



CARISMA

A multi-disciplinary research centre focussed on understanding, modelling, quantification, management and control of RISK

- **School of Information Systems, Computing and Mathematics**
- **Brunel Business School**
- **School of Engineering**
- **School of Social Sciences**



TECHNICAL REPORT

CTR/82/08

Metaheuristic Approaches for the Quartet Method of Hierarchical Clustering

Sergio Consoli, Ken Darby-Dowman, Gijs Geleijjnse, Jan Korst, Steffan Pauws

Metaheuristic approaches for the quartet method of hierarchical clustering

Sergio Consoli, Kenneth Darby-Dowman

CARISMA and NET-ACE, School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom, {sergio.consoli@brunel.ac.uk, kenneth.darby-dowman@brunel.ac.uk}

Gijs Geleijnse, Jan Korst, Steffen Pauws

User Experiences (UX) group, Computational Intelligence cluster, Philips Research Eindhoven, High Tech Campus 34, Eindhoven 5656 AE, The Netherlands, gijs.geleijnse@philips.com; jan.korst@philips.com; steffen.pauws@philips.com

Given a set of objects and their pairwise distances, we wish to determine a visual representation of the data. We use the quartet paradigm to compute a hierarchy of clusters of the objects. The method is based on an NP-hard graph optimization problem called the Minimum Quartet Tree Cost problem. This paper presents and compares several metaheuristic approaches to approximate the optimal hierarchy. The performance of the algorithms is tested through extensive computational experiments and it is shown that the Reduced Variable Neighbourhood Search metaheuristic is the most effective approach to the problem, obtaining high quality solutions in short computational running times.

Key words: artificial intelligence; cluster analysis; networks; graphs; heuristics

1. Introduction

Hierarchical clustering methods have been employed in many different disciplines, such as social science, engineering, medicine, biology, planning, management, and even literature (Kaufman and Rousseeuw, 2005). This paper focusses on the *quartet method* of hierarchical clustering (Cilibrasi and Vitányi, 2005, 2006) which, given a set of objects to be classified, produces a hierarchy of the objects according to a specific cost evaluation, without knowing a priori the number of clusters to be produced. Given $n \geq 4$ objects to cluster, the quartet method of hierarchical clustering accepts as input a *distance matrix*, which is a matrix containing the distances, taken pairwise, among the n objects. It is assumed that the $n \times n$ distance matrix is symmetric, with non-negative reals, normalized between 0 and 1, as entries. The value 1 represents the largest distance between two objects. As output, the quartet method produces a *full unrooted binary tree* with $n \geq 4$ leaves (Diestel, 2000).

In a full unrooted binary tree, all the internal nodes have degree exactly three and there is no distinction between parent and child nodes (Furnas, 1984; Diestel, 2000). In order to visually represent the distance matrix as well as possible, the quartet method of hierarchical clustering places the n objects to be clustered as leaves of the full unrooted binary tree, such that objects with a short relative distance will be represented close to each other in the tree. A full unrooted binary tree with $n \geq 4$ leaves will have exactly $n - 2$ internal nodes, and consequently will have a total of $2n - 2$ nodes. Full unrooted binary trees are of primary interest in clustering contexts because, of all trees with a fixed number of nodes, they have the richest internal structure. They are therefore the most sensitive for representing the structure of a set of objects (Furnas, 1984).

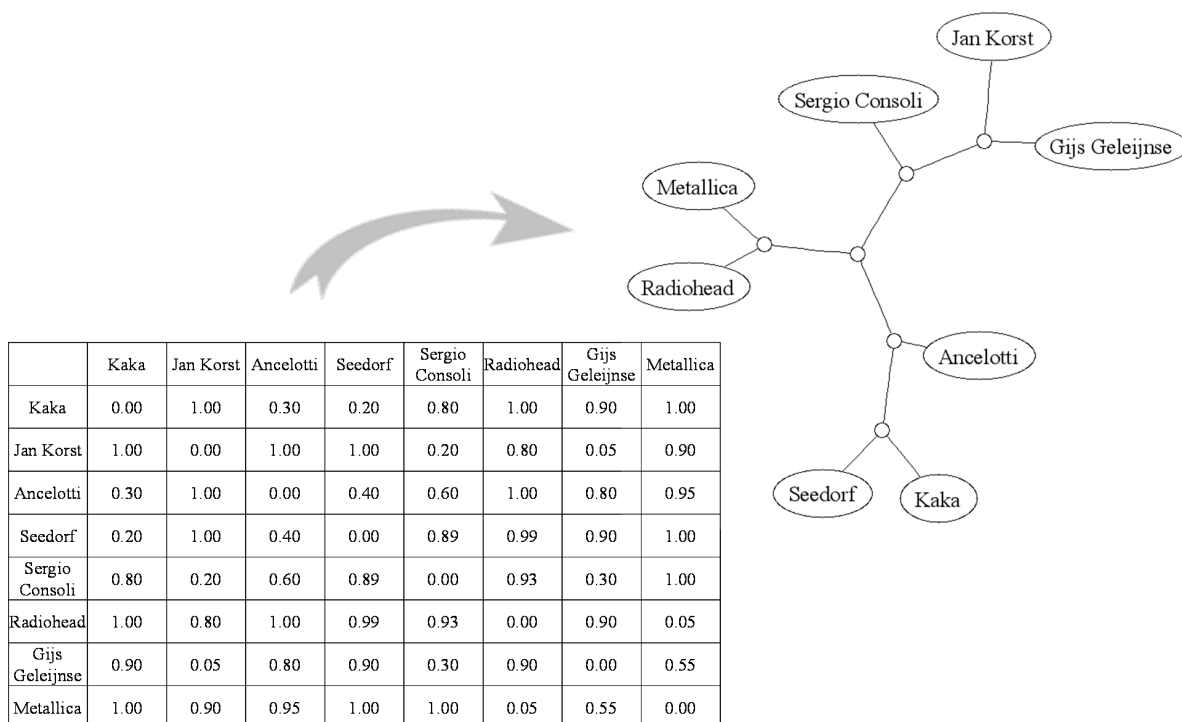


Figure 1: The left part shows an example of a distance matrix in input to the quartet method. The right part shows the full unrooted binary tree representing the optimal hierarchy.

Figure 1 shows a simple example on how the quartet method by Cilibrasi and Vitányi (2005, 2006) classifies $n = 8$ objects from completely different domains by means of a full unrooted binary tree. The left part of Figure 1 is an example of an input distance matrix created arbitrarily by the authors, with the optimal hierarchy shown on the full unrooted binary tree in the right part. The optimal hierarchy is the best representation of the distance matrix by means of a full unrooted binary tree with the n objects clustered as leaves. The two

famous bands Metallica and Radiohead form a cluster. Then Kaka, Seedorf, and Ancelotti, who belong to the same football club (A.C. Milan) form another cluster, with Kaka and Seedorf closer together as players, and coach Ancelotti further away. The final cluster is that of Sergio Consoli, Gijs Geleijnse, and Jan Korst, who are co-authors of this paper.

The rest of the paper is organised as follows. In Section 2, the quartet method and the related literature are described in depth. This method constructs the full unrooted binary tree approximating the optimal hierarchy according to the input distance matrix. The quartet method of hierarchical clustering is based on an NP-hard graph optimization problem, called the *Minimum Quartet Tree Cost* (MQTC) problem (Cilibrasi and Vitányi, 2005, 2006). In Section 3, we present details of several metaheuristics which find approximate solutions to the problem: the heuristic recommended in the literature (the *Randomized Hill Climbing* by Cilibrasi and Vitányi (2005, 2006)), and four new approaches that we introduce for the quartet method (*Greedy Randomized Adaptive Search Procedure*, *Simulated Annealing*, *Variable Neighbourhood Search*, and *Reduced Variable Neighbourhood Search*). Section 4 includes the experimental analysis of the evaluation of these metaheuristics, and the paper ends with some conclusions (Section 5). For a survey on the basic concepts of metaheuristics and combinatorial optimisation, the reader is referred to (Glover and Kochenberger, 2003).

2. The quartet method of hierarchical clustering

A fundamental problem in computational biology that has been widely studied in recent years is the construction of evolutionary trees from biological data (Diestel, 2000). These trees shows the evolutionary relationships between various biological species or other entities that are believed to have a common ancestor. The compelling need for having efficient computational tools to solve this biological problem has attracted much attention to the analysis of the quartet paradigm for inferring evolutionary trees (Felsenstein, 1981). Quartet methods utilize topological information on sets of four objects to infer an evolutionary tree. Given a set N of $n \geq 4$ objects, the number of sets of four objects from the set N is $\binom{n}{4}$. For each set of four objects $\{a, b, c, d\} \subseteq N$, there exist exactly three different full unrooted binary trees with four leaves (i.e. two internal nodes), also known as *simple quartet topologies*: $ab|cd$, $ac|bd$, $ad|bc$ (Figure 2). The vertical bar in a simple quartet topology divides the two pairs of objects, where each pair is represented by two leaf nodes, labelled by the corresponding objects and attached to the same internal node. Thus, considering the set N of $n \geq 4$

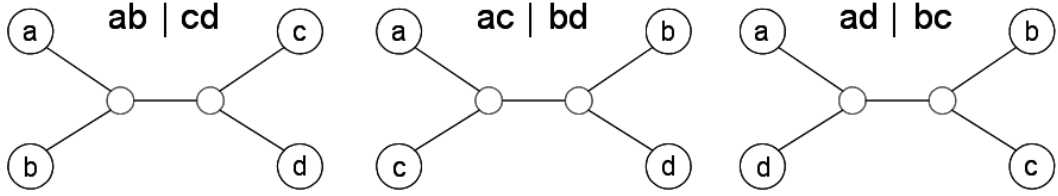


Figure 2: The three different simple quartet topologies of the generic set $\{a, b, c, d\}$ of objects.

objects, the total number of possible simple quartet topologies is $3 \cdot \binom{n}{4}$.

The quartet methods proceed by first estimating the topology of each quartet of objects and then recombining the inferred simple quartet topologies into an evolutionary tree. A major difficulty in this approach derives from the fact that quartet methods often make mistakes that result in the inconsistent inference of simple quartet topologies. These mistakes are also referred to as quartet errors. An evolutionary tree t is *consistent* with respect to a simple quartet topology $ab|cd$ if and only if the path from a to b does not cross the path from c to d (Felsenstein, 1981). We refer to $ab|cd$ as a simple quartet topology being *embedded* in the tree t . For example, the tree in Figure 1 is consistent with the simple quartet topology *Seedorf, Radiohead | Sergio Consoli, Jan Korst*. However, it is not consistent with the quartet topology *Ancelotti, Sergio Consoli | Metallica, Jan Korst*.

