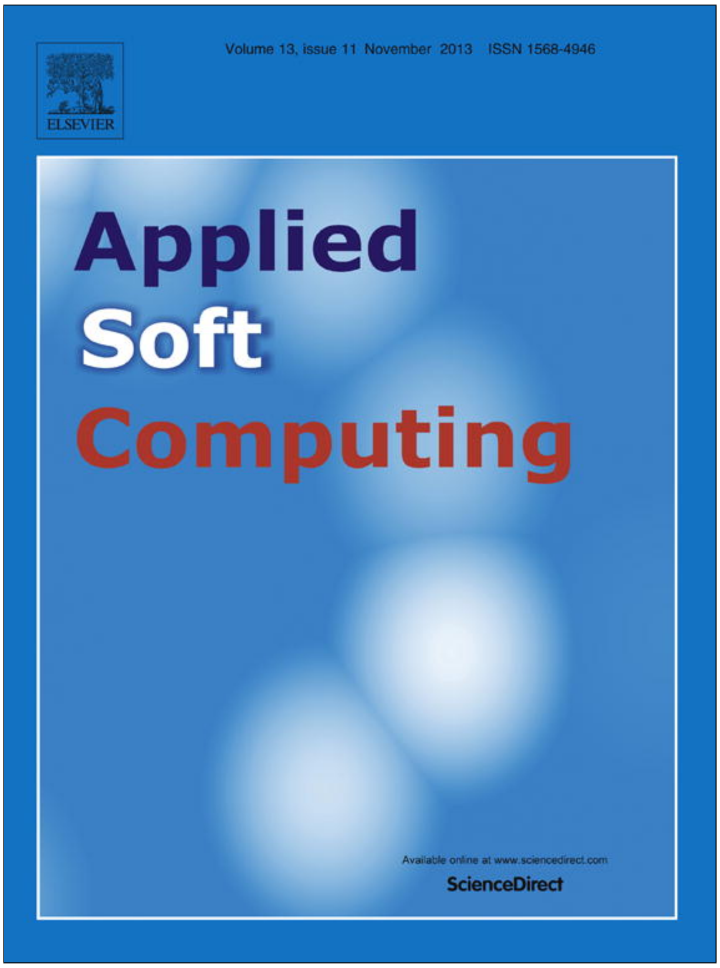

Volume 13, issue 11 November 2013 ISSN 1568-4946
ELSEVIER
Applied
Soft
Computing
Available online at www.sciencedirect.com
ScienceDirect

Applied Soft Computing 13 (2013) 4303–4319

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A traffic-based evolutionary algorithm for network clustering

Maurizio Naldi[a,*,1], Sancho Salcedo-Sanz[b,2], Leopoldo Carro-Calvo[b,2], Luigi Laura[a,1], Antonio Portilla-Figueras[b,2], Giuseppe F. Italiano[a,1]

[a] *Università di Roma "Tor Vergata", Dipartimento di Ingegneria Civile e Ingegneria Informatica, Via del Politecnico 1, 00133 Rome, Italy*
[b] *Department of Signal Theory and Communications, Universidad de Alcalá, Madrid, Spain*

ARTICLE INFO

Article history:
Received 27 September 2012
Received in revised form 23 April 2013
Accepted 19 June 2013
Available online 5 July 2013

Keywords:
Clustering
Traffic matrices
Genetic algorithms

ABSTRACT

Network clustering algorithms are typically based only on the topology information of the network. In this paper, we introduce traffic as a quantity representing the intensity of the relationship among nodes in the network, regardless of their connectivity, and propose an evolutionary clustering algorithm, based on the application of genetic operators and capable of exploiting the traffic information. In a comparative evaluation based on synthetic instances and two real world datasets, we show that our approach outperforms a selection of well established evolutionary and non-evolutionary clustering algorithms.

1. Introduction

Clustering, organizing a collection of items into groups on the basis of their similarity, is a well known problem in many different areas. Its applications span fields as different as image segmentation, object and character recognition, customer classification for marketing, and genomic, just to name a few. In the two survey papers by Jain, written more than ten years apart, we can recognize the expansion of clustering, both in methods and in applications [1,2].

Graphs also can be clustered into groups of nodes, each cluster including vertices that are strongly interconnected among them: there should be many edges within each cluster and relatively few between clusters [3]. Graphs can represent a number of interrelated entities. For example, when a graph represents a social network, the vertices are individuals, and the edges represent relationships among them. The problem of detecting communities of individuals within the whole society is itself a clustering problem. In the case of a communications network, the vertices represent network nodes (routers or other switching devices), and the edges represent transmission links. In the following, we refer to nodes and links rather than vertices and edges.

In spite of its importance, the problem of network clustering has been approached so far mainly by considering topology information only. The criterion employed is then based on the number of links lying respectively inside a cluster and among different clusters: the relation between two nodes is entirely embodied by their sharing a link. Adding a weight to each link (so that the relation between two nodes may be stronger or weaker) recognizes the relevance of the intensity of the relationship, but, even in weighted networks, two nodes are related only if they are connected by a direct link. Variations of this approach consider the energy consumption (related to the physical distance) to minimize the total consumption in a wireless network [4,5].

In [6], we put forward the use of traffic information to cluster the nodes of a network. Such information is contained in the traffic matrix of a network, and represents the actual intensity of the communication between two nodes, regardless of the network topology and the route employed to get the messages from the sender to the receiver: the more two nodes communicate between them, the larger their traffic is. The range of applications in which adding traffic information should lead to improvements is very wide: basically all networks in which traffic does not flow exclusively between neighboring nodes. For example, that's been shown in [7] for telephony traffic. Another example is given by social networks in which many relationships are indirect, and an individual is used as a transfer means to convey information (or any other mode of

* Corresponding author. Tel.: +39 0672597269.
E-mail addresses: naldi@disp.uniroma2.it (M. Naldi), sancho.salcedo@uah.es (S. Salcedo-Sanz), Leopoldo.Carro@uah.es (L. Carro-Calvo), laura@dis.uniroma1.it (L. Laura), antonio.portilla@uah.es (A. Portilla-Figueras), italiano@disp.uniroma2.it (G.F. Italiano).

[1] The work of Maurizio Naldi, Luigi Laura, and Giuseppe F. Italiano has been partially supported by the Italian Ministry of Education, University, and Research through the ALGODEEP Project, and by the European Union under the EuroNF Network of Excellence.

[2] The work of Sancho Salcedo-Sanz, Leopoldo Carro-Calvo, and José Antonio Portilla has been partially supported by the Spanish Ministry of Science and Innovation, under a project number ECO2010-22065-C03-02.

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.06.022

relationship, so that traffic is meant here in a broad sense as anything that is exchanged by two nodes) between two other parties (see, e.g., [8] for the discovery of hidden relationships and [9] for gossip networks). In [6], we adapted two quality metrics from the context of topology-based clustering algorithms, to make them applicable in a traffic-based approach, namely the *Traffic-aware Scaled Coverage Measure* and the *Modularity* measure. In [10], we proposed a preliminary version of an evolutionary clustering algorithm, and performed a first comparison against the Spectral Filtering algorithm, a major non-evolutionary clustering algorithm.

In this paper we fully embrace the traffic-based approach for network clustering, and propose a novel evolutionary algorithm based on the use of genetic operators, which we name EC (Evolutionary Clustering). We embed the quality metrics recalled above in the fitness function of the evolutionary procedure, so that our algorithm aims at maximizing the quality of the clustering solution as evaluated through those metrics. We have tested our approach against four competing topology-based clustering algorithms and against an existing evolutionary approach (EvoCluster [11]) on synthetic and real world datasets. We fully describe our evolutionary algorithm and report the results of that comparative evaluation. We compare the first two statistical moments of the two metrics (which represent a measure of central tendency and dispersion), while in previous works the comparison was limited to the scatterplot of the metrics. In this paper, we also compute the percentage of success of the Evolutionary Algorithm against its non-genetic competitors. The comparison performed in this paper is completed by a thorough analysis of the computational cost. The key results obtained in this work are: (1) We show that our evolutionary algorithm achieves better values of both quality metrics than the topology-based alternative clustering algorithms; (2) We provide an analysis of the computational cost of the EC algorithm; (3) we show that its computational cost is lower than that of the topology-based competitors (excepting K-means). We have also performed a comparison with the above selection of non-evolutionary algorithms and the evolutionary algorithm EvoCluster, using a synthetic dataset with larger traffic matrices. Also in the case of synthetic matrices, our EC algorithm outperforms all the other algorithms, with the only exception of Newman's and K-means for the larger traffic matrices when the modularity metric is used.

The rest of the paper is organized as follows. In Section 2 we recall the notion of traffic matrix and its use in the context of network clustering. In Sections 3 and 4 we describe respectively our traffic-based Evolutionary algorithm and the topology-based competitors employed in our comparative evaluation. Sections 5.1 and 5.2 are devoted to set the performance evaluation context, respectively through the definition of the quality metrics and the description of the real world datasets. Finally, in Sections 5.3–5.5, we describe the results of the comparison, under the two viewpoints of the quality of the clustering solutions (for the real world datasets and the synthetic one) and the computational cost.

2. Traffic information and clustering

Network clustering is traditionally performed on the basis of topology information. Roughly speaking, two nodes belong to the same cluster if they are strongly interconnected. In this paper, we advocate instead the use of traffic information to partition the network into clusters. In this section, we review the tools that gather respectively the topology information used for clustering in the traditional approach, and the traffic information employed in ours; we compare them, and provide motivations for the use of traffic information.

When we cluster a network on the basis of the connectivity information only, we employ the adjacency matrix A. The generic element A_{ij} of that matrix equals 1 if the nodes i and j are connected by a link, and 0 otherwise. When links are bidirectional (which is usually the case in communications networks), the adjacency matrix is symmetric. Though two nodes may be considered strongly related if they are directly connected, clustering based on the adjacency matrix fails to consider the case where two nodes have a strong relationship even if they are topologically distant.

For this reason, we introduce traffic matrices as the basis for network clustering. Traffic represents the intensity of the relation between two nodes, regardless of the way those nodes are connected. Nodes that communicate heavily between them, as indicated by the traffic matrix, should be put into the same cluster, though they are not directly connected. In a traffic matrix X, the element X_{ij} provides the traffic originated by node i and destined for node j. Despite the use of the term *traffic*, there are several possibilities as to the actual quantity used to represent traffic. In [12], a two-level taxonomy of traffic matrices is proposed, based on the spatial representation of network traffic used and the aggregation level for the sources and destinations engaging in traffic exchanges. In addition, we may consider either intensity values (averages over a measurement time window, typically an hour long) or volume values (accumulated over a typically much longer observation window, e.g., over a month), depending on the purpose of the traffic matrix [13]. The resulting matrix is generally asymmetric, even for a network with all bidirectional links. Their asymmetry makes methods employed for weighted networks unsuitable, since they typically assume a symmetric weight matrix [14]. Traffic matrices are dense, usually complete, as opposed to the usually sparse structure of adjacency matrices. The elements of traffic matrices are real numbers, rather than Boolean values. In addition, they often vary considerably even over small time frameworks, while the topology is much stabler, with changes due typically to failures or planned interventions. Contrary to adjacency matrices, traffic matrices are independent of the internal topology of the network. Moreover, when the nodes i and j are supposed to be respectively the ultimate source and destination of that traffic, the traffic matrix is also insensitive to routing changes, which represents a further advantage in their use and a spur to estimate them as accurately as possible [15,16].

3. The evolutionary clustering algorithm

The main objective of this paper is to introduce a new evolutionary algorithm to perform a partitional clustering of a network on the basis of traffic matrices and a fitness function that describes the quality of the clustering solution. By partitional clustering we mean an approach where each node is assigned to a single cluster: clusters do not overlap and represent a partition of the network. In this section, we describe that algorithm.

A primary issue in any clustering algorithm is the choice of the number of clusters. Some algorithms need it to be defined a priori, while others include the number of clusters as a variable to be optimized during the clustering process, jointly with the composition of each cluster. The review in [17] adopts that feature to classify clustering algorithms into two classes: algorithms with either a fixed or a variable number of clusters.

Our algorithm does not require the number of clusters to be decided a priori. Rather, the candidate solutions can be composed of different numbers of clusters, so that it can be considered to belong in the latter class. Nevertheless, our algorithm requires the maximum number k^* of clusters to be set, so that the candidate solutions can be made of any number of clusters in the range between one and that maximum. The number of clusters $k \leq k^*$ is therefore an outcome of the algorithm.

