Speeding-Up Hierarchical Agglomerative Clustering in Presence of Expensive Metrics

Mirco Nanni

ISTI-CNR, Pisa, Italy
mirco.nanni@isti.cnr.it

Abstract. In several contexts and domains, hierarchical agglomerative clustering (HAC) offers best-quality results, but at the price of a high complexity which reduces the size of datasets which can be handled. In some contexts, in particular, computing distances between objects is the most expensive task. In this paper we propose a pruning heuristics aimed at improving performances in these cases, which is well integrated in all the phases of the HAC process and can be applied to two HAC variants: single-linkage and complete-linkage. After describing the method, we provide some theoretical evidence of its pruning power, followed by an empirical study of its effectiveness over different data domains, with a special focus on dimensionality issues.

1 Introduction

In several domains, hierarchical agglomerative clustering algorithms are able to yield best-quality results. However, this class of algorithms is characterized by a high complexity which reduces the size of datasets which can be handled. In the most standard cases, such complexity is $O(dN^2+N^2 \log N)$, N being the number of objects in the dataset and d the cost of computing the distance between two objects, which is the result of $O(N^2)$ distance computations followed by $O(N^2)$ selection steps, each having cost $O(\log N)$. In typical settings, d is either a constant or very small w.r.t. $\log N$, so that the algorithm complexity is usually simplified to $O(N^2 \log N)$.

In some contexts, however, computing distances can be a very expensive task, such as in the case of high-dimensional data or complex comparison functions, e.g., the edit distance between long strings. In these cases, the computation of all object-to-object distances dominates the overall cost of the clustering process, and so any attempt to improve performances should aim at saving a significant portion of distance computations. To the best of our knowledge, this aspect has not been explicitly studied in literature, yet, despite the fact that it has been marginally mentioned in several works (e.g., most of those described in Section 2.2).

In this work, we will consider two popular instances of the general hierarchical agglomerative algorithms family, namely the single- and complete-linkage versions, and propose a simple pruning strategy that improves their performances

T.B. Ho, D. Cheung, and H. Liu (Eds.): PAKDD 2005, LNAI 3518, pp. 378–387, 2005.

by reducing the number of object-to-object distances to compute without affecting the results. A formal proof of its effectiveness under some assumptions will also be given, together with an extensive experimental session to test it on different contexts and conditions.

2 Background and Related Work

In this section we will provide a short description of the general hierarchical agglomerative clustering schema, instantiating it to the two specific cases discussed in this paper. Finally, a brief summary of related work will follow.

2.1 Hierarchical Agglomerative Clustering (HAC)

The objective of hierarchical clustering algorithms is to extract a multi-level partitioning of data, i.e., a partitioning which groups data into a set of clusters and then, recursively, partitions them into smaller sub-clusters, until some stop criteria are satisfied [3]. Hierarchical algorithms can be divided in two main categories: agglomerative and divisive. Agglomerative algorithms start with several clusters containing only one object, and iteratively two clusters are chosen and merged to form one larger cluster. The process is repeated until only one large cluster is left, that contains all objects. Divisive algorithms work in the symmetrical way. In this paper we will focus on the former class of algorithms.

Algorithm: Hierarchical Agglomerative Clustering
Input: a dataset D
Output: a tree structure T

1. $C := \{\{o\} |\ o \in D\}$ and $T = \emptyset$;
2. **while** $|C| > 1$ **do**
3. Select best couple (a, b) s.t. $a, b \in C$;
4. Create a new cluster $c = a \cup b$, and let a and b be children of c in T;
5. $C := C \cup \{c\} \setminus \{a, b\}$;
6. **foreach** $x \in C$ **do**
7. Compute the distance between x and c;
8. **return** T;