Define Q to be the set of all the possible simple quartet topologies, and Q_t to be the set of consistent simple quartet topologies being embedded in an evolutionary tree t . The problem of recombining the quartet topologies of Q to form an estimate of the correct evolutionary tree is naturally formulated as an optimization problem that looks for an evolutionary tree t maximizing the number of consistent simple quartet topologies Q_t (i.e. $\max Q \cap Q_t$). This problem, referred to as *Maximum Quartet Consistency* (MQC) problem, has been shown to be NP-hard (Steel, 1992). Jiang et al. (2000) proved that the MQC problem admits a polynomial time approximation scheme by using the technique of smooth integer polynomial programming and by exploiting the natural denseness of the set Q . However, this scheme only guarantees an evolutionary tree that may deviate from Q by εn^4 quartet topologies for any small constant $\varepsilon > 0$, where n is the number of objects.

Due to these results, most quartet methods are heuristics that attempt to solve the MQC problem, or some variants of the MQC problem with weaker optimization requirements. For example, Strimmer and von Haeseler (1996) formulated the MQC problem as a “tree-puzzling problem” by providing the simple quartet topologies with a probability value to be inferred.

Then, a set of simple quartet topologies is selected at random according to these probabilities to form the maximum-likelihood evolutionary tree. Berry et al. (1999) reported an interesting result. They presented two “quartet cleaning” algorithms for correcting bounded numbers of quartet errors for many popular quartet methods. Exact approaches to the MQC problem are presented in (Ben-Dor et al., 1998), where the problem is solved by using dynamic programming and a geometric algorithm, and in (Weyer-Menkhoff et al., 2005), where the problem is reformulated as an integer linear programming problem. However, these approaches are not able to solve problems with more than 15-20 objects.

Cilibrasi and Vitányi (2005, 2006) introduced a quartet method for hierarchically clustering data from different domains, not necessarily evolutionary data. They showed that their quartet method of hierarchical clustering is based on the *Minimum Quartet Tree Cost* (MQTC) problem, and provided a Randomized Hill Climbing metaheuristic to obtain approximate solutions. Several experiments with natural data, like genomic and phylogenetic data, texts or music, and data of completely different types, were further presented. The Randomized Hill Climbing produced good approximate solutions for small sets of objects (up to 40-50 objects), but for larger sets the performance was poor.

2.1. Mathematical formulation

Considering a set N of $n \geq 4$ objects, the quartet method of hierarchical clustering associates a real valued cost with each simple quartet topology by means of a *cost function* $C : Q \rightarrow \mathfrak{R}^+$, where Q is the set of simple quartet topologies. The cost assigned to each simple quartet topology is defined as the sum of the distances (taken from the distance matrix) between each pair of neighbouring leaves (Cilibrasi and Vitányi, 2005, 2006). For example, the cost associated with the simple quartet topology $ab|cd$ is: $C_{ab|cd} = d(a, b) + d(c, d)$, where $d(a, b)$ and $d(c, d)$ indicate, respectively, the distances between the two neighbouring objects (a and b) and (c and d), obtained from the distance matrix.

Consider the set Γ of full unrooted binary trees with $2n - 2$ nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the n objects to cluster as leaf nodes of the trees. For each tree $t \in \Gamma$, precisely one of the three possible simple quartet topologies for any set of four leaves is consistent. Thus, for each $t \in \Gamma$, there exist precisely $\binom{n}{4}$ consistent quartet topologies (one for each set of four objects) embedded in t . Let Q_t be the set of such $\binom{n}{4}$ quartet topologies embedded in t . Then, the *cost associated with a full unrooted binary tree*

$t \in \Gamma$ is defined as the sum of the costs of its $\binom{n}{4}$ consistent simple quartet topologies, that is: $C_t = \sum_{\forall \{ab|cd\} \in Q_t} C_{ab|cd}$.

In most cases, it is not possible to create a full unrooted binary tree which embeds all the simple quartet topologies with the minimum cost for all the sets of four objects (especially for a large number of objects n), due to inconsistency. Thus, it is a matter of making the most balanced choice of the quartet topologies to embed. This is the goal of the quartet method of hierarchical clustering: trying to find (or approximate as closely as possible) the full unrooted binary tree $t \in \Gamma$ with the minimum total cost. This full unrooted binary tree t will embed the combination of $\binom{n}{4}$ “possible” (consistent) simple quartet topologies of Q with the minimum costs, with respect to a full unrooted binary tree representation of the distance matrix. This optimization problem is called *Minimum Quartet Tree Cost* (MQTC) problem (Cilibrasi and Vitányi, 2005, 2006), and can be formally defined as follows:

MQTC problem: Given a set N of $n \geq 4$ objects to be clustered, and a symmetric distance matrix $n \times n$ containing their pairwise distances, find the full unrooted binary tree $t \in \Gamma$ with the minimum total cost C_t , i.e. $\min C_t = \min \left(\sum_{\forall \{ab|cd\} \in Q_t} C_{ab|cd} \right)$.

In a hierarchical clustering context, we do not even have a priori knowledge that certain simple quartet topologies are objectively true and must be embedded. Thus, the quartet method of hierarchical clustering assigns a cost value to each simple quartet topology, in order to express the relative importance of the simple quartet topologies to be embedded in the full unrooted binary tree having the n objects as leaves. The tree $t \in \Gamma$ with the minimum cost C_t , produced by the quartet method, balances the importance of embedding different quartet topologies against others, leading to a full unrooted binary tree that visually represents the symmetric distance matrix $n \times n$ as well as possible.

The MQTC may be normalized as follows (Cilibrasi and Vitányi, 2005, 2006). Consider the list of all possible four-tuples of $n \geq 4$ objects in N under consideration. For each set of four objects $\{a, b, c, d\} \subseteq N$, among the three possible simple quartet topologies, extract the one with the minimum cost, $m_{abcd} = \min \{C_{ab|cd}, C_{ac|bd}, C_{ad|bc}\}$, and that with the maximum cost, $M_{abcd} = \max \{C_{ab|cd}, C_{ac|bd}, C_{ad|bc}\}$.

The *best (minimal) total cost*, m , associated with $t \in \Gamma$ is calculated as the sum of the $\binom{n}{4}$ minimum costs m_{abcd} of each set of four objects $\{a, b, c, d\} \subseteq N$, that is: $m = \sum_{\forall \{a,b,c,d\} \subseteq N} m_{abcd}$. Similarly, the *worst (maximal) total cost*, M , associated with $t \in \Gamma$ is the sum of the $\binom{n}{4}$ maximum costs M_{abcd} of each set of four objects $\{a, b, c, d\} \subseteq N$, that is: $M = \sum_{\forall \{a,b,c,d\} \subseteq N} M_{abcd}$. In most cases, these cost values m and M can not be really attained

for any $t \in \Gamma$ (especially with a large number of objects n) and represent, respectively, a lower bound (m) and an upper bound (M) for the cost function C_t , that is: $m \leq C_t \leq M, \forall t \in \Gamma$. For a better and more uniform comparison of the costs associated with different full unrooted binary tree representations of different numbers of objects, the cost function is now rescaled linearly such that the best (minimal) cost maps to 1, and the worst (maximal) cost maps to 0. The rescaled cost function is called *normalized tree benefit score* S_t , and is defined as follows:

$$S_t = \frac{M - C_t}{M - m} \in [0, 1], \quad \forall t \in \Gamma. \quad (1)$$

The goal of the quartet method of hierarchical clustering is to find a full unrooted binary tree $t \in \Gamma$ with a maximum value of S_t , which is to say, the lowest total cost C_t . In order to compare uniformly the solutions of instances of the quartet method with different sizes, the MQTC can be reformulated with respect to the normalized tree benefit score as follows (Cilibrasi and Vitányi, 2005, 2006):

MQTC problem: Given a set N of $n \geq 4$ objects to be clustered, and a symmetric distance matrix $n \times n$ containing their pairwise distances, find the full unrooted binary tree $t \in \Gamma$ with the maximum normalized tree benefit score S_t (i.e. $\max S_t$).

Considering a set N of $n \geq 4$ objects, all the possible representations of the distance matrix by means of a full unrooted binary tree $t \in \Gamma$ will have a best normalized tree benefit score less than one in most of cases ($S_t < 1$, that is $C_t > m$), especially for a large number of objects n and noise in the distance matrix. The value $(1 - S_t)$ gives an estimation on how large is the distortion produced by a full unrooted binary tree representation of the distance matrix, resulting from the quartet method of hierarchical clustering. Trying to find the full unrooted binary tree $t \in \Gamma$ with the maximum S_t value (minimum C_t value) is the goal of the MQTC problem. This tree t will visually represent the distance matrix $n \times n$ as faithfully as possible by using the quartet method representation. As shown in (Cilibrasi and Vitányi, 2005, 2006), the Minimum Quartet Tree Cost problem is an NP-hard optimization problem by reduction from the Maximum Quartet Consistency problem (Steel, 1992; Jiang et al., 2000). Therefore, any practical approach to obtain or approximate the optimal solution requires heuristics. In the next section, several metaheuristics for the considered problem are presented and discussed in detail.

3. Exploited metaheuristics

This section describes the main features of the metaheuristics considered in this paper for the Minimum Quartet Tree Cost problem. First, the best performing method from the literature is reported, the Randomized Hill Climbing (RHC) by Cilibrasi and Vitányi (2005, 2006). The remaining heuristics are new approaches to the quartet method of hierarchical clustering. They are a Greedy Randomized Adaptive Search Procedure (GRASP), a Simulated Annealing (SA) approach, a Variable Neighbourhood Search (VNS), and a Reduced Variable Neighbourhood Search (RVNS).