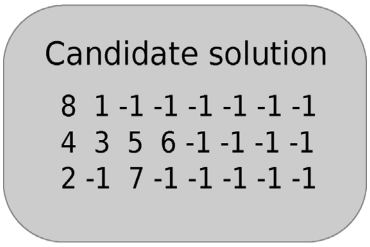
In order to obtain a partition of the network into k clusters, our algorithm goes iteratively through the steps listed in Algorithm 1 and described in detail in the following. The algorithm stops when there has been no change in the clustering solution in the latest 20 iterations. The steps followed by the algorithm may be arranged into the classical phases of evolutionary algorithms [18]: an initialization phase, followed by the repeated application of operators (*Selection*, *Crossover*, and *Mutation*) and the evaluation of the fitness function. In the following, we define the format of candidate solutions (the encoding) and describe in detail the three evolutionary operators. We postpone the description of the fitness function to Section 5.1.

Algorithm 1 *(Evolutionary Clustering (EC)).*

```
Input: A set of $n \times n$ traffic matrices $X$
Output: A partition of the $n$ nodes into $k$ clusters.
1:  Define an initial set of candidate solutions.
2:  repeat
3:    Compute the fitness function for all candidate solutions.
4:    Compute the average value of the fitness function.
5:    Count the number $N_{less}$ of candidate solutions with
      lower-than-average values of the fitness function and remove
      them.
6:    for $j = 1$ to $N_{less}$ do
7:      Select randomly two parent solutions among the survived
        ones.
8:      Copy the matrix of the parent A in the child solution.
9:      Select randomly 50% of the elements of parent B and keep only
        the non empty elements.
10:     Find in the child matrix the selected nodes in B and replace
        them with $-1$.
11:     Add the selected nodes to the child in the same cluster they
        belong to in parent B.
12:   end for
13:   Select all the children generates in Steps 6–12.
14:   Swap two random positions in two different clusters in each
      candidate solution identified in Step 13.
15: until Convergence
16: Select the candidate solution exhibiting the largest value of the
    fitness function.
```

3.1. Encoding

The first issue to be solved is the encoding used to represent each candidate solution (or genotype), i.e., the assignment of nodes to clusters. A number of alternatives are indicated in [17]. We have opted for an integer encoding, where each possible assignment of the n nodes to k clusters is represented as a $k \times n$ matrix. In our encoding approach, each cluster in a genotype is represented individually and is described by a string of n integers taking values on the alphabet $\{1, \ldots, n\} \cup \{-1\}$. The values in the former set represent the node ID. The string representing a cluster shows therefore the nodes belonging to that cluster through their ID. Since a cluster can include at most $n - 2k + 2$ nodes (if we exclude both empty and singular clusters), each row of the encoding matrix will have at least $2k - 2$ empty positions (i.e., not assigned to a node), which are filled by the -1 value. This encoding approach employs $k \times n$ positions, while other more efficient approaches can use n (e.g., the label-based encoding where the position i in a string of length n provides the cluster to which node i belongs). In addition, it is of the one-to-many type (each candidate solution can be represented in $k!n!$ equivalent ways). However, it has the advantage of showing immediately the composition of each cluster. An example is shown in Fig. 1, where a candidate solution with 3 clusters for a network of 8 nodes is shown, with Cluster 1 being composed of nodes 1 and 8. Since our clustering has to represent a partition of the network, each node has to appear in a single position over the whole $k \times n$ matrix, and there have to be $n(k - 1)$ empty positions (i.e., filled with -1 values).



Candidate solution

8 1 -1 -1 -1 -1 -1 -1
4 3 5 6 -1 -1 -1 -1
2 -1 7 -1 -1 -1 -1 -1

Fig. 1. Example of solution encoding.

3.2. Selection

The first evolutionary operator to be applied to the population of candidate solutions is the *selection* one, which preserves a fraction of the current group of parents and children to be used as parents for the next generation of the algorithm. Several procedures have been proposed in the literature for that purpose [17]. Here we have opted for the *truncation selection* method. This method employs the fitness function and applies a deterministic procedure, by saving for the future steps just the candidate solutions exhibiting the highest fitness. Namely, in the truncation selection approach we go through the following steps: (1) the fitness function is computed for each candidate solution; (2) the candidate solutions are ordered according to their value of fitness from the largest (best fit) to the smallest (worst fit); (3) the top τ candidate solutions are retained. These steps corresponds to Steps 3–5 of Algorithm 1. The truncation selection method is known to reduce strongly the variance of the population's fitness [19], but, when employed with a large value of τ, guarantees at the same time that the population is wide enough and that the best individuals are preserved. Here we have employed a variant of truncation selection where the value of τ is not set a priori, but is rather such that just the individuals exhibiting a larger-than-average value of the fitness function are retained. When the distribution of values of the fitness function over the set of candidate solution is symmetric, this is tantamount to setting τ equal to 50% of the population, as in [20].

3.3. Crossover

In order to replenish the set of candidate solutions after the trimming due to the selection phase, we replace the missing candidates through the *crossover* operator. With crossover, two candidate solutions (parents, which hereafter we call respectively Parent A and Parent B) produce one new candidate solution (child), which is added to the set of candidate solutions. The parents are chosen at random among the individuals in the population of survivors after the selection operator, using a simple sampling-with-replacement scheme: a couple of parents are selected from an ideal urn and returned to the urn before the next extraction. Such sampling scheme is uniform over the set of candidate solutions, and allows for multiple instances of any candidate solution in the mated couples. The number of children generated at this step is exactly equal to the number of individuals eliminated in the selection phase. In Fig. 2, we show an example of the application of our crossover operator as described through Steps 6–12 of Algorithm 1. The child is first generated as a copy of Parent A (Step 1), but 50% of the locations inside Parent B are then chosen randomly, and the nodes appearing in the non-empty locations (i.e., filled with values in the $\{1, \ldots, n\}$ set) are retained (Step 2). In the first version of the child, the locations containing the nodes identified in Step 2 are then emptied, i.e., filled with -1 values (Step 3). At this point the child contains all the nodes excepting those identified in Step 2 (coming from Parent B), which are then reinserted in the clusters to which they belonged in Parent B (Step 4).

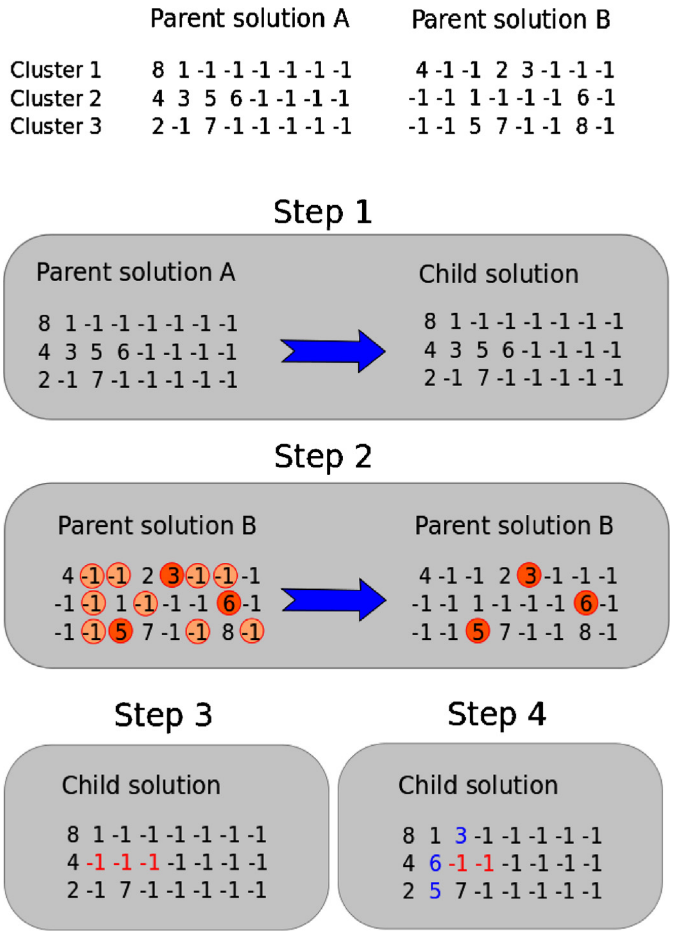


Fig. 2. Example of crossover.

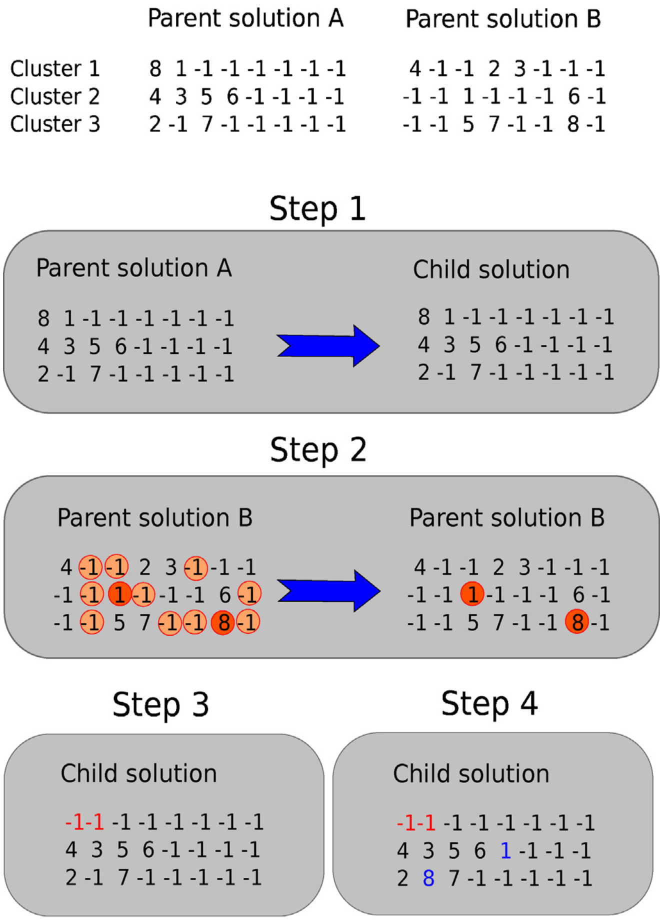


Fig. 3. Example of solution with a lower number of clusters.

Note that the application of the crossover operator (precisely, Steps 3 and 4 in Fig. 2) may result in a candidate solution with a number of clusters strictly lower than the maximum ($k < k^*$). In fact, in Step 3 a cluster may get completely empty, while the subsequent reinsertion of nodes at Step 4 takes place on other clusters. An example is provided in Fig. 3, where the candidate solution resulting after crossover is made of just two clusters, Cluster 1 having been emptied. This represents an opportunity to explore candidate solutions with a lower number of clusters, so that the search space is significantly enlarged with respect to algorithms employing a fixed number of clusters.

3.4. *Mutation*

After the crossover operator, a mutation operator is applied to generate more diversity in the population and avoid that the algorithm remains stuck in local minima of the objective function. A swap-type operator is used (see Fig. 4): each candidate solution among the children generated in the Crossover phase is selected in turn, and two nodes of that solution are swapped (each one is assigned to the cluster of the other) to generate a mutated individual. In the example of Fig. 4, node 7 is moved from Cluster 3 to Cluster 2. In the process, Cluster 3 remains as a degenerate cluster, made of a single node.

4. Alternative algorithms

The performance of our evolutionary algorithm has to be compared against alternative algorithms. In this section we provide a brief description of all those algorithms we have considered in our comparative evaluation. We have considered the same algorithms as in [6]:

- *K*-means;
- Fast Singular Value Decomposition;
- Newman;
- Spectral Filtering.

4.1. *The K-means algorithm*

The *K*-means algorithm is probably the best-known clustering algorithm [21,22]. Each item to be clustered is represented by a vector of attributes, each cluster has a centroid (whose vector of attributes is the average of the vectors pertaining to the items belonging to that cluster), and each item is assigned to the cluster according to the distance from its centroid.