Fig. 1. General schema for hierarchical agglomerative clustering algorithms

The general structure of an agglomerative clustering algorithm can be summarized as in Figure 1. As we can notice, there are two key operations in the general schema which still need to be instantiated: the choice of the *best* couple of clusters, and the computation of the distances between the new cluster and the existing ones. Each different instantiation of these two operations results into a different agglomerative clustering algorithm. In this paper, the cluster

selection in step 3 is performed by selecting the closest pair of clusters, while the distance computation in step 7 is performed in two alternative ways: by extracting the distance between the closest pair of objects (excluding couples belonging to the same cluster), which yields a so called *Single-linkage* algorithm; and by extracting the distance between the farthest pair of objects, which yields a *Complete-linkage* algorithm. In particular, the complete-linkage algorithm in general produces tightly bound or compact clusters, while the single-link algorithm, on the contrary, suffers from a *chaining effect*, i.e., it has a tendency to produce clusters that are straggly or elongated [3].

2.2 Related Work

Efficiency is a strong issue in hierarchical clustering, and it has been treated in literature in many different ways. In the following we summarize some of the main approaches to the problem.

Some approaches seek slight computational complexity improvements for the HAC problem. For example, [2] introduces a data structure for dynamic closest pair retrieval, which is directly applicable to hierarchical clustering, and which is shown to reach a $O(n^2)$ complexity for simple aggregation operators (e.g., maximum, minimum and average). For specific contexts, even faster solutions have been proposed, such as a sub-quadratic single-linkage method for low-dimensional data [4], and a $O(n \log n)$ complete-linkage solution for $\mathcal{R}^d$ spaces ($d \geq 1$) with L_1 and L_∞ metrics [5]. We remark that these approaches do not take into account the (pragmatic) possibility of having very expensive distance computations, which is exactly the context we will focus on in this paper. When some degree of approximation in the hierarchical clustering structure can be tolerated, several approximation approaches can be followed, which mainly try to reduce the size of data: from data simple sampling methods to data aggregation solutions, such as (i) *grid-based* clustering solutions for vectorial datasets [3], and (ii) the *data bubbles* approach [1], which extends the grid-based approach to non-vectorial data.

3 HAC with Enhanced Distance Management

The basic assumption of our method is that our distance function is a metric. Then, the key idea is that from the exact distances of a limited number of couples it is possible to derive useful approximated values for all object-to-object distances. Such approximations can be easily updated at each iteration of the HAC algorithm, and can be used to effectively limit the number of exact distance computations needed along the whole process.

3.1 Distance Approximations

As basic means for estimating unknown distances, we propose to use the triangular inequality, a property satisfied by all metrics: $\forall a, b, p \in D : \; d(a, b) \leq$

$d(a,p)+d(p,c)$, where d is a metric defined over a domain D. With some simple math and exploiting the symmetry property of metrics, we can rewrite it as

$$\forall a,b,p \in D: \ |d(p,a) - d(p,c)| \leq d(a,b) \leq d(p,a) + d(p,c) \tag{1}$$

Now, assuming to know all $|D|$ distances $d(p,a)$ for some fixed element $p \in D$, which we will call *pivot*, the above formula can be directly used to provide a bounding interval for the distance between any couple (a,b) of objects. Henceforth, we will refer to such bounding intervals as *approximated distances* or simply *approximations*. In particular, we notice that if some object a is very close to the pivot, the $d(p,a)$ values in (1) will be very small, and therefore the approximation of any distance $d(a,b)$ from a will be very tight.

In our approach, the computation of all object-to-object distances, usually performed at the beginning of HAC algorithms, is replaced by (i) the computation of the $|D|$ exact distances relative to a randomly chosen pivot, and (ii) the approximation of all other distances by following the method outlined above.

3.2 Enhanced Distance Management

The method shown in the previous section can be used to provide an initial set of approximations aimed at replacing as much as possible the full matrix of distances. In the following we will describe: (i) how such approximations can be used to save exact distance computations in the couple selection phase (step 3 in Figure 1); (ii) how they can be composed to derive approximations for a newly created cluster (steps 6–7); and (iii) how to exploit them also in the *on demand* computation of exact distances between compound clusters, when they are required in the couple selection phase.