Before examining these methods in detail, it is useful to specify the notation used within the implementations of these algorithms. Given a full unrooted binary tree, its internal nodes can be classified as *terminal nodes*, which are internal nodes connected to two leaves and another internal node, *transition nodes*, which are internal nodes connected to one leaf node and two other internal nodes, and *cross nodes*, which are internal nodes connected to three other internal nodes (no attached leaf nodes). For example, the full unrooted binary tree in Figure 1 has three terminal nodes which are connected to pairs of leaves with labels Seedorf and Kaka, Metallica and Radiohead, Gijs Geleijnse and Jan Korst, two transition nodes which are connected to the leaves with labels Sergio Consoli and Ancelotti, and one cross node which is not connected to any leaf. Furthermore, a *branch* of a full unrooted binary tree is defined as the subgraph, delimited between one terminal node and one cross node, containing only transition nodes. For example, the full unrooted binary tree of Figure 1 contains three branches, each one rooted at the only cross node of the tree and finishing with one of the three terminal nodes. The first branch contains the leaves Metallica and Radiohead, another contains the leaves Sergio Consoli, Gijs Geleijnse, and Jan Korst, and the last branch contains the leaves Ancelotti, Seedorf, and Kaka.

3.1. Randomized Hill Climbing (Cilibrasi and Vitányi, 2006)

The Randomized Hill Climbing (RHC) proposed by Cilibrasi and Vitányi (2005, 2006) for the quartet method of hierarchical clustering combines a basic Hill Climbing heuristic with randomization by using parallelized Genetic Programming (Glover and Kochenberger, 2003), where undirected trees evolve in a random walk driven by a prescribed fitness function. The details of this RHC for the quartet method are specified in Algorithm 1.

The algorithm starts by selecting at random a full unrooted binary trees $t \in \Gamma$ with $2n - 2$

Algorithm 1: Randomized Hill Climbing for the quartet method of hierarchical clustering

Input: A symmetric distance matrix d containing the $n \times n$ pairwise distances among $n \geq 4$ objects;

Output: A full unrooted binary tree t with $2n - 2$ nodes;

Initialisation:

- Let Γ be the class of full unrooted binary trees with $2n - 2$ nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the $n \geq 4$ objects to cluster as leaves;

- For each $x \in \Gamma$, let $S_x \in [0, 1]$ be the normalized tree benefit score of x ;

- Let $t' \in \Gamma$ be a full unrooted binary tree used as support solution at each iteration;

begin

 Generate the initial full unrooted binary tree $t \in \Gamma$ at random: $t \leftarrow \text{Generate-At-Random}(\Gamma)$;

 Evaluate the normalized tree benefit score of t : $S_t \leftarrow \text{Evaluate}(t)$;

repeat

 Set $t' \leftarrow t$;

 Select the number k of simple mutations with fat-tail probability distribution $p(k) = c/k(\log k)^2$ where $1/c = \sum_{k=1}^{\infty} 1/k(\log k)^2$;

for $i = 1$ to k **do**

 Apply a simple mutation to t' : $t' \leftarrow \text{Simple-Mutation}(t')$;

 Increase i : $i = i + 1$;

end

 Evaluate the normalized tree benefit score of t' : $S_{t'} \leftarrow \text{Evaluate}(t')$;

if $S_{t'} > S_t$ **then**

 Move $t \leftarrow t'$;

end

until *termination conditions* ;

\Rightarrow The full unrooted binary tree $t \in \Gamma$.

end

nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the $n \geq 4$ objects to cluster as leaves. This tree t is used as basis for further searching. The costs of the consistent quartet topologies embedded in t are calculated, and then the normalized tree benefit score S_t is computed ($S_t \leftarrow \text{Evaluate}(t)$). Afterwards, solution t is assigned to another full unrooted binary tree t' , which will be used as a support solution at each iteration of the search process. Then, a number k is picked up by a *fat-tail probability distribution* $p(k)$:

$$p(k) = \frac{c}{k(\log k)^2}, \text{ where } \frac{1}{c} = \sum_{k=1}^{\infty} \frac{1}{k(\log k)^2}. \quad (2)$$

A fat tail probability distribution $p(k)$ with the fattest tail possible has been chosen, in order to concentrate maximal probability also on the larger values of k , trying to minimize the likelihood of being trapped at a local minimum. For more details see (Cilibrasi and Vitányi, 2005, 2006).

In order to search for a better solution, a *k-mutation* is applied to the support solution t' . A *k-mutation* is defined as a sequence of k *simple mutations*, where a simple mutation, or *1-mutation*, is one of three possible operations (Cilibrasi and Vitányi, 2005, 2006):

1. A *leaf swap*, which consists of randomly choosing two leaf nodes and swapping them;
2. A *subtree swap*, which consists of randomly choosing two internal nodes and swapping the subtrees rooted at those nodes;

3. A *subtree transfer*, whereby a randomly chosen subtree (possibly a leaf node) is detached and reattached in another place.

Note that each of these simple mutations keeps the number of leaf nodes and internal nodes in the tree invariant. Only the structure of the full unrooted binary tree and the positions of the nodes are changed. Considering the support full unrooted binary tree t' , a k -mutation is composed by choosing one of the three possible simple mutations with equal probability. Leaves and internal nodes for each simple mutation are selected completely at random. Full unrooted binary trees which are close to t' , in terms of number of simple mutation steps in between, are examined often, intensifying the search process, while full unrooted binary trees that are far away from the original tree will eventually be examined, but not very frequently, diversifying the search process.

The normalized tree benefit score of the new solution t' , obtained by the k -mutation, is evaluated ($S_{t'}$), and is compared to the normalized tree benefit score (S_t) of the best solution to date t . If an improved full unrooted binary tree is obtained ($S_{t'} > S_t$), the best solution to date is updated with the new solution ($t \leftarrow t'$), otherwise the search restarts with the current t . This procedure continues iteratively until the termination conditions imposed by the user are satisfied and, at the end of the algorithm, the best full unrooted binary tree to date $t \in \Gamma$ is produced as output of the procedure.

3.2. Greedy Randomized Adaptive Search Process

The GRASP (Greedy Randomized Adaptive Search Procedure) methodology was developed in the late 1980s, and the acronym was coined by Feo and Resende (1989). Basically, it is a multi-start two-phase metaheuristic, consisting of a *construction phase* and a *local search* phase. The details are specified in Algorithm 2.

The algorithm starts by selecting at random a full unrooted binary trees $t \in \Gamma$ with $2n - 2$ nodes, obtained by placing the $n \geq 4$ objects to cluster as leaves. The costs of the consistent quartet topologies embedded in t are evaluated, and then the normalized tree benefit score S_t is computed ($S_t \leftarrow Evaluate(t)$). Then, the *Construction phase*(t', RCL_α) procedure builds another full unrooted binary tree $t' \in \Gamma$ by using a greedy randomized mechanism, whose randomness allows solutions in different areas of the solution space to be obtained. This greedy randomized mechanism obtains a full unrooted binary tree t' by iteratively creating a candidate list of distances (RCL_α : Restricted Candidate List of length α), and then by randomly selecting a distance from this list and connecting the corresponding objects in t' .

Algorithm 2: Greedy Randomized Adaptive Search Procedure for the quartet method of hierarchical clustering

Input: A symmetric distance matrix d containing the $n \times n$ pairwise distances among $n \geq 4$ objects;

Output: A full unrooted binary tree t with $2n - 2$ nodes;

Initialisation:

- Let Γ be the class of full unrooted binary trees with $2n - 2$ nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the $n \geq 4$ objects to cluster as leaves;

- For each $x \in \Gamma$, let $S_x \in [0, 1]$ be the normalized tree benefit score of x ;

- Let $t' \in \Gamma$ be a full unrooted binary tree used as support solution at each iteration;

- Let RCL_α be the restricted candidate list of length α ;

begin

Generate the initial full unrooted binary tree $t \in \Gamma$ at random: $t \leftarrow \text{Generate-At-Random}(\Gamma)$;

Evaluate the normalized tree benefit score of t : $S_t \leftarrow \text{Evaluate}(t)$;

repeat

Set $t' \leftarrow \emptyset$;

Construction phase(t', RCL_α);

Local search(t');

Evaluate the normalized tree benefit score of t' : $S_{t'} \leftarrow \text{Evaluate}(t')$;

if $S_{t'} > S_t$ **then**

Move $t \leftarrow t'$;

end

until termination conditions ;

\Rightarrow The full unrooted binary tree $t \in \Gamma$.

end

The candidate list is created by evaluating the distances between the objects that are not yet connected within the partial full unrooted binary tree t' , and then by including the shortest α of such distances in the list. At each iteration one new distance is randomly selected from RCL_α , the corresponding pair of objects are connected within the current full unrooted binary tree t' , and the candidate list is updated. It is important to make a good tuning of α in order to obtain an optimal balance between the intensification and diversification capabilities of the search process. Our experience indicates that $5 \leq \alpha \leq 10$ produces good results for the quartet method of hierarchical clustering.

The construction phase stops when a full unrooted binary tree t' is obtained. The produced solution t' is not necessarily locally optimal, so the *Local search(t')* procedure tries to improve it. This phase uses a local search mechanism which, iteratively, tries to replace the current full unrooted binary tree t' with a better neighbouring full unrooted binary tree, until no better solution can be found. Different strategies may be used in order to evaluate the neighbourhood structure. In our implementation, we consider each internal node and the neighbouring nodes having Manhattan distance equals to one with respect to the considered node (*one-neighbourhood structure* with respect to the Manhattan distance), that is we consider the internal nodes which are directly connected to the considered internal node. Then, a transformation of each pair of selected internal nodes is performed, aimed at producing small changes in the topology of the considered tree t' , checking whether these modifications improve the normalized tree benefit score of t' .

The internal nodes are selected following a specific order. First, all the terminal nodes are evaluated in order to improve each single branch of the current full unrooted binary tree t' . After selecting a terminal node, all the successive transition nodes belonging to the corresponding branch of the tree are evaluated, starting from the ones that are closer to the terminal node and stopping when the cross node delimiting the current branch is reached. For each selected internal node, the algorithm tries to exchange its attached leaf (or leaves in case of the terminal node) with the leaves attached to the one-neighbouring internal nodes (according to the Manhattan distance). The exchange of two leaves is retained if the normalized tree benefit score of t' improves. After selecting all the terminal nodes and trying to improve the corresponding branches, the algorithm selects all the remaining cross nodes. For each cross node, the algorithm tries to move each one-neighbouring terminal node from the corresponding branch containing the terminal node to the two other branches rooted at the selected cross node. In the case of another neighbouring cross node, the algorithm alternatively swaps one branch of one cross node with another branch of the other cross node. Again, each modification of the tree t' is retained if it produces a benefit in the normalized tree benefit score $S_{t'}$.