In our case, the items to be clustered are the nodes. The vector of attributes for each node is obtained as the corresponding row vector of the two-way traffic matrix $M = X + X^T$, whose element m_{ij}

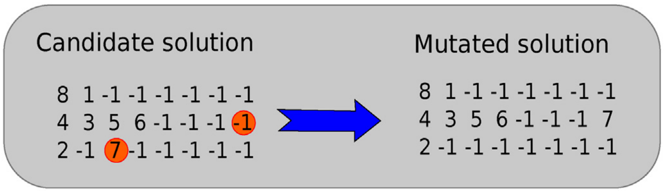


Fig. 4. Example of mutation.

represents the overall traffic exchanged between the two nodes i and j. For a network of n nodes, the vector of attributes is made of n components. The distance between any two nodes can be computed as the Euclidean distance between their vectors of attributes, excluding from the computation the elements containing the traffic exchanged between the two nodes.

For each cluster, a clusterhead can be determined, which is the node having the least average distance from the other nodes in its cluster (a.k.a. the centroid). The aim of the K-means algorithm is to assign nodes to clusters so that the within-cluster sum of distances to the clusterhead is minimized. The algorithm is carried out in an iterative fashion, since each assignment modifies the clusterhead. The algorithm stops when no cluster changes are needed: each node is assigned to the cluster whose clusterhead is the nearest to it.

The pseudo-code of the K-means algorithm is provided as Algorithm 2.

Algorithm 2 *(The K-means algorithm).*

```
Input: A traffic matrix $X$
Input: The number of clusters $k$
Output: A partition of the $n$ nodes into $k$ clusters.
Compute the two-way traffic matrix $M = X + X^T$
Randomly choose $k$ nodes and make them the $k$ clusterheads
repeat
  for $i = 1$ to $n$ do
    Compute the Euclidean distance from node $i$ to each of the clusterheads
    Assign node $i$ to the nearest clusterhead
  end for
  for $j = 1$ to $k$ do
    Compute the centroid for cluster $j$ and make it the new clusterhead
  end for
until Convergence
```

4.2. Derandomized fast singular value decomposition

Drineas et al. proposed the use of Singular Value Decomposition as a clustering tool in [23], through the relaxation of the original discrete problem. The aim is to minimize the sum of the squared distance from each point to its cluster center (equivalent to minimizing the variance of clusters). While the discrete version is known to be $\mathcal{NP}$-hard, the latter can be solved efficiently using a projection onto the top k left singular values, obtained by computing the Singular Value Decomposition (SVD) of the adjacency matrix. As the computation of SVD for large matrices can be time and space consuming, Drineas et al. proposed a heuristic, called Fast SVD, to speed up this algorithm by calculating the SVD only on a random column-sampled submatrix.

We considered a derandomized version of Fast SVD (hence DF-SVD) where the column sampling just selects the top c columns with largest weights. The pseudo-code is given in Algorithm 3.

Algorithm 3 *(Deterministic and Fast SVD (DF-SVD)).*

```
Input: A traffic matrix $X$ for a network of $n$ nodes, an integer $c \le n$, an integer $k \le c$.
Output: A partition of the $n$ nodes into $k$ clusters.
{ Phase 1: Column Sampling }
for $i = 1$ to $n$ do
  Compute the column weights $w_i = |X^{(i)}|^2 / ||X||_F^2$
end for
Sort the weights $w_i$
Let $B$ be the matrix consisting of the $c$ columns of $X$ with the largest weights
{ Phase 2: Build Clusters }
Compute the SVD of matrix $B$
Find the top $k$ left singular vectors $u_1, \ldots, u_k$
Let $C$ be the matrix whose $j$th column is given by $u_j$, i.e., the first $k$ columns of $U$
Place row $i$ in cluster $j$ if $C_{ij}$ is the largest entry in the $i$th row of $C$
```

4.3. Newman's algorithm

Newman and Girvan have proposed an algorithm specifically devoted to find communities within a network [24].

Their algorithm falls into the class of hierarchical divisive ones, where clusters are identified starting with the network as it is, and progressively removing links, one at a time. Removing links reduces the original network to a set of connectivity islands, which represent the clusters (though the removal of a single link does not necessarily alter the connectivity of the network, the overall effect of a sequence of removals is just that).

The links to be removed are chosen according to their *betweenness*. Though different definition of betweenness are possible, Newman and Girvan have opted for the shortest-path betweenness: the betweenness of any particular link is computed by identifying the shortest paths for each couple of nodes in the network, and counting how many times (i.e., on how many shortest paths) that link appears. The rationale is that a link with a large betweenness connects a large number of nodes, and probably serves as a bridge between different communities: its removal exposes the communities it connected. At any step in the process, the links are therefore ranked by their betweenness, and the link exhibiting the largest value is removed.

The process could go on till the removal of all the links, which reduces the network to its nodes, but this is of course not the purpose of clustering. In order to properly stop the process, Newman and Girvan employ the modularity metric (defined in Section 5.1.2) as a quality indicator for the clustering process. They have shown that, during the divisive procedure, the modularity first increases as links are removed, and then decreases with further removals. Link removal is stopped when the modularity stops increasing.

The algorithm is described in full as Algorithm 4.

Algorithm 4 *(Newman).*

```
Input: A two-way traffic matrix $X + X^T$
Output: A partition of the $n$ nodes into $k$ clusters.
while Modularity increases do
  Calculate betweenness scores for all links in the network.
  Find the link with the highest score and remove it from the network.
  Compute the modularity
end while
Identify the connected portions of the network
```

4.4. Spectral filtering

In [25] a network decomposition method has been proposed to analyze Internet topology at the Autonomous System level. Here we consider that method to perform clustering among the alternatives to our EC algorithm, with the modifications introduced in [6] to consider the actual traffic exchanged by nodes rather than just the connectivity.

The entry at location (i, j) of a symmetric matrix can be said to represent the correlation between the entity at row i and the entity at column j of the matrix (which in our case are both nodes of the network). At the same time, every real symmetric matrix of size n has a spectrum of n orthonormal eigenvectors. Such spectrum is related to the graph connectivity, as thoroughly studied, e.g., in [26]; the eigenvectors associated to the largest eigenvalues of the matrix of interest can be examined to infer the cluster decomposition. Such spectral filtering has been applied to cluster nodes in a network by working on the adjacency matrix [25]. A rationale for the use of the spectral method is that it represents an efficient heuristic to maximize the following clustering efficiency metric, given by the ratio between the number of edges within a cluster and the number of edges incident on that cluster:

$$E = \sum_{l=1}^{k} \frac{|\{(i,j) : A_{ij} = 1; i, j \in \hat{C}_l|}{|\{(i,j) : A_{ij} = 1; i \in \hat{C}_l, j \in \{1, \ldots, n\}\}|}. \quad (1)$$

When considering traffic matrices instead of adjacency matrices, there are however a number of differences with respect to the basic problem:

- In a traffic matrix the entries are generally different from either 0 or 1;
- Traffic matrices are generally asymmetric.

We have therefore to apply some modifications to the basic spectral filtering to make it suitable for traffic matrices. Such modifications aim at removing the asymmetry and reducing the large range of traffic values to the basic unitary interval. The resulting procedure goes through the following steps:

1. Apply the symmetry transformation;
2. Apply stochastic normalization;
3. Extract the k largest eigenvalues and their corresponding eigenvectors;
4. Select one of the eigenvalues identified at Step 3;
5. Build clusters of nodes by using the K-means algorithm.

The first step consists of making the matrix symmetric, so that we can work with a spectrum of orthonormal eigenvectors. We employ the following symmetry transformation

$$M = X + X^T. \tag{2}$$

We prefer this transformation to the more widely deployed $M = X \cdot X^T$ since the transformation (2) maintains the same dimensionality as the original traffic matrix and has a specific physical meaning. In fact, after the transformation, the element m_{ij} of the resulting matrix simply contains the overall traffic exchanged between the two nodes i and j. At this point, we have a symmetric matrix but possibly with a wide range of values. It has been noted that spectral filtering applied to adjacency matrices deteriorates rapidly when the frequency of nonzero elements in the adjacency matrix varies widely (an occurrence related to the varying degrees of the nodes) [27]. We expect similar problems to appear with the traffic matrix, given the wide imbalance existing in traffic between different nodes. Stochastic normalization has been proposed to avoid such problems [25]. In stochastic normalization each matrix element m_{ij} is divided by the sum $\sum_{j=1}^{n} m_{ij}$ of all the elements in the same row. After this normalization, all the elements in a row sum up to 1, making the matrix a stochastic one. In addition we set all the diagonal elements of the traffic matrix equal to 1/2 and multiply all other elements by 1/2. The effect of that operation is to shift the range of the eigenvalues to (0, 1). Now we have an $n \times n$ real symmetric matrix M' that exhibits n eigenvalues and the corresponding eigenvectors. We consider the largest eigenvalues and extract the weights present in the eigenvectors. Each weight is associated to a node of the network. The weights (hence the corresponding nodes) can then be grouped into clusters containing weights of similar value. For such purpose, we use the well-known K-means algorithm in its unidimensional version, which is described in detail, e.g., in [22].

The Spectral Filtering algorithm is described through the pseudo-code reported in Algorithm 5.

Algorithm 5 *(Spectral filtering).*

```
Input: A traffic matrix $X$
Input: The number of clusters $k$
Output: A partition of the $n$ nodes into $k$ clusters.
Compute the two-way symmetric traffic matrix $M = X + X^T$
Perform the stochastic normalization $m'_{ij} = m_{ij} / \sum_{i,j} m_{ij}$.
Extract eigenvalues and eigenvectors of $M'$
Sort eigenvalues
Select one of the largest eigenvalues and the corresponding eigenvector
Cluster nodes according to the eigenvector weight through a unidimensional $K$-means algorithm
```

5. Performance analysis

5.1. Performance metrics

In Sections 3 and 4 we have defined respectively the evolutionary algorithm we propose for clustering and a number of alternative algorithms established in the literature. In order to compare their performance, we have to define a performance metric. Several such metrics have been defined in the past. Unfortunately, performance indices defined in the context of topology-based clustering may be inapplicable. In fact, in a topology-based approach the information employed for clustering is just the adjacency indicator, embodied by a Boolean variable. Instead, in a traffic-based approach the variables employed for clustering are traffic volumes or intensities, represented by real non-negative values. However, the concepts embodied by metrics defined for a topology-based approach may be transferred with some adaptations to the new context. In [6] we have adapted two of such topology-based metrics to a traffic-based context. In this section, we define those performance indices, namely the Traffic aware Scaled coverage metric (hereafter referred to as TS) and the Modularity metric. In addition to the performance metrics used to evaluate the algorithms, we also define the fitness function employed by our EC algorithm to progress toward the solution.

5.1.1. Traffic aware scaled coverage measure

In the context of topology-based clustering the *Scaled Coverage Measure* (SCM) had been defined in [28]. The basic idea of SCM is that each node should be clustered only with its neighbors, minimizing the number of non-neighboring nodes within its cluster as well as the neighboring nodes excluded from its cluster. That metric relies on the concept of neighboring node, defined through a binary variable taking the values 1 or 0 (two nodes are neighbors iff they share a link). Here, we consider instead the Traffic aware Scaled coverage metric, defined in [6], where the concept of neighbor node is replaced by that of close node, represented by a variable (the degree of closeness) taking values in the [0,1) range, to represent a relation between any two nodes, which is not as sharp as that defined by adjacency but rather continuous as that defined by traffic. However, the value of the adjacency indicator could be directly evaluated on the network graph, while traffic matrices provide traffic values, which must be converted into values for the degree of closeness. For that purpose, we employ a logistic transformation, which maps values in the $[0, \infty)$ semi-infinite range into values in the [0, 1) finite co-domain. We define V_{ij}, the degree of closeness of node j to node i, as a variable in the [0, 1) range, derived from a logistic transformation of the traffic intensity X_{ij}, the traffic sent from node i to node j:

$$V_{ij} = \frac{2}{1 + e^{-\alpha \cdot X_{ij}}} - 1. \tag{3}$$

The degree of closeness of node j to node i is 0 iff node i sends no traffic to node j, and is a growing function of traffic:

$$\begin{aligned} V_{ij} = 0 \quad &\Leftrightarrow X_{ij} = 0, \\ \lim_{X_{ij} \to \infty} V_{ij} \quad &= 1. \end{aligned} \tag{4}$$

We note that, in general, $V_{ij} \neq V_{ji}$, since the traffic flows are generally asymmetrical. The coefficient α may be set to match the corresponding values of X_{ij} and V_{ij}. In fact, by inverting the definition (3), we obtain

$$\alpha = \frac{1}{X_{ij}} \ln \left(\frac{1 + V_{ij}}{1 - V_{ij}} \right). \tag{5}$$

For example, if we set the degree of closeness $V_{ij} = 0.7$ when the traffic is $X_{ij} = 10$ erl, the value of the coefficient α is $\alpha \simeq 0.173$. We employ the concept of close node in the clustering metric, since we want to put into the same cluster nodes that exchange most of the traffic with one another, while we don't want to include in the same cluster nodes that exchange little or no traffic with one another. In the TS metric, we pursue that goal by employing two error indicators. Clustering can err in two ways: by including in the cluster nodes that are not close to the node at hand, or by excluding from the cluster those nodes that are close to the node at hand. If we introduce the set C_i as the cluster to which node i is assigned, we can define for a generic node i two quantities representing respectively the inclusion error and the exclusion one:

$$W_i = \sum_{j \in C_i, j \neq i} (1 - V_{ij}), \tag{6}$$

$$Z_i = \sum_{j \notin C_i} V_{ij}. \tag{7}$$

In fact, the inclusion error W_i grows if we include in C_i nodes that have a low degree of closeness to node i. And the exclusion error Z_i grows if we exclude from C_i nodes having a large degree of closeness to node i. Since the number of terms in the sums defining W_i and Z_i is respectively $|C_i| - 1$ and $n - |C_i|$, the two errors are upper bounded as follows

$$\begin{aligned} W_i &\leq |C_i| - 1 \\ Z_i &< n - |C_i| \end{aligned} \tag{8}$$

The upper bound for the inclusion error is achieved if all the nodes belonging to cluster C_i (excluding node i) send no traffic to node i.

