2015 IEEE Symposium Series on Computational Intelligence

Scalable Hierarchical Clustering: Twister Tries with a Posteriori Trie Elimination

Michael Cochez*, and Ferrante Neri†*
*Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
Email: {miselico@jyu.fi, fneri@dmu.ac.uk}
†Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University,
The Gateway, Leicester LE1 9BH, England, United Kingdom

***Abstract*—Exact methods for Agglomerative Hierarchical Clustering (AHC) with average linkage do not scale well when the number of items to be clustered is large. The best known algorithms are characterized by quadratic complexity. This is a generally accepted fact and cannot be improved without using specifics of certain metric spaces. Twister tries is an algorithm that produces a dendrogram (i.e, outcome of a hierarchical clustering) which resembles the one produced by AHC, while only needing linear space and time. However, twister tries are sensitive to rare, but still possible, hash evaluations. These might have a disastrous effect on the final outcome. We propose the use of a metaheuristic algorithm to overcome this sensitivity and show how approximate computations of dendrogram quality can help to evaluate the heuristic within reasonable time. The proposed metaheuristic is based on an evolutionary framework and integrates a surrogate model of the fitness within it to enhance the algorithmic performance in terms of computational time.**

I. Introduction

Clustering is used in a wide variety of domains and is often one of the first unsupervised learning tasks introduced in many data mining courses. The overall task is to place items into clusters such that items which are similar are placed in the same cluster, while items which have a low similarity are placed in different clusters. One way to perform this task is so-called hierarchical clustering. As opposed to others, the outcome will not just be a partition of the items, but a dendrogram (see fig. 1), showing a nested clustering.

The drawback of hierarchical clustering is that most algorithm scale poorly in the number of items clustered. Hence, for the increasing data sizes of today's application domains, more scalable algorithms are needed. The scalable approach which we will enhance further in this paper is called *twister tries*, which was proposed by Cochez and Mou [1]. This algorithm produces an approximate dendrogram and poses linear time and space requirements.

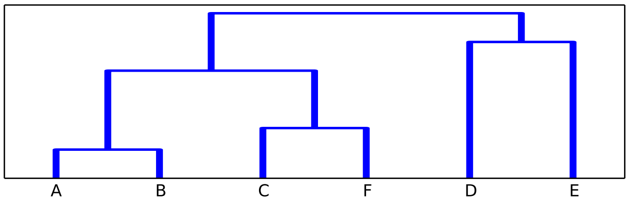

Fig. 1. An example of a dendrogram.

978-1-4799-7560-0/15/$31 ©2015 IEEE

In the core of twister tries there is a collection of tries. Encoded in these tries are the outcomes of specific locality-sensitive [2] hash functions. After this hashing stage, there is the twisting stage in which the information in the tries is converted into a dendrogram, which is the final outcome of the algorithm. The locality-sensitive hash functions are chosen such that they will usually improve the quality of the final dendrogram. However, in a rare occasion it can happen that the outcomes of some of these functions has a devastating effect on the clustering.

In this paper we explore the idea of solving this issue after the fact. We first perform the hashing step of the twister tries algorithm and then attempt to find out which hash outcomes to ignore in order to improve the quality of the dendrogram. Note that it is important that this a posteriori correction is performed fast and with low space requirements. If the correction would be slow or consume much memory, it would be better not to use an approximation in the first place and tackle the whole clustering problem exactly.

The main contributions of this paper are: *1*) Introduction and comparison of approximate ways to evaluate a dendrogram (see section IV-C and tables I and II); *2*) A better clustering of two datasets, demonstrating an improvement over the existing twister tries algorithm with reasonable time and space overhead (see section VI and in particular figs. 3 and 4). A further minor point of interest is theorem 1 about the computational complexity of the Joining Distance Ratio which was introduced by Kull and Vilo [3].

The article is structured as follows: in the next sections we introduce agglomerative hierarchical clustering and the twister tries algorithm. Then we propose approximate methods for measuring the quality of dendrograms. These will then be used in the metaheuristic proposed in section V to result in the main algorithm developed in this paper. In section VI the results of our experimentation are shown. We end the paper with a short conclusion and an outlook on future research.

II. Background: Hierarchical Clustering

Clustering is a well-known, unsupervised learning task which has the objective of grouping similar items together. These groups are called clusters and they are to be discovered in the dataset. This is as opposed to classification where the classes into which the items have to be placed are predetermined. The clusters formed should be such that the similarities between the items in the same cluster are high, while the similarities between items in different clusters are relatively low.

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.12

A specific type of clustering algorithms are the hierarchical ones. These do not just produce a partition of the items, but a dendrogram (see fig. 1). One interpretation of such dendrogram is that it shows partitions of sizes ranging from one until the number of items to be clustered. These partitions are obtained by slicing the dendrogram at a given height. This is useful in application were the number of clusters to be obtained is not known beforehand. Some clustering approaches will determine the number of clusters automatically using a heuristic, but users might not trust these or it might be difficult to find a suitable one. Hierarchical approaches further allow the user to browse trough the produced hierarchy, which might give them a deeper insight in the structure of the data.