Enhanced Couple Selection. Both the single- and complete-linkage algorithms, at each iteration find the couple of clusters with minimal distance, and merge them. A simple method for searching such couple exploiting the approximated distances, is the following:

1. Select the couple (a,b) which has the lowest-bounded approximation;
2. **if** the approximation is perfect
3. **then** return (a,b);
4. **else** compute the exact $d(a,b)$ and return to step 1;

Essentially, a two-steps selection is performed: a first selection of the most promising candidate couple is performed by means of the known approximations; if the *best* approximation is perfect, then all other couples certainly have an equal or greater distance, and therefore we can safely choose the selected couple for the merging phase; otherwise, another step is necessary, i.e., the exact distance of the couple needs to be computed and checked to be still the best candidate. The last test is implicitly performed by immediately repeating the selection step.

Deriving New Approximations. When two clusters are merged, all distances from the resulting new cluster have to be computed, exact or approximated, so that it can be considered in the next iterations of the selection-merging process. Analogously to the case of exact distances, the approximations for the new cluster can be derived by aggregating the already known approximations of the two clusters it originated from. In particular, we exploit a simple property of the max and min aggregation operators, that are used in the single- and complete-linkage HAC algorithms[1]:

Proposition 1. *Let* $x, y, l_1, u_1, l_2, u_2 \in \mathcal{R}$, $x \in [l_1, u_1]$ *and* $y \in [l_2, u_2]$. *Then:*

$$\min\{x, y\} \in [\min\{l_1, l_2\}, \min\{u_1, u_2\}] \tag{2}$$

$$\max\{x, y\} \in [\max\{l_1, l_2\}, \max\{u_1, u_2\}] \tag{3}$$

In the single-linkage algorithm, the distance between two clusters c and c' is computed as the minimum of the object-to-object distances between elements of the two clusters, i.e., $d(c, c') = \min_{a \in c, b \in c'} d(a, b)$. If c is obtained by merging clusters c_1 and c_2, then we can write $d(c, c') = \min_{a \in c_1 \cup c_2, b \in c'} d(a, b)$, and therefore $d(c, c') = \min\{d(c_1, c'), d(c_2, c')\}$. This property, together with (2), provides a straightforward means for approximating all distances $d(c, c')$ from c, given that we know an approximation for both its components c_1 and c_2. A completely symmetrical reasoning can be repeated for the complete-linkage algorithm, which makes use of inequality (3).

Enhanced Distance Computation. In the (enhanced) selection step it is often necessary to compute the exact distance between two clusters. That happens whenever the best candidate couple found is associated with only an approximated distance. The trivial way to do it, consists in computing all distances between each object in the first cluster and each object in the second one and aggregating them with the proper operator (min or max). An obvious drawback of this solution is that it easily leads to compute all $\frac{|D| \cdot (|D|-1)}{2}$ object-to-object distances, which is exactly what we wanted to avoid. A surprisingly effective enhancement can be obtained by exploiting the following simple fact:

Proposition 2. *Let* c_1, c_2, c' *be three distinct clusters and* $c = c_1 \cup c_2$, $d(c_1, c') \in [l_1, u_1]$, $d(c_2, c') \in [l_2, u_2]$. *If* $u_1 \leq l_2$, *then: (i) in the single-linkage algorithm it holds that* $d(c, c') = d(c_1, c')$, *and (ii) in the complete-linkage* $d(c, c') = d(c_2, c')$.

The basic idea is to compute the distance between compound clusters by recursively analyzing their components (i.e., the two sub-clusters they originated from), until we reach simple objects. At each step of the recursion, the above property allows to prune unnecessary distance computations. The process for single-linkage HAC can be summarized as in Figure 2. If the clusters to compare

[1] Due to space limits, all the proofs are omitted here, and can be found in [8].