At this point, we can consider an overall clustering error as given by the sum of the inclusion error and the exclusion one, and define the local TS for node i as

$$TS_i = 1 - \frac{W_i + Z_i}{n - 1}, \tag{9}$$

where the clustering error is normalized to 1 by dividing it by the sum of the upper bounds derived from the inequalities (8), since $W_i + Z_i < n - |C_i| + |C_i| - 1 = n - 1$. We obtain therefore that the local TS obeys the constraint $0 < TS_i < 1$. The global TS is computed as the average of the TS of the single nodes

$$TS = \frac{1}{n} \sum_{i=1}^{n} TS_i, \tag{10}$$

and lies therefore again in the [0, 1) range.

5.1.2. Modularity

The Modularity metric has been introduced by Newman and Girvan [24] to evaluate the quality of community structure in networks. Since we deal with traffic matrices, which are weighted and represent directed (and, therefore, generally asymmetric) networks, we refer to the definition of modularity given by Newman in [14] for the case of weighted matrices, and extend it naturally to asymmetric matrices. Since weights are represented by non-negative real numbers (like the entries of the traffic matrix), we can simply re-use that modularity definition by replacing the weights with the traffic matrix entries. By introducing the sum of the traffic matrix elements $m = \sum_{ij} X_{ij}$, the modularity Q is given by

$$Q = \frac{1}{m} \sum_{ij} \left[X_{ij} - \frac{\sum_j X_{ij} \sum_i X_{ji}}{m} \right] \delta(C_i, C_j) \tag{11}$$

where the δ-function $\delta(C_i, C_j)$ equals 1 if nodes i and j belong in the same cluster (i.e., $C_i = C_j$), and 0 otherwise. The modularity metric takes values in the [−1,1] range. It takes the value 0 if the partition has no more traffic than one would expect from the random distribution of traffic over all possible origin-destination pairs.

5.1.3. The fitness function

The EC algorithm employs a fitness function to measure the goodness of the candidate solutions. Since we have defined two performance metrics, it is natural to adopt a fitness function that reflects them. As a fitness function, we consider the linear combination of the two metrics:

$$F = \gamma TS + (1 - \gamma) Q. \tag{12}$$

The results reported in Section 5.3 have been obtained with a balanced mix where $\gamma = 0.5$. Note that we have decided to deal with a single-objective formulation of the problem to compare our evolutionary approach with other existing algorithms in the literature, which are also thought to be implemented as single-objective solvers. Thus, we have chosen a value of γ that balances both metrics. A multi-objective formulation of the problem is of course possible, but it is out of the scope of the present study. Related to this point, in [29] a multi-objective formulation of a software module clustering problem is proposed. Though the application is different, the authors introduce several new metrics to solve the problem, some of them similar to the ones used in this paper. The interested reader can consult details about that multi-objective approach to clustering in graphs in [29].

5.2. The dataset

In Sections 3 and 4, we have described respectively the evolutionary algorithm we propose and the alternative non-traffic-based algorithms for network clustering. In order to compare our proposal with the set of competing algorithms, we have carried out an extensive performance analysis of them all, by employing the two performance metrics defined in Section 5.1 on two large real world datasets. These datasets are made of a large number of traffic matrices gathered respectively on two networks, both of continental size: Géant and Abilene. In this section, we describe the two datasets.

5.2.1. The Géant network dataset

Géant is the pan-European research network, serving Europe's research and education community. It was co-founded in 2000 by the European National Research & Education Networks (NRENs) and the European Community. Over the years, it has progressively grown, reaching over 50,000 km of network infrastructure in 2010, including 12,000 km of optical/dark fiber across Europe, with an overall NREN partner access capacity to the network of 258 Gbit/s. A traffic measurement campaign was conducted on Géant for four months in 2004. The result of that campaign is a full set of traffic matrices, built by employing several traffic engineering algorithms and using full IGP (Interior Gateway Protocol) and BGP (Border Gateway Protocol) routing information, with sampled Netflow data. Such traffic matrices provide the traffic volume (in bytes) for each origin/destination router, collected at 15 min intervals. At the time of the measurement campaign, Géant was composed of 23 routers interconnected using 38 links, and the average degree of routers was 3.3. A map of the topology at that time is shown in Fig. 5, where it can be seen that a number of PoPs (Points of Presence), i.e., stub nodes, were present in addition to the routers. Both the dataset and the measurement procedure are publicly available [30]. The whole dataset is made of 11460 traffic matrices. We selected the first batch of 1000 for our performance comparison.

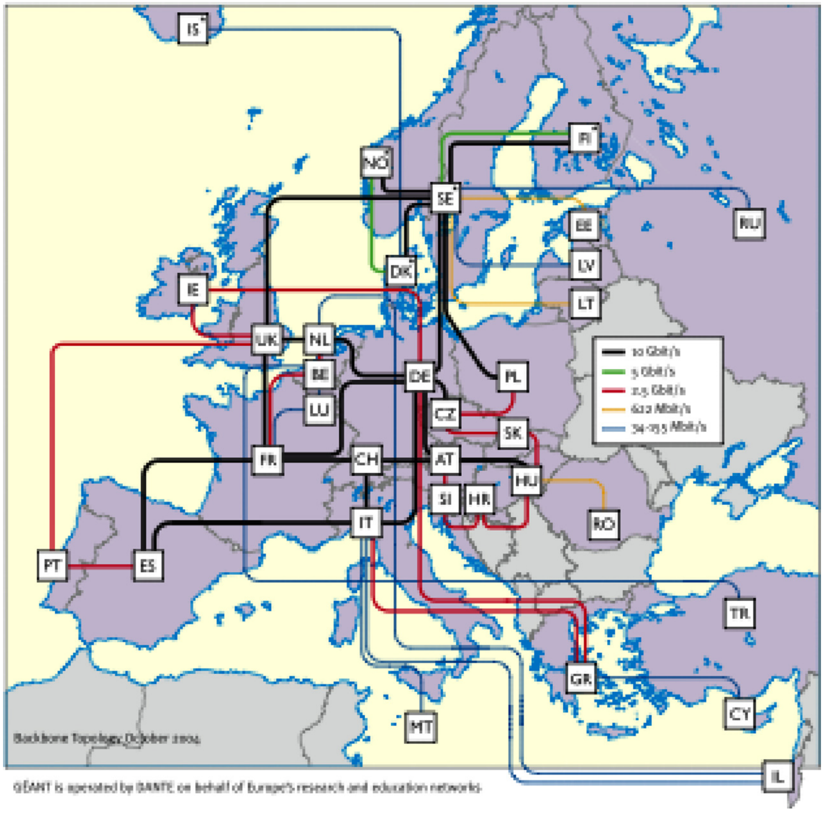


Fig. 5. Topology of the GÉANT network at the time of measurements (2004).

5.2.2. *The Abilene network*

Abilene (now known as Internet2 Network) is the U.S. high-performance backbone network created by the Internet2 community (more than 220 member institutions, mostly universities and some corporate institutions). The data were collected over 6 months in 2004 with a resolution of 5 min, and concerned 12 routers interconnected by 30 links, with an average degree of 5. A picture of the topology at the time of the measurement campaign is shown in Fig. 6, where Atlanta hosts two nodes. The whole Abilene dataset is made of 11425 matrices. We selected the first batch of 1000 for our performance comparison.

5.3. *Performance comparison*

After having described the clustering algorithms under comparison (in Sections 3 and 4), the performance metrics (in Section 5.1), and the datasets (in Section 5.2), we can now provide the result of the performance analysis. In the case of the EC algorithm, we

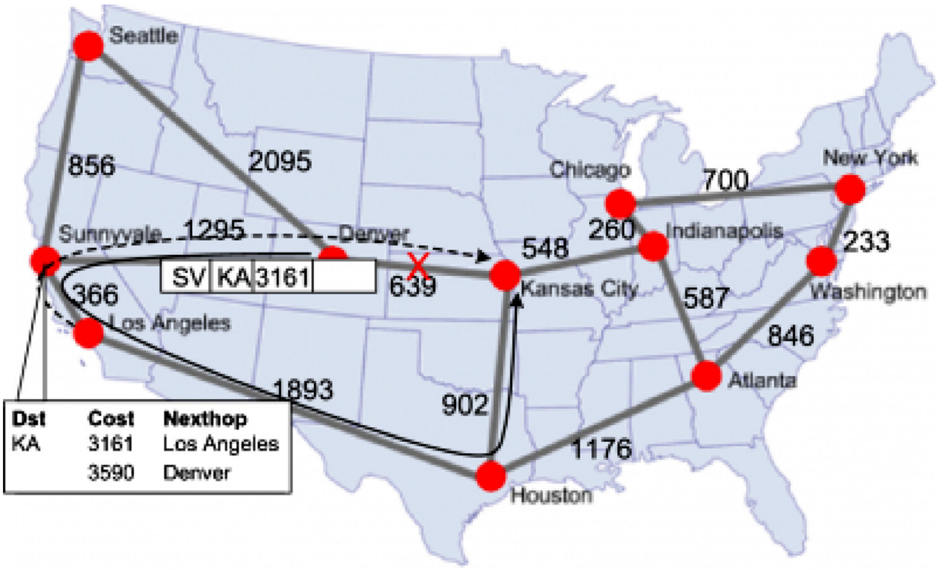
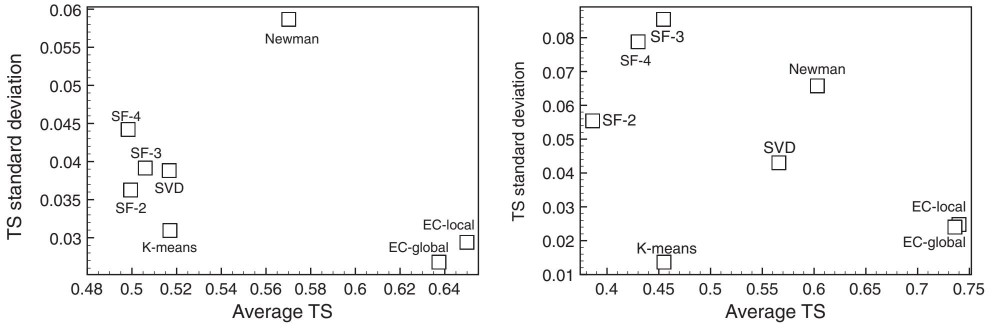


Fig. 6. Topology of the Abilene network at the time of measurements (2004).



(a) Abilene (b) Géant

Fig. 7. Comparison under the TS metric.

report the results for two alternative cases. In fact, our evolutionary algorithm can provide an individual clustering solution for each traffic matrix; we label this case as Local Optimization (synthetically indicated in the following as EC-local). Alternatively, we can go for Global Optimization (EC-global), where our evolutionary algorithm provides a single clustering solution, optimized over the whole set of traffic matrices. The EC-local is expected to outperform EC-global, since it provides an individual solution tailored to each traffic matrix. It is to be noted that, in the non-evolutionary algorithms, a clustering solution is provided individually for each traffic matrix. For a fair comparison, we should therefore consider the local version of EC.