There are two major approaches to hierarchical clustering. First there are the divisive ones, where the items to be clustered are observed as a whole and split in two parts or sub-clusters. Each of these parts is then recursively split in ever smaller sub-clusters until each item is inside its own cluster. The other group of approaches are called agglomerative. The reason is that these algorithms perform the clustering by first placing each item in its own cluster and then repeatedly merging (or agglomerating) two clusters together to form a new one. This process stops when only one cluster is left. In this paper we will be dealing with the later group of approaches (i.e., the agglomerative ones).

The agglomerative algorithm is know by many names such as Globally Closest Pair (GCP) clustering [4], Sequential Agglomerative Hierarchical Non-overlapping (SAHN) clustering [5], [6], or the term which we will adopt Agglomerative Hierarchical Clustering (AHC) [1], [3], [7]. As mentioned, the algorithm works by repeatedly merging two clusters together into a larger one. The algorithm always merges the closest clusters in terms of a predefined linkage. This linkage defines the distance between clusters of items. Examples include single linkage (the shortest distance between any of the items), complete linkage (the longest distance between any of the items), average linkage (the average distance between the items), etc. (see also [8]). In this paper average linkage is used as the measure for cluster similarity.

Algorithm 1 Naive AHC algorithm

```
procedure NAIVE_AHC(C, $d_c$)
    while $size(C) > 1$ do
        $d_{min} \leftarrow \infty$
        for $c_1 \in C$ do
            for $c_2 \in C \setminus \{c_1\}$ do
                $distance \leftarrow d_c(c_1, c_2)$
                if $distance < d_{min}$ then
                    $d_{min}, c_L, c_R \leftarrow distance, c_1, c_2$
                end if
            end for
        end for
        $C \leftarrow C \setminus \{c_L, c_R\}$
        $c_{new} \leftarrow merge\,(c_L, c_R)$
        add merger of $c_L$ and $c_R$ to $den$
        $C \leftarrow C \cup \{c_{new}\}$
    end while
    return den
end procedure
```

C is the set of all clusters, where initially each item is put into a singleton cluster. d_c is the distance measure for clusters of items like, for instance, the average distance. den is the resulting dendrogram.

The most naive approach for AHC is show in algorithm 1. The code is a direct application of what was described above. Since there are $O(n)$ iterations with each $O(n^2)$ iterations inside, the computational complexity of this algorithm is $O(n^3)$. The memory overhead is, however, constant (if we ignore the dendrogram produced). Note that the distances between clusters which are not involved in the merge do not change. Therefore, a better approach would be to not re-compute these distances after each merge, but only update the ones which have actually changed. Using this fact was, for instance, done in what Müllner [8] called the primitive AHC algorithm. However, despite a practical speedup, the algorithm still requires $O(n^3)$ time. Some gain can still be made by, for instance, using a priority queue to store the closest items, but in order to get closer to the optimal boundaries, more advanced algorithms are needed. The work by Eppstein [9] is exemplary for exact approaches which are at the theoretical efficiency boundary for generic distance metrics. His algorithm performs a clustering in $O(n^2)$ time with a space requirement of $O(n^2)$. A less space consuming approach is described by the same author (posing linear memory requirements) but the time complexity of this algorithm rises to $O(n^2 \log^2 n)$. In our experimental evaluation below we used the $fastcluster$ implementation by Müllner [6].

All in all, despite the efforts done to improve exact AHC, the algorithms do not scale well in terms of the number of items. Therefore, besides the work on exact approaches, several authors have focused on producing a clustering which resembles the correct one closely, but consuming less resources. In some works heuristics, exploitation of specific properties of given metrics, or predefined parameters have been used to limit the number of distance calculations required [3], [5], [10]. Other works exploited some form of quantization of the space. One example is the work by Gilpin et al. [7] who proposed the use of angular quantization for approximate AHC. Several authors have exploited locality-sensitive hashing [2] to speed up the clustering task, for example Koga et al. [11] for single linkage and Cochez and Mou [1] for average linkage. In the next section we will summarize the work of the later article and in the rest of this paper we will present an improved version of the algorithm. For a more elaborated review on recent work related to AHC, see the Related Work section in [1].

III. Problem Formulation: Improving the Quality of the Dendrogram produced by Twister Tries

Twister tries were proposed as an algorithm which can overcome the scalability bottleneck of AHC with as a trade-off the production of an approximate dendrogram. [1] The algorithm works for average linkage and at least cosine, Jaccard and Hamming distance can be used. Further, linear time and space guarantees are provided. In this section, we will not present every detail of the algorithm, instead we will focus on the parts related to this paper. Readers interested in the details of the algorithm are referred to the original paper.

In order to understand the inner workings of the twister tries algorithm, we first need to introduce the notion of a proportionally sensitive hash function.

Definition 1 (Proportionally sensitive family)**.** *Let $\mathcal{H}$ be a family of hash functions mapping from $\mathcal{D}$ to some universe $\mathcal{U}$ and d be a distance metric defined on $\mathcal{D}$. Then, given $k > 0$, $\mathcal{H}$ is called k-proportionally sensitive with respect to the distance metric d if for every two $p, q \in \mathcal{D}$ and every $h \in \mathcal{H}$*

$$\Pr\left[h\left(p\right) = h\left(q\right)\right] = 1 - kd(p, q)$$

This means that a hash function is proportionally sensitive if the probability that it hashes points p and q at distance $d(p, q)$ to the same value is equal to $1 - kd(p, q)$, with k a predetermined constant. Several examples of proportionally sensitive hash functions (i.e., for cosine, Jaccard, and Hamming distance) were shown in the original twister tries paper. For this work we will be using random hyperplane hashing (RHH) [12] for the cosine distance and minhash [13] for Jaccard distance.

