Solving the Edge-Disjoint Paths Problem using a two-stage method

Bernardo Martín[a], Ángel Sánchez[a], Cesar Beltran-Royo[a], Abraham Duarte[a,*]

[a] *Universidad Rey Juan Carlos, Dept. Computer Sciences, C/Tulipán s/n, Móstoles (Madrid)*

Abstract

There exists a wide variety of network problems where several connection requests occur simultaneously. In general, each request is attended by finding a route in the network, where the origin and destination of such a route are those hosts that wish to establish a connection for information exchange. As it is well documented in the related literature, the exchange of information through disjoint routes increases the effective bandwidth, velocity, and the probability of receiving the corresponding information. The definition of disjoint paths may refer to nodes, edges, or both. As far as we know, the most studied variant is the one where disjointness implies not to share edges. This optimization problem is usually known as the maximum edge-disjoint paths (EDP) problem. This $\mathcal{NP}$-hard optimization problem has applications in real-time communications, VLSI-design, scheduling, bin packing, or load balancing. The proposed approach hybridizes an Integer Linear Programming formulation of the problem with an Evolutionary Algorithm. Empirical results using 174 previously reported instances show that the proposed procedure compares favorably to previous metaheuristics for this problem. We finally confirm the significance of the results by conducting non-parametric statistical tests.

1. Introduction

In communication networks, a node may either be a data communication equipment (modem, hub, bridge, or switch) or a data terminal equipment (telephones, printers, or host computers). A network is usually modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of nodes $\mathcal{V}$ corresponds to terminal/communication equipments and the set of edges $\mathcal{E}$ corresponds to the physical links. For the sake of clarity, the number of nodes and edges are denoted as $|\mathcal{V}| = V$ and $|\mathcal{E}| = E$, respectively. Each edge $e \in \mathcal{E}$, with $e = (i, j)$ and $i, j \in \mathcal{V}$, has a weight $w(e) \in \mathbb{R}^+$, which usually represents the physical distance between both endpoints.

Let $\mathcal{T} = \{[i_k, j_k] \mid i_k \neq j_k \in \mathcal{V}, \quad 1 \leq k \leq K\}$ be a set of pairs of nodes, called terminal nodes or terminals, that request to be connected with a disjoint route, that is, an edge-disjoint path P_k in $\mathcal{G}$. According to [36], the maximum edge-disjoint paths (EDP) problem consists in maximizing the number of such edge-disjoint paths.

Figure 1 shows an example of an EDP problem with 16 nodes and 16 edges. Terminal and transshipment nodes correspond to the white and black ones, respectively. In this example, there are four pairs of terminals $[i_1, j_1], [i_2, j_2]$, $[i_3, j_3]$, $[i_4, j_4]$, that request to be connected with disjoint paths.

In Figure 2, we depict a feasible solution (i.e., each edge is used exclusively by one path), where involved paths have been highlighted with dotted lines. As it is easy to check, this is a feasible solution composed of two paths $\{i_1, 1, 2, 3, 4, j_1\}$ and $\{i_4, 8, 7, 6, 5, j_4\}$. Therefore, for this feasible solution, the value of the objective function is equal to 2.

This problem presents a wide range of applications in real-time communications, where several connection requests occur simultaneously [36]. Specifically, each request is attended by finding a route in the

*Corresponding author

Email addresses: b.martinn@alumnos.urjc.es (Bernardo Martín), angel.sanchez@urjc.es (Ángel Sánchez), cesar.beltran@urjc.es (Cesar Beltran-Royo), abraham.duarte@urjc.es (Abraham Duarte)

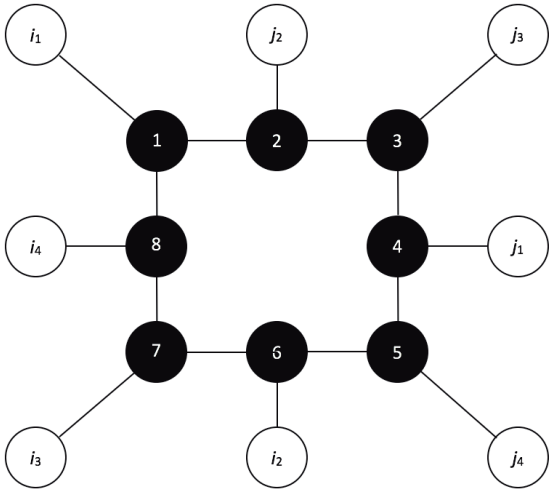

Figure 1: Example of EDP problem (graph).