We conduct a three-fold analysis. First, we report the first two statistical moments of the two metrics, which provide an overall picture of how the algorithms perform. Second, we go into more depth by performing a matrix-by-matrix comparison of the results. Third, we take a look at the distribution of values of the performance metrics.

5.3.1. Moment analysis

We consider the first two statistical moments of the quality metrics for the two networks, namely the average value and the standard deviation. A good clustering algorithm should provide a large average quality metric. On the other hand, a small standard deviation of the quality metrics is also a desirable feature, since it shows that the algorithm's performance is quite stable and doesn't depend on the dataset instance. In order to consider these two characteristics at the same time, we plot for each algorithm (and for each combination of network dataset and quality metric) the standard deviation of the metric vs. its average value. Since the analysis concerns the whole dataset, we obtain a single couple of values for each algorithm. For Spectral Filtering, we report the results obtained by selecting the second, third, and fourth largest eigenvalue (labeled respectively as SF-2, SF-3, and SF-4). Symbols appearing in the lower right-hand corner of the picture represent the best algorithms (high average and low variability). Instead, symbols appearing in the higher left-hand corner represent the worst algorithms (low average and large variability).

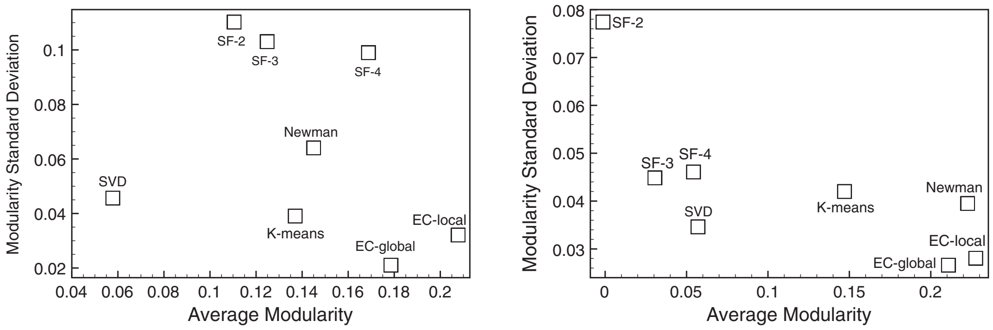
We report the results in Figs. 7 and 8 for the TS metric and modularity respectively.

For the Abilene network, we see that the TS metric is very low for all the non-evolutionary algorithms, with the exception of Newman's algorithm, which, however, exhibits a large standard deviation as well. The evolutionary algorithms perform best in both its versions (Local and Global Optimization). As expected, the Local Optimization version achieves better average performances, though with a slightly larger standard deviation.

The results for the Géant network shown in Fig. 7(b) mimic those obtained for Abilene: the EC algorithm scores quite better than all the rest, in both the local and the global version, with Newman's algorithm in a distant second place. With respect to the Abilene network, the SVD algorithm performs slightly better, progressing from 0.516 to 0.566.

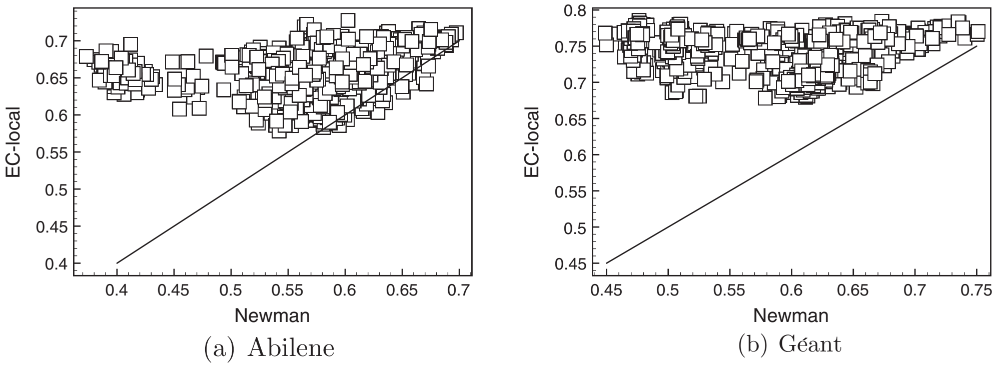
Under the Modularity metric, the performances are more scattered. In the Abilene network, we see again that the EC algorithm exhibits the largest average values and the lowest standard deviation at the same time. Now, however, the Spectral Filtering, in the version with the fourth largest eigenvalue, is the best competitor, though with a large standard deviation. Newman's algorithm is however a good runner-up in the group of non-evolutionary algorithms, with a lower average modularity, but also a lower standard deviation.

A slight surprise comes when comparing the modularity values obtained for Géant. Here, though the local version of the EC algorithm is the absolute best, Newman's algorithm exhibits an average value larger than the global version of the EC algorithm. However,



(a) Abilene (b) Géant

Fig. 8. Comparison under modularity.



(a) Abilene

(b) Géant

Fig. 9. Individual comparison for the traffic-aware scaled coverage.

its variability is much larger than that scored by the global EC. And we remind that a fair comparison should be carried out with the local version of EC, since all the non-evolutionary algorithms provide an individual clustering solution for each traffic matrix.

Summarizing, on the basis of the average results shown so far, we can conclude that the proposed EC algorithm outperforms the non-evolutionary approaches. The best runner up is Newman's algorithms for all cases, except in the Abilene network when the Modularity metric is employed, in which case Spectral Filtering is the best competitor.

5.3.2. Individual comparison

The moment analysis performed in Section 5.3.1 tells us that the proposed EC algorithm is the best algorithm among all those compared, in average, and also that it has the lowest dispersion (as represented by the standard deviation). However, the variability of the obtained results may make some other algorithm better for some traffic matrix. In order to perform a thorough comparison, we now analyze the results obtained matrix by matrix. In this section, we provide such comparison between the local version of our EC algorithm and the best competitor, which is the one exhibiting the largest average value of the metric in the group of non-evolutionary algorithms, identified at the end of Section 5.3.1.

We report on a graph a square dot for each traffic matrix. Each dot represents the metric value for that matrix: its x- and y-coordinates are given respectively by the value of the metric obtained by the best competitor and by the local EC. We do that again for each of the four cases given by the combination of dataset and metric. On each graph, we also plot the bisectrix line. If the square dot lies above the bisectrix, the EC algorithm achieves a larger value of the metric for the traffic matrix represented by that dot. We compute the percentage of EC success as the percentage of traffic matrices for which the EC algorithm scores better than the best competitor.

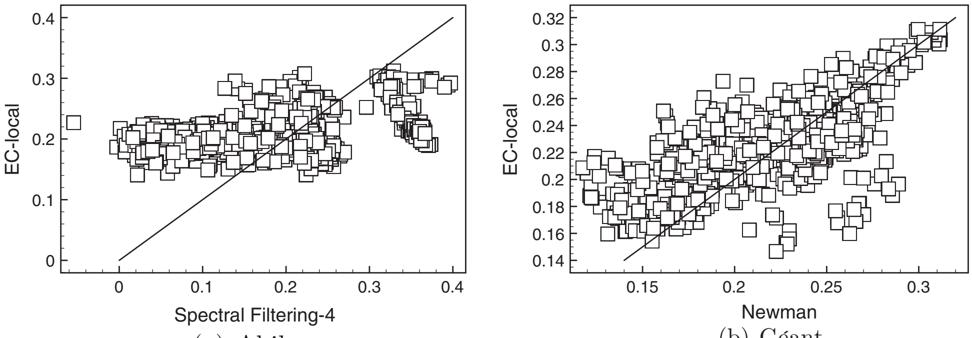
Table 1
Percentage of EC success.

	TS	Modularity
Abilene	97.7%	63.3%
Géant	100%	52.7%

We report the results in Figs. 9 and 10 for the TS metric and modularity respectively. We see that in one case (the Géant dataset analyzed through the TS metric) the EC algorithm is better for any single traffic matrix. In the remaining three cases, the best competitor achieves better scores in some cases (but very few in the case of the Abilene network with the TS metric). But we remind that the EC algorithm also boasts a much smaller variability of results, so that it is quite more reliable than its competitors. In Figs. 9 and 10(a), this is self-evident through the elongated nature of the cloud of square dots.

We report the percentage of EC success in Table 1, where we see that the EC algorithm achieves a striking dominance under the TS metric, while it scores worse in a significant fraction of the dataset under the Modularity metric. We have to remind two relevant conditions: (1) the fitness function employed in EC is a balanced combination of the two metrics (if tuned for the Modularity metric only, it would achieve much better scores for that metric); (2) Newman's algorithm has been specifically devised to achieve a good Modularity value.

Though Table 1 and Figs. 9 and 10 give us respectively an aggregate and an individual (for each traffic matrix) performance measure, our comparison still considers the two performance measures separately. We can gain a further insight into the relative merits of our EC algorithm if we identify where it is better than the



(a) Abilene

(b) Géant

Fig. 10. Individual comparison for modularity.

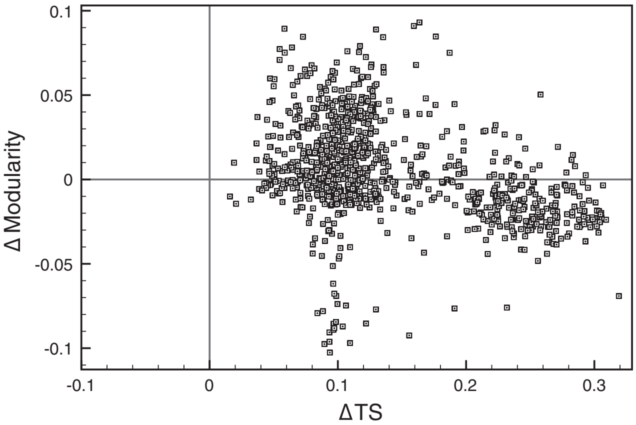


Fig. 11. Pareto dominance analysis for the Géant network.

competing algorithms on both quality metrics. By borrowing the concept of Pareto dominance from game theory [31], we say that the EC algorithm dominates (resp. is dominated by) an algorithm *Z* if both the Modularity and the Traffic-aware Scaled Coverage values achieved by EC are higher (resp. lower) than those achieved by algorithm *Z*. Since the EC algorithm performs worse in the Géant case, we report the results of the Pareto dominance analysis just for this worst case. As said above, the strongest competitor in that case is Newman's algorithm, and we limit ourselves to comparing EC with Newman's. Precisely, we compute the quantities ΔM and ΔTS, given by the differences between the values achieved by EC and by Newman's algorithm for the two quality metrics, and plot ΔM vs. ΔTS. If ΔM is positive, the EC algorithm performs better than Newman's algorithm under the Modularity metric; the same can be said for ΔTS under the Traffic-aware Scaled Coverage. The results are reported in Fig. 11, where each square represents a single traffic matrix. In order to analyze those results, we resort to the usual classification of the regions of the Cartesian system into four quadrants. When a square falls into the first quadrant ($\Delta TS > 0$ and $\Delta M > 0$), the EC algorithm is better for both metrics. The reverse takes place when a square falls into the third quadrant ($\Delta TS < 0$ and $\Delta M < 0$), where Newman's algorithm is the clear winner. The first and the third quadrants represent regions of Pareto dominance. Instead, in the second and fourth quadrant the EC algorithm is better under just one of the two quality metrics. If we now look at Fig. 11, we see that a large proportion of squares falls in the first quadrant, but none in the third one: the EC algorithm dominates Newman's for a large fraction of the traffic matrices, and is never dominated.

After this final comparison of the EC algorithm against its competitors, we can safely conclude that EC, through its fitness function, is capable of achieving the best results not just on a single metric, but in a multi-criterion context.

5.3.3. *Distribution of values*

The analysis of the variance of the performance metrics allows us to evaluate the dispersion of values around the average. We want to examine the distribution of values. We limit ourselves to an exploratory analysis, through a non-parametric estimation method. We estimate the probability density function by using the kernel method, with a Gaussian kernel [32]. The bandwidth *h* of the kernel is set using the empirical formula provided in Section 4.3.2 of [32]:

$$h = 1.06\sigma \quad d^{-0.5}, \tag{13}$$

where *d* is the number of data points used for the estimation, and σ is the standard deviation of those data. We report in Fig. 12 the curves obtained for the Modularity with the Abilene dataset; similar curves are obtained for the other combinations of dataset and metric.