The main data structure used in twister tries is a forest consisting of tries (also called prefix trees). To create one such trie of height k_{max}, one needs to select k_{max} proportionally sensitive hash functions, at random. Then, the data items are inserted into the trie, such that the labels encountered on the arcs when following the path from the root node to the corresponding leafs are the outcomes of the hash function evaluations.

Once this structure is in place, the twister trie algorithm works somewhat similar to the naive AHC algorithm. The leafs of the tries represent the items inserted in the forest, these are now interpreted as one item clusters. Then the following steps are repeated: *a*) the lowest point where any of the tries split is located, *b*) two clusters connected to this node are removed from the tries, *c*) these clusters are merged by taking their union (recorded in the dendrogram), and *d*) the merged cluster is re-inserted into the forest using the same hash functions, but applied on items randomly selected from the merged cluster.

It has been proven that that selecting items randomly still provides certain guarantees of correctness. However, twister tries are still sensitive to unlikely hash outcomes. These unlikely, but possible, outcomes could affect the final dendrogram produced in a disastrous way. In practice, this would mean that a trie contains a pair of clusters which splits much lower than it should, which will affect the quality of the whole clustering negatively. It might also happen (although with a much lower probability) that a pair of clusters does not branch low enough in any of the tries, causing their clustering to happen too late.

It seems unfeasible to see directly from a trie that this type of issues exists, in a reasonable amount of time (i.e., less time than it would take to perform an exact clustering of the data). Moreover, if we would discover a trie which looks bad in isolation, it might still contribute positively in the overall clustering because of the effects of other tries. What we will do instead of looking inside the tries, is treating each trie as a black box. Then, we attempt to find the combination of tries which produces the dendrogram with the best quality. In the next section we will develop a way to decide the quality of a dendrogram. In the section after that we will propose a metaheuristic which allows us to find a good combination of tries within reasonable time.

IV. SIDESTEP: MEASURING THE QUALITY OF A DENDROGRAM

As discussed in the previous section, we need a way to determine the quality of a dendrogram. To be more precise, for the execution of the proposed algorithm we need to be able to assess the quality of dendrograms without knowing a dendrogram produced by an exact algorithm, whereas to evaluate the performance, we will also need to compare the quality of the approximate dendrograms with an exact one. Furthermore, since we are working with fairly large dendrograms, the methods used have to be scalable.

Traditional ways of assessing a hierarchical clustering algorithm work by comparing its outcome to a gold standard or to the dendrogram produced by a known (exact) algorithm. Comparing to a gold standard (i.e., a natural clustering of the data made by a specialist) is not appropriate in what we try to achieve. Doing this type of analysis would show that the clustering approach is suitable to express the semantic meaning of specific datasets. However, it would not show that the approach which we propose is able to closely resemble what AHC with average linkage offers. In practice, this would mean that if it is known that AHC with average linkage works for a given problem, there would be no guarantee whatsoever that our method can be used. For the comparison with the outcome produced by a known algorithm there are several options. A thorough approach is the measure developed by Fowlkes and Mallows [14]. However, the interpretation of produced results is hard to be performed automatically. Furthermore, the use of this measure for any larger evaluation is infeasible due to its computational complexity. The high computational cost is due to the calculation of an exact dendrogram, as well as due to the computation of the metric itself (whose complexity is $O(n^3)$). The issue of scaling metrics for dendrogram comparison is discussed in depth in the original twister tries paper [1].

Given these issues with most metrics, we investigated the use of the metric proposed by Kull and Vilo [3], since it seems computable and can be adapted for the computation of the quality of a dendrogram in isolation (i.e., without an exact dendrogram available). In the next subsection we introduce this metric, analyze its computational properties, and propose computationally more attractive alternatives.

A. Joining Distance Ratio

At each step of the naive AHC algorithm, the pair of clusters with the smallest distance is merged. This implies that an exact dendrogram minimizes the sum of these distances. If we now obtain an approximate dendrogram, we can evaluate its performance by measuring how closely the sum of its joining distances resembles the sum obtained for the exact dendrogram. Based on this idea, the joining distance ratio [3] is defined as follows:

Definition 2 (Joining distance ratio (JDR))**.** *The joining distance ratio (*JDR*) is the proportion of the sum of the distance at each step of the standard AHC dendrogram and the sum of the distances of the approximate algorithm.*

$$\mathrm{JDR} = \frac{\sum\limits_{I,J \in \textit{AHC dendrogram}} d_A(I, J)}{\sum\limits_{I,J \in \textit{approximate dendrogram}} d_A(I, J)}$$

The AHC is expected to be below 1 for an approximate dendrogram. This is because a non-optimal merging decision might have been taken and hence the sum of the distances will be larger than in the optimal case. Note, however, that a result larger than 1 is possible if the exact algorithm performs a clustering at a lower level forcing it to make a unfavorable clustering at a higher level. This might, for instance, happen if two clusters are at the same distance and the exact algorithm can choose among them. However, a benefit of this metric is that it is indifferent regarding insignificant re-orderings of the dendrogram. This means that if a re-ordering occurs which does not change the shape of the dendrogram, then the JDR will produce the same outcome.