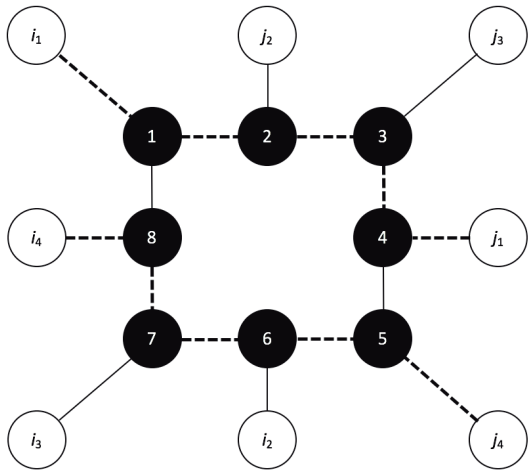

Figure 2: Example of EDP problem (feasible solution).

network, where the origin and destination of such a route are those hosts that wish to establish a connection for information exchange. That interchange is performed by means of disjoint routes, since it increases the effective bandwidth, velocity, and the probability of receiving the corresponding information. Simultaneously, congestion and possible failures are reduced (see [5][6] for further details). Regarding the practical application areas of the EDP problem, we can mention the two following ones: very-large-scale-integration (VLSI) design [11] and virtual circuit routing [2][31]. In VLSI design, the EDP problem is similar to the escape problem [11], where it is required to find a maximal number of wiring disjoint paths in grids (i.e., that ones which avoid the overlaps) between pairs of terminals. In virtual circuit routing, a path for each communication request is needed to be reserved in advance, in the sense that once a communication is established no interruptions will occur. Virtual circuit routing presents a practical application [2] for database servers and large-scale video servers.

The contribution of this paper corresponds to propose a two-stage approach for the EDP problem. Specifically, this approach combines in sequence an Integer Linear Programming (ILP) model and an Evolutionary Algorithm (EA). The computational results show that our procedure obtains better performance than previous approaches in both, quality of the solution and computing time.

The rest of the paper is organized as follows. Section 2 summarizes the state-of-the-art of the EDP problem and related work. Sections 3 and 4 describe the respective ILP and EA formulations for the considered EDP problem. Section 5 introduces the proposed two-stage approach devised for this problem. Section 6 describes the computational experience conducted to test the effectiveness of our approach. Finally, the paper ends in Section 7 by drawing the most relevant conclusions.

2. Related work

The EPD problem is different from maximizing the number of edge-disjoint paths that connect a single pair of terminal nodes, say $[s, t]$, which is known in the literature as the Edge-Disjoint Menger (EDM) problem [8]. The paths obtained after solving the EDM problem are $s-t$ paths, but there is no guarantee of connecting any pair of nodes from the set of pairs $\mathcal{T}$. By Menger theorem ([1], Theorem 6.7) the maximum number of edge-disjoint $s-t$ paths equals the minimum number of edges whose removal from the network disconnects all $s-t$ paths (min cut), which can be computed by solving a maximum flow problem (max-flow min-cut theorem [1]). Therefore, the EDM problem is solvable in polynomial time, while the EDP problem is $\mathcal{NP}$-complete for general graphs as it was proven in [32]. In fact, it is one of the classical $\mathcal{NP}$-complete problems described in [25]. The problem remains $\mathcal{NP}$-complete even when considering specific graphs. In particular, for directed graphs where the number of pairs of terminals is larger than one [23], for meshes [38], planar graphs of maximum degree 3 [46], or in directed/undirected complete graphs [22].