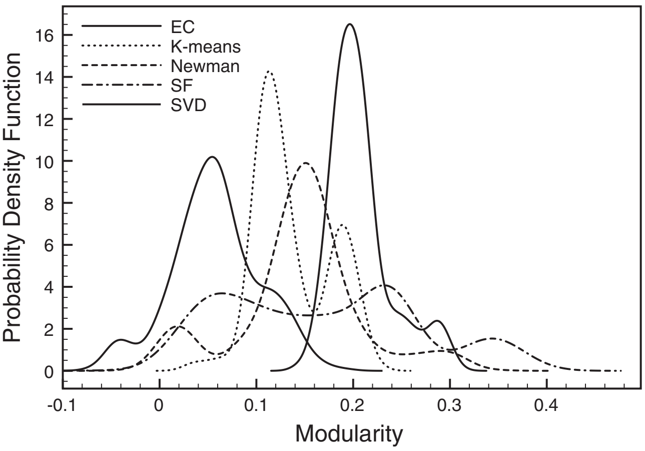


Fig. 12. Empirical probability density function of Modularity for the Abilene network.

We see that all the curves exhibit a bimodal nature, which is slight for the EC algorithm, Newman's, and singular Value Decomposition, but becomes more pronounced for the *K*-means and Spectral Filtering (where the two modes are of nearly equal amplitude). The presence of two modes is associated to the time variability of the traffic matrices. We have a dataset made of 1000 consecutive matrices, sampled at 15 min interval. The two peaks in the probability density function pertain to two different times of the day.

5.4. *Synthetic traffic matrices and an alternative evolutionary approach*

In order to further evaluate the capabilities of the proposed EC approach, we have carried out an additional experimental analysis of the algorithm. We have focused on comparing the proposed approach in larger networks, including a comparison with an alternative evolutionary algorithm for clustering, *EvoCluster*, presented in [11]. First of all, we propose a model for generating new network clustering problems in larger networks, starting from the Géant network previously analyzed in this paper. The main characteristics of the EvoCluster algorithm in [11] are presented next. Finally, we show the results obtained in these new experiments.

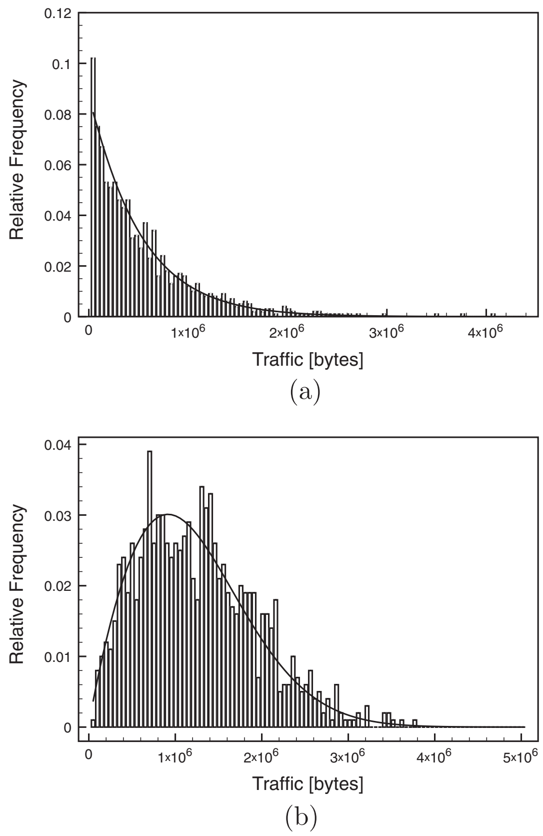
5.4.1. *Modeling synthetic networks to generate larger instances*

In order to obtain a realistic modeling of synthetic networks, we start considering the Géant network used in the performance evaluation of our EC algorithm. First, it can be observed that the traffic in Géant network follows approximately a Weibull distribution. This can be seen by plotting the traffic among every pair of nodes in the network, in this case for the 1000 traffic matrices of the Géant network considered above. Fig. 13 shows two examples of traffic distribution, respectively between nodes 5 and 13, and between nodes 22 and 12. We show the histogram obtained with 1000 traffic matrices as well as the fitted Weibull distribution.

The Weibull probability density function has the following expression:

$$f(x; \lambda, k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} \qquad x > 0, \tag{14}$$

where λ is the scale parameter and *k* is the shape parameter (both have to be positive). Note that the Weibull distribution can take quite different shapes, and both the exponential and the Rayleigh



(a)

(b)

Fig. 13. Two examples of traffic between nodes in the Géant network: (a) traffic between nodes 5 and 13; (b) traffic between nodes 22 and 12.

model are special cases of the Weibull one. Back to our example of the traffic matrices from the Géant network, once we have established the Weibull distribution as the statistical reference to model the traffic of this network, we can estimate both λ and k from the Géant network's traffic. After estimating both parameters, we can see in Fig. 14 (which reports the values obtained for 1000 traffic matrices) that a relation exists between them. This curve can be approximated as:

$$k^* = \frac{(\lambda^*)^2}{32} - 0.9 \qquad (15)$$

Our best fit curve is plot as a solid line in Fig. 14.

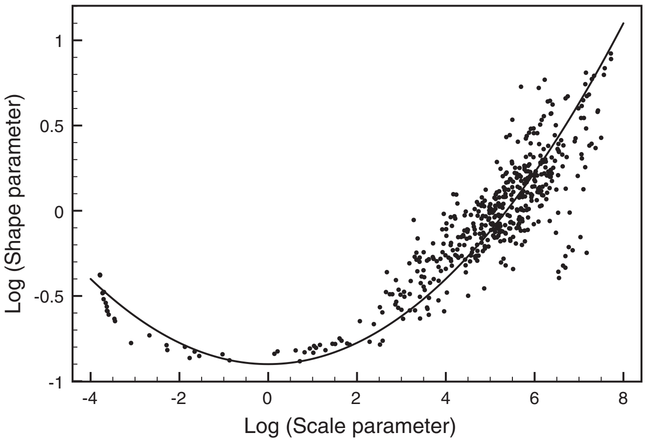
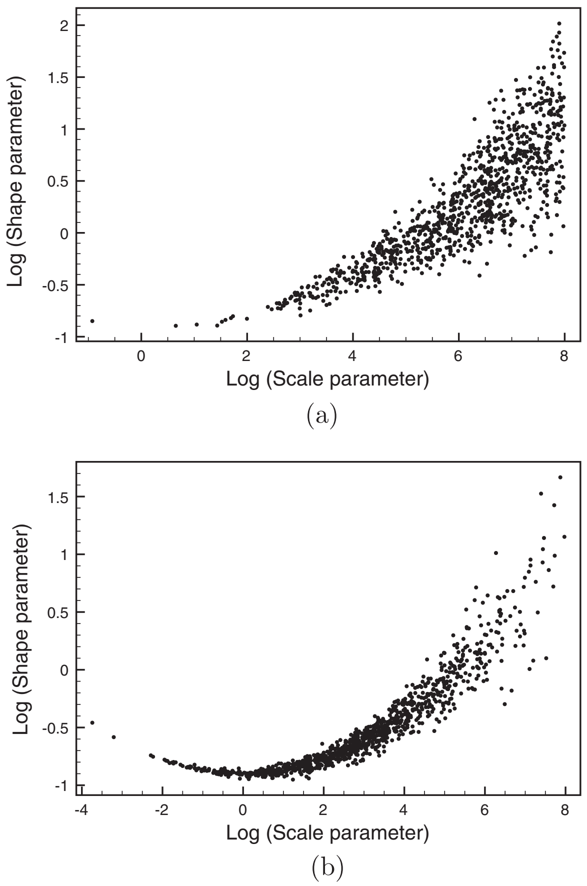


Fig. 14. Relation between Weibull parameters and best fit approximation for the Géant network.



(a)

(b)

Fig. 15. Weibull parameters in the synthetic traffic matrices for a 100-node network: (a) nodes belonging to the same cluster; (b) nodes belonging to different clusters.

We can exploit Eq. (15) to generate realistic couples of (k, λ) values. Thus, we can propose the following synthetic model to generate larger network clustering problems. We assign each node of the network to one of 5 different groups (clusters). Then, for any two nodes of a network i, j, we assume the scale parameter of the Weibull distribution to be normally distributed, precisely $\lambda^* = N(7, 4)$ if i and j belong to the same cluster, whereas $\lambda^* = N(3, 4)$ if i and j belong to different clusters. Note that this procedure can be applied to networks of any number of nodes. We restrict the values of λ in the interval [0, 8] (negative values of λ are censored, whereas values above 8 represent cases of extremely high traffic). By using this simple model, we can randomly generate values of λ^* for each pair of nodes i and j in a given network, and then calculate the corresponding value of k^* using Equation (15). We have applied this synthetic model to the generation of network clustering problems of size 30, 40, 50, 75, and 100 nodes, producing 1000 different matrices for each size, to carry out experiments similar to the ones run for Abilene and Géant networks above. Fig. 15 shows an example of λ^* vs. k^* figure obtained between two nodes belonging to the same cluster (Fig. 15(a)) and between two nodes belonging to different clusters (Fig. 15(b)), for the case a 100 nodes synthetically generated traffic matrix.

5.4.2. An alternative evolutionary clustering algorithm

In [11] an interesting evolutionary-based algorithm is proposed for clustering problems (EvoCluster). That approach has several adaptations to improve the evolution of solutions in clustering problems. First, the algorithm uses a grouping-based encoding, where each chromosome (solution to the cluster problem) includes a variable number of clusters, and each cluster is formed by the

Table 2
Modularity and TS measures of the predefined clusters for the synthetically generated networks.

Network	Objective clusters	
	Mod	TS
30	0.5452	0.8337
40	0.5449	0.8228
50	0.4893	0.8037
75	0.4822	0.8125
100	0.5013	0.8066

labels of elements associated to it. Reproduction in EvoCluster consists in the application of crossover and mutation operators. Two different crossover operators are defined: guided and unguided. In the guided crossover operator, the exchange of grouping information is not totally random, but some information about the best clusters is preserved during the crossover process. On the other hand, in the unguided crossover operator, the exchange of the grouping information between clusters is performed randomly. In EvoCluster, six different mutation operators have been defined, since guided and unguided mutation are considered, and the mutation process may involve either just the removal and reclassification of labels in clusters or the merge and split of the whole cluster. The objective function used in EvoCluster is based on a statistical study of the clusters encoded in each chromosome. The chromosome's fitness is indeed evaluated in two steps. A first step attempts to discover statistically significant association patterns in the clusters. A subset of records from different clusters encoded in a chromosome is randomly selected to form a training set for pattern discovery. In a second step, those records not selected in the first step are reclassified into one of the clusters based on the discovered patterns, to determine the *reclassification accuracy* of the chromosome. This measure can be used as the fitness value for each chromosome. Further information and implementation details about the EvoCluster algorithm can be found in [11].

5.4.3. Results for synthetic traffic matrices

Here we present the new results obtained by the proposed EC approach in the larger synthetically generated network clustering problems, compared to all the alternative algorithms considered so far, including the EvoCluster approach. First, Table 2 shows the Modularity and TS measures of the synthetic problems according to the cluster assignment procedure adopted in the generation process (note that this is only possible in synthetic problems, where we know the correct classification of each node in a given cluster). These measures can be taken as a reference of the algorithm's accuracy when tackling these synthetic network clustering problems.

Tables 3 and 4 show the results obtained by the proposed EC in the larger synthetically generated network clustering problems (Modularity and TS metrics, respectively). A comparison with the previously considered algorithms for network clustering, including the EvoCluster algorithm, is also presented in these tables. There are several interesting points to highlight after this round of experiments. First, the proposed EC approach outperforms the compared approaches in the network clustering problem. When using the modularity metric as objective, the EC approach obtains better values than the rest of approaches, excepting the instance of 75 nodes, where Newman's algorithm obtains slightly better values, precisely better by 1.5% (but we recall that Newman's algorithm has been specifically devised to optimize modularity). In the case of the TS metric, the EC algorithm obtains better results than all the other methods under comparison. In this case the Newman approach seems to be the second best option. The comparison of the proposed EC algorithm with the existing EvoCluster algorithm also exhibits interesting features. Note that the EvoCluster algorithm is quite competitive under both metrics, outperforming other approaches to network clustering. In fact, using the TS metric, the EvoCluster approach is the third best option, after the proposed EC and the Newman approach. Note also that the EvoCluster algorithm improves its performance in larger networks. There is a good explanation to this behavior of the algorithm: the EvoCluster is based on a statistical model of the clustering provided by each solution (by means of a training and validation sets), so this statistical model will be better for large networks, and worse when there are few nodes in the network, as in Abilene or Géant.