From the original JDR, we derive a metric which we will call the Joining Distance (JD). The JD is defined as

Definition 3 (Joining Distance)**.** *The joining distance (*JD*) is the sum of the distances between the clusters at each merging step of the dendrogram.* $\mathrm{JD} = \sum_{I,J \in \textit{dendrogram}} d_A(I,J)$

It is easily seen that the JDR is obtained by dividing the JD of the exact dendrogram by the JD of the approximation. Note that the JD measure does only make sense in the context of a specific clustering task. In other words, it does not make sense to compare the JD computed from a dendrogram for one clustering task and compare it to the JD obtained from another one. Moreover, it is important that the JD is only computed for complete dendrograms. When comparing two JD values, the dendrogram which resulted in the lowest value is better than the one with the higher value. In the next subsection we analyze the complexity of this metric.

B. Complexity

It is clear that the computation of the JD does only require a constant amount of memory. However, it is not immediately obvious what the computational complexity is. The JD is the sum of $n-1$ terms, since the hierarchical clustering of n items will always have $n-1$ clustering steps. However, the analysis gets complicated because the computation of each of these terms involves the items which are in the left and right sub-cluster of the mergers. In the next theorem, and accompanying proof, we show that the total number of distance calculations needed will always be $\frac{n(n-1)}{2}$ and how this leads to a time complexity of $O(n^2)$ for both JD and JDR.

Theorem 1. *If the computation of the distance between two items can be performed in a constant time, then the computation of the joining distance for average distance is in $O(n^2)$ (with n the number of items or leaf nodes in the dendrogram).*

Proof: We will show by induction on the number of items n that the computation of JD for a dendrogram under average distance involves $\frac{n(n-1)}{2}$ individual distance computations.

Base case: calculating the JD of a dendrogram with two leafs (and hence one merging step) needs $\frac{n*(n-1)}{2} = 1$ step.

Induction: Let $n > 2$ and suppose the hypothesis is true for all $k < n$. Call the number of elements in the left sub-dendrogram of the root l and in the right one r. Note that, $n = l + r$, $l < n$, and $r < n$. The JD of the whole tree is the sum of the JD of the left and right sub-dendrogram and the average distance measured at the root itself. Hence, calculating the JD requires $\frac{l*(l-1)}{2} + \frac{r*(r-1)}{2} + l*r$ distance calculations. This is equal to $\frac{l^2-l+r^2-r+2rl}{2} = \frac{(l+r)(l+r-1)}{2} = \frac{n(n-1)}{2}$.

Conclusion: By the principle of strong induction, it follows that the hypothesis is true for all n.

Now, to compute the JD we need these distance computations and the addition of $n-1$ terms. Hence, altogether $\frac{(n+2)(n-1)}{2}$ constant time operations are needed, which implies that the joining distance is computed in $O(n^2)$. ■

The consequence of this analysis is that the JDR is also computable in $O(n^2)$ and somewhat scalable. Therefore we will use it for analyzing the quality of the produced dendrograms of moderate sizes. However, if we want to use a metric for improving the quality of the clustering, we will need a faster alternative. The reason is that if $O(n^2)$ operations are needed for the evaluation of one dendrogram, it is better to use this computational power to compute the exact dendrogram directly. Therefore, we will propose several approximations for the JD in the next section. These approximations have a lower practical or theoretical complexity.

C. A Computationally More Attractive Alternative

We established in the previous subsection that the JD is reasonably computable for moderate datasets. Unfortunately, within the context of optimization algorithms, a quadratic growth of the complexity ($O(n^2)$) can lead to very demanding tasks when large scale cases are taken into consideration. In the latter case the computational time to achieve an optimal solution can be unacceptable.

Therefore, we developed alternatives which are computationally less demanding but still preserve the properties required to find a satisfactory solution. These functions, which we will call surrogates or approximations, work well if they are able to assign a larger value to a dendrogram with a larger joining distance. Or, in other words, the values produced by the surrogate should allow an ordering of the dendrograms which is the same as (or at least similar to) the order of the dendrograms by joining distance. The quality of the surrogates is measured as the fraction of dendrogram pairs the surrogate manages to order correctly.

Definition 4 (Surrogate quality)**.** *Given a surrogate function* JD_s *(which computes an approximated value for the* JD*) and a set of dendrograms $\mathcal{D}$, if we indicate with $K = \binom{\mathcal{D}}{2}$, the quality* $\mathrm{Q}(\mathrm{JD}_s, \mathcal{D})$ *of the surrogate with respect to this set is given by 1-the normalized Kendal Tau distance [15]:*

$$\frac{\displaystyle\sum_{(a,b)\in K} \begin{cases} 1 : \mathrm{JD}(a) \leq \mathrm{JD}(b) \vee \mathrm{JD}_s(a) \leq \mathrm{JD}_s(b) \\ 1 : \mathrm{JD}(a) > \mathrm{JD}(b) \vee \mathrm{JD}_s(a) > \mathrm{JD}_s(b) \\ 0 : \textit{otherwise} \end{cases}}{|K|}$$

Note that the closer the JD of the dendrograms in $\mathcal{D}$ to each other, the more difficult it will be to obtain a good quality. The reason is that that when the dendrograms are close, the surrogate becomes more likely to make mistakes in the ordering.

