Les Cahiers du GERAD ISSN: 0711–2440

An iterative algorithm for the solution of very large-scale diameter clustering problems

D. Aloise
C. Contardo

G–2015–140

December 2015

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

An iterative algorithm for the solution of very large-scale diameter clustering problems

Daniel Aloise [a]
Claudio Contardo [b]

[a] *GERAD & Department of Computer Engineering, Universidade Federal do Rio Grande do Norte, Natal, Brazil*

[b] *GERAD & Department of Management and Technology, ESG-UQAM, Montréal (Québec) Canada, H2X 3X2*

aloise@dca.ufrn.br
contardo.claudio@uqam.ca

December 2015

Les Cahiers du GERAD
G–2015–140

Abstract: We introduce an iterative algorithm for the solution of the diameter minimization clustering problem (DMCP). Our algorithm is based upon two observations: 1) subsets induce lower bounds on the value of the optimal solution of the original problem; and 2) there exists a subset whose optimal clustering has the same value as that of the original problem. We also describe how to adapt our algorithmic framework for the solution of other clustering problems, namely the minimum sum-of-diameters clustering problem (MSDCP), the split maximization clustering problem (SMCP) and the maximum sum-of-splits clustering problem (MSSCP). A parallel implementation of our algorithm can solve problems containing almost 600,000 entities while consuming only moderate amounts of time and memory. The size of the problems that can be solved using our algorithm is two orders of magnitude larger than the largest problems solved by the current state-of-the-art algorithm.

Key Words: Clustering, big data, optimization.

Acknowledgments: D. Aloise has been partially financed by CNPq-Brazil grant 308887/2014-0. C. Contardo has been partially financed by the Fonds de recherche du Québec – Nature et technologies (FRQNT) under Grant no 181909, and the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant no 435824-2013. These supports are gratefully acknowledged.

1 Introduction

Clustering is among the most important data mining tasks. Given a set of objects without any prior knowledge, clustering consists in finding subsets, called *clusters*, which are homogeneous and/or well separated. Homogeneity means that entities in the same cluster should be similar whereas separation means that entities in different clusters should differ one from the other.

The problem of clustering n objects into k clusters can be posed as a mathematical optimization problem as follows:

$$\min(\text{or}\max) \quad f(x) \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{k} x_{ij} = 1, \qquad \forall i = 1, \ldots, n \tag{2}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i = 1, \ldots, n; \quad \forall j = 1, \ldots, k. \tag{3}$$

The decision variables x_{ij} express the assignment of the object i to the cluster j. Thus, constraints (2) assure that each object is assigned to exactly one cluster.

The function f in (1) defines how homogeneity and separation are expressed in the clusters to be found [18]. Moreover, f is determinant to the computational complexity of the associated clustering problem. For example, the split of a cluster consists in maximizing the minimum dissimilarity across all pair of objects assigned to different clusters. Clustering via split maximization is polynomially solved in time $O(n^2)$ [10].

Among the many available criterion used in cluster analysis, the *diameter minimization* (DM) is the most natural to express homogeneity of the clusters found. The diameter (D_j) of a cluster $j \in \{1, \ldots, k\}$ is defined as:

$$D_j = \max\{d_{ii'} : 1 \leq i < i' \leq n, x_{ij} = x_{i'j} = 1\}, \tag{4}$$

where $d_{ii'}$ represents the dissimilarity between objects i and i'. The diameter minimization problem is then expressed as

$$\min \max_{1 \leq j \leq k} D_j. \tag{5}$$

DM tends to produce *compact* clusters due to its objective of producing clusters with small intra-cluster dissimilarities among their objects. For $k = 2$, the problem can be solved in polynomial time via a repulsion algorithm developed by Rao [26]. Brucker [6] showed that DM is polynomially solvable when data is located over a line. In two or more dimensions, DM is proved NP-hard by a reduction to the graph coloring problem [17]. Gonzalez [16] and Hochbaum & Shmoys [20] showed that the DM has a 2-approximation algorithm. Hochbaum [19] showed that DM cannot be approximated within a factor better than 2. Feder and Greene [12] showed that for objects located on the plane DM cannot be approximated within a ratio of 1.969.

The classic book of Garey and Johnson [15] refers to the DM clustering problem (henceforth DMCP) as *the clustering problem* to prove that the associated decision problem is, in general, strongly NP-complete. This highlights the importance the DM which is perhaps the most intuitive criterion for clustering. Besides, DM does not have the tendency to produce clusters of particular sizes [8], which may be important for datasets with a potential large cluster and several small ones. One disadvantage is that many distinct optimal solutions often exist, leaving to the analyst the burden of selecting one of them. Alternatives to automate this decision include combining DM with a second criterion in a multi-objective setting [7, 10].

Existing methods for large-scale hierarchical clustering are only heuristic, and often fail to find an optimal clustering for large problems. Moreover, many of those algorithms do not scale well and cannot be applied on large problems due to time or memory limitations. Regarding exact algorihtms, the landscape is even bleaker. The state-of-the-art algorithm for DM [9] can solve problems containing a few thousand nodes and rapidly runs out of memory beyond that.

In this paper, we propose an algorithm that aims at solving very large DM clustering problems and that consumes only moderate CPU time and memory. Our algorithm's running time is comparable to the time needed to compute the whole dissimilarity matrix, and sometimes it proves even faster. In terms of the memory space required to solve such problems, the requirements are minimal. Indeed, the proposed algorithm does never store the dissimilarity matrix explicitly. Our algorithm can, on most cases, detect an optimal clustering in a few seconds, and spend a long time to prove the optimality of such clustering. Thus, a simple modification of our algorithm can also be used to solve problems with an arbitrarily large number of observations in a heuristic fashion.

2 Literature review

DM has a quite old history in data analysis. A popular heuristic for DM is the classical complete-linkage hierarchical clustering algorithm [21, 29]. Complete-linkage merges at each step of its hierarchical construction the two subsets for which the maximum dissimilarity between their objects is minimum. The desired solution is then obtained by cutting the resulting tree at k clusters. However, the method seldom finds optimal solutions as reported by [3, 17].

The FPF (*Furthest Point First*) method introduced in [16] works with the idea of *head* objects for each cluster of the partition to which each object is assigned. In the first iteration, an object is chosen at random as the head of the first cluster and all objects assigned to it. In the next iteration, the furthest object from the first head is chosen as the head of the second cluster. Then, any object which is closer to the second head than to the first one is assigned to the second cluster. The algorithm continues for k iterations, choosing the object which is the furthest to its head as the head of the new cluster. The FPF heuristic is guaranteed to find a partition with diameter at most two times larger than the diameter of the optimal partition.