Table 3
Modularity results in synthetic instances of different sizes.

Network	EC	EvoCluster	Fast SVD	*K*-means	SF	Newman
Abilene (12)	0.2077	0.0745	0.0578	0.1366	0.1688	0.1450
Géant (23)	0.2276	0.0187	0.0570	0.1474	0.0543	0.2226
30	0.5555	0.0150	0.0281	0.0088	−0.0260	0.4071
40	0.5162	0.0349	0.0306	0.0464	−0.0143	0.3812
50	0.4601	0.1599	0.0011	0.0211	−0.0105	0.4121
75	0.4513	0.1002	0.0101	0.2672	−0.0034	0.4580
100	0.4793	0.2756	0.0011	0.3316	−0.0214	0.4788

Table 4
TS results obtained in synthetic instances of different sizes.

Network	EC	EvoCluster	Fast SVD	*K*-means	SF	Newman
Abilene (12)	0.6499	0.5540	0.5167	0.5174	0.5060	0.5702
Géant (23)	0.7399	0.6843	0.5660	0.4548	0.4546	0.6032
30	0.8227	0.6911	0.7155	0.4752	0.6669	0.8013
40	0.8146	0.7299	0.7233	0.5611	0.6391	0.8346
50	0.7999	0.7424	0.7212	0.5354	0.6354	0.7836
75	0.8091	0.7327	0.7239	0.5409	0.5653	0.7969
100	0.8036	0.7695	0.7151	0.6213	0.6642	0.7729

5.5. Computational cost analysis

In Section 5.3 we have compared the quality of the clustering solutions obtained through our EC algorithm with the alternative algorithms. That comparison was based on the quality metrics defined in Section 5.1. Another dimension that has to be considered in the comparison is the computational cost associated to those algorithms. In this section we provide an asymptotic analysis of the computational cost of all the algorithms described in Section 4 and of our EC algorithm. We recall that, in the following, we consider a network with m links and n nodes, which are grouped into k clusters.

5.5.1. Newman

In the same paper in which they describe their algorithm, Newman and Girvan provide an indication of its computational cost [24], which is $O(m^2n)$. This value depends on the connectivity of the network. We can obtain lower and upper bounds for that value. In fact, the minimum number of links for a network of n nodes is $n-1$, and corresponds to a network with linear topology. The opposite extreme case corresponds to a fully meshed network, where the number of links is $n(n-1)/2$. Correspondingly, the lower and upper bounds for the computational cost are respectively $O(n^3)$ and $O(n^5)$, with the cost growing with the nodes' average degree. We recall that, in the networks we have considered, the average node degree was respectively 3.3 for Géant and 5 for Abilene. As examples of other networks, we can mention Skitter, with an average degree of 6.34 [33], and the AS inter-domain topology, with an average degree ranging between 2.67 and 2.99 [34]. These examples show that there is not a direct relation between the size of the network and the degree of nodes. In turn, that means that the number of links can be deemed to be roughly proportional to the number of nodes, as in the linear topology rather than in the full mesh topology. And the computational cost of Newman's algorithm is therefore nearer to its lower bound ($O(n^3)$) than to its upper one ($O(n^5)$).

5.5.2. *K-means*

The general expression for the computational cost of the multidimensional *K*-means algorithm to find the global optimum in a deterministic way, when each node has *d* attributes, is $O(n^{dk+1}\log n)$ [35]. However, we have used a standard iterative heuristic, so that we can evaluate the computational cost in a straightforward way. Most of the time is spent by computing distances. Each distance computation requires a cost $O(n)$, since the number of attributes of each node equals the number of nodes. When reassigning each node to a cluster, we compute its distance to each centroid, so that the overall cost of reassignment is *kn* times the cost of a single computation, i.e., $O(kn^2)$. Finally, the association of a group of nodes to a centroid has a cost $O(n^2)$. The overall cost is therefore $O(kn^2)$ for each iteration, which, in our case, can be simplified to $O(n^2)$. Though it has been shown that, in the very worst case, the algorithm requires an exponential number of iterations [36], it has been recognized since long that the performance of the *K*-means algorithm exhibits a stark contrast between practical observations and theoretical bounds [37]. In our case, we have observed an approximately constant number of iterations, so that we can safely assume that the overall cost of the *K*-means algorithm is $O(n^2)$.

5.5.3. *Singular value decomposition*

In the case of the Fast SVD, Drineas et alii show that the computational cost is $O(k^3+k^2r/\epsilon^4)$, where *r* is the maximum number of non-zero entries in the adjacency matrix, and $\epsilon > 0$ is a given error parameter [38]. The worst case corresponds to a network with a full mesh topology, whose adjacency matrix is full excepting its diagonal. In that case, the number of non-zero entries is $n(n-1)$ and the computational cost becomes $O(k^3+k^2n^2/\epsilon^4)$.

5.5.4. *Spectral filtering*

As to the computational cost of Spectral Filtering, its proposers have not provided it. Here, we provide an evaluation based on the decomposition of tasks required by that algorithm. Namely, for the purpose of evaluating the computational cost, we consider the following sequence of tasks:

1. Computation of the two-way traffic matrix;
2. Stochastic normalization;
3. Eigen-decomposition of the normalized two-way traffic matrix;
4. Sorting of the eigenvalues and selection of one of the top eigenvalues and the corresponding eigenvector;
5. Application of the *K*-means algorithm to the group of eigenvector weights.

The first step consists in the sum of the traffic matrix and of its previously computed transpose. Both the transposition and the sum are operations whose cost is $O(n^2)$, since it is proportional to the number of matrix elements. Stochastic normalization consists in summing all the elements of each row in the matrix and dividing each element of the matrix by the row sum pertaining to the row that element belongs to, again a task of cost $O(n^2)$. The cost of the eigen-decomposition depends on the accuracy we aim for. For an approximation within 2^{-b}, the computational cost has been shown to be $O(n^3+n^2\log^2 n\log b)$ [39]. As to sorting, no algorithm that sorts by comparing elements can do better than $O(n\log n)$, but the worst case result for most algorithms is $O(n^2)$ [40]. Finally, the determination of *k* clusters through the application of the unidimensional *K*-means algorithm to the *n* weights of the selected eigenvector requires a cost of $O(n^{k+1}\log n)$. The costs of all the tasks are reported in Table 5. The overall cost is then due to eigen-decomposition and *K*-means, i.e, $O(n^3+n^2\log^2 n\log b)+O(n^{k+1}\log n)$. However, when the number of clusters is at least 3 (a condition almost surely satisfied for networks larger than a few tens of nodes), the dominant cost is that due to the final application of the *K*-means algorithm.

Table 5
Computational cost of spectral filtering.

Task	Computational cost
Two-way traffic matrix	$O(n^2)$
Normalization	$O(n^2)$
Eigen-decomposition	$O(n^3+n^2\log^2 n\log b)$
Sorting	$O(n^2)$
K-means	$O(n^{k+1}\log n)$

5.5.5. *EC algorithm*

We now turn to the EC algorithm. As for the Spectral Filtering algorithm, we identify the following sequence of computational tasks:

1. Initialization;
2. Iteration of the following tasks till convergence
   (a) Computation of the fitness function
   (b) Application of the Selection operator
   (c) Application of the Crossover operator
   (d) Application of the Mutation operator

The initialization consists in assigning randomly the *n* nodes to the clusters, and setting all the remaining elements of the encoding matrix to -1. This can be done by first setting the encoding matrix of size $k\times n$, and then setting the values of the *n* elements different from -1. The cost of setting the matrix elements equal to -1 in the first place is $O(kn)$. We have to generate N_{sol} candidate solutions (which, though random during the iterative process, fluctuates around the pre-set value, which is independent of the number of nodes), so that the overall computational cost of initialization is $O(kn)$.

The fitness function is a linear combination of the TS metric and of the Modularity; its computation requires therefore the computation of both quality metrics. We now derive the computational costs pertaining to them.

As to the TS, the combination of Eqs. (10) and (9) give us the following overall expression:

$$
\begin{aligned}
TS &= \frac{1}{n}\sum_{i=1}^{n}\left(1-\frac{W_i+Z_i}{n-1}\right) = 1-\frac{1}{n(n-1)}\sum_{i=1}^{n}(W_i+Z_i)\\
&= 1-\frac{1}{n(n-1)}\sum_{i=1}^{n}\left[\mid C_i\mid -1-\sum_{j\in C_i, j\neq i}V_{ij}+\sum_{j\notin C_i}V_{ij}\right]\\
&= \frac{n}{n-1}+\frac{\sum_{i=1}^{n}\left(\sum_{j\in C_i, j\neq i}V_{ij}-\sum_{j\notin C_i}V_{ij}-\mid C_i\mid\right)}{n(n-1)} \qquad (16)
\end{aligned}
$$

The first term of this sum can be computed once and for all. The term involving the elements V_{ij} consists of summing all the extra-diagonal elements of the matrix *V* with a sign which is 1 if the nodes *i* and *j* belong to the same cluster and -1 otherwise. This is tantamount to multiplying term by term the matrix *V* and the matrix *C* obtained by the following rule, and then summing all the elements:

$$
c_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ belong to same cluster}\\ -1 & \text{if nodes } i \text{ and } j \text{ belong to different clusters}\\ 0 & \text{if } i=j \end{cases}
$$

Table 6
Computational cost of modularity.

Task	Computational cost
Row sums of traffic matrix	$O(n^2)$
Overall sum of traffic matrix	$O(n)$
Identification of cluster composition	$O(kn)$
Terms pertaining to all couples	$O(n^2)$

All the operations involved require a cost $O(n^2)$. Instead, the last term is equal to the sum of the squared cardinalities of all the clusters. The cardinality for each set can be obtained by scanning the encoding matrix and counting the number of elements different from -1 in each row, which has a cost $O(kn)$. Squaring each cardinality has a cost linear in the number k of clusters. And summing all the squared cardinalities has again a cost $O(k)$. The computational cost for this term is then $O(kn)$. Since the number of clusters is lower than the number of nodes, the dominant cost is that due to the term involving the V matrix: the cost of computing the TS metric is then $O(n^2)$.

For the Modularity, we go back to the definition (11). We see that the definition involves considering for each cluster just the terms in the double sum concerning the nodes belonging to that cluster. Most of the terms can be computed beforehand, since they are used repeatedly in the double sum. Namely, the terms $\sum_j X_{ij}$ and $\sum_i X_{ji}$ represent just the row sums pertaining to nodes i and j in he traffic matrix. There are just n rows in the traffic matrix matrix, so that there are just n such terms. Computing the modularity requires the following tasks

1. Computation of the sums $\sum_j X_{ij}$ for each node
2. Computation of the overall sum $m = \sum_i \sum_j X_{ij}$
3. Identification of the nodes belonging to each cluster
4. Computation of the sum $X_{ij} - \sum_j X_{ij} \sum_i X_{ji}/(2m)$ for each couple of nodes belonging to the same cluster

We now examine separately each step. In the first step, we need to scan the traffic matrix row by row and compute a sum of n terms for each row. The computational cost is then $O(n^2)$. Once we have all the row sums, computing the overall sum m (Step 2) consists in summing the n row sums, and has then a computational cost $O(n)$. The nodes belonging to each cluster have already been identified when computing the TS metric; anyway their cost is $O(kn)$ as shown above. Finally, the computation involved in Step 4 has to be carried out for all the clusters and a number of times equal to the number of couples of nodes belonging to each cluster. If we indicate by $|\hat{C}_i|$ the cardinality of the ith cluster, the number of couples of nodes in cluster i is $|\hat{C}_i|(|\hat{C}_i| - 1)/2$, so that the overall number of terms to be summed in Step 4 is upper bounded by $\sum_{i=1}^{k} (|\hat{C}_i|)^2 < n^2$, since $\sum_{i=1}^{k} |\hat{C}_i| = n$. The computational cost of Step 4 is then safely upper bounded by $O(n^2)$. In Table 6, we summarize the computational costs of the tasks involved in computing the modularity. The dominant cost is $O(n^2)$.