We propose several options for approximating quality and evaluate their performance. The main target is to limit the number of pairwise distances computed in the process by estimating the average distance for each merge by a lower number of individual distance calculations.

We propose the following approximations:

fix b For each merge step, we have a fixed budget b of distance calculations. Meaning that to estimate the average distance we sample b pairs and calculate their average.

sqrt Instead of a fixed budget, we use sample $\sqrt{\text{number of pairs}}$ pairs to make an estimate.

sub At each internal node, we create pairs by pairing all items from the smallest subtree with a randomly selected item from the largest subtree. The result is the average of their distances.

w b At each merge step we select b pairs by traversing the dendrogram downwards and at each branching, continue to the left or right with equal probability. Then, in the average distance computation we weigh the pairs. The weight of a pair is the product of the depths in the sub-dendrograms. The rationale behind this choice is that if an item is deeply nested in the dendrogram, then it is likely that *a*) there are many other ones which are close to this item and *b*) items in this dense cluster are unlikely to be chosen randomly.

In our evaluation we compare the performance of the above approximations in practice, thus obtaining a fitness modified objective function which we indicate with $\tilde{f}$. In the next section we discuss a metaheurisic which uses the approximations developed in this subsection.

V. A Metaheuristic Approach: A Tailored Evolutionary Algorithm

Modern technologies often impose the solution of complex and multivariate optimization problems also when an explicit representation of the objective function is not available, see e.g. [16]–[18]. Whenever derivatives cannot be calculated, or due to other reasons such as a high problem dimensionality, the application of a metaheuristic approach is the only viable option. This is the case of the present work where the result depends on the result of a simulation process. As a further complication, a well-known fact within the optimization community is that, as a consequence of the No Free Lunch Theorem, a universal optimizer does not exist, see [19]. On the contrary, a high performance in optimization is achieved by designing an optimizer around the problem features, see [20]–[23].

Thus, although optimization is not the focus of this paper, we have attempted an algorithmic design tailored to the problem. The binary nature of the problem, the active/inactive trie associated to a certain index, allows a natural binary encoding of the information. For example, the binary representation $[0, 1, 1]$ corresponds to the first trie being disabled, the second and third one active (see also fig. 2). In order to evaluate a candidate solution, the quality of the corresponding dendrogram is evaluated according to one of the procedures described in section IV.

The chosen algorithmic structure is that of an Evolutionary Algorithm (EA), see [24] because the long and discontinuous binary structure requires a high diversity which can be achieved by using a population based system, see [25]. The algorithmic features are listed in the following. At the beginning of the optimization process n binary strings are sampled at random. The length is the same as the number of tries. Twenty tries are used on the basis of the experiments shown in the original twister trie article [1]. The population size n has been set equal to 25. We have chosen this setting in order to have a fairly exploitative behavior without having an excessively high risk to converge prematurely (as it would be for a population size of very few individuals).

At each generation, parents are selected by means of proportionate fitness selection with a roulette wheel. This choice can be justified by the large size of the decision space (the total amount of possible combinations) and the time constraints due to the simulator (each fitness evaluation requires up till 10 seconds, or more for certain approximations applied on large dendrograms). In other words, in order to quickly achieve an improvement, a high selection pressure has been applied although this choice might result into a premature convergence.

To enhance the diversity and on the basis of a set of preliminary results, a three parent crossover approach which generates one offspring has been designed for this problem. For each bit (gene), the value that occurs more often in the parent solutions is copied into the offspring. This approach is obviously highly exploitative and is counterbalanced by a mutation operation occurring after the generation of the offspring solution. The chosen mutation scheme is the biased mutation reported in [26]. In this mutation scheme the likelihood that a bit gets flipped from zero to one is higher than the opposite direction. The rationale behind this type of mutation is purely problem based: the chance that adding (moderately) more tries to a solution is likely to improve the result. This mutation can be seen as a shallow local search that makes use of problem information, see e.g. [27]. Finally, parent and offspring solutions are merged for survivor selection. The survivor selection occurs, in the fashion of a plus strategy of Evolution Strategies, by retaining those n individuals which display the highest fitness. For this study, since a clear convergence criterion was hard to find we ran the algorithm for 15 minutes and report the evolution of the real quality of the dendrogram corresponding to the most fit individual as reported by the surrogate.

Algorithm 2 describes, in greater details, the implementation of the proposed metaheuristic approach. An overview of the way the fitness function works can be found in fig. 2. First the tries which have a zero bit in the vector are inactivated. Then, the twister tries algorithm is used to generate an approximate dendrogram. Finally, this dendrogram is evaluated using a dendrogram quality function, which produces a result which resembles the joining distance.

VI. Experiments

To get a deeper insight into the approximations of the joining distance and to evaluate our algorithm we performed two series of experiments. In the first series we investigate how useful the proposed approximations are by applying them on realistic dendrograms and comparing their outcome with the one obtained using the exact joining distance. We also discuss the execution times and select reasonable ones for the second

Algorithm 2 The proposed metaheuristic optimization algorithm

```
procedure FIND BEST DENDROGRAM(forest, b, f, n)
    $pop \leftarrow n$ random vectors of length $b$
    for individual in pop do
        $fitness[individual] \leftarrow f(individual, forest)$
    end for
    $best \leftarrow \max_{individual} fitness[individual]$
    while budget condition do
        $matingPool \leftarrow$ select $n$ individuals
        $newPop \leftarrow$ 3-parent_cross_over $(matingpool)$
        biased_mutate$(newPop)$
        for i in newPop do
            $fitness[i] \leftarrow f(i, forest)$
        end for
        $best \leftarrow \max_{individual} fitness[individual]$
        $pop \leftarrow n$ fittest from $pop \cup newPop$
    end while
    return best
end procedure
```

The procedure has four parameters: $forest$ which is the collection of tries containing the hash outcomes, b is the number of tries in the forest, f which is the function to be optimized (i.e., the approximation of the quality), and n the population size.