Substantial effort has been devoted to the identification of polynomial-time solvable special cases. We refer the reader to [24, 53] for detailed descriptions. For example, the EDP version where k is fixed *a priori* can be polynomially solved [50]. Similarly, for acyclic digraphs where K is fixed, for planar graphs where the number of pairs of terminals are bounded by the number of faces of the graph, for interval graphs [28], undirected/directed chains [20], or undirected trees [26].

Another interesting line of research is based on the design of approximation algorithms. For instance, the shortest-path-first greedy algorithm is proposed in [37]. This procedure connects pairs of terminals in non-decreasing order of the shortest path (i.e., instead of considering an arbitrary order). As it is proven in [20], this method has an approximation ratio at most $\sqrt{E}$ for the EDP problem in directed or undirected graphs. This factor is not right for dense graphs due to the large number of edges. This theoretical result is further improved in (Chekuri and Khanna, 2003 [12]), establishing a new approximation ratio $\mathcal{O}(\min\{\sqrt{E}, V^{\frac{2}{3}}\})$ in undirected graphs and $\mathcal{O}(\min\{\sqrt{E}, (V \log V)^{\frac{2}{3}}\})$ in directed graphs. Another interesting result is reported in [29] for directed graphs, where it is proved that there cannot be a $\rho = E^{(\frac{1}{2}-\epsilon)}$-approximation algorithm for any $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$. On the other hand, in specific graphs, [21, 18] demonstrated that there exists ρ-approximation algorithms for bidirected trees ($\rho = \frac{5}{3} + \epsilon$), undirected trees of rings ($\rho = 3$) and directed trees of rings ($\rho = 5 + \epsilon$).

Kleinberg and Tardos [35] found a randomized $\mathcal{O}(1)$-approximation algorithm for the EDP problem in two-dimensional meshes. Furthermore, they generalized the algorithm to obtain an approximation ratio of

$\mathcal{O}(1)$ also for the class of densely embedded, nearly-Eulerian graphs. Notice that there were some previous works in this line for general meshes [3, 34].

We focus our research on those procedures that can deal with general graphs. Specifically, general graphs have been addressed in [8] by means of an Ant Colony Optimization algorithm. The procedure is based on a decomposition of the problem into simpler subproblems. In the computational experience, these authors showed that the proposed method outperformed a multi-start greedy procedure in both, solution quality and computing time. The same authors improved the previous algorithm in [9]. The new approach incorporated a parallel construction of all paths and the use of candidate list strategies for the exploitation of the promising choices at each construction step. Additionally, the authors improved a local search strategy by considering two different criteria of the objective function, and the partial destruction of the currently best solution as an escape mechanism.

In [16], a constraint-based local search framework for Constrained Optimum Tree (COT) and Constrained Optimum Path (COP) applications is introduced. We focus on COP since the EDP problem belongs to this family of optimization problems [16]. The proposed method introduces an effective neighborhood structure based on replacing edges in a routed spanning tree. The authors also present a second neighborhood based on more complex moves. Specifically, they define chained independent moves as those that the application of one move does not affect the rest of the moves. Local search methods, that use these neighborhoods, minimize the number of violated constraints for the corresponding COP. In order to further specialize the general COP method to the EDP context, the authors hybridize it with a method that explicitly removes those paths that have one or more edges in common. The experimental results show that the proposed method gives better solutions than the naive greedy algorithm (based on a multi-start version of the shortest path algorithm).

