computing times.

9 Conclusions

Connectivity constraints are among the most frequent types of constraints encountered in cluster analysis. Maximizing the split, i.e., the measure of separation used in the single-linkage algorithm, subject to such constraints is strongly NP-hard. However, it is possible to solve exactly many such problems with possibly several hundred entities using tools from combinatorial optimization (graph theory and set covering algorithms) provided the number of clusters is not too large. Moreover, if the contiguity graph is a tree, this problem can be solved in $\Theta(N^2)$ time with algorithm CTREE. Very large problems, or difficult ones for which partitions into many clusters are sought, can be solved heuristically. A heuristic version of the set covering type algorithm, HCOVER, can give very good results but may be very time consuming. The 3 other heuristics, which are different adaptations of CTREE, provide in a very short amount of time results whose quality however depends on the instances. In view of these short computational times, an improved procedure can be obtained by running the 3 heuristics and keeping the best solution of each (actually we can consider only HTREE and DISTREE as PATHTREE turns out to be then dominated).

References

[1] A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.

[2] R. E. Bellman and S. E. Dreyfus. *Applied dynamic programming*. Princeton University Press, Princeton, N. J., 1962.

[3] C. Berge. *Graphs and hypergraphs*. North Holland, Amsterdam, 1973.

[4] D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, 5:724–742, 1976.

[5] H. E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1:7–24, 1984.

[6] M. Delattre and P. Hansen. Bicriterion cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI*, 2(4):277–291, 1980.

[7] E. Diday and collaborateurs. *Optimisation en classification automatique. Tome 1,2*. Le Chesnay: Institut National de Recherche en Informatique et Automatique, 1979.

[8] A. Ferligoj and V. Batagelj. Clustering with relational constraints. *Psychometrika*, 47(4):413–426, 1982.

[9] A. Ferligoj and V. Batagelj. Some types of clustering with relational constraints. *Psychometrika*, 48(4):541–552, 1983.

[10] M. L. Fisher and P. Kedia. Optimal solution of set covering / partitioning problems using dual heuristics. *Management Science*, 36(6):674–688, 1990.

[11] W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53:789–798, 1958.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[13] A. D. Gordon. Classification in the presence of constraints. *Biometrics*, 29:821–827, 1973.

[14] A. D. Gordon. Methods of constrained classification. In R. Tomassone, editor, *Analyse de Données et Informatique*, pages 161–171. INRIA, Le Chesnay, 1979.

[15] A. D. Gordon. *Classification: Methods for the Exploratory Analysis of Multivariate Data*. Monographs on Applied Probability and Statistics. Chapman & Hall, New-York, 1981.

[16] A. D. Gordon. A survey of constrained classification. *Computational statistics & Data Analysis*, 1:17–29, 1996.

[17] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18:54–61, 1969.

[18] P. Hansen, B. Jaumard, and K. Musitu. Weight-constrained maximum split clustering. *Journal of Classification*, 7:217–240, 1990.

[19] J. A. Hartigan. *Clustering Algorithms*. Wiley, New-York, 1975.

[20] L. Hubert. Monotone invariant clustering procedures. *Psychometrika*, 38(1):47–62, 1973.

[21] N. Jardine and R. Sibson. *Mathematical taxonomy*. Wiley, New-York, 1971.

[22] L. Kaufman and P. T. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New-York, 1989.

[23] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.*, 7:48–50, 1956.

[24] L. Lebart. Programme d'agrégation avec contraintes. *Les Cahiers de l'Analyse des Données*, III(3):275–287, 1978.

[25] L. P. Lefkovitch. Conditional clustering. *Biometrics*, 36:43–58, 1980.

[26] M. Lucertini, Y. Perl, and B. Simeone. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*, 42(2-3):227–256, 1993.

[27] M. Maravalle and B. Simeone. A greedy algorithm for maximum split clustering on trees. Research Report 4 (serie A), Dipartimento di Statistica, Probabilità e Statistiche Applicate, Università degli studi di Roma "La Sapienza", 1985.

[28] M. Maravalle and B. Simeone. A spanning tree heuristic for regional clustering. *Communications in Statistics - Theory and Methods*, 24:625–639, 1995.

[29] M. Maravalle, B. Simeone, and R. Naldini. Clustering on trees. *Computational Statistics and Data Analysis*, 24:217–234, 1997.

[30] P. Monestiez. Méthode de classification automatique sous contraintes spatiales. *Statistique et Analyse des Données*, 3:75–84, 1977.

[31] F. Murtagh. A survey of algorithms for contiguity-constrained clustering and related problems. *The Computer Journal*, 28:82–88, 1985.

[32] C. Perruchet. Constrained agglomerative hierarchical classification. *Pattern Recognition*, 16(2):213–217, 1983.

[33] M. R. Rao. Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, 66:622–626, 1971.

[34] P. Rosenstiehl. L'arbre minimum d'un graphe. In P. Rosenstiehl, editor, *Théorie des Graphes, Rome, I.C.C.*, pages 357–368. Dunod, 1967.

[35] A. J. Scott. *Combinatorial Programming, Spatial Analysis, and Planning*. Methuen, London, 1971.

[36] B. Simeone. Optimal graph partitioning. *Atti Giornate di Lavoro AIRO, Urbino*, pages 57–73, 1978.

[37] H. Späth. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*. Chichester, Horwood, 1980.