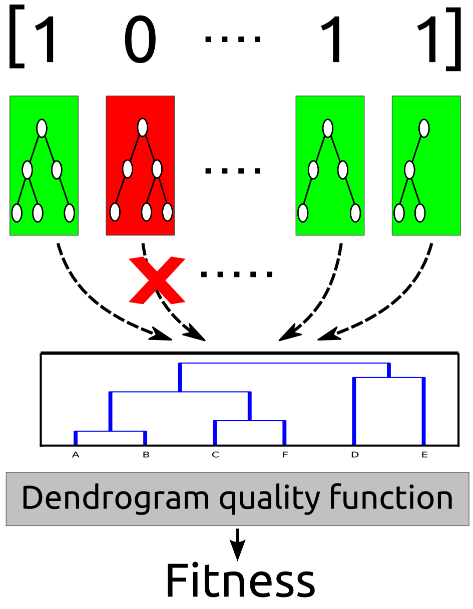

Fig. 2. Overview of the proposed a posteriori trie removal.

series of experiments. In the second series of experiments we look at the overall performance of the proposed algorithm. We cluster various portions of data selected form two different datasets. Then, we observe how the real joining distance of the best individual (according to the used surrogate) evolves.

The server used in our experiments has two Intel Xeon E5-2670 processors. Except for exact clustering (using Python, numpy, and fastcluster) all evaluations are performed using an OpenJDK 8 64-bit Server VM limited to use a maximum of 120 GB RAM.

A. Datasets

To allow comparison with the original twister tries paper we obtained the same datasets for our evaluation. The first dataset is the $cifar$-10 dataset[1]. This dataset contains 60,000 32x32 pixel images, resulting in 60,000 vectors of 3,072 integers. The distance between images is defined by the cosine distance between their vectors. The second dataset is a collection of newspaper articles called TRC2. This set consists of 1.8 million articles which we preprocessed by splitting on whitespace, removing punctuation, converting to lowercase, and applying Porter[2] stemming. Then, stop words, single characters, and numbers were removed. This procedure resulted in 1.68 million sets of words. The distance between two articles is defined as the Jaccard distance between these sets.

B. Evaluation of surrogates

To find out whether the proposed surrogates resemble the actual joining distance we performed two large experiments. The experiments differ in the dataset and the number of items used. In the first experiment, we used 20 dendrograms (190 pairs) representing the clustering of 10,000 images from the $cifar$-10 dataset. While in the second experiment, we used 20 dendrograms for 30,000 newspaper articles. The moderate number of items used in these experiments is due to the fact that we need to compare the findings to exactly computed joining distances. The dendrograms created for these experiments are such that their joining distances are pairwise close to each other. (This is achieved by using the original twister tries algorithm with different seeding.) For each of the surrogates proposed above, and for several possible settings of their parameters, we measure the quality as defined in section IV-C. Further, we measure the (average) time needed to evaluate the surrogate function. The results of these experiments are in tables I and II.

TABLE I. QUALITY AND RUNTIME OF SURROGATES FOR DENDROGRAMS CREATED FROM 10,000 IMAGES. ALL COMPUTATIONS ARE PERFORMED SERIALLY TO AVOID MEASUREMENTS BIAS IN THE TIMINGS.

measure	quality	time (ms)	measure	quality	time (ms)
exact	100	291740			
fix 1	90	44	w 1	81	51
fix 2	91	73	w 2	86	100
fix 3	94	100	w 3	91	124
fix 4	94	131	w 4	88	119
fix 5	93	154	w 5	89	151
fix 7	95	212	w 7	90	204
fix 10	96	294	w 10	91	275
fix 20	97	562	w 20	92	545
fix 50	97	1377	w 50	91	1317
sub	89	198	sqrt	95	403

What we observe from these experiments is that overall the surrogates perform fairly well, while using much less time than the exact computation. Further, the fixed and weighted computations use time linear in the number of samples performed and, with a few exceptions, the quality raises when more samples are taken. The exceptions are most likely due to

[1] http://www.cs.toronto.edu/~kriz/cifar.html

[2] http://tartarus.org/martin/PorterStemmer/

TABLE II. Quality and runtime of surrogates for dendrograms created from 30,000 images. All computations are performed serially to avoid measurements bias in the timings.

measure	quality	time (ms)	measure	quality	time (ms)
exact	100	3416174			
fix 1	94	292	w 1	92	260
fix 2	96	498	w 2	95	461
fix 3	96	698	w 3	95	655
fix 4	98	893	w 4	95	845
fix 5	97	1088	w 5	96	1034
fix 7	97	1463	w 7	94	1403
fix 10	98	2023	w 10	97	1951
fix 20	98	3865	w 20	96	3739
fix 50	98	9324	w 50	96	9015
sub	96	1564	sqrt	97	3432

random behavior. From a quality perspective, the sub and sampling the square root of the number of pairs samples surrogates perform poorly. They need more time than other surrogates which produce results with better quality.