Regarding exact methods, Hansen and Delattre [17] explore the relationship between DM and graph-coloring to devise a branch-and-bound method to solve problems of non-trivial size. Brusco and Stahl [8] proposes a backtracking algorithm, denoted Repetitive Branch-and-Bound Algorithm (RBBA), that branches by assigning each object to one of the possible k clusters. Pruning is performed whenever: (i) the number of unassigned objects is smaller than the number of empty clusters; (ii) the assignment of an object to a particular cluster yields a partition with a diameter larger than the diameter of the best current branch-and-bound solution; and (iii) there is an unassigned object that cannot be assigned to any of clusters of the partial solution. Recently, Dao et al. [9] proposed a constraint programming approach for DM whose computational results outperformed the previous exact approaches found in the literature. In particular, the method obtained the optimal partition regarding the DM criterion for a real-world dataset with 5,000 objects grouped in three clusters.

We would like to mention that our algorithm is inspired from the need of handling extremely large datasets in the context of *big data* applications. We are not the first to elaborate upon the need of developing a scalable algorithm able to handle large datasets while avoiding as much as possible the scanning of the whole data. Zhang et al. [33] use a tree structure to estimate and store the distribution of the dissimilarities. Their algorithm then relies on this estimator to proceed to cluster the data. In Bradley et al. [5], the authors introduce an algorithm for the k-means clustering problem that uses compression and discarding of data to reduce the number of observations. Fraley et al. [14] introduce a sampling-based algorithm to solve problems containing large amounts of data in a sequential fashion. At every iteration of their algorithm, typically much smaller datasets are considered. Unlike our method, all these algorithms are heuristic by nature.

It is always important to compare clustering algorithms using the same datasets since the hardness of a clustering problem depends not only on the number of objects (n), but also on the number of sought clusters (k) and on how the objects are spread in the space. For example, if objects are embedded in an Euclidean space and located in very separated clusters, any reasonable exact method should find the optimal solution regardless of the number of objects.

3 Preliminaries and notation

Let V be the complete set of nodes, of size n. We let $E = \{\{u, v\} : u, v \in V, u < v\}$ be the set of edges linking all nodes in V, of size $m = n(n-1)/2$. For each edge $e = \{u, v\} \in E$, we denote by d_e the dissimilarity between nodes u and v. The dissimilarity matrix of V is denoted by $D(V) = [d_e]_{e \in E}$. A feasible solution of the DMCP is a partition $\mathcal{C} = (C_i)_{i=1}^k$ of V. The diameter of a cluster C_i is denoted by $d(C_i)$ and is equal to $\max\{d_e : e = \{u, v\} \in E, u, v \in C_i\}$. The cost associated with $\mathcal{C}$ is denoted by ω and is equal to $\max\{d(C_i) : i = 1, \ldots, k\}$. Our algorithm to solve the DMCP relies on the following results and hypothesis.

Lemma 1 *Let $U \subset V$ be a subset of V, and let $D(U)$ be its dissimilarity matrix. The optimal solution of the DMCP on $(U, D(U))$ provides a lower bound on the optimal solution of the DMCP on $(V, D(V))$.*

Proof. The proof is by induction on the size of $V \setminus U$. Let $|V \setminus U| = 1$. Let ω^V, ω^U be the values of two optimal solutions of the DMCP on $(V, D(V))$ and on $(U, D(U))$, respectively. Let $v^* = V \setminus U$. If $\omega^U > \omega^V$, the one can remove v^* from the cluster containing it in $(V, D(V))$ and get a solution for $(U, D(U))$ of cost smaller than or equal to ω^V, which contradicts the optimality of ω^U. If $|V \setminus U| = k > 1$ then one can apply the same argument k times. □

Proposition 1 *Let $U \subset V$, and let $\mathcal{C}^U$ be an optimal solution for $(U, D(U))$ of cost ω^U. If one can find a clustering $\mathcal{C}^V$ of V of cost ω^U, then that clustering is optimal for V.*

Proof. From the lemma, ω^U is a lower bound on the optimal solution value of DMCP for the set V. If ω^U is also the value of an upper bound (a feasible solution), then ω^U is optimal. □

Hypothesis 1 *There exists a subset $U \subset V$ of size $n' \ll n$ such that Proposition 1 holds for U.*

The algorithm introduced in this article provides an efficient way of finding such set U. Note that this hypothesis does not hold on problems in which the addition of a node into a cluster entails necessarily an increase in the objective function. Thus, our algorithm, in the way it is presented, is limited to problems presenting large degrees of degeneracy.

4 The general framework

We begin by selecting (using some criterion) an initial set U^0 containing κ points from V, where κ is a positive integer parameter, usually small. Let us set $i \leftarrow 0$. At iteration i, one solves problem $(U^i, D(U^i))$ to optimality using some method (e.g. branch-and-price, constraint programming). Let $\mathcal{C}^i, \omega^i$ denote the optimal solution and its value, respectively, for problem $(U^i, D(U^i))$. We then run a heuristic and try to build a feasible solution for problem $(V, D(V))$ of cost ω^i. Our heuristic proceeds as follows: The nodes in $V \setminus U^i$ are sorted using some ordering cheap to compute. Every node $v \in V \setminus U^i$ is then iteratively inserted into $\mathcal{C}^i$ on some cluster that does not sees its diameter increased. If for some node v^* this is not possible, the heuristic stops. We set $U^{i+1} \leftarrow U^i \cup \{v^*\}$, $i \leftarrow i + 1$ and the algorithm is restarted. If, on the contrary, all nodes in $V \setminus U^i$ could be inserted into $\mathcal{C}^i$, then an optimal clustering of $(V, D(V))$ has been found. The following pseudo-code illustrates our method:

Algorithm 1 Iterative clustering

Input: Problem $P = (V, D(V))$
Output: Optimal clustering $\mathcal{C}^V = \{C_j : j = 1, \ldots, k\}$ of P
  Build U by selecting κ points from V
  $\omega^U, \omega^V \leftarrow \infty$
  $\mathcal{C}^U, \mathcal{C}^V \leftarrow \emptyset$
  $W \leftarrow \emptyset$
  repeat
    $U \leftarrow U \cup W$
    $(\omega^U, \mathcal{C}^U) \leftarrow$ `OptimalClustering`$(U, D(U))$
    $(\omega^V, \mathcal{C}^V, W) \leftarrow$ `HeuristicCompletion`$(\mathcal{C}^U, V \setminus U)$
  until $W = \emptyset$
  return $\mathcal{C}^V$

In this algorithm, procedure `OptimalClustering`$(U, D(U))$ solves the clustering problem restricted to the nodes in U to optimality. It returns the optimal clustering $\mathcal{C}^U$ and its objective value ω^U. The function `HeuristicCompletion`$(\mathcal{C}^U, V \setminus U)$ completes the solution found by `OptimalClustering`$(U, D(U))$ and returns the resulting clustering $\mathcal{C}^V$ (not necessarily an optimal one), its value ω^V and a set $W = \{w\}$ containing a node that could not be inserted into $\mathcal{C}^U$ without augmenting its cost ($\emptyset$ if no such node exists).