After exhausting all the cross nodes, the local search stops because all the internal nodes have been evaluated (best improvement strategy) and, hopefully, the obtained full unrooted binary tree t' will represent an improved solution with respect to the tree previously obtained by the construction phase. Afterwards, if the normalized tree benefit score of t' is better than that of the best full unrooted binary tree to date t (i.e. $S_{t'} > S_t$), the best full unrooted binary tree to date is updated with the new solution ($t \leftarrow t'$). The entire algorithm proceeds iteratively until the user termination conditions are satisfied, and produces the best full unrooted binary tree to date $t \in \Gamma$ as output of the procedure.

3.3. Simulated Annealing

Simulated Annealing (SA) is a descent heuristic with non-deterministic search developed by Kirkpatrick et al. (1983). In contrast to classical descent methods, where only modifications to the current solution that decrease the cost function value are accepted, modifications that increase the value of the cost function are allowed in SA.

SA exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system, forming the basis of an efficient optimisation technique for combina-

torial and other problems. SA seeks to minimise an energy function (the cost function); free variables in SA are like particles in the metal, and “low energy” configurations correspond to high quality solutions of the problem, obtained by slowly reducing a *temperature parameter* (T) by means of a *cooling rule* (or *cooling schedule*). The dependency is such that the current solution is always replaced by a new one if this modification reduces the cost function value (downhill move), while a modification increasing the cost function value (uphill move) by Δ is only accepted with a probability $\exp(-\Delta/T)$, referred to as Boltzmann function, using the temperature T as a control parameter. At the beginning of the algorithm, at a high temperature T , the probability of accepting an increase in the cost function value is high, allowing many worse moves to be accepted. Conversely, this probability gets lower as the temperature T is decreased during the search process by means of the cooling rule.

The details of the implementation of our Simulated Annealing for the quartet method are specified in Algorithm 3. For the considered problem, we implemented a *non-monotonic SA cooling schedule* (Osman, 1993), which requires specification of the following: (i) starting and final temperatures (T_s and T_f); (ii) decrement rule for updating the temperature T after each iteration; (iii) occasional increment rule for updating the temperature T every N_{reset} iterations with a reset temperature T_{reset} (in order to avoid the system being locked at local optima).

The algorithm starts by selecting at random a full unrooted binary tree $t \in \Gamma$ with $2n - 2$ nodes, obtained by placing the $n \geq 4$ objects to cluster as leaves, with cost C_t and normalized tree benefit score S_t . Then, the starting and final temperatures, T_s and T_f , are set to the maximum and minimum estimated modifications of the cost function, Δ_{max} and Δ_{min} , evaluated heuristically by means of the *Test-Cycle*(t) procedure. This procedure considers the *base moves* that each internal node of t can perform with its neighbouring internal nodes (*one-neighbourhood structure* with respect to the Manhattan distance). The alterations of the cost function corresponding to the performed base moves are evaluated, retaining the maximum and the minimum modifications in Δ_{max} and Δ_{min} .

Given an internal node and its neighbouring internal nodes, the possible base moves that can be performed depend on the types of internal node pairs. In the case of:

1. two transition nodes \rightarrow either the attached leaves are exchanged, or they are transformed into one cross node and one terminal node connected to the corresponding leaves;
2. one terminal node and one transition node \rightarrow the leaf of the transition node is exchanged with one of the two leaves of the terminal node;

Algorithm 3: Simulated Annealing for the quartet method of hierarchical clustering

Input: A symmetric distance matrix d containing the $n \times n$ pairwise distances among $n \geq 4$ objects;

Output: A full unrooted binary tree t_{best} with $2n - 2$ nodes;

Initialisation:

- Let Γ be the class of full unrooted binary trees with $2n - 2$ nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the $n \geq 4$ objects to cluster as leaves;

- For each $x \in \Gamma$, let C_x be the cost associated with x and $S_x \in [0, 1]$ the corresponding normalized tree benefit score;

- Let Δ_{\min} and Δ_{\max} be the minimum and the maximum estimated modifications of the cost function;

- Let T be the temperature parameter, T_s be the starting temperature value, T_f be the final temperature value, T_{best} be the best temperature value, T_{reset} be the reset temperature value, α be the geometric cooling rate;

- Let i be the number of iterations of the algorithm;

- Let $N_{\text{reset}} = \frac{2.5 \cdot 10^5}{n^2} + 200$ be the number of reset iterations;

- Let $t \in \Gamma$ be the full unrooted binary tree used at each iteration;

- Let $t' \in \Gamma$ be a full unrooted binary tree used as support solution at each iteration;

begin

Generate the initial full unrooted binary tree $t \in \Gamma$ at random: $t \leftarrow \text{Generate-At-Random}(\Gamma)$;

Evaluate the cost of t and its normalized tree benefit score: $(C_t, S_t) \leftarrow \text{Evaluate}(t)$;

Evaluate the minimum and the maximum estimated modifications of the cost function:

$(\Delta_{\min}, \Delta_{\max}) \leftarrow \text{Test-Cycle}(t)$;

Move $t_{\text{best}} \leftarrow t$, and set $T = T_{\text{reset}} = T_s = \Delta_{\max}$, $T_f = \Delta_{\min}$, $i = 1$;

repeat

Move $t' \leftarrow t$;

Select at random an integer between 0 and $n - 2$: $\lambda = \text{Random}(0, n-2)$;

for $j = 1$ **to** λ **do**

Perform a base move with respect to t' : $t' \leftarrow \text{Base-Move}(t')$;

Increase j : $j = j + 1$;

end

Evaluate the cost of t' and its normalized tree benefit score: $(C_{t'}, S_{t'}) \leftarrow \text{Evaluate}(t')$;

if $S_{t'} > S_t$ **then**

Move $t \leftarrow t'$;

if $S_t > S_{t_{\text{best}}}$ **then**

Move $t_{\text{best}} \leftarrow t$ and set $T_{\text{best}} = T$;

end

else

Select at random a real number between 0 and 1: $\xi = \text{Random}(0, 1)$;

if $\xi < \exp\left(-\frac{C_{t'} - C_t}{T}\right)$ **then**

Move $t \leftarrow t'$;

end

end

Geometric decrement rule for the temperature: $T = T_s \cdot \alpha^{(i \bmod N_{\text{reset}})/N_{\text{reset}}}$, where $\alpha = T_f/T_s$;

if $(i \bmod N_{\text{reset}}) = 0$ **then**

Occasional increment rule for the temperature: $T_{\text{reset}} = \max(T_{\text{reset}}/2, T_{\text{best}})$;

Set $T = T_s = T_{\text{reset}}$;

end

Increase the number of iterations: $i = i + 1$;

until *termination conditions* ;

\Rightarrow The full unrooted binary tree $t_{\text{best}} \in \Gamma$.

end

3. one terminal node and one cross node \rightarrow they are transformed into two transition nodes with the two leaves of the terminal node attached;

4. one transition node and one cross node \rightarrow the transition node is moved in one of the other two branches of the cross node;

5. two cross nodes \rightarrow one branch of one cross node is swapped with a branch of the other cross node.

When T_f and T_s are evaluated, the algorithm continues by assigning the value of T_s to the current temperature T and to the reset temperature T_{reset} , and by making a copy of t

to another full unrooted binary tree t' that will be modified by means of a *random move*. A random move is defined as a set of consecutive base moves, whose number is a random integer λ selected between 0 and $n - 2$. The random move starts by selecting a random internal node and one of its neighbouring internal nodes, and performing a base move with this pair of nodes. Then, to perform the successive base move, the algorithm selects one of the two considered internal nodes, and another neighbouring internal node that must be different from the two already considered internal nodes. The procedure continues until λ consecutive base moves are produced.

The cost and the normalized tree benefit score of the new full unrooted binary tree t' are evaluated, $(C_{t'}, S_{t'}) \leftarrow \text{Evaluate}(t')$. If $S_{t'} > S_t$, the solution t is assigned to the tree t' , storing the best solution to date in t_{best} , and the temperature at which this full unrooted binary tree is obtained in T_{best} . Otherwise, if the new full unrooted binary tree t' is worse than t ($S_{t'} < S_t$), the algorithm moves to t' with a probability that depends on the Boltzmann function $\exp(-\Delta/T) = \exp(-(C_{t'} - C_t)/T)$.

The non-monotonic SA cooling schedule that we use for the quartet method, decreases, at each iteration i of the algorithm, the temperature T according to the following geometric cooling rule: $T = T_s \cdot \alpha^{(i \bmod N_{\text{reset}})/N_{\text{reset}}}$, where $\alpha = T_f/T_s < 1$, and where $(i \bmod N_{\text{reset}})$ represents the arithmetic remainder of the integer division between the number of iterations i and the number of reset iterations N_{reset} . Every N_{reset} iterations (i.e. when $(i \bmod N_{\text{reset}}) = 0$) the temperature T and the starting temperature T_s are reset to a larger value, T_{reset} , to allow the algorithm to escape from local optima ($T = T_s = T_{\text{reset}}$). T_{reset} is chosen as the maximum value between $T_{\text{reset}}/2$ and T_{best} , while N_{reset} is a user defined parameter (our experience indicates that the value $N_{\text{reset}} = (2.5 \cdot 10^5)/n^2 + 200$ produces good results). This cooling schedule and its implementation is in contrast to classical SA schemes. From our experience, the considered non-monotonic cooling schedule outperformed other different SA cooling schedules for the quartet method. Note that the importance of non-monotonic search has been widely discussed in (Glover, 1986) as a basic feature of Tabu Search methods.

Subsequently, the algorithm restarts with the same procedure by setting $t' \leftarrow t$, continuing iteratively until the user termination conditions are satisfied. At the end, the best full unrooted binary tree to date, T_{best} , is produced as the output of the SA algorithm.

3.4. Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) is a recent metaheuristic for solving combinatorial optimization problems based on dynamically changing neighbourhood structures during the search process (Hansen and Mladenović, 2003). VNS does not follow a trajectory, but it searches for new solutions in increasingly distant neighbourhoods of the current solution, jumping only if a better solution than the current best solution is found. The proposed VNS for the quartet method of hierarchical clustering is specified in Algorithm 4.