Hsu and Cho [30] presented a genetic algorithm (GA) for solving the EDP problem on general graphs. Specifically, each individual in the population is encoded as a set of $|\mathcal{T}|$ paths and each path is represented by a string of real values with length $|\mathcal{V}|$ in the interval $[0, 1]$. Each of these real values denotes the "priority" of a node to be selected in the path under construction. More precisely, given a pair of terminals, $[i_k, j_k]$, the path starts at the origin node i_k. The candidate nodes to be incorporated in the path are those ones (not previously used) with larger values of priority. The method then checks whether the inclusion of the selected node produces a feasible solution (including it in the path) or not (backtracking as many steps as necessary). This construction procedure stops when the selected node is j_k. The crossover operator produces an offspring by using a linear combination of two parents. The mutation operator reverses the value of the corresponding real value (i.e., large priorities become small priorities and vice versa). The GA considers also a self-adaptation operator that improves the quality of a particular individual. Specifically, it tries to connect those terminal nodes not connected in the solution by updating the real values with information of the shortest path, barely used edges, or length of the path, among others.

Recently, Altarelli et al. [2] propose a distributed method to solve the EDP based on message-passing techniques. This algorithm has been previously used in other scientific fields [44, 45, 13]. The authors first describe how the EDP problem can be easily solved in trees. Then, they derive a min-sum recursive equation that can be used for message-passing. In case of locally tree-like graphs, such as sparse graphs, the method becomes approximate, but still useful for the EDP problem. In order to further improve the performance of the method, the proposed algorithm transforms the aforementioned recursive equation to the computation of the maximum weight matching problem [40] on an auxiliary complete graph. This transformation allows the message-passing algorithm to execute each iteration of the method in polynomial time (in the number of edges) and linear time (on average) for sparse random graphs.

3. Integer linear programming formulation for the EDP problem

In the EDP problem there are, essentially, a set of pairs of terminal nodes $\mathcal{T}$ that one wishes to connect by edge-disjoint paths. The EDP problem can be modeled by means of a path-based Integer Linear Programming (ILP) formulation [12, 36]. However, this formulation requires a number of variables which is exponential in the size of the graph and, therefore, it does not result to be practical for the exact solution of the EDP problem. In this paper, we propose a binary multi-commodity network flow formulation of the

Parameter	Values	Description	
$\mathcal{V}$	$\{1,\ldots,K,\ldots,V\}$	Set of nodes	
$\mathcal{V}^-$	$\{1,\ldots,K\}$	Supply (source) nodes	
$\mathcal{V}^+$	$\{K+1,\ldots,K+K\}$	Demand (sink) nodes	
$\mathcal{V}^0$	$\mathcal{V}\setminus(\mathcal{V}^-\cup\mathcal{V}^+)$	Transshipment nodes	
i, j, v	-	Indexes for nodes	$i,j,v\in\mathcal{V}$
$\mathcal{E}$	-	Set of edges	
(i,j)	-	Index pair for edges	$(i,j)\in\mathcal{E}$
$\mathcal{G}$	$(\mathcal{V},\mathcal{E})$	Undirected graph	
$\mathcal{K}$	$\{1,\ldots,K\}$	Set of *commodities*	
k	-	Index for *commodities*	$k\in\mathcal{K}$
x_{ijk}	$\{0,\ 1\}$	Commodity-k flow from i to j (adjacent nodes)	$i,j\in\mathcal{V},\ k\in\mathcal{K}$

Table 1: Notation for the ILP formulation.

EDP problem, which requires a polynomial number of variables in the size of the graph. As far as we know, this is the first polynomial size ILP formulation of the EDP problem that has been proposed in the literature. This new formulation allows for the exact solution of medium size EDP instances by using standard MILP solvers as, for example, CPLEX and GUROBI (see Section 6). Thinking in terms of flow, this formulation is focused on sending a single unit of flow from terminal node i_k to terminal node j_k, for all pairs $[i_k, j_k]\in\mathcal{T}$. In this way, the commodity-k flow defines a path from node i_k to node j_k associated to the k-th commodity, for all $k\in\mathcal{K}=\{1,\ldots,K\}$.