The exactness and finiteness of our approach is based upon the following observation. At every iteration, the heuristic either proves the optimality of the current subproblem (supported by Proposition 1), or the set V is augmented. In the worst case, U will grow up to become equal to V, in which case procedure `OptimalClustering`$(V, D(V))$ is guaranteed to return an optimal solution for problem $(V, D(V))$. A simple worst-case time analysis of our method based upon this observation would suggest that it is indeed theoretically slower than solving the complete problem at once. In practice, our algorithm exploits the extremely large degeneracy of these DM problems and typically proves optimality by solving problems several orders of magnitude smaller than the original one.

5 Selection of an initial set U^0

The choice of the initial set U^0 in our algorithm is as follows. We first compute the centroid $\overline{u}$ of the points in V. If the i-th dimension of the problem is a numerical variable, we let $\overline{u}_i = \frac{1}{|V|} \sum_{u \in V} u_i$. On the contrary, if it is a categorical variable, we let $\overline{u}_i$ to be the median of the values $(u_i)_{u \in V}$.

We let then T be the set containing the $\rho = \min\{5000, |V|\}$ farthest points in V from $\overline{u}$. For each $p \in T$, we let T_p be built iteratively as follows: in step 1, $T_p = \{p\}$. From step 2 to κ, we insert into T_p the node $v \in V \setminus T_p$ whose minimum dissimilarity with respect to a node in T_p is maximum.

Following the ordering defined by the dissimilarity with respect to the centroid (the farthest first), we execute for T_p the following constructive heuristic: we sort the edges in $E_p = E(T_p)$ in non-decreasing order of dissimilarity, so as to favor the appearance of solutions with small diameters as early as possible during the process. For a given edge $e = \{u, v\}$ of dissimilarity d_e, we try to iteratively build clusters whose diameters do not exceed d_e, as follows. We begin with a single cluster containing the nodes u, v. Then, we always try to insert into the current cluster a node that will prevent the cluster from exceeding a diameter of d_e. If not possible, we create a new cluster unless the maximum number of clusters has been reached, in which case the current clustering is simply deemed unfeasible. If it is not possible to build a clustering of maximum diameter d_e, we continue to the next edge.

The solution for a given set T_p is then compared to the solutions obtained for previous sets. The set T_p that provides the maximum possible diameter is always retained. Indeed, it corresponds to the best possible estimate of the optimal clustering value. The algorithm is thus executed at most ρ times. Indeed, we abort the execution of the algorithm as soon as 100 different candidate sets have been considered without providing an increment of the objective value.

6 Optimal clustering

In this section we present a branch-and-price algorithm for the exact solution of the DMCP. The branch-and-price is based upon the solution of a set-covering formulation by column generation. Integrality is enforced through branch-and-bound.

6.1 Set-covering formulation of the DMCP

A feasible cluster can either be equal to a singleton $\{u\}$ or to a pair $(U, e) \in \mathcal{P} \times E$, where e represents one possible edge of maximum diameter within the nodes in U. The set of all feasible clusters containing two or more nodes is denoted by $\mathcal{T}$. For a cluster $t = (U, e) \in \mathcal{T}$ we denote $U_t = U, e_t = e$.

For every $u \in V$, we let s_u be a binary variable equal to 1 iff node u is clustered alone, and we let d_u be its diameter (as we will explain in Section 6.5, a node can have a strictly positive diameter following a branching decision). For every $t \in \mathcal{T}$ we consider a binary variable z_t that will take the value 1 iff the cluster t is retained. For every node $u \in V$ we let a_{ut} be a binary constant that takes the value 1 iff $u \in U_t$. Also, for every edge $e \in E$ we let b_{et} be a binary constant that takes the value 1 iff $e = e_t$. We finally define a continuous variable ω equal to the maximum diameter among the chosen clusters. The following set-covering formulation is valid for the DMCP:

$$\min_{s,\omega,z} \quad \omega \tag{6}$$

subject to

$$\omega - \sum_{t\in\mathcal{T}} b_{et} d_e z_t \geq 0 \qquad e \in E \tag{7}$$

$$\omega - d_u s_u \geq 0 \qquad u \in V \tag{8}$$

$$\sum_{t\in\mathcal{T}} a_{ut} z_t + s_u \geq 1 \qquad u \in V \tag{9}$$

$$\sum_{t\in\mathcal{T}} z_t + \sum_{u\in V} s_u \leq k \tag{10}$$

$$z_t \geq 0 \qquad t \in \mathcal{T} \tag{11}$$

$$z_t \text{ is integer} \qquad t \in \mathcal{T} \tag{12}$$

This formulation contains an exponentially large number of variables and cannot be explicitly solved even for small graphs. Column generation is a technique that allows the solution of a linear program with a large number of variables by adding them in a dynamic fashion. In our case, we consider the linear relaxation of formulation (6)–(12) by relaxing constraints (12). Primal feasibility is then enforced through branching.

6.2 Reduced costs

Let us assume that problem (6)–(11) has been solved restricted to a subset $\mathcal{T}' \subset \mathcal{T}$ of feasible clusters. We can extract dual variables $(\sigma_e)_{e\in E}, (\alpha_u)_{u\in V}, (\lambda_u)_{u\in V}$ and γ for constraints (7), (8), (9) and (10), respectively. The reduced cost of variables s_u, z_t are denoted as $\overline{c_u}, \overline{c_t}$, respectively, and are equal to

$$\overline{c_u} = d_u \alpha_u - \lambda_u - \gamma \tag{13}$$

$$\overline{c_t} = d_{e_t} \sigma_{e_t} - \sum_{u\in V} a_{ut} \lambda_u - \gamma \tag{14}$$

6.3 Pricing subproblem

The pricing sub-problem is performed in two steps. In the first step, we search for single-node clusters of negative reduced cost. This can be done by simple inspection by evaluating expression (13) for every possible $u \in V$.