Algorithm: EDC
Input: two clusters a and b
Output: the exact distance $d(a, b)$

1. **if** a and b contain only one object **then** Stop and **return** $d(a, b)$;
2. **if** a contains only one object **then** Swap a and b;
3. Let a_1, a_2 be the clusters which compose a, i.e., $a = a_1 \cup a_2$;
4. Let $d(a_1, b) \in [l_1, u_1]$ and $d(a_2, b) \in [l_2, u_2]$;
5. **if** $l_1 > l_2$ **then** Swap a_1 and a_2;
6. $d_1 := EDC(a_1, b)$;
7. **if** $d_1 < l_2$ **then** Stop and **return** d_1;
8. $d_2 := EDC(a_2, b)$;
9. **return** $\min\{d_1, d_2\}$;

Fig. 2. Enhanced Distance Computation (EDC) for single-linkage HAC

contain single objects, then the algorithm simply computes their distance (step 1), otherwise it breaks down one of the compound clusters into its components (steps 2–4), and recursively analyzes them. In the analysis of sub-components, priority is given to the *most promising* one, i.e., that with the smaller lower bound distance (step 5), to the purpose of maximizing the pruning opportunities offered by Proposition 2. Step 7 implements that by avoiding to compute the distance for the *less promising* component when it is not strictly necessary.

The complete-linkage version of the algorithm is essentially the same, and can be obtained by just modifying the conditions of step 5 and 7 with, respectively, $(u_1 < u_2)$ and $(d_1 > u_2)$, and by replacing min with max in step 9.

3.3 Selecting Pivots

As we noticed in Section 3.1, the approximations computed before the clustering process can have variable tightness. In particular, the approximations for objects close to the *pivot* will be tight, while the others will be looser. A natural extension of the method consists in choosing more than one pivot, so that a larger number of objects will have a pivot near to them, and therefore a larger quantity of approximated distances will result tight. The expected consequence is that the pruning strategies described in the previous sections will be more effective.

Choosing several pivots, we obtain several approximations for the same distance – one for each pivot – so they need to be composed together in some way. The approximation computed by means of each pivot represents a constraint that the real distance must satisfy. Therefore, the composition of approximations corresponds to the conjunction of the constraints they represent, which is simply implemented by intersecting of the available approximations.

A more difficult problem is the choice of the pivots. While a simple, repeated random choice would be a possible solution, it provides no guarantee on the results. On the contrary, assuming that a dataset is really composed of a number

Algorithm: Pivots Selection
Input: a dataset D and an integer n
Output: a set P of n pivots

1. Randomly select an object $p_0 \in D$;
2. $P := \{p_0\}$;
3. **while** $|P| < n$ **do**
4. $\quad p = \arg\max_{o \in D}\{\min_{p' \in P} d(p', o)\}$;
5. $\quad P := P \cup \{p\}$;
6. **return** P;

Fig. 3. Algorithm for selecting the initial pivots

of clusters, an optimal choice for pivots would assign at least one pivot to each cluster. The key idea of our pivot selection heuristics is the following: assuming to have very well defined clusters in our data, each point is expected to be far from the objects of other clusters, at least if compared with the distance from other objects in the same cluster. Therefore, given a set of pivots, we can reasonably search a new good pivot, i.e., a pivot which belongs to an *uncovered* cluster, among those objects which are far from all existing pivots. These are essentially the same ideas applied in [6], where a similar approach has been used for approximated clustering. Figure 3 shows our pivot selection method.

The very first pivot is chosen randomly (steps 1–2), while the following ones are chosen as mentioned above. In particular, the *furthest* object from the existing set of pivots is selected, i.e., the object which maximizes the distance from the closest pivot (step 4). This simple algorithm seems to capture reasonably well the cluster structure of data, at least for clean-cut clusters, as indicated by the property proven below.

Definition 1 (δ-separateness). *Given a set of objects D and a distance $d()$, D is called* δ-separated *if it can be split into at least two clusters, such that the following holds: $\forall a, b, a', b' \in D$: if a and b belong to the same cluster while a' and b' do not, then $d(a', b') > \delta \cdot d(a, b)$.*

Essentially, δ-separateness requires that the minimum distance between clusters is at least δ times larger than the maximum diameter of clusters.