Algorithm 4: Variable Neighbourhood Search for the quartet method of hierarchical clustering

Input: A symmetric distance matrix d containing the $n \times n$ pairwise distances among $n \geq 4$ objects;

Output: A full unrooted binary tree t with $2n - 2$ nodes;

Initialisation:

- Let Γ be the class of full unrooted binary trees with $2n - 2$ nodes (i.e. n leaves and $n - 2$ internal nodes), obtained by placing the $n \geq 4$ objects to cluster as leaves;

- For each $x \in \Gamma$, let $S_x \in [0, 1]$ be the normalized tree benefit score of x ;

- Let $t' \in \Gamma$ be a full unrooted binary tree used as support solution at each iteration;

- Let k be the current size of the shaking phase, and k_{max} be the maximum size of the shaking phase;

- Let i be the number of iterations between two successive improvements;

- Let $i_{update} = \frac{1.25 \cdot 10^5}{n^2} + 50$ be the number of update iterations for k_{max} ;

begin

Generate the initial full unrooted binary tree $t \in \Gamma$ at random: $t \leftarrow \text{Generate-At-Random}(\Gamma)$;

Evaluate the normalized tree benefit score of t : $S_t \leftarrow \text{Evaluate}(t)$;

Set $i = 0$ and $k_{max} = 2$;

repeat

Set $k = 1$;

while $k < k_{max}$ **do**

Move $t' \leftarrow t$;

for $j = 1$ **to** k **do**

Shake t' by performing a base move: $t' \leftarrow \text{Base-Move}(t')$;

Increase j : $j = j + 1$;

end

Local search(t');

Evaluate the normalized tree benefit score of t' : $S_{t'} \leftarrow \text{Evaluate}(t')$;

if $S_{t'} > S_t$ **then**

Restart with the first neighbourhood structure: $k = 1$;

Move $t \leftarrow t'$;

Set $i = 0$;

else

Increase the current size of the shaking phase: $k = k + 1$;

Increase the number of iterations between two successive improvements: $i = i + 1$;

end

end

if $i \geq i_{update}$ **then**

Increase the maximum size of the shaking phase: $k_{max} = k_{max} + 1$;

Set $i = 0$;

end

until *termination conditions* ;

\Rightarrow The full unrooted binary tree $t \in \Gamma$.

end

At the starting point, a full unrooted binary tree $t \in \Gamma$ with $2n - 2$ nodes, obtained by placing the $n \geq 4$ objects to cluster as leaves, is generated at random. Then, the *shaking phase*, which represents the core idea of VNS, is applied to t . A shaking phase of size k consists of the random selection of another full unrooted binary tree t' within

the neighbourhood $N_k(t)$ of the current solution t . To obtain t' from $N_k(t)$, the algorithm performs k consecutive base moves, already defined in Section 3.3. The first base move is performed to a randomly selected internal node and one of its neighbouring internal nodes (one-neighbourhood structure with respect to the Manhattan distance). Then, to perform the successive base move, the algorithm selects one of the two considered internal nodes, and another neighbouring internal node that must be different from the two already considered internal nodes, and so on. The procedure is repeated until k consecutive base moves are performed.

The shaking phase aims to change the neighbourhood structure when the algorithm is trapped at a local optimum. The solution t' is generated at random in order to avoid cycling, which might occur if a deterministic rule is used. Suitable neighbourhood structures need to be defined for the shaking phase. The simplest and most common choice consists of neighbourhoods with increasing cardinality: $|N_1(\cdot)| < |N_2(\cdot)| < \dots < |N_{k_{max}}(\cdot)|$, where k_{max} represents the maximum size of the shaking phase. Let k be the current size of the shaking phase. The algorithm starts by selecting the first neighbourhood ($k = 1$) and, at each iteration, it increases the parameter k if a better solution is not obtained ($k = k + 1$), until the largest neighbourhood is reached ($k = k_{max}$). The process of changing neighbourhoods when no improvement occurs diversifies the search. In particular, the choice of neighbourhoods of increasing cardinality yields a progressive diversification of the search process.

The full unrooted binary tree t' produced by the shaking phase, represents the starting point for the successive *local search phase*, which tries to improve, if possible, the solution t' . The considered local search (*Local search(t')*) is a first improvement strategy. It considers each internal node of t' and each of its neighbouring internal nodes, and computes all the base moves that can be performed with the selected pair of nodes (as with the shaking phase, the local search phase uses a one-neighbourhood structure with respect to the Manhattan distance for the selection of the neighbouring internal nodes). The local search stops either when a base move which improves the normalized tree benefit cost of t' , $S_{t'}$, is produced, or when all the internal nodes of t' have been evaluated without having improved $S_{t'}$.

If an improved full unrooted binary tree t' is produced by the shaking and the local search phases ($S_{t'} > S_t$), it becomes the best solution to date ($t \leftarrow t'$) and the algorithm restarts from the first neighbourhood ($k = 1$) of the best solution t . Otherwise, if no improvements are obtained ($S_{t'} < S_t$), the neighbourhood structure is increased ($k = k + 1$) giving a progressive diversification of the search process. Parameter k is increased until

the maximum size of the shaking phase, k_{max} , is reached. When this happens, k is re-initialized to the first neighbourhood ($k = 1$). The correct setting of k_{max} is an important user task. For the quartet method, a simple reactive schema for the efficient tuning of k_{max} has been implemented (Battiti et al., 2008). At the starting point, k_{max} is set to a small value ($k_{max} = 2$) and is increased ($k_{max} = k_{max} + 1$) every i_{update} iterations between two consecutive improvements. Our experience indicates that the value $i_{update} = (1.25 \cdot 10^5)/n^2 + 50$ produces good results. For more details on reactive search techniques, the reader is referred to (Battiti et al., 2008). The algorithm proceeds iteratively until the user termination conditions are satisfied, producing the best full unrooted binary tree to date, t , as the output of the procedure.

3.5. Reduced Variable Neighbourhood Search

Reduced Variable Neighbourhood Search (RVNS) is a variant of the basic VNS algorithm, that has been shown to be successful for many combinatorial problems where local optima with respect to one or several neighbourhoods are relatively close to each other (Hansen and Mladenović, 2003). The Reduced Variable Neighbourhood Search is obtained from VNS, where random solutions are selected from the neighbourhoods $N_k(\cdot)$ of the current solution, but without being followed by a local search phase. Therefore, it is a typical example of a pure stochastic heuristic. In practice, RVNS is akin to a classic Monte-Carlo method, but is a more systematic approach (Mladenović et al., 2003). It is useful especially for very large problem instances for which the local search of the basic VNS is costly, as in the case of the quartet method of hierarchical clustering.

RVNS starts by selecting at random a full unrooted binary tree $t \in \Gamma$ with $2n - 2$ nodes, obtained by placing the $n \geq 4$ objects to cluster as leaves, with normalized tree benefit score S_t . Then, the same shaking phase of the VNS specified in Algorithm 4 is applied. It selects at random another full unrooted binary tree t' from the neighbourhood $N_k(t)$ of the current solution t , by performing k consecutive base moves. Again, a one-neighbourhood structure with respect to the Manhattan distance, for the selection of the internal nodes, is used. At the beginning, the first neighbourhood ($k = 1$) is selected and, at each iteration, the parameter k is increased ($k = k + 1$) whenever the solution obtained is not an improvement to the current best solution, until the maximum size of the shaking phase (k_{max}) is reached. Hansen and Mladenović (2003) observed that, in RVNS, the best values for the maximum size of the shaking phase are often small values. Thus, parameter k_{max} is set arbitrarily by the user

through computational experience (say $k_{max} = 2$ or 3). As already stated, no local search phase is applied after the shaking phase. Throughout the execution of the algorithm, the best solution to date is stored as the full unrooted binary tree t , which will be produced as output of the algorithm when the user termination conditions are met.

4. Experimental results

In this section, the metaheuristics proposed for the quartet method of hierarchical clustering are compared in terms of solution quality and computational running time. We identify the metaheuristics with the abbreviations: RHC (Randomized Hill Climbing), GRASP (Greedy Randomized Adaptive Search Procedure), SA (Simulated Annealing), VNS (Variable Neighbourhood Search), RVNS (Reduced Variable Neighbourhood Search). All the algorithms that we propose have been implemented using the C++ programming language (Microsoft Visual C++ 2005). For the Randomized Hill Climbing, we have used the open-source software released in the public domain by the authors (Cilibrasi, 2007). All the computations have been made on a Pentium Centrino microprocessor at 2.0 GHz with 512 MB RAM.

In our experiments, we considered 26 different datasets with a number of objects to cluster (n) from 10 up to 224. Data from different fields have been considered in order to evaluate how the algorithms are influenced by the nature of the objects. First we considered data without inconsistency, that is data for which the exact solution is known and have a normalized tree benefit score equals to one, in order to test the accuracy of the quartet-based tree reconstruction. These data were produced artificially as described in Section 4.1. Then, in Section 4.2 we considered some examples from nature obtained from (Cilibrasi, 2007), concerning a study in genomics with DNA sequences of different placental mammalian species. Section 4.3 contains data with real geographic distances between famous cities, while Section 4.4 contains data obtained by mining of the WWW through an automatic web information extraction method by Geleijnse et al. (2006). Specifically, we have focussed on data concerning musical artists. All the instances of the problem are available online from the authors (Consoli, 2008).

For each dataset, given a full unrooted binary tree t produced by the quartet method, solution quality is evaluated by means of its normalized tree benefit score $S_t \in [0, 1]$. The quartet method of hierarchical clustering tries to find the solution which maximizes the S_t value, which is to say, the lowest total cost C_t . A maximum allowed CPU time (*max-CPU-*

time), determined with respect to the dimension of the problem instance, is chosen as the stopping condition for all the metaheuristics. Experimentally, for problem instances with a number of objects $n \leq 100$, we set *max-CPU-time* to one hour. For larger instances ($n \geq 100$), *max-CPU-time* is set to 10 hours. Selection of the maximum allowed CPU time as the stopping criterion is made in order to have a direct comparison of the metaheuristics with respect to the quality of their solutions.

Our results are reported in Tables 1 - 4. In each table, the first column shows the number of objects of the considered datasets (n), while the kind of data determines the different tables. Last row shows the averages, respectively, of the normalized tree benefits score and of the computational running times among the considered group of data instances. All the metaheuristics run for *max-CPU-time* and, in each case, the normalized tree benefit score of the best solution is recorded. The computational times reported in the tables are the times at which the best solutions are obtained. The reported times have precision of ± 1 sec. Analysing the performance of the considered algorithms, for a single dataset a metaheuristic should be considered *better* than another if either it obtains a larger normalized tree benefit score, or an equal normalized tree benefit score but in a smaller computational running time.