If no such cluster exist, the second step of the pricing subproblem searches for a feasible cluster t of minimum reduced cost, this is such that expression (14) is minimized. We formulate this problem as an integer program, as follows. For every $u \in V$, we let x_u be a binary variable equal to 1 iff $u \in U_t$. For every edge $e \in E$ we let y_e be a binary variable equal to 1 iff $e = e_t$. We consider the following integer program:

$$\min_{x,y} \quad \phi \quad = \quad \sum_{e \in E} \sigma_e d_e y_e - \sum_{u \in V} \lambda_u x_u \tag{15}$$

subject to

$$\sum_{f \in E} d_f y_f - d_e(x_u + x_v - 1) \geq 0 \qquad e = \{u, v\} \in E \tag{16}$$

$$2y_e - x_u - x_v \leq 0 \qquad e = \{u, v\} \in E \tag{17}$$

$$\sum_{e \in E} y_e = 1 \tag{18}$$

$$x_u \in \{0, 1\} \qquad u \in V \tag{19}$$

$$y_e \in \{0, 1\} \qquad e \in E \tag{20}$$

Now, this pricing subproblem will select one edge $e \in E$ (associated to a variable $y_e = 1$) and construct a node subset accordingly (associated to the set $\{u \in V : x_u = 1\}$). We would like to highlight that, if the edge $e_t = \{u_t, v_t\}$ is chosen in advance (meaning that $y_{e_t} = 1$), one can *a priori* fix to zero all variables x_u such that $\max\{d_{uu_t}, d_{uv_t}\} > d_{e_t}$ and to one variables x_{u_t}, x_{v_t}. Let then $V^{e_t} = \{u \in V \backslash \{u_t, v_t\} : \max\{d_{uu_t}, d_{uv_t}\} \leq d_{e_t}\}$. The optimal set of nodes for a fixed e_t can be found by solving the following integer program:

$$\max_{x} \quad \phi'(e_t) \quad = \quad \sum_{u \in V^{e_t}} \lambda_u x_u \tag{21}$$

subject to

$$x_u + x_v \leq 1 \qquad u, v \in V^{e_t}, u < v, d_{uv} > d_{e_t} \tag{22}$$

$$x_u \in \{0, 1\} \qquad u \in V^{e_t} \tag{23}$$

The relationship between ϕ and $\phi'(\cdot)$ is as follows:

$$\phi = \min\{\sigma_e d_e - (\phi'(e) + \lambda_u + \lambda_v) : e = \{u, v\} \in E\} \tag{24}$$

6.4 Solution of the pricing subproblem

The solution of the pricing subproblem exploits the aforementioned property to derive an efficient heuristic and exact method. It relies on the sorting of the edges in E in increasing order according to the quantity $\sigma_e d_e - \lambda_u - \lambda_v$, with $e = \{u, v\} \in E$.

6.4.1 Greedy heuristic

Our greedy heuristic starts, for every edge $e = \{u, v\} \in E$, with a cluster containing the nodes u and v, and expanding it while possible.

Set $k \leftarrow 1$, and let $e_k = \{u_k, v_k\}$ be the k-th element of E. We let $C = \{u_k, v_k\}$, $S \leftarrow V^{e_k}$. Let us set $j \leftarrow 1$. At any iteration $j \geq 1$, C represents be the current cluster and S the set of nodes that can be inserted into C without exceeding the diameter given by d_{e_k}. At iteration $j+1$, we will set $v^* \leftarrow \arg\max\{\lambda_v : v \in S\}$. We then update C and S as follows: the new cluster becomes $C \leftarrow C \cup \{v^*\}$; the new set S becomes $S \leftarrow S \setminus \{v \in S : d_{vv^*} > d_{e_k}\}$. If S is empty, we check if the reduced cost associated to cluster (C, e_k) is negative. If so, stop and return (C, e_k). Otherwise, set $k \leftarrow k + 1$ and restart unless $k > |E|$, in which case the algorithm is stopped and no column is returned.

6.4.2 Exact algorithm

Let $\overline{w}$ be the linear relaxation value associated with the current set of columns. Let w^* be the current upper bound of the problem. We define the absolute gap as the difference $w^* - \overline{w}$ and denote it by `gap`.

Let $k \leftarrow 1$ and let $e_k = \{u_k, v_k\}$ be the next edge to inspect. Our exact algorithm computes a maximum weighted clique on the graph $G^k = (V^k, E^k)$, with $V^k = V^{e_k}$ and $E^k = \{e = \{u, v\} : e \in E, u, v \in V^k, d_e \leq d_{e_k}\}$. Each node $u \in V$ has a weight equal to λ_u. We compute $\phi'(e_k)$ using CLIQUER [24]. Let us denote $\overline{c_k} \leftarrow \sigma_{e_k} d_{e_k} - (\phi'(e_k) + \lambda_{u_k} + \lambda_{v_k}) - \gamma$. Depending in the value of $\overline{c_k}$, three possible cases arise:

1. If $\overline{c_k} < 0$, then a column (a feasible cluster) of negative reduced cost has been detected. Stop and return the associated cluster.
2. Else if $0 \leq \overline{c_k} <$ `gap`, then do nothing.
3. Else (i.e. if $\overline{c_k} \geq$ `gap`), then no column $t \in \mathcal{T}$ such that $e_t = e_k$ can be part of an optimal solution improving upon the incumbent. Therefore, edge e_k can be removed from the computation of ϕ in (24) without compromising the exactness of the algorithm.

In cases 2 and 3, we set $k \leftarrow k + 1$ and restart the procedure with the next available edge. If none, we abort the algorithm and return *nil*.