Proposition 3. *Let D be a 1-separated dataset composed of n clusters, and $k \geq n$. Then, PivotsSelection(D,k) returns at least one object from each cluster.*

4 Performance Evaluation

In this section we provide some experimental and theoretical evaluations of the performances of the HAC algorithms with enhanced distance management described in this work.

4.1 Theoretical Evaluation

While any realistic context usually shows some kind of irregularity, such as noise (i.e., objects that do not clearly belong to any cluster) and dispersion (i.e., largely dispersed clusters, possibly without clear boundaries), it is useful to have some theoretical estimation of performances also on ideal datasets: on one hand, it provides at least a comparison reference for empirical studies; on the other hand, it helps to understand where are the weak and strong points of the algorithm analyzed. In this section, we provide one of such theoretical hints.

First of all, we introduce a slight variant of HAC algorithms:

Definition 2 (k-HAC). *Given a HAC algorithm and a parameter k, we define the corresponding k-HAC algorithm as its variant which stops the aggregation process when k clusters are obtained. That corresponds to replace step 2 in the general HAC algorithm (Figure 1) with the following:* **while** $|C| > k$ **do**.

In practice, such generalization is quite reasonable, since usually it is easy to provide some a priori lower bound on the number of clusters we are interested in – obviously at least 2, but often it is much larger.

Proposition 4. *Given a 3-separated dataset D with n clusters, and a parameter $k \geq n$, the execution of an optimized k-HAC algorithm over D with k initial pivots requires $O(N_1^2 + \cdots + N_k^2)$ object-to-object distance computations, where $(N_i)_{i=1,\ldots,k}$ are the sizes of the k top level clusters returned by the algorithm.*

In summary, when clusters are very compact our pruning strategy allows to limit the distance computations just to couples within the same cluster. That results in a considerable reduction factor, as stated by the following:

Corollary 1. *Under the assumptions of Proposition 4, if $k = n$ and the clusters in D have balanced sizes (i.e., $\forall i : N_i \sim N/k$), then the k-HAC algorithm with enhanced distance computation requires a fraction $O(1/k)$ of the distances required by the simple HAC algorithm.*

We notice that the above analysis does not take in consideration the pruning capabilities of the Enhanced Distance Computation algorithm. As the next section will show, in some cases this second component allows to obtain much larger reduction factors.

4.2 Experimental Evaluation

In order to study the effectiveness of our pruning heuristics, and to understand which factors can affect it, we performed several experiments over datasets of different nature with corresponding distance functions:

- 2D points: the dataset contains points in the $\mathcal{R}^2$ space, and the standard Euclidean distance is applied. Data were randomly generated into 10 spherical, normally-distributed clusters with a 5% of random noise. Although Euclidean metrics are not expensive, this kind of metric space provides a good example of low-dimensional data, so it is useful to evaluate the pruning power of our heuristics on it and to compare the results with the other data types.

– Trajectories: each element describes the movement of an object in a bi-dimensional space, and is represented as a sequence of points in space-time. The distance between two objects is defined as the average Euclidean distance between them. Data were synthesized by means of a random generator, which created 10 clusters of trajectories (see [7] for the details).