C. Evaluation of the overall algorithm

We evaluated the overall algorithm with several surrogates selected based on the experiments above. The selection consists of fix 1, fix 3, fix 50 and w 10. The reason to select these is that they span a broad range of quality and timing values. Now, using these surrogates we run the metaheuristic optimization using the same portions of data as used in the previous experiment.

During the experiment, we use 20 tries of height 20 and 30 for the TRC2 and $cifar$-10 dataset, respectively. In addition to these (to observe the effect of the trie removal) we introduce a possible, but unlikely trie into the forest about which we know that it has unfavorable properties. The trie we introduced has the same hash function at each level and, as a result, many splitpoints will be lower than desired. We measure how the (real) quality of the best individual, as reported by the surrogate, evolves over time. Further, we contrast this to the joining distance of the dendrogram produced by the standard twister tries and exact algorithm and the time needed for the exact algorithm. The result of these experiments can be found from figs. 3 and 4. Note that the budget for the metaheuristic was a runtime of 15 minutes.

What we notice from the graphs is that the a posteriori trie removal is not only able to remove the bad trie from the forest; it also improves beyond the result of the standard twister tries algorithm. This means that the metaheuristic also removes other tries (which were not intentionally bad, but just created randomly) from the forest and improves the result further. As was noted by Cochez and Mou [1], once the quality is at a certain level making a small improvement requires a lot more and higher tries, or in other words resources. Hence, these improvements are significant.

When looking at the performance of the different surrogates, we notice that fix 50 is slow, but always results in a monotonically decreasing joining distance. Fix 1 on the other hand gives fast results, but seems slightly less reliable. This can be seen from the $cifar$-10 figure around 300 seconds: the surrogate orders the individuals wrongly and the (real) quality of the best individual reported lowers. However, fix 1 is able to perform many more evaluations than fix 50 and overcomes the slightly lower performance in ordering results. Broader experiments would be needed to give support for a general recommendation, but our experiments suggest that using a cheap, but reasonably good surrogate (e.g., fix 1) is more beneficial than using a slow but more accurate one (e.g., fix 50).

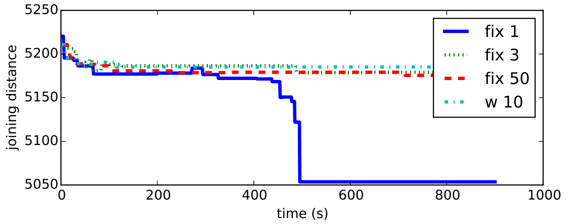

Fig. 3. Evolution of the real quality of the best individual as reported by metaheuristic using the surrogate. Results are for 10,000 images of the $cifar$-10 dataset. For comparison: the computation of the distance matrix took about 14.3 hours (using Python/numpy). From this matrix the exact dendrogram is computed in a couple of seconds, resulting in a joining distance of 4130.61. This exact computation only used one core. Normal twister tries would produce a JD of 5991.87. When the bad trie is taken away manually the result becomes 5217.85.

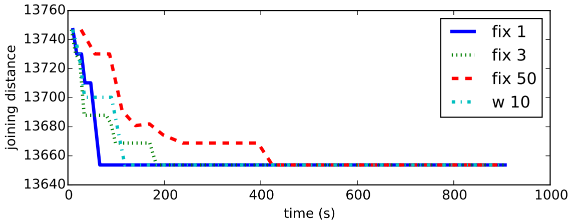

Fig. 4. Evolution of the real quality of the best individual as reported by metaheuristic using the surrogate. Results are for 30,000 articles of the TRC2 dataset. For comparison: the computation of the distance matrix took about half an hour. From this matrix the exact dendrogram is computed in about 10 seconds, resulting in a joining distance of 11814.27. This exact computation only used one core. Normal twister tries would produce a JD of 20040.32. When the bad trie is taken away manually the result becomes 13668.78.

VII. Conclusion

Twister tries usually produce reasonable approximate dendrograms for average linkage. However, since it is a probabilistic algorithm, there is a possibility that the randomly chosen hash functions result into tries with unwanted properties. In effect, these tries will cause the overall algorithm to produce a dendrogram with low quality. We proposed a metaheuristic which is capable of identifying bad tries. In order to make this metaheuristic useful, we first proposed and evaluated several surrogates which give a fast, but rough estimation of the dendrogram quality. Then we evaluated the metaheuristic and noticed (in a controlled experiment) that it is able to disable the trie and produce a better result than normal twister tries would. Moreover, not only the metaheuristic could disable the artificially inserted trie, but also was able to further improve the performance of twister tries by disabling specific other (random) tries which had a slightly negative effect on the final dendrogram. In conclusion, we recommend to always use some form of a posteriori trie removal in order to improve the quality of the dendrogram. We noticed in our experiments that the use of a simple surrogate (which produces reasonable outcomes) is more beneficial than a slower one (which delivers even better accuracy). Future work could attempt to combine multiple surrogates in the process by somehow determining that running a more expensive surrogate is worthwhile to make an important decision. Another path for further exploration is that the current approach is in some sense an all or nothing approach. Either a trie is enabled or disabled, even if this is only due to a couple of hash functions which cause the trie to be bad. It might be possible to save the good hash values if it would be possible to somehow find out which hash functions have a bad influence and then ignore these outcomes.