6.5 Branch-and-bound

Let $((\overline{s_u})_{u \in V}, (\overline{z_t})_{k \in \mathcal{T}})$ be the optimal solution at the end of the column generation process. Let us define, for every $e \in E$, $\overline{y_e} = \sum_{t \in \mathcal{T}} b_{eT} \overline{z_t}$. We also define, for every edge $\{u, v\} \in E$, $\overline{x_e} = \sum_{t \in \mathcal{T}} a_{ut} a_{bt} \overline{z_t}$. In addition, we define, for every node $u \in V$, $\overline{r_u} = \sum_{v \in V \setminus \{u\}} \overline{y_e}$. The solution might be declared unfeasible if $\overline{s_u}, \overline{y_e}, \overline{x_e}$ or $\overline{r_u}$ are fractional for some edge e or node u. Depending on the branching decision chosen, the two children problems are built as follows:

1. If $\overline{s_u} \in]0, 1[$ is the chosen branch: We create two children nodes, namely $s_u = 0$ and $s_u \geq 1$. For $s_e = 0$ we remove the variable s_u from the child master and from the computation of the reduced costs. For the branch $s_u \geq 1$, we simply add the inequality to the child node. As the variable is already in the current node master, there is no need to remove it and therefore no need to consider its dual in the computation of the reduced costs when using expression (13).
2. If $\overline{y_e} \in]0, 1[$ is the chosen branch: We create two children nodes, namely $y_e = 0$ and $y_e \geq 1$. For $y_e = 0$ we remove the edge e from the computation of ϕ in (24) and also remove all clusters $\{(U, e) \in \mathcal{T}'\}$ from the master problem. For the branch $y_e \geq 1$, we simply add the following inequality to the child node: $\sum_{t \in \mathcal{T}} b_{et} z_t \geq 1$. Its dual must be taken into account for the computation of the reduced costs.
3. If $\overline{x_e} \in]0, 1[$ is the chosen branch, with $e = \{u, v\}$: We create two children nodes, namely $x_e = 0$ and $x_e = 1$. For $x_e = 0$, we need to assure that u and v will never be on the same cluster. Thus, we remove from the master problem all clusters containing both nodes. In addition, we modify their dissimilarity to be $d_{uv} \leftarrow +\infty$. For the branch $x_e = 1$, we need to assure that every cluster containing u will also contain v. We proceed to *shrink* these two nodes into a single node $\{uv\}$, and update its diameter to $d_{\{uv\}} \leftarrow d_u + d_v$. Also, for every $w \in V \setminus \{u, v\}$, we let $d_{\{u,v\}w} \leftarrow \max\{d_{uw}, d_{vw}, d_{\{uv\}}\}$. We also remove from the master problem all clusters containing one of the nodes but not both.
4. If $\overline{r_u} \in]0, 1[$ is the chosen branch: We create two children nodes, namely $r_u = 0$ and $r_u \geq 1$. For $r_u = 0$, we remove from the master all clusters $(U, e) \in \mathcal{T}'$ such that one of the endpoints of e is equal to u. All edges $\{e = \{v, w\} \in E : v = u \text{ or } w = u\}$ are removed from the computation of ϕ in 24. For the branch $r_u \geq 1$, we simply add the following inequality to the child node: $\sum_{t \in \mathcal{T}} \sum_{\{e = \{u,v\} \in E : v \in V\}} b_{et} z_t \geq 1$.

We have implemented a hybrid branching strategy that performs strong branching for the first 1,000 node separations, and pseudo-cost branching afterwards. Some technical details for a proper implementation of this procedure are perhaps necessary, but for the sake of simplicity of the presentation we prefer to omit the details.

7 Completion heuristic

To prove the optimality of the solution associated to the current set U or that another iteration of the algorithm is necessary, we have implemented the following completion heuristic. Let $D(U), \mathcal{C}^U, \omega^U$ be the associated dissimilarity matrix, the optimal clustering of those nodes and the optimal solution value, respectively. For every node $u \in V \setminus U$ and for every cluster $C \in \mathcal{C}^U$, we let $\nu(u, C) = \max\{d_{uv} : v \in C\}$. For each $u \in V \setminus U$, the clusters are sorted in non-decreasing order of $\nu(u, C)$, this is $\nu(u, C^1(u)) \leq \nu(u, C^2(u)) \leq \cdots \leq \nu(u, C^k(u))$. In case of a tie among two or more clusters, we proceed as follows. For two clusters sharing the same value for $\nu(u, C^i(u))$, we consider the following tie-breaker quantity: $\nu(u, C)' = \max\{0, d(C) - \nu(u, C)\}$. Two clusters sharing the same value of $\nu(u, C)$ are sorted in non-decreasing order of $\nu(u, C)'$. Following ties are broken arbitrarily.

The next problem is to define an ordering of the nodes in $V \setminus U$. The intuition behind our approach is to sort the nodes so as to favor the detection of unfeasibilities in the early stages of the algorithm. The reason for doing so lies on the need of avoiding an exhaustive computation of the dissimilarity matrix. For every $u \in V \setminus U$, let $C^1(u)$ be the first cluster considered in the previously defined ordering for node u. We consider the following quantities:

$$l_1(u) = \max\{0, \omega^U - d(C^1(u) \cup \{u\})\} \tag{25}$$
$$l_2(u) = \max\{0, d(C^1(u) \cup \{u\}) - d(C^1(u))\} \tag{26}$$
$$l_3(u) = d(C^1(u) \cup \{u\}) \tag{27}$$
$$l_4(u) = d(C^1(u)) \tag{28}$$

The nodes are sorted in non-increasing order of l_1. Ties are broken by considering l_2, l_3 and l_4, in that order. If it is still not possible to break a tie, we consider the second cluster for each node, namely $C^2(\cdot)$ and apply the same reasoning. Further ties are broken using $C^3(\cdot), C^4(\cdot), \ldots, C^k(\cdot)$, in that order.

Let $i \leftarrow 1$. Let $u_i \in V \setminus U$ be the next node to be inserted according to the mentioned ordering. We try to insert u_i in some cluster using the ordering defined by the values $(\nu(u, C^l(u)))_{l=1}^k$. The node is inserted into the first cluster where it can be added without augmenting the value of the maximum diameter given by ω^U. If no such cluster exists, the algorithm stops. We check whether the insertion of node u_i becomes unfeasible because of another previously inserted node $u_j, j < i$. If no such u_j exists, we return $W = \{u_i\}$, otherwise we return $W = \{u_j, u_i\}$. If the insertion is, however, possible, we update the diameter of the cluster involved, let $i \leftarrow i + 1$ and restart, unless i reaches $|V \setminus U| + 1$ in which case the algorithm stops and the global optimal clustering is returned.

8 Acceleration techniques

To accelerate the whole procedure, some remarks are in order.

First, we perform a preprocessing of the nodes by sorting them in lexicographical order and then by removing duplicates. While for some problems this procedure did not help to reduce the number of points, for others the reduction reached a 70% of the nodes (*KDD cup 10%*, from 494,020 to only 145,583 nodes). In addition, we perform dimension reduction by removing dimensions whose standard deviations are equal to 0.