4.1. Testing the quartet-based tree reconstruction

In this section, we test whether the quartet-based tree reconstruction heuristic is reliable and accurate on clean consistent data with known solutions. We used the same procedure by Cilibrasi and Vitányi (2005, 2006) to generate data instances with corresponding optimal full unrooted binary trees t having normalized tree benefit score equal to one, $S_t = 1$. To obtain these data, we used the “rand” pseudo-random number generator from the C++ programming language (Microsoft Visual C++ 2005), and derived a metric from it by defining the distance, $d(x, y)$, between two objects x and y , as follows:

$$d(x, y) = \begin{cases} \frac{L(x, y) + 1}{n} & \text{if } x \neq y, \\ 0 & \text{otherwise,} \end{cases} \quad (3)$$

where $L(x, y)$ is the length of the path from x to y , expressed by the number of edges which connect the leaves of the full unrooted binary tree where the two objects are assigned. Obviously, the entries in the diagonal of the distance matrix are all zeros, since $d(x, y) = 0$ if $x = y$. All the full unrooted binary trees t constructed artificially with this procedure have optimal score $S_t = 1$. We generated data instances with a number of objects n from

10 to 100, setting the *max-CPU-time* for the heuristics to one hour. Computational results, reporting the normalized tree benefit scores found by the heuristics and the corresponding computational times, are presented in Table 1.

Table 1: Computational results for artificial data with optimal normalized tree benefit score equals to one

| Normalized tree benefit score | | | | | |
|--|---------|---------|---------|---------|---------|
| n | RHC | GRASP | SA | VNS | RVNS |
| 10 | 1 | 1 | 1 | 1 | 1 |
| 20 | 1 | 1 | 1 | 1 | 1 |
| 30 | 0.99441 | 1 | 1 | 1 | 1 |
| 40 | 0.98297 | 0.99234 | 1 | 1 | 1 |
| 50 | 0.92642 | 0.99641 | 1 | 1 | 1 |
| 60 | 0.75907 | 0.99308 | 1 | 1 | 1 |
| 70 | 0.71672 | 0.99956 | 1 | 1 | 1 |
| 80 | 0.58044 | 0.99289 | 1 | 1 | 1 |
| 90 | 0.45588 | 0.98964 | 1 | 1 | 1 |
| 100 | 0.39074 | 0.98332 | 1 | 1 | 1 |
| Average: | 0.78066 | 0.99472 | 1 | 1 | 1 |
| Computational running times (sec) | | | | | |
| n | RHC | GRASP | SA | VNS | RVNS |
| 10 | 4.81 | 0.21 | 0.31 | 0.24 | 0.04 |
| 20 | 666.37 | 11.18 | 1.71 | 7.48 | 0.42 |
| 30 | 2749.09 | 1.12 | 7.93 | 9.95 | 0.86 |
| 40 | 3272.73 | 100.32 | 24.66 | 39.01 | 8.41 |
| 50 | 3331.79 | 663.61 | 42.62 | 187.35 | 10.61 |
| 60 | 3411.07 | 517.13 | 181.30 | 180.9 | 38.72 |
| 70 | 3569.81 | 838.13 | 115.68 | 272.49 | 38.86 |
| 80 | 3524.89 | 266.56 | 248.79 | 723.59 | 66.88 |
| 90 | 3419.11 | 871.79 | 255.65 | 570.28 | 101.51 |
| 100 | 3492.53 | 978.05 | 3491.96 | 932.67 | 115.013 |
| Average: | 2744.22 | 424.81 | 437.06 | 292.396 | 38.13 |

Looking at this table, for $n = 10$ and $n = 20$ all the heuristics obtained the exact solution ($S_t = 1$). However, RHC was considerably slower than the other metaheuristics. For $n > 20$, the performance of RHC was extremely poor, obtaining solutions with extremely low quality in very high computational running times. SA, VNS, and RVNS always produced the exact solutions ($S_t = 1$) for all the considered instances of Table 1, in very short computational times. In particular, RVNS was always faster than the other heuristics among all the datasets, indicating an optimal tuning between intensification and diversification of

the search process, while SA was extremely slow for the instance $n = 100$ with a time of 3491.96 sec. The performance of GRASP is between the poor performing RHC and the high performing SA, VNS, RVNS. Solution quality of GRASP decreases as the problem instance increases, but not as bad as for RHC, while computational times are comparable with those of SA. Summarizing, the average values of the normalized tree benefit score of the meta-heuristics among the instances of Table 1, ranking from the best to the worst performing algorithm, are: RVNS = 1, VNS = 1, SA = 1, GRASP = 0.99472, RHC = 0.78066 (in case of ties in the average normalized tree benefit scores, an algorithm is considered better than another if it has a smaller average computational time).

4.2. Testing on examples from nature

In evolutionary biology the timing and origin of the major extant *placental clades* (groups of organisms that have evolved from a common ancestor) continues to fuel debate and research (Rokas et al., 2003). As the complete genomes of various species become available, it has become possible to do whole genome phylogeny (Felsenstein, 1981; Ben-Dor et al., 1998). Traditional phylogenetic methods on individual genes depended on multiple alignment of the related proteins and on the model of evolution of individual amino acids. Neither of these is practically applicable to the genome level. In absence of such models, a method which can compute the shared information between two sequences is useful because biological sequences encode information, and the occurrence of evolutionary events (such as insertions, deletions, point mutations, rearrangements, and inversions) separating two sequences sharing a common ancestor will result in the loss of their shared information (Rokas et al., 2003).

This section considers a study in genomics with DNA sequences of different placental mammalian species, obtained from (Cilibrasi, 2007). The distance matrices from the genomic data were computed by using the automated software method by Cilibrasi and Vitányi (2005, 2006), who downloaded the whole mitochondrial genomes of the placental mammalian species from the GenBank Database on the World Wide Web. Three sets of data with $n = 10$, $n = 24$, and $n = 34$ were considered, with a *max-CPU-time* for the heuristics of one hour. Computational results are reported in Table 2.

Looking at the table, all the heuristics obtained almost the same normalized tree benefit scores. However, as in the previous set of instances, RHC was considerably slower than the other metaheuristics, showing limited intensification and diversification capabilities of the search process. The average values of the normalized tree benefit scores, ranking from

Table 2: Computational results for examples from nature

| Normalized tree benefit score | | | | | |
|-----------------------------------|---------|---------|---------|---------|---------|
| n | RHC | GRASP | SA | VNS | RVNS |
| 10 | 0.99979 | 0.99979 | 0.99979 | 0.99979 | 0.99979 |
| 24 | 0.99575 | 0.99588 | 0.99588 | 0.99588 | 0.99588 |
| 34 | 0.98488 | 0.98782 | 0.98792 | 0.98792 | 0.98792 |
| Average: | 0.99347 | 0.99450 | 0.99453 | 0.99453 | 0.99453 |
| Computational running times (sec) | | | | | |
| n | RHC | GRASP | SA | VNS | RVNS |
| 10 | 6.72 | 0.078 | 1.84 | 0.172 | 0.00 |
| 24 | 934.42 | 16.56 | 6.69 | 4.48 | 2.08 |
| 34 | 3352.01 | 228.702 | 32.78 | 65.28 | 10.61 |
| Average: | 1431.05 | 81.78 | 13.77 | 23.31 | 4.23 |

the best to the worst performing algorithm, are: RVNS = 0.99453, VNS = 0.99453, SA = 0.99453, GRASP = 0.99450, RHC = 0.99347 (again, in case of ties in the average normalized tree benefit scores, an algorithm is considered better than another if it has a shorter average computational time). RHC obtains the worst average normalized tree benefit score, and the worst average computational running time (1431.05 sec). The best performance in terms of solution quality and computational running time is obtained again by RVNS. Figure 3 shows

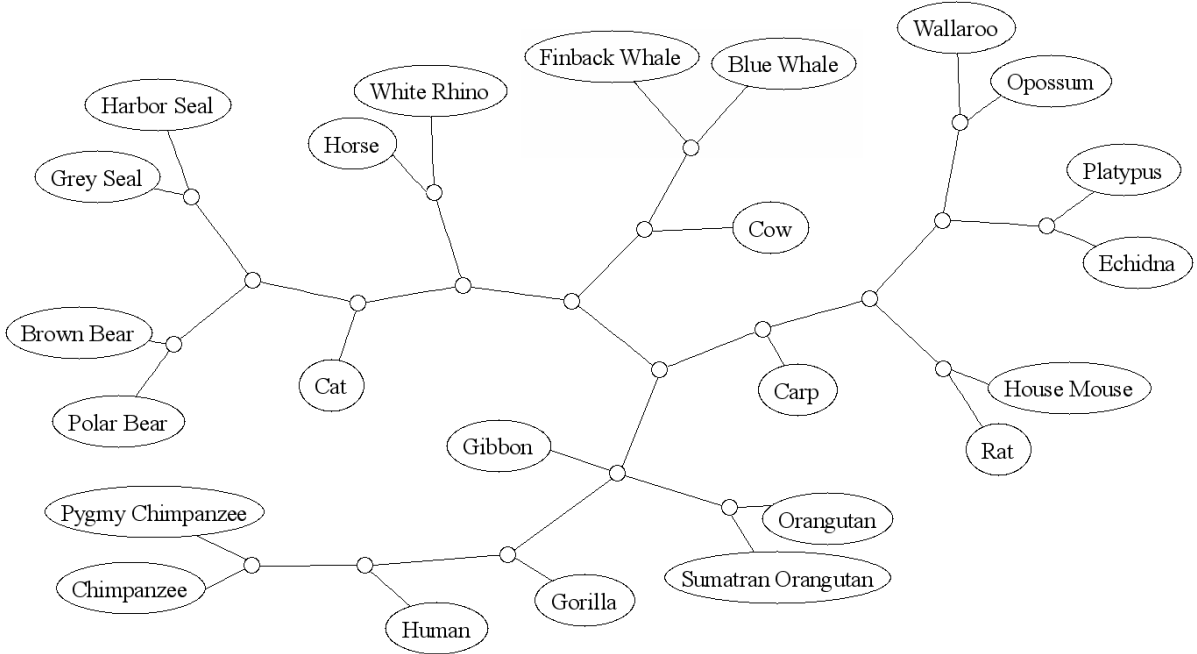


Figure 3: The full unrooted binary tree t obtained by RVNS for the instance with $n = 24$ mammals, with a normalized tree benefit score of $S_t = 0.99588$ obtained in 2.08 sec.

the full unrooted binary tree t obtained by RVNS for the instance with $n = 24$ placental mammals, with a normalized tree benefit score of $S_t = 0.99588$ obtained in just 2.08 sec. The interpretation is that objects in a given subtree are pairwise closer (more similar) to each other than any of those objects in a disjoint subtree. Roughly, it is possible to identify the following groups among the considered placental mammals: *Primates* (Chimpanzee, Pygmy Chimpanzee, Human, Gorilla, Orangutan, Sumatran Orangutan, Gibbon); *Ferungulates* (Grey Seal, Harbor Seal, Brown Bear, Polar Bear, Cat, Horse, White Rhino, Cow, Finback Whale, Blue Whale); *Marsupionta* (Wallaroo, Opossum, Platypus, Echidna, House Mouse, Rat, Carp).