For each data domain, datasets of different sizes were generated (from 400 to 3200 objects) and the single- and complete-linkage versions of a 10-HAC algorithm were applied, with a variable number of pivots (from 4 to 48). Figure 4 depicts the results of our experiments for single-linkage, which are evaluated by means of the ratio between the total number of distance computations required by the basic HAC algorithms and the number of distances computed by their enhanced version. We will refer to such ratio as *gain factor*, and each value is averaged over 16 runs. Due to space limitations, the results for the complete-linkage algorithm are not reported here, since they are quite similar to the single-linkage case. The interested reader can find them in [8], together with tests on other datasets. We can summarize the results as it follows:

– For 2D data (Figure 4 left), a very high gain factor is obtained for all settings of the parameters. In particular, the gain factor grows very quickly with the size of the database, and the best results are obtained with the minimal number of pivots. The latter fact essentially means that the pruning power of the EDC procedure (Figure 2) is so high in this context, that only a very small number of exact distances are needed to capture the structure of data, and so the $k|D|$ distances computed in the initialization phase ($|D|$ for each pivot) become a limitation to the performances.
– For trajectory data (Figure 4 right), the gain factor is moderately high, and the enhanced HAC algorithms reduce the number of computed distances of around an order of magnitude. In this contexts, we notice that the best results are obtained with a number of pivots around 10–20, and both smaller and higher values yield a decrease in performances.

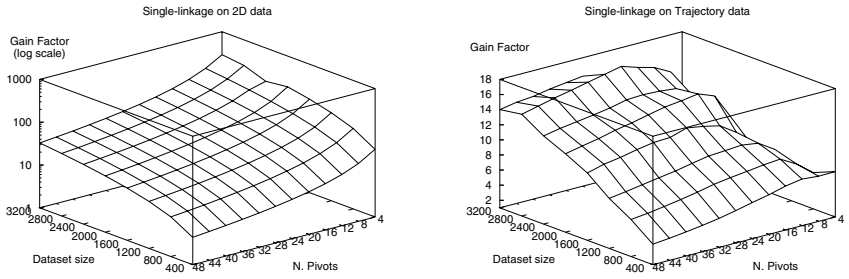

Fig. 4. Gain Factor of Single-linkage HAC on 2D and trajectory data

Due to space limitations, no analysis of execution times is provided here. We summarize our results as follows: with 2D data the gain in running times is slightly negative, because the Euclidean metric is extremely cheap, and then, even though the overhead of our heuristics results to be small, the distances saved cannot compensate it; with other data, instead, the gain in running times is almost identical to the gain factor, since the distances are more complex, and saving even a few of them is enough to balance all the overhead.

5 Conclusions

In this paper we introduced an optimization technique for two popular hierarchical clustering algorithms, and studied its potentialities and its limitations by means of both theoretical and empirical means. Our optimization technique tries to save as many distance computations as possible, which is particularly important for contexts where distances are time-consuming, and we showed that on reasonably dense datasets it is able to achieve good performances.

As future work, we plan (i) to perform a systematic study aimed at understanding more precisely which statistical properties of data influence the performances of our pruning heuristics (as suggested in the previous section and confirmed by additional tests in [8], dimensionality is one of them); (ii) to empirically evaluate the pruning power of the heuristics over several real world datasets, having different characteristics; and, finally, (iii) to extend the heuristics to other variants of HAC and, if possible, to other clustering approaches.

References

1. M. M. Breunig, H.-P. Kriegel, P. Krüger, and J. Sander. Data bubbles: quality preserving performance boosting for hierarchical clustering. In *SIGMOD '01: Proc. of the 2001 ACM SIGMOD Int' Conf. on Management of data*, pages 79–90, 2001.
2. David Eppstein. Fast hiearchical clustering and other applications of dynamic closet pairs. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 619–628, 1998.
3. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
4. D. Krznaric and C. Levcopoulos. The first subquadratic algorithm for complete linkage clustering. In *ISAAC '95: Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 392–401. Springer-Verlag, 1995.
5. D. Krznaric and C. Levcopoulos. Optimal algorithms for complete linkage clustering in d dimensions. *Theor. Comput. Sci.*, 286(1):139–149, 2002.
6. R. R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1–3):35–60, 2004.
7. M. Nanni. *Clustering methods for spatio-temporal data*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2002.
8. M. Nanni. Hierarchical clustering in presence of expensive metrics. Technical report, ISTI-CNR, 2005. `http://ercolino.isti.cnr.it/mirco/papers.html`.