Second, we warm-start the branch-and-price procedure by computing lower and upper bounds for the problem, as follows: An upper bound can be obtained by executing the constructive heuristic introduced in Section 5. Moreover, we have implemented an *iterated local search* method to improve this solution that uses the LS presented in Fioruci et al. [13], with a perturbation based on the random selection of diameter nodes (two nodes that have maximum intra-cluster dissimilarity) and the reassignment of those nodes to randomly selected clusters. The value achieved by the ILS is then used as a cut-off value for the branch-and-price algorithm. The ILS, in all of our tests, consumed no more than a 5% of the total computing time. Also, the optimal solution from the previous iteration of the branch-and-price provides a lower bound on the value of

the optimal solution at the current iteration, thanks to Lemma 1. Let ω' be that lower bound. We enforce it by adding to the master problem the inequality

$$\sum_{t\in\mathcal{T}}\sum_{e\in E'} b_{et}z_t \geq 1, \tag{29}$$

where $E' = \{e \in E : d_e \geq \omega'\}$. Certainly, one needs to consider its dual variable in the computation of the reduced costs for the column generation process.

Third, we also provide a simple but effective parallelization of our algorithm. Indeed, we have realized that the bottleneck of our algorithm in most cases is the completion heuristic that takes time $O(n^2)$ as it needs to compute the dissimilarity between a node $u \in V \setminus U$ and all the nodes in a cluster to update its diameter. We simply divide this computation among the available cores.

9 A heuristic implementation of the algorithm

Our algorithm might fail into solving extremely large problems due mainly to the $O(n^2)$ time spent by the completion heuristic. However, we have realized that the completion heuristic takes this extremely long time only at the last iteration to prove optimality. Indeed, the preprocessing performed to sort the nodes before the execution of the heuristic is such that an unfeasible completion is usually detected after the insertion of a few nodes (less than 1,000 in most cases). Otherwise, the heuristic often succeeds to prove the optimality of the current solution. Moreover, when this happens, the sorting of the clusters for each node is such that most nodes can be effectively inserted into the first cluster according to this ordering. Thus, if one decides to abort the completion heuristic after, say, 1,000 successful node insertions and assign the remaining nodes to the first cluster according to the ordering defined in the preprocessing, the result is a heuristic that provides very near-optimal solutions.

10 Solution of other clustering problems

In this section we briefly describe how to adapt our algorithm to handle clustering problems using other criteria. Namely, we consider the minimum sum-of-diameters clustering problem (MSDCP), the split maximization clustering problem (SMCP) and the maximum sum-of-splits clustering problem (MSSCP). We have decided not to implement these modifications because of two main reasons: 1) in the case of the MSDCP, this criterion is known to favor the appearance of one extremely large cluster + several singletons or extremely small clusters, and is therefore not interesting in practice; 2) in the case of SMCP and MSSCP, these problems can already be solved in time $O(n^2)$. While our algorithmic approach may still be advantageous against this algorithm, as it avoids an explicit computation of the dissimilarity matrix, we think that the potential speed-up may not be significant to justify the use of our algorithm. We encourage, however, enthusiast researchers to prove us wrong or to support our claim with strong evidence.

For the case of the MSDCP, we propose to modify the objective function on problem (6)–(12) to

$$\min \quad \sum_{t\in\mathcal{T}} d(U_t)z_t. \tag{30}$$

In addition, one may adapt the constructive heuristic and ILS method to favor the minimization of this criterion. The correctness of the approach is supported by observing that Lemma 1 and Proposition 1 also hold for this criterion, as well as Hypothesis 1.

For SMCP and MSSCP, it does not make sense anymore to implement an implicit enumeration algorithm such as branch-and-bound to solve problems that can be solved in polynomial-time using the single-linkage algorithm [27]. Just as for the MSDCP, the constructive heuristic and ILS should also be adapted to favor the maximization of these criteria. In addition, it is easy to see that Lemma 1 and Proposition 1 also hold for these two criteria by replacing the maximization of the (minimum / sum-of-) splits by a minimization of the additive inverse of the (minimum / sum-of-) splits. Finally, it is easy to see that, due to the large degree of degeneracy usually found on these two problems, Hypothesis 1 is also likely to hold.

11 Computational results

The proposed algorithm has been programmed in C++ using the GNU g++ compiler v5.2. The general purpose linear programming solver used within the branch-and-price procedure is CPLEX 12.6. The algorithm has been compiled and executed on a machine powered by an Intel Core i7-4710HQ CPU @ 2.50GHz×8 with 16GB of RAM, running the Ubuntu 15.10 Operating System. The parallel implementation of our algorithm uses the OpenMP standard and is executed on the same machine using the 8 available cores.

In our experimental analysis, we consider some classical problems from clustering, classification and regression tasks. In Table 1 we provide the detail of the datasets considered in our computational study. The number of problems ranges from very small instances with 150 points up to very large-scale problems containing 581,012 observations. In this table, n, m, k and d stand for the number of points, the number of edges (computed as $n(n-1)/2$), the number of classes and the dimension of the problem, respectively. Column labeled *Ref* contains a reference to the source of the problem. The selected problems contain mixed numerical and categorical data. The dissimilarity between two points $u = (u_1, \ldots, u_d)$ and $v = (v_1, \ldots, v_d)$ is computed as:

$$d_{uv} = \sqrt{\sum_{i=1}^{d} f(u_i, v_i)}, \tag{31}$$

where $f(x, y)$ is equal to $(x-y)^2$ if x and y are two numerical values, equal to 1 if x and y are two categorical values whose string representations differ, and equal to 0 if x and y are two categorical values whose string representations are the same.