4.3. Testing on geographic distances

In this section, the metaheuristics were compared by considering some famous cities as objects to cluster. Thus, the distances between the objects are real geographic distances between the considered cities, normalized in the interval $[0, 1]$. We considered data instances with a number of objects n from 13 to 37, setting the *max-CPU-time* for the heuristics to one hour. Computational results are presented in Table 3.

Table 3: Computational results for geographic distances between cities

| Normalized tree benefit score | | | | | |
|--|---------|---------|---------|---------|---------|
| n | RHC | GRASP | SA | VNS | RVNS |
| 13 | 0.96843 | 0.96843 | 0.96843 | 0.96843 | 0.96843 |
| 22 | 0.93507 | 0.93507 | 0.93507 | 0.93507 | 0.93507 |
| 24 | 0.92459 | 0.92429 | 0.92459 | 0.92459 | 0.92459 |
| 25 | 0.98760 | 0.98760 | 0.98760 | 0.98760 | 0.98760 |
| 35 | 0.98203 | 0.94395 | 0.98367 | 0.98367 | 0.98367 |
| 37 | 0.90552 | 0.88094 | 0.91973 | 0.91973 | 0.91973 |
| Average: | 0.95054 | 0.94004 | 0.95318 | 0.95318 | 0.95318 |
| Computational running times (sec) | | | | | |
| n | RHC | GRASP | SA | VNS | RVNS |
| 13 | 67.46 | 6.21 | 5.42 | 0.55 | 0.27 |
| 22 | 1365.22 | 198.49 | 15.42 | 17.26 | 3.14 |
| 24 | 803.61 | 311.80 | 15.46 | 17.61 | 3.29 |
| 25 | 1752.89 | 25.36 | 8.98 | 52.81 | 2.84 |
| 35 | 2686.73 | 996.63 | 89.72 | 43.27 | 10.75 |
| 37 | 3434.06 | 1480.82 | 74.94 | 53.53 | 32.94 |
| Average: | 1684.99 | 503.22 | 34.99 | 30.84 | 8.87 |

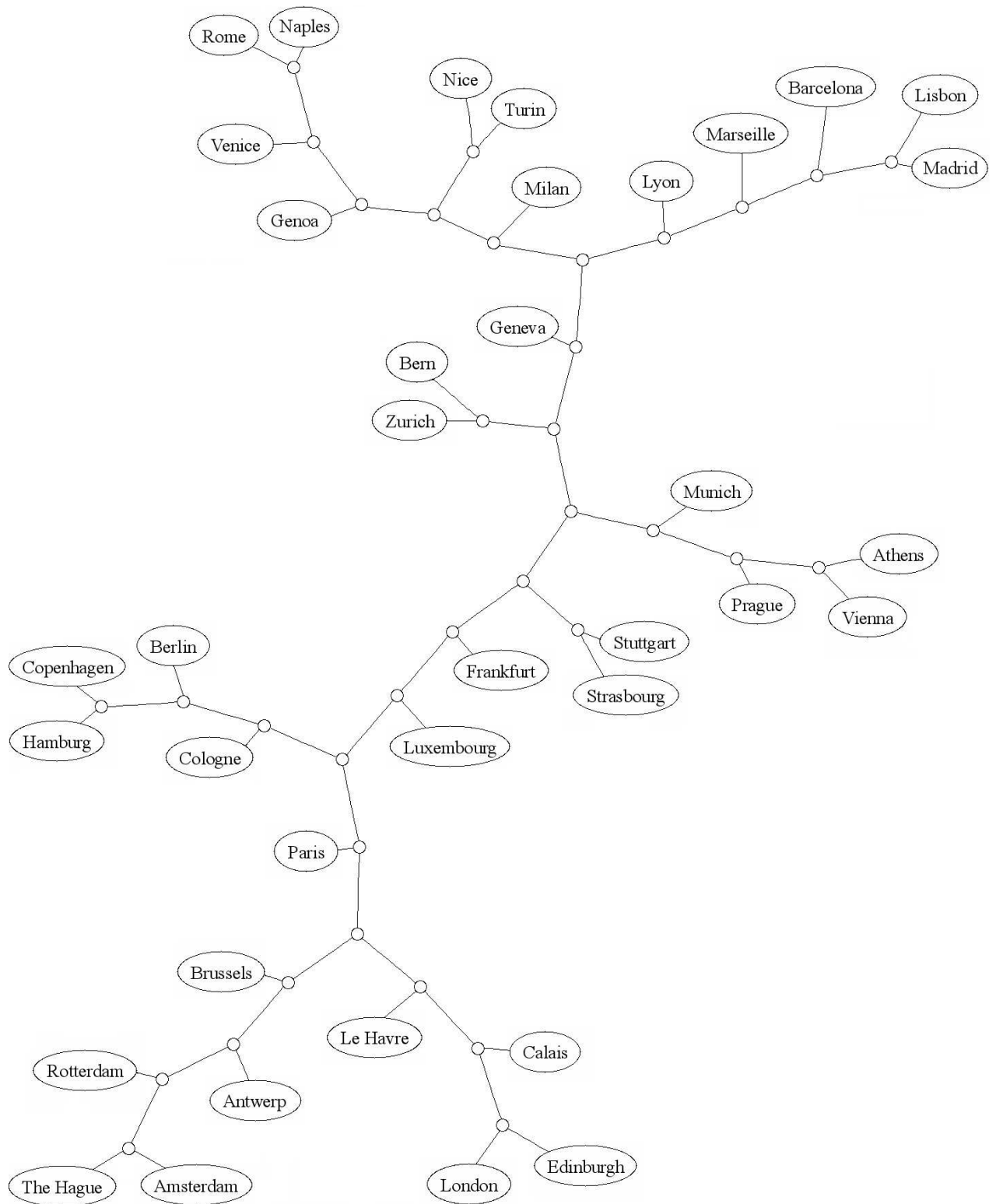


Figure 4: The full unrooted binary tree t with $S_t = 0.91973$ obtained by RVNS in 32.94 sec for the instance with $n = 37$ European cities.

We observe as the normalized tree benefit score obtained by the heuristics deteriorates by increasing the size of the problem instance to cluster (n), as a result of a higher inconsistency produced by the full unrooted binary tree representation of the distance matrices used by the quartet method. The average values of the normalized tree benefit scores, ranking from the best to the worst performing algorithm, are: RVNS = 0.95318, VNS = 0.95318, SA = 0.95318, RHC = 0.95054, GRASP = 0.94004 (as in the previous sections, in case of ties in the average normalized tree benefit scores, an algorithm is considered better than another if it has a shorter average computational time). Again, the best performance are obtained by RVNS, VNS, and SA, which obtain the largest normalized tree benefit scores in the shortest computational running times. In particular, the best performing heuristic is again RVNS, which is considerable faster (average computational time: 8.87 sec) than VNS and SA (average computational times: 30.84 sec and 34.99 sec, respectively). The performance of RHC and GRASP are quite poor. RHC is considerably slower than all the other metaheuristics (average computational time: 1684.99 sec), but it produces slightly better solutions with respect to GRASP in terms of normalized tree benefit score. For these data instances, GRASP produces solutions of poor quality although being faster than RHC, as a result of a poor diversification capability and an excessive intensification capability which sometimes do not allow the search process to escape from local optima.

In Figure 4, the full unrooted binary tree t obtained by RVNS for the instance with $n = 37$, which contains the distances among some famous European cities, is illustrated. The normalized tree benefit score of this example is $S_t = 0.91973$, obtained by RVNS in 32.94 sec. Figure 4 represents an intuitive visual example of the way of clustering data hierarchically by means of the quartet method. Cities that have short relative distances are assigned to close positions of the full unrooted binary tree. For instance, the Italian cities of Rome, Naples, Venice, Genoa, are placed in close positions of t , followed by Nice (that belongs to France but is extremely close to the Italian border) and Turin, and then Milan. Similarly, the Netherlands cities of Amsterdam, The Hague, Rotterdam, Antwerp, and Brussels belongs to the same group, and so on.

4.4. Testing on data extracted from the World Wide Web

In this section, we consider data obtained by mining of the WWW through an automatic web information extraction method by Geleijnse et al. (2006). Specifically, we have focussed

on data concerning musical artists, in order to easily show subjective artist categories such as genre of the music that they produce.

Geleijnse et al. (2006) use the assumption that related artists often share the same category (*working hypothesis*). Alternatively, if two artists are both known for the same category (e.g. romantic music), it is expected that they would occur often in the same context within the World Wide Web. To obtain a metric which expresses the similarity between each pair of artists a and b , selected from a given set of artists A , Geleijnse et al. (2006) count the number of *co-occurrences* of a and b , $co(a, b)$, within the WWW by means of, either a *Page-count-based mapping* (PCM), a *Pattern-based mapping* (PM), or a *Document-based mapping* (DM). In this paper a PCM is used, where the number of co-occurrences of a pair of artists $(a, b) \in A$ is the number of Google hits for queries “ a ”, “ b ”. Geleijnse notes that the estimated numbers of Google hits can fluctuate which may lead to unexpected results.