We have used the notation shown in Table 1. Basically, graph nodes are classified into three groups: supply nodes ($\mathcal{V}^-$), demand nodes ($\mathcal{V}^+$), and transshipment nodes ($\mathcal{V}^0$). The nodes of the first group are labeled by the commodity index k and the nodes of the second group are labeled by $K+k$. With this notation, each pair or terminal nodes $[i_k, j_k]$ is labeled as $[k, K+k]$ for all $k\in\mathcal{K}$. Notice that any node $v\in\mathcal{V}^-$ is a supply node for commodity $k=v$, but it is also a transshipment node for the other commodities (see Equation 4). Similarly, any node $v\in\mathcal{V}^+$ is a demand node for commodity $k=v-K$, but it is also a transshipment node for the other commodities (see Equation 8).

We propose the following ILP formulation for the EDP problem:

$$\max_{x} \quad z_{EDP} = \sum_{v\in\mathcal{V}^-} \sum_{(v,j)\in\mathcal{E}} x_{vjv} \tag{1}$$

$$\text{s.t.} \quad \sum_{(v,j)\in\mathcal{E}} x_{vjv} \leq 1 \qquad v \in \mathcal{V}^- \tag{2}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivv} = 0 \qquad v \in \mathcal{V}^- \tag{3}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v \in \mathcal{V}^-, \quad k \in \mathcal{K}\backslash\{v\} \tag{4}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v \in \mathcal{V}^0, \quad k \in \mathcal{K} \tag{5}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} \leq 1 \qquad v \in \mathcal{V}^+, \quad k = v - K \tag{6}$$

$$\sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v \in \mathcal{V}^+, \quad k = v - K \tag{7}$$

$$\sum_{(i,v)\in\mathcal{E}} x_{ivk} - \sum_{(v,j)\in\mathcal{E}} x_{vjk} = 0 \qquad v \in \mathcal{V}^+, \quad k \in \mathcal{K}\backslash\{v - K\} \tag{8}$$

$$\sum_{k\in\mathcal{K}} x_{ijk} + \sum_{k\in\mathcal{K}} x_{jik} \leq 1 \qquad (i,j) \in \mathcal{E}, \quad i < j \tag{9}$$

$$x_{ijk} \in \{0,1\} \qquad (i,j) \in \mathcal{E}, \quad k \in \mathcal{K}, \tag{10}$$

where we would like to highlight:

- Eq. (1): We maximize the total flow sent from source nodes. At the end of this section, we will see that this is equivalent to maximizing the number of edge-disjoint paths.
- Eq. (2): Each source node v can send at most one unit of commodity-v flow.
- Eq. (3): Each source node v cannot receive commodity-v flow. This avoids the generation of cycles beginning and ending at source nodes.
- Eq. (4): At each source node v and for each commodity k, the flow supply is equal to the flow demand, for all $k \in \mathcal{K}\backslash\{v\}$.
- Eq. (5): At each transshipment node and for each commodity, the flow supply is equal to the flow demand.
- Eq. (6): Each sink node v can receive at most one unit of commodity-$(v-K)$ flow. Notice that sink node $v = K + k$ is meant to receive commodity-k flow with $k = v - K$.
- Eq. (7): Each sink node v cannot send commodity-$(v-K)$ flow.
- Eq. (8): At each sink node v and for each commodity k, the flow supply is equal to the flow demand, for all $k \in \mathcal{K}\backslash\{v-K\}$.
- Eq. (9): The joint flow capacity of each edge is one. Notice that commodity flows can use each edge $(i,j) \in \mathcal{E}$ in any of the two directions (but not both).

The validity of formulation (1)-(10) for the EDP problem can be seen as follows. First, let us see that, given a source node $\bar{v} \in \mathcal{V}$ and an edge $(\bar{v}, j) \in \mathcal{E}$, if we have $x_{\bar{v}j\bar{v}} = 1$ then, there exists one path connecting the pair of terminal nodes $[\bar{v}, K+\bar{v}]$. The details are as follows: The equality $x_{\bar{v}j\bar{v}} = 1$ means that the source