Acknowledgments

The authors would like to thank the Department of MIT and the Industrial Ontologies Group of the University of Jyväskylä for making this work possible. This research was also financed in part by the TEKES N4S SHOK in collaboration with Steeri Oy. For the experiments, we used the "Thomson Reuters Text Research Collection (TRC2)".

References

[1] M. Cochez and H. Mou, "Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time," in *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015.

[2] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327494

[3] M. Kull and J. Vilo, "Fast approximate hierarchical clustering using similarity heuristics," *BioData mining*, vol. 1, no. 1, p. 9, 2008.

[4] I. Gronau and S. Moran, "Optimal implementations of UPGMA and other common clustering algorithms," *Information Processing Letters*, vol. 104, no. 6, pp. 205–210, 2007.

[5] N. Kriege, P. Mutzel, and T. Schäfer, "SAHN clustering in arbitrary metric spaces using heuristic nearest neighbor search," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science, S. Pal and K. Sadakane, Eds. Springer International Publishing, 2014, vol. 8344, pp. 90–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-04657-0_11

[6] D. Müllner, "fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python," *Journal of Statistical Software*, vol. 53, no. 9, pp. 1–18, 5 2013. [Online]. Available: http://www.jstatsoft.org/v53/i09

[7] S. Gilpin, B. Qian, and I. Davidson, "Efficient hierarchical clustering of large high dimensional datasets," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, pp. 1371–1380. [Online]. Available: http://doi.acm.org/10.1145/2505515.2505527

[8] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.

[9] D. Eppstein, "Fast hierarchical clustering and other applications of dynamic closest pairs," *J. Exp. Algorithmics*, vol. 5, Dec. 2000. [Online]. Available: http://doi.acm.org/10.1145/351827.351829

[10] B. Patra, N. Hubballi, S. Biswas, and S. Nandi, "Distance based fast hierarchical clustering method for large datasets," in *Rough Sets and Current Trends in Computing*, ser. Lecture Notes in Computer Science, M. Szczuka, M. Kryszkiewicz, S. Ramanna, R. Jensen, and Q. Hu, Eds. Springer Berlin Heidelberg, 2010, vol. 6086, pp. 50–59. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13529-3_7

[11] H. Koga, T. Ishibashi, and T. Watanabe, "Fast hierarchical clustering algorithm using locality-sensitive hashing," in *Discovery Science*, ser. Lecture Notes in Computer Science, E. Suzuki and S. Arikawa, Eds. Springer Berlin Heidelberg, 2004, vol. 3245, pp. 114–128. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30214-8_9

[12] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02. New York, NY, USA: ACM, 2002, pp. 380–388. [Online]. Available: http://doi.acm.org/10.1145/509907.509965

[13] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences 1997. Proceedings*. IEEE, 1997, pp. 21–29.

[14] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, pp. 553–569, 1983. [Online]. Available: http://amstat.tandfonline.com/doi/abs/10.1080/01621459.1983.10478008

[15] M. G. Kendall, "Rank correlation methods." 1948.

[16] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "A fast adaptive memetic algorithm for on-line and off-line control design of PMSM drives," *IEEE Transactions on System Man and Cybernetics-part B*, vol. 37, no. 1, pp. 28–41, 2007.

[17] N. Salvatore, A. Caponio, F. Neri, S. Stasi, and G. L. Cascella, "Optimization of Delayed-State Kalman Filter-based Algorithm via Differential Evolution for Sensorless Control of Induction Motors," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 1, pp. 385–394, 2010.

[18] E. Mininno, F. Neri, F. Cupertino, and D. Naso, "Compact Differential Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 32–54, 2011.

[19] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[20] F. Caraffini, F. Neri, and L. Picinali, "An analysis on separability for memetic computing automatic design," *Information Sciences*, vol. 265, pp. 1–22, 2014.

[21] G. Iacca, F. Caraffini, and F. Neri, "Multi-strategy coevolving againg particle optimization," *International Journal of Neural Systems*, vol. 24, no. 01, p. 1450008, 2014.

[22] M. Epitropakis, F. Caraffini, F. Neri, and E. Burke, "A separability prototype for automatic memes with adaptive operator selection," in *Foundations of Computational Intelligence (FOCI), 2014 IEEE Symposium on*, Dec 2014, pp. 70–77.

[23] G. Zhang, H. Rong, F. Neri, and M. J. Perez-Jimenez, "An optimization spiking neural p system for approximately solving combinatorial optimization problems," *International Journal of Neural Systems*, vol. 24, no. 05, p. 1440006, 2014.

[24] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computation*. Berlin: Springer Verlag, 2003, pp. 175–188.

[25] A. Prügel-Bennett, "Benefits of a Population: Five Mechanisms That Advantage Population-Based Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 500–517, 2010.

[26] T. Nakashima and H. Ishibuchi, "Learning of fuzzy reference sets in nearest neighbor classification," in *Fuzzy Information Processing Society, 1999. NAFIPS. 18th International Conference of the North American*, Jul 1999, pp. 357–360.

[27] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*, ser. Studies in Computational Intelligence. Springer, 2011, vol. 379.