After having collected the number of co-occurrences for each pair of artists in A , Geleijnse derives a similarity metric among the artists by defining a scoring function, $T(a, b)$, between two artists $(a, b) \in A$, as follows:

$$T(a, b) = \frac{co(a, b)}{1 + \sum_{y \in A, y \neq a} co(a, y) \cdot \sum_{x \in A, x \neq b} co(x, b)}. \quad (4)$$

This similarity metric is inspired by the theory of “pointwise mutual information” (for more details see (Manning and Schütze, 1999)). It is symmetric and lies between 0 and 1. The value 1 represents the highest similarity measure between two artists; as the metric between two objects approaches zero, the less the similarity between the two artists. All the elements in the diagonal of the distance matrix are equal to 1. For each $(a, b) \in A$, the similarity metric $T(a, b)$ is converted into a distance metric $d(a, b) = 1 - T(a, b)$. In this way, a symmetric distance matrix, suitable input for the quartet method, is produced.

Our results are presented in Table 4, which considers data instances with number of artists n from 15 to 224. For small problem instances ($n \leq 100$), *max-CPU-time* for the heuristics is set to one hour, while for the last two large instances with $n > 100$, (i.e. $n = 150$ and $n = 224$), a *max-CPU-time* of 10 hours is imposed. The average values of the normalized tree benefit scores, ranking from the best to the worst performing algorithm, are: RVNS = 0.88990, SA = 0.88788, VNS = 0.88689, GRASP = 0.84860, RHC = 0.64829; while the average computational running times, from the fastest to the slowest, are (in sec): GRASP = 6524.97, RVNS = 9019.74, SA = 10278.07, VNS = 10442.23, RHC = 10767.61.

Table 4: Computational results for data concerning distances between musical artists extracted from the World Wide Web

| Normalized tree benefit score | | | | | |
|--|----------|----------|----------|----------|----------|
| n | RHC | GRASP | SA | VNS | RVNS |
| 15 | 0.95273 | 0.95273 | 0.95273 | 0.95273 | 0.95273 |
| 25 | 0.92218 | 0.92080 | 0.92218 | 0.92190 | 0.92218 |
| 50 | 0.75077 | 0.90511 | 0.92244 | 0.92244 | 0.92252 |
| 100 | 0.43476 | 0.85988 | 0.88736 | 0.88212 | 0.88731 |
| 150 | 0.42591 | 0.74047 | 0.84214 | 0.84132 | 0.84614 |
| 224 | 0.40341 | 0.71262 | 0.80045 | 0.80080 | 0.80849 |
| Average: | 0.64829 | 0.84860 | 0.88788 | 0.88689 | 0.88990 |
| Computational running times (sec) | | | | | |
| n | RHC | GRASP | SA | VNS | RVNS |
| 15 | 41.21 | 1.25 | 1.63 | 0.78 | 0.25 |
| 25 | 1107.81 | 12.68 | 13.13 | 19.13 | 2.34 |
| 50 | 3469.89 | 103.51 | 65.79 | 112.19 | 35.37 |
| 100 | 3525.28 | 94.91 | 3033.53 | 2735.55 | 884.31 |
| 150 | 34809.11 | 4580.45 | 34261.51 | 24425.52 | 17896.95 |
| 224 | 21652.34 | 34357.02 | 24292.81 | 35360.20 | 35299.22 |
| Average: | 10767.61 | 6524.97 | 10278.07 | 10442.23 | 9019.74 |

As in the previous experimental analysis, the table shows approximately the same relative behaviour for all the considered metaheuristics. RVNS obtains the solutions with the best normalized tree benefit scores, followed by SA and VNS, then GRASP, and finally RHC, which produces extremely poor results (average normalized tree benefit score: 0.64829). In addition, the computational running times (average computational time: 10767.61 sec) are poor. For the data instances considered in this section, GRASP is on average faster than the other algorithms, because it converges prematurely to local optima from where it is not able to escape, producing solutions of poor quality. SA and VNS produce results close to those of RVNS in terms of solution quality and computational running times, indicating an optimal tuning between intensification and diversification of the search process, which evidently is not obtained by GRASP and RHC. VNS obtains slightly worse solutions than those obtained by SA, perhaps lacking a bit in terms of exploration of the search space with respect to the SA approach. As in Section 4.3, it is interesting to note the effect of the data inconsistency in the normalized tree benefit score obtained by the heuristics as n becomes larger. For example, for $n = 224$, it is not possible to produce a solution having normalized tree benefit score larger than 0.80849, that is obtained by RVNS in a very high

computational time (35299.22 sec)! This further analysis underlines the limit of the quartet method to process data instances larger than, approximately, $n = 100$ objects to cluster. For $n > 100$, the heuristics often produce results with inadequate normalized tree benefit scores in very high computational running times.

Summarizing, for all the considered problem instances containing objects to cluster of different nature, analysed in Sections 4.1 - 4.4, all the metaheuristics that we propose (RVNS, VNS, SA, GRASP) clearly outperformed the Randomized Hill Climbing by Cilibrasi and Vitányi (2005, 2006), the heuristic recommended in the literature for the quartet method of hierarchical clustering. In particular, the best performance in terms of normalized tree benefit score and computational running time were obtained by RVNS. This is the most effective heuristic for the Minimum Quartet Tree Cost problem. As shown in our experiments, RVNS is able to produce the most accurate full unrooted binary trees, capable of representing the symmetric distance matrices. From our analysis, it has been shown that our Reduced Variable Neighbourhood Search is fast and particularly effective for the quartet method of hierarchical clustering.

5. Conclusions

In this paper we considered the quartet method of hierarchical clustering which, given a set of objects to be classified and a symmetric distance matrix containing their pairwise distances, produces an optimal hierarchy of the objects without knowing a priori the number of clusters to be produced. The optimal hierarchy produced by the quartet method is visualized by means of a full unrooted binary tree, which visually represents the distance matrix as closely as possible, according to a specified cost evaluation.

In order to produce the optimal hierarchy through a full unrooted binary tree, the quartet method of hierarchical clustering needs to solve a graph optimization problem, called the Minimum Quartet Tree Cost problem. A Greedy Randomized Adaptive Search Procedure, a Simulated Annealing approach, a Variable Neighbourhood Search, and a Reduced Variable Neighbourhood Search have been presented for this problem. Considering a wide range of problem instances, we compared these metaheuristics with the Randomized Hill Climbing by Cilibrasi and Vitányi (2005, 2006), the most popular heuristic in the literature for the quartet method of hierarchical clustering. Based on this experimental analysis, all the proposed procedures clearly outperformed the Randomized Hill Climbing and, in particular,

the best performance was obtained by Reduced Variable Neighbourhood Search. Reduced Variable Neighbourhood Search was shown to be a fast, simple, and particularly effective metaheuristic for the quartet method of hierarchical clustering, obtaining high-quality solutions in short computational running times. This analysis provides further evidence of the ability of variable neighbourhood heuristics to deal with NP-hard combinatorial problems.

References

- Battiti, R., M. Brunato, F. Mascia. 2008. *Reactive search and intelligent optimization, Operations Research/Computer Science Interfaces Series*, vol. 45. Springer-Verlag, New York.
- Ben-Dor, A., B. Chor, D. Graur, R. Ophir, D. Pelleg. 1998. Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships. *Journal of Computational Biology* **5** (3) 377–390.
- Berry, V., T. Jiang, P. Kearney, M. Li, T. Wareham. 1999. Quartet cleaning: Improved algorithms and simulations. H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel, eds., *Algorithms - Proceedings 7th European Symposium on Algorithms (ESA'99), Lecture Notes in Computer Science*, vol. 1643. Springer-Verlag, Berlin, Germany, 313–324.
- Cilibrasi, R. 2007. The Complearn toolkit. [online]. URL <http://www.complearn.org/>.
- Cilibrasi, R., P. M. B. Vitányi. 2005. Clustering by compression. *IEEE Transactions on Information Theory* **51** (4) 1523–1545.
- Cilibrasi, R., P. M. B. Vitányi. 2006. A new quartet tree heuristic for hierarchical clustering. D. V. Arnold, T. Jansen, M. D. Vose, J. E. Rowe, eds., *Theory of Evolutionary Algorithms. Dagstuhl Seminar Proceedings, Dagstuhl, Germany*. URL <http://drops.dagstuhl.de/opus/volltexte/2006/598>.
- Consoli, S. 2008. Test datasets for the quartet method of hierarchical clustering. [online]. URL <http://www.sergioconsoli.com/Quartet.htm>.
- Diestel, R. 2000. *Graph theory*. Springer-Verlag, New York.
- Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution* **17** (6) 368–376.

- Feo, T. A., M. G. C. Resende. 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* **8** 67–71.
- Furnas, G. W. 1984. The generation of random, binary unordered trees. *Journal of Classification* **1** (1) 187–233.
- Geleijnse, G., J. Korst, V. de Boer. 2006. Instance classification using co-occurrences on the web. *Proceedings of the ISWC 2006 workshop on Web Content Mining (WebConMine)*. Athens, GA. URL <http://www.dse.nl/~gijsg/webconmine.pdf>.
- Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* **13** 533–549.
- Glover, F., G. A. Kochenberger. 2003. *Handbook of metaheuristics*. Kluwer Academic Publishers, Norwell, MA.
- Hansen, P., N. Mladenović. 2003. Variable neighbourhood search. F. Glover, G. A. Kochenberger, eds., *Handbook of metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 145–184.
- Jiang, T., P. Kearney, M. Li. 2000. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM Journal on Computing* **30** (6) 1942–1961.
- Kaufman, L., P. J. Rousseeuw. 2005. *Finding groups in data: An introduction to cluster analysis (Wiley Series in Probability and Statistics)*. John Wiley & Sons, Chichester.
- Kirkpatrick, S., C. D. Gelatt, M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* **220** 671–680.
- Manning, C. D., H. Schütze. 1999. *Foundations of statistical natural language processing*. The MIT Press, Cambridge, MA.
- Mladenović, N., J. Petrović, V. Kovačević-Vujčić, M. Čangalović. 2003. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research* **151** (2) 389–399.
- Osman, I. H. 1993. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research* **41** 421–451.

- Rokas, A., B. L. Williams, N. King, S. B. Carroll. 2003. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature* **425** (6960) 798–804.
- Steel, M. A. 1992. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification* **9** 91–116.
- Strimmer, K., A. von Haeseler. 1996. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution* **13** (7) 964–969.
- Weyer-Menkoff, J., C. Devauchelle, A. Grossmann, S. Grünewald. 2005. Integer linear programming as a tool for constructing trees from quartet data. *Computational Biology and Chemistry* **29** (3) 196–203.