The application of the Selection operator requires the computation of the average value of the fitness function, and the identification and removal of all the candidate solutions whose fitness function is lower than that average. The former task requires $N_{\text{sol}} - 1$ additions and a division. The identification of the candidate solution requires N_{sol} comparisons. The computational cost of both tasks doesn't depend on the size of the problem and can therefore be considered as running in constant time.

We recall that the Crossover operator consists in the following tasks:

1. Duplication of encoding matrix of Parent A

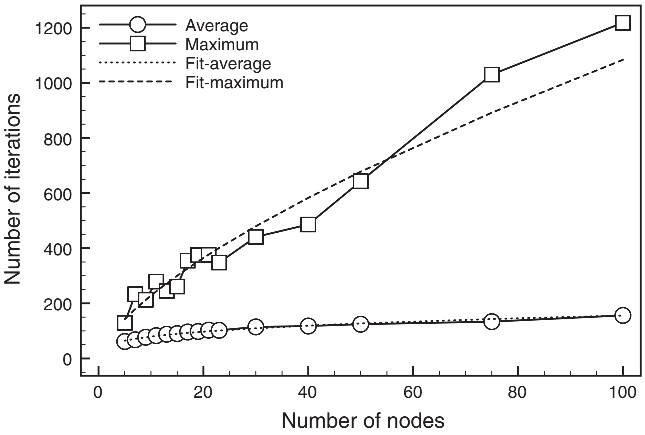


Fig. 16. Iterations to convergence in the EC algorithm.

2. Random selection of half of the elements of the encoding matrix of Parent B
3. Identification of those elements different from -1
4. Localization in the encoding matrix of the child of the elements identified in Step 3
5. Addition of the elements removed in Step 4

All the above tasks act on a matrix of size kn, by accessing, writing, or comparing a fraction of the elements of the encoding matrix, and are therefore of complexity $O(kn)$.

Even simpler is the case of the Mutation operator, which acts by just swapping two elements of two encoding matrices, and is therefore running in constant time.

After comparing the costs of the Selection, Crossover, and Mutation operators, we see that the dominant cost for each iteration of the genetic algorithm is $O(n^2)$. However, the number of iterations needed for convergence may be large, and we expect it to be dependent on the size of the network. In order to evaluate the order of dependence, we have extracted subnetworks of different size from the Géant topology, and applied the genetic algorithms to those subnetworks with the corresponding dataset of traffic matrices. In Fig. 16 we have plotted the average and the maximum number of iterations over the dataset of 1000 traffic matrices. We see that the number of iterations grows with the size of the network, embodied by the number of nodes, in a sublinear fashion, though we are aware that we cannot draw conclusions on the asymptotic behavior from such a limited data sample. We hypothesize that the number of iterations N_{iter} grows with the number of nodes according to a power law

$$N_{\text{iter}} = an^b. \tag{17}$$

By a least-squares fitting procedure, we find the values of the two fitting parameters reported in Table 7. In the same Fig. 16 we see the two best-fitting curves. The fit is nearly perfect for the average number of iterations, and quite good for the trend of the maximum value, though the latter shows significant oscillations around the trend, and larger values than the trend when the number of nodes is high (but the shape of the trend is preserved). If we consider the worst case, given by the maximum number of

Table 7
Best fit parameters for the number of iterations.

Statistic	Factor a	Exponent b
Average	40.411	0.293
Maximum	48.027	0.677

Table 8
Computational cost.

Algorithm	Computational cost
Newman	$O(m^2n)$
K-means	$O(n^2)$
Fast SVD	$O\left(n^3+\frac{k^2n^2}{\epsilon^4}\right)$
Spectral Filtering	$O(n^{k+1}\log n)$
Evolutionary	$O(n^{2.677})$

iterations, the overall computational cost of the EC algorithm is then $O(n^{2+b}) = O(n^{2.677})$.

The computational costs of all the algorithms are summarized in Table 8. They do not appear to be directly comparable, since some of them depend on other factors (additionally to the number of nodes), which are the number of links and the number of clusters. However, we see straight that the cost of K-means grows with the square of the number of nodes. Instead, the cost of Fast SVD grows faster than the cube of the number of nodes. The cost of Spectral Filtering is again faster than that cube as soon as the number of clusters is larger than two, a situation so frequent to be certain, excepting very small networks. As to the cost of Newman's algorithm, it depends on the relation between the number of links and that of nodes. Since the number of links grows at least proportionally to the number of nodes, Newman's algorithm also has a computational cost which grows as $\Omega(n^3)$, i.e., at least as fast as $O(n^3)$. Hence, all those algorithms, excepting K-means, exhibit a computational cost larger than our EC algorithm.

6. Conclusions

We have proposed a new evolutionary algorithm to cluster networks, based on traffic matrices, where existing clustering algorithms rely just on topology information, though augmented with link weights.

We have compared the performance of our EC algorithm with a different evolutionary algorithm (EvoCluster) as well as a selection of non-evolutionary algorithms well established in the literature: K-means, Spectral Filtering, Newman's, and Fast SVD. In the comparison we have considered both the aspects of the quality of the solution and the computational cost. Two metrics have been considered for the quality evaluation: the Traffic-aware Scaled Coverage Measure (deriving from the Scaled Coverage Measure, which we have adapted to work in a traffic-based context), and the Modularity measure. The comparison has been conducted on two real world datasets (pertaining to two networks of small size) and on a synthetic dataset (with larger size traffic matrices).

Our EC algorithm outperforms all the competing algorithms under all the aspects considered for the real world datasets. It exhibits a larger average score under both quality metrics, accompanied by a standard deviation lower than all the competitors, which means that its performance are quite stable. In the synthetic datasets, it outperforms all other methods for all sizes and under both metrics, with the only exception of Newman's algorithm when the modularity is used and the network is made of 75 nodes (and by just 1.5% even in that case). The computational cost, evaluated through an asymptotic analysis, is lower than all the competitors, excepting K-means.

References

[1] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Computing Surveys 31 (1999) 264–323.

[2] A.K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters 31 (2010) 651–666.

[3] S.E. Schaeffer, Graph clustering, Computer Science Review 1 (2007) 27–64.

[4] H. Ali, W. Shahzad, F.A. Khan, Energy-efficient clustering in mobile ad-hoc networks using multi-objective particle swarm optimization, Applied Soft Computing 12 (2012) 1913–1928.

[5] B.A. Attea, E.A. Khalil, A new evolutionary based routing protocol for clustered heterogeneous wireless sensor networks, Applied Soft Computing 12 (2012) 1950–1957.

[6] L. Laura, M. Naldi, G.F. Italiano, Traffic-based network clustering, in: A. Helmy, P. Mueller, Y. Zhang (Eds.), Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, IWCMC 2010, June 28–July 2, 2010, Caen, France, ACM, 2010, pp. 321–325.

[7] M. Sinclair, Improved model for European international telephony traffic, Electronics Letters 30 (1994) 1468–1470.

[8] T. Crnovrsanin, C.D. Correa, K.-L. Ma, Social network discovery based on sensitivity analysis, in: 2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, 20–22 July 2009, Athens, Greece, 2009, pp. 107–112.

[9] M.-J. Lin, K. Marzullo, Directional Gossip: Gossip in a Wide Area Network, Springer, 1999.

[10] S. Salcedo-Sanz, M. Naldi, L. Carro-Calvo, L. Laura, A. Portilla-Figueras, G.F. Italiano, An evolutionary algorithm for network clustering through traffic matrices, in: Proceedings of the 7th International Wireless Communications and Mobile Computing Conference, IWCMC 2011, 4–8 July, Istanbul, Turkey, 2011, pp. 1580–1584.

[11] P.C.H. Ma, K.C.C. Chan, X. Yao, D.K.Y. Chiu, An evolutionary clustering algorithm for gene expression microarray data analysis, IEEE Transactions on Evolutionary Computation 10 (2006) 296–314.

[12] A. Medina, C. Fraleigh, N. Taft, S. Bhattacharyya, C. Diot, Taxonomy of IP traffic matrices, in: V.F.Z.-L. Zhang (Ed.), Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 4868 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, 2002, pp. 200–213.

[13] G. Fiche, G. Hébuterne, Communicating Systems & Networks: Traffic & Performance, Kogan Page Science, 2004.

[14] M.E.J. Newman, Analysis of weighted networks, Physical Review E 70 (2004) 056131.

[15] P. Conti, L.D. Giovanni, M. Naldi, Blind maximum likelihood estimation of traffic matrices under long-range dependent traffic, Computer Networks 54 (2010) 2626–2639.

[16] P. Conti, L. De Giovanni, M. Naldi, Estimation of traffic matrices in the presence of long memory traffic, Statistical Modelling 12 (2012) 29–65.

[17] E.R. Hruschka, R.J.G.B. Campello, A.A. Freitas, A.C.P.L.F. de Carvalho, A survey of evolutionary algorithms for clustering, IEEE Transactions on Systems, Man, and Cybernetics, Part C 39 (2009) 133–155.

[18] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing (Natural Computing Series), Springer, 2008.

[19] E. Cantú-Paz, Order statistics and selection methods of evolutionary algorithms, Information Processing Letters 82 (2002) 15–22.

[20] H. Mühlenbein, D. Schlierkamp-Voosen, Predictive models for the breeder genetic algorithm, I: Continuous parameter optimization, Evolutionary Computation 1 (1993) 25–49.

[21] M. Kantardzic, Data Mining, Wiley-Interscience, 2003.

[22] J. Hartigan, Clustering Algorithms, Wiley, New York, 1975.

[23] P. Drineas, A.M. Frieze, R. Kannan, S. Vempala, V. Vinay, Clustering large graphs via the singular value decomposition, Machine Learning 56 (2004) 9–33.

[24] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69 (2004) 026113.

[25] C. Gkantsidis, M. Mihail, E.W. Zegura, Spectral analysis of internet topologies, in: INFOCOM 2003, San Francisco,30 March–3 April 2003, vol. 1, pp. 364–374.

[26] F. Chung, Spectral Graph Theory (CBMS Regional Conference Series in Mathematics No. 92) (Cbms Regional Conference Series in Mathematics), American Mathematical Society, 1997.

[27] P. Husbands, H. Simon, C. Ding, On the use of the singular value decomposition for text retrieval, in: Computational information retrieval, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001, pp. 145–156.

[28] S.V. Dongen, A Cluster Algorithm for Graphs, Technical Report INS-R0010, CWI, National Research Institute for Mathematics and Computer Science, 2000, 31 May.

[29] K. Praditwong, M. Harman, X. Yao, Software module clustering as a multi-objective search problem, IEEE Transactions on Software Engineering 37 (2011) 264–282.

[30] S. Uhlig, B. Quoitin, J. Lepropre, S. Balon, Providing public intradomain traffic matrices to the research community, SIGCOMM Computer Communication Review 36 (2006) 83–86.

[31] F. Vega-Redondo, Economics and the Theory of Games, Cambridge University Press, 2003.

[32] B. Silverman, Density Estimation, Chapman & Hall, 1986.

[33] H. Haddadi, M. Rio, G. Iannaccone, A.W. Moore, R. Mortier, Network topologies: inference, modeling, and generation, IEEE Communications Surveys and Tutorials 10 (2008) 48–69.

[34] R. Govindan, A. Reddy, An analysis of internet inter-domain topology and route stability, in: INFOCOM, 1997, pp. 850–857.

[35] M. Inaba, N. Katoh, H. Imai, Applications of weighted Voronoi diagrams and randomization to variance-based-clustering (extended abstract), in: Symposium on Computational Geometry, 1994, pp. 332–339.

[36] A. Vattani, *K*-means requires exponentially many iterations even in the plane, Discrete & Computational Geometry 45 (2011) 596–616.

[37] D. Arthur, B. Manthey, H. Röglin, Smoothed analysis of the *K*-means method, Journal of ACM 58 (2011) 19.

[38] P. Drineas, A.M. Frieze, R. Kannan, S. Vempala, V. Vinay, Clustering large graphs via the singular value decomposition, Machine Learning 56 (2004) 9–33.

[39] V.Y. Pan, Z.Q. Chen, The complexity of the matrix eigenproblem, in: STOC, 1999, pp. 507–516.

[40] G.T. Heineman, G. Pollice, S.M. Selkow, Algorithms in a Nutshell – A Desktop Quick Reference, O'Reilly, 2009.