node $\bar{v}$ sends a unit of commodity-$\bar{v}$ flow to node j. Equations (3), (5) and (7) enforce that all the nodes of the graph behave as transshipment nodes for commodity $\bar{v}$, except the source node $\bar{v}$ and the corresponding sink node $K+\bar{v}$. The number of edges is finite and each edge admits, at most, one unit of flow. Therefore, this flow transmission initiated at source node $\bar{v}$ will necessarily end up at sink node $K+\bar{v}$. As a consequence, we will have $x_{ivk} = 1$ for some $i \in \mathcal{V}$, for $v = K+\bar{v}$ and for $k = \bar{v}$. The commodity-$\bar{v}$ flow just described defines a path connecting the pair of terminal nodes $[\bar{v}, K+\bar{v}]$.

Second, let us see now, given two source nodes $\bar{u}$ and $\bar{v}$, if we have $x_{\bar{u}i\bar{u}} = x_{\bar{v}j\bar{v}} = 1$ then the paths connecting the corresponding pairs of terminal nodes are disjoint. This is directly enforced by Equation (9).

Let us define EDP^* as the maximum number of edge-disjoint paths associated to a pair $(\mathcal{G}, \mathcal{T})$ and let us consider z^*_{EDP}, the optimal value of the corresponding ILP formulation (1)-(10). So far we have shown that $z^*_{EDP} \leq EDP^*$.

Third, let us see now that $EDP^* \leq z^*_{EDP}$. This is clear, since given a pair of terminal nodes, say $[\bar{v}, K+\bar{v}]$ connected by an edge-disjoint path P, it is possible to send one unit of commodity-$\bar{v}$ flow from node $\bar{v}$ to node $K+\bar{v}$ through P. In the ILP formulation (1)-(10), this flow corresponds to set equal to one some variables, in particular $x_{\bar{v}i\bar{v}} = 1$ for some edge $(\bar{v}, i) \in \mathcal{E}$, such that $\bar{v} \in \mathcal{V}^-$, which implies $EDP^* \leq z^*_{EDP}$ as we wanted to see.

All in all, we have seen that $EDP^* = z^*_{EDP}$. Since by construction, the ILP formulation (1)-(10) generates a set of edge-disjoint paths with cardinality z^*_{EDP}, we conclude that it is a valid formulation of the EDP problem.

4. Evolutionary Algorithm for the EDP problem

Evolutionary Algorithms (EAs) [27] are random search procedures inspired by the Darwinian theory of natural evolution (i.e., the notion of survival of the fittest). In this context, potential solutions to a specific optimization problem are coded by a chromosome structure, called individual. The set of individuals is usually known as population. In order to evolve the initial population, an EA successively transforms it by means of random operators (selection, crossover, and mutation).

4.1. Solution representation and initial population

In this paper, we use an EA approach to solve the EDP problem, where each individual of the population is an array of size K (being K the number of pairs of terminals). We store in each item of the array information about paths between terminals. More precisely, let p be an individual in the population $\mathcal{P}$ and let $[i_k, j_k]$ be a pair of terminals in $\mathcal{T}$. Then, considering the typical array notation, if $p[k] = \emptyset$ (with $1 \leq k \leq K$) there is not a disjoint path between i_k and j_k in the corresponding solution; otherwise, $p[k] = \{i_k, \ldots, j_k\}$ is a valid disjoint path in $\mathcal{G}$. Let us consider the example shown in Figures 1 and 2, where there are four pairs of terminals (i.e., $K = 4$) that should be connected with a disjoint path. Then, the feasible solution in Figure 2 is represented as $p = [\{i_1, 1, 2, 3, 4, j_1\}; \emptyset; \emptyset; \{i_4, 8, 7, 6, 5, j_4\}]$.

The proposed EA procedure starts by generating a population of different individuals, where each one represents a feasible solution. As before-mentioned, a solution of the EDP problem is a set of disjoint paths, where each path connects a particular pair of terminals. In order to initialize our EA procedure, we propose to construct λ shortest paths between each pair of terminals by using the algorithm described in [54], where λ is a predetermined number. This procedure uses two sets, A and B. The first one contains the actual shortest path, while the second one stores tentative/candidate shortest paths. At the beginning of the algorithm, the set A is initialized with the shortest path by using, in our case, the well-known Dijkstra algorithm. The method then exploits the idea that the current shortest paths (say for instance k) may share edges and sub-paths (i.e., a path from a given source to any intermediary node within the path) with the previous one.