Table 1: Problems details

Problem	n	m	k	d	Ref
Iris	150	11,175	3	4	[23]
Wine	178	15,753	3	13	[23]
Glass	214	22,791	7	9	[23]
Ionosphere	351	61,425	2	34	[23]
User knowledge	403	81,003	4	5	[22]
Breast cancer	569	161,596	2	30	[23]
Synthetic control	600	179,700	6	60	[1]
Vehicle	846	357,435	4	18	[28]
Yeast	1,484	1,100,386	10	8	[23]
Mfeat (morph)	2,000	1,999,000	10	6	[9]
Multiple features	2,000	1,999,000	10	649	[23]
Segmentation	2,000	1,999,000	7	19	[9]
Image segm	2,310	2,666,895	7	19	[23]
Waveform (v1)	5,000	12,497,500	3	21	[23]
Waveform (v2)	5,000	12,497,500	3	40	[23]
Ailerons	13,750	94,524,375	10	41	[30]
Magic	19,020	180,870,690	2	10	[23]
Krkopt	28,056	393,555,540	17	6	[23]
Shuttle	58,000	1,681,971,000	7	9	[23]
Connect-4	67,557	2,281,940,346	3	42	[23]
SensIt (acoustic)	96,080	4,615,635,160	3	50	[11]
Twitter	140,707	9,899,159,571	2	77	[23]
Census	142,521	10,156,046,460	3	41	[23]
HAR	165,633	13,717,062,528	5	18	[31]
IJCNN1	191,681	18,370,707,040	2	22	[25]
Cod-Rna	488,565	119,347,635,330	2	8	[32]
KDD cup 10%	494,090	122,062,217,005	23	41	[23]
Cover type	581,012	168,787,181,566	7	54	[4]

To assess the efficiency of our method, we have designed three experiments. In the first experiment, summarized in Table 2, we consider a sequential implementation of our algorithm and compare it against Dao et al.'s CP algorithm [9], the RBBA of Brusco and Stahl [8] and the BB of Delattre and Hansen [10]. We restrict our analysis to the problems used in the experimental analysis of CP. As one can see from this

table, our algorithm takes two seconds or less to solve all the problems, including problem *Waveform (v2)* that passed from being solved in almost a minute, to only two seconds.

Table 2: Running times (in seconds) on small datasets

Problem	Opt	RBBA	BB	CP	IC
Iris	2.58	1.4	1.8	< 0.1	< 0.1
Wine	458.13	2.0	2.3	< 0.1	< 0.1
Glass	4.97	8.1	42.0	0.2	0.2
Ionosphere	8.6		0.6	0.3	0.2
User knowledge	1.17		3.7	0.2	1.2
Breast cancer	2,377.96		1.8	0.5	0.2
Synthetic control	109.36			1.6	0.4
Vehicle	264.83			0.9	0.2
Yeast	0.67			5.2	1.7
Mfeat (morph)	1,594.96			8.59	0.6
Segmentation	436.4			5.7	0.6
Waveform (v2)	15.58			50.1	2.0

Our second experiment compares the efficiency of the parallelization scheme used in the completion heuristic, against a sequential implementation of the algorithm. We have decided to omit from our analysis all problems that were solved in one minute or less by the sequential implementation of our algorithm. In Table 3 we describe the Wall times (in minutes) required to solve each problem using either approach, as well as the speed up provided by the paralllelization, computed as $t_{sequential}/t_{parallel}$, where $t_{sequential}, t_{parallel}$ are the reported Wall times. As one can see, the algorithm takes advantage of the parallelization, which can reduce the times by a factor of 1.84 on average with respect to the sequential implementation of the algorithm.

Table 3: Sequential vs Parallel implementations

Problem	Wall time (minutes)		Speed up
	Sequential	Parallel	
Multiple features	1.22	0.67	1.82
Shuttle	3.87	3.38	1.14
Connect-4	7.26	2.73	2.66
SensIt (acoustic)	19.46	13.14	1.48
Twitter	53.41	28.77	1.86
Census	95.20	33.95	2.80
HAR	35.96	19.25	1.87
IJCNN1	20.52	13.36	1.54
Cod-Rna	141.62	123.62	1.15
KDD Cup 10%	62.99	28.43	2.22
Cover type	280.60	162.94	1.72
Average			1.84

Our third experiment, summarized in Table 4 aims at reporting and analyzing the performance of our algorithm on very large datasets. We restrict our analysis to the parallel implementation of the algorithm, and to problems containing 5,000 nodes or more. In this table, label *it* represents the number of main iterations of our algorithm, this is the number of times that the completion heuristic needs to be executed. Label n' represents the number of nodes in the last iteration of the algorithm right before the final completion heuristic takes place. The time (in minutes) spent in the last completion heuristic is reported under column labeled *lch*. The total wall time spent by our algorithm (in minutes) is shown under column labeled *t*. Finally, we report in a separate column (as it is not part of the algorithm bur rather reported for analyses purposes) labeled *dmc*, the time (in minutes) needed to compute the whole dissimilarity matrix. In all cases, times lower than 0.1 minute (6 seconds) are reported as "< 0.1".

As shown in this table, our method is capable of solving all these problems within reasonable time limits. Typically, only very small graphs need to be handled by the exact clustering submethod at every iteration of

Table 4: Detailed results on iterative clustering

Problem	Opt	Iterative clustering				dmc
		it	n'	lch	t	
Waveform (v1)	13.74	10	21	< 0.1	< 0.1	< 0.1
Waveform (v2)	15.58	9	22	< 0.1	< 0.1	< 0.1
Ailerons	230.71	34	49	< 0.1	0.2	0.17
Magic	692.44	3	12	0.33	0.37	0.27
Krkopt	2.00	60	77	< 0.1	0.39	0.47
Shuttle	6,157.44	5	14	3.23	3.38	2.95
Connect-4	3.87	11	20	2.31	2.73	6.45
SensIt (acoustic)	4.47	6	15	12.72	13.14	12.16
Twitter	80,734	2	11	28.19	28.77	27.91
Census	100,056	3	13	33.27	33.95	33.00
HAR	1,078.73	8	18	18.70	19.25	24.76
IJCNN1	3.97	5	14	12.98	13.36	17.90
Cod-Rna	934.68	3	12	122.86	123.62	97.26
KDD cup 10%	144,165	26	53	25.50	28.43	23.71
Cover type	3,557.3	129	143	122.5	162.94	393.35

the algorithm. In addition, if the last completion heuristic is aborted and the remaining nodes are added into their closest cluster (according to a predefined ordering that is cheap to compute), the algorithm becomes a heuristic that finds near optimal solutions in a matter of minutes even for extremely large datasets. Finally, we would like to highlight that our algorithm's running time is comparable to the time needed to compute the dissimilarity matrix. Moreover, on some very large datasets, it proves even faster. This demonstrates that in the future, algorithms for DM and related criteria need to rely on the necessity of avoiding the pre-computation and storage of the dissimilarity matrix.

12 Algorithm limitations

Unfortunately, not everything is good news regarding the iterative clustering approach. In this section, we would like to report some of the difficulties that we have encountered and that we have not been able to satisfactorily deal with.

First, IC is not suitable, at least from a straightforward modification, to handle problems in which Hypothesis 1 does not hold. This is the case of, for instance, some important clustering criteria as the minimum sum-of-squares [2]. Whether our algorithmic approach can be generalized to handle these types of problems should be a matter of future research.

Another limitation of our algorithm is related to the handling of noise in problems with a large number of clusters. We have encountered this issue on problems *pendigits*, *letter*, *letter-recognition*, *birch1*, *birch2* and *birch3*. Our algorithm when applied to these problems showed poor convergence, which resulted in too many iterations of the main loop and thus in subproblems large enough to become untractable for our branch-and-price method. Whether this limitation can be overcome by using an alternate algorithm for optimal clustering (like the CP of [9], for instance) remains unknown and should certainly be the matter of future research.

At last, we would like to highlight an inherent limitation of our algorithm related to the complexity of *reading the data*. We have shown that our algorithm does not need to *a priori* compute the dissimilarity matrix explicitly. However, the running time of our algorithm is still comparable to the $O(n^2)$ time of performing this task. Whether it is possible to reduce this to some theoretical bound strictly lower than $O(n^2)$ is unknown.

13 Conclusion

We have presented a novel algorithmic framework for the solution of very large-scale DM clustering problems. Our algorithm outperforms the state-of-the-art algorithms for this problem by solving datasets that are two orders of magnitude larger than those handled by previous algorithms. Moreover, we show how to adapt our algorithm to derive a fast and efficient heuristic that consumes only a fraction of the time and provides in many cases the optimal solution. In addition, we show that our algorithmic framework can also be generalized to solve other closely related clustering problems, such as the sum-of-diameters clustering problem, the split maximization clustering problem and the maximum sum-of-splits clustering problem. We can identify three potential avenues for future research: 1) to adapt this algorithmic framework to solve clustering problems where Hypothesis 1 does not hold; 2) to work on the robustness of the algorithm to allow the solution of problems with noise; and 3) to assess the efficiency of the heuristic implementation of our algorithm to solve arbitrarily large problems, as for those encountered on online applications in which large amounts of data arrive in streams.

References

[1] Alcock, R., Manolopoulos, Y.: Time-series similarity queries employing a feature-based approach. In: 7th Hellenic Conference on Informatics (1999)

[2] Aloise, D., Hansen, P., Liberti, L.: An improved column generation algorithm for minimum sum-of-squares clustering. Mathematical Programming 131(1–2), 195–220 (2012). http://dx.doi.org/10.1007/s10107-010-0349-7

[3] Alpert, C.J., Kahng, A.B.: Splitting an ordering into a partition to minimize diameter. Journal of Classification 14(1), 51–74 (1997)

[4] Blackard, J.A.: Comparison of neural networks and discriminant analysis in predicting forest cover types. Ph.D. thesis (1998)

[5] Bradley, P.S., Fayyad, U.M., Reina, C.: Scaling clustering algorithms to large databases. In: KDD, 9–15 (1998)

[6] Brucker, P.: On the complexity of clustering problems. In: Optimization and operations research, 45–54. Springer (1978)

[7] Brusco, M.J., Dennis Cradit, J.: Bicriterion methods for partitioning dissimilarity matrices. British Journal of Mathematical and Statistical Psychology 58(2), 319–332 (2005)

[8] Brusco, M.J., Stahl, S.: Branch-and-bound applications in combinatorial data analysis. Springer Science & Business Media (2006)

[9] Dao, T.B.H., Duong, K.C., Vrain, C.: Constrained clustering by constraint programming. Artificial Intelligence (in press)

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

[11] Duarte, M., Hu, Y.H.: Vehicle classification in distributed sensor networks. Journal of Parallel and Distributed Computing 64, 826–838 (2004)

[12] Feder, T., Greene, D.: Optimal algorithms for approximate clustering. In: Proceedings of the twentieth annual ACM symposium on Theory of computing, 434–444. ACM (1988)

[13] Fioruci, J.A.A., Toledo, F.M., Nascimento, M.A.C.V.: Heuristics for minimizing the maximum within-clusters distance. Pesquisa Operacional 32, 497–522 (2012)

[14] Fraley, C., Raftery, A., Wehrens, R.: Incremental model-based clustering for large datasets with small clusters. Journal of Computational and Graphical Statistics 14(3), 529–546 (2005). http://dx.doi.org/10.1198/106186005X59603

[15] Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to NP-completeness. WH Freeman New York (1979)

[16] Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38, 293–306 (1985)

[17] Hansen, P., Delattre, M.: Complete-link cluster analysis by graph coloring. Journal of the American Statistical Association 73(362), 397–403 (1978)

[18] Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. Mathematical programming 79(1–3), 191–215 (1997)

[19] Hochbaum, D.S.: When are NP-hard location problems easy? Annals of Operations Research 1(3), 201–214 (1984)

[20] Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Mathematics of operations research 10(2), 180–184 (1985)

[21] Johnson, S.C.: Hierarchical clustering schemes. Psychometrika 32(3), 241–254 (1967)

[22] Kahraman, H.T., Sagiroglu, S., Colak, I.: Developing intuitive knowledge classifier and modeling of users' domain dependent data in web. Knowledge Based Systems 37, 283–295 (2013)

[23] Lichman, M.: UCI machine learning repository (2013). http://archive.ics.uci.edu/ml

[24] Östergård, P.R.: A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120(1), 197–207 (2002)

[25] Prokhorov, D.: IJCNN 2001 neural network competition (2001)

[26] Rao, M.: Cluster analysis and mathematical programming. Journal of the American statistical association 66(335), 622–626 (1971)

[27] Sibson, R.: SLINK: an opoptimal efficient algorithm for the single-link cluster method. The Computer Journal 16, 30–34 (1973)

[28] Siebert, J.P.: Vehicle recognition using rule based methods. Research Memorandum TIRM-87-018, Turing Institute (1987)

[29] Sørensen, T.: A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. Biol. Skr. 5, 1–34 (1948)

[30] Torgo, L.: Regression datasets (2009). http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[31] Ugulino, W., Cardador, D., Vega, K., Velloso, E., Milidiu, R., Fuks, H.: Wearable computing: Accelerometers' data classification of body postures and movements. In: Proceedings of 21st Brazilian Symposium on Artificial Intelligence, Lecture Notes in Computer Science, 52–61. Springer Berlin/Heidelberg (2012)

[32] Uzilov, A.V., Keegan, J.M., Mathews, D.H.: Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. BMC Bioinformatics 7 (2006)

[33] Zhang, T., Ramakrishnan, R., Livny, M.: Birch: A new data clustering algorithm and its applications. Data Mining and Knowledge Discovery 1(2), 141–182 (1997). http://dx.doi.org/10.1023/A%3A1009783824328