Specifically, this procedure considers the $(k-1)$-th shortest paths and makes each node in the path unreachable (by removing a particular edge that has as endpoint the corresponding node). Once the node is unreachable, the procedure finds the shortest path from the preceding node to the destination. Then a new path is created by appending the common sub-path (from source to the preceding node of the unreachable

node) and adds the new shortest path from preceding node to destination. This route is then added to the set B, testing that it was not previously constructed in either A or B. After repeating this strategy for all nodes in the route, we select the shortest path in set B and move it to A. This procedure is repeated for λ iterations obtaining the λ shortest paths between an origin and a destination.

Given the pair $[i_k, j_k] \in \mathcal{T}$, we construct λ paths from i_k to j_k and store them in the set $SP(k)$, where k identifies the specific pair of terminals. We assume that the paths are sorted in ascending order (from the shortest to the longest. For the sake of clarity, we represent this set as $SP(k) = \{SP(k, x) \quad | \quad 1 \leq x \leq \lambda\}$, where $SP(k, 1)$ is the shortest path between i_k and j_k, $SP(k, 2)$ is the second shortest path between i_k and j_k, and so on. Notice that the set $SP(k)$ for each pair of terminals $[i_k, j_k]$ is only computed once as a preprocessing step at the beginning of the procedure.

Once we have constructed λ paths between each pair of terminals, the goal is to construct a feasible solution (i.e., individual of the population). Specifically, the proposed algorithm randomly selects one of the pairs (say for instance $[i_x, j_x]$). Then, this pair is joined with the shortest path, i.e., $SP(x, 1)$. This path is then included in the solution under construction. After that, a different pair of terminals, $[i_y, j_y]$, is selected also at random. Paths in $SP(y)$ are scanned from the shortest, $SP(y, 1)$, to the longest, $SP(y, \lambda)$, verifying whether the inclusion of the corresponding path produces a feasible solution (i.e. it does not share any edge with the paths already incorporated in the solution under construction) or not. If so, the path is actually included in the partial solution; otherwise it is discarded. This process finally ends when all the pairs of terminals have been scanned, returning the solution as a set of disjoint paths. The process described in this paragraph is successively repeated a number of $size$ times to generate the initial population of $size$ individuals or solutions.

4.2. Crossover operator

Classical crossover operators usually consist in combining a pair of solutions randomly selected from the current population [27]. This strategy usually obtains high-quality results in optimization problems where solutions are coded with basic data structures (i.e., string of binary numbers, array of integer/real values, etc.). Considering the proposed solution representation for the EDP, the combination of solutions is a really hard task. In this paper, we propose a crossover operator where each new individual is constructed by combining three different individuals: the incumbent one and two other individuals selected at random from the whole population.

The aforementioned mechanism is further improved by taking ideas from Particle Swarm Optimization (PSO) [17]. In particular, instead of combining the incumbent individual with other two individuals selected at random, we propose to combine it with other two fit individuals. More precisely, being this an iterative process, let p^t be individual p of the population $\mathcal{P}$ at iteration t. We call p^t the t-th evolution of individual p. Otherwise said, we consider that individual p evolves at each iteration: $p^1, p^2, \ldots, p^t, \ldots$. Similarly, let l^t be the so far best historical solution (in terms of the objective function) of individual p (the best one from p^1 to p^t). Finally, let g^t be the best individual found so far in the whole population. Assuming that the objective function value of these individuals (solutions) can be computed with the cardinal operator, $|\cdot|$ (number of pairs of terminals connected), it trivially holds that $|p^t| \leq |l^t| \leq |g^t|$.

The proposed crossover operator is designed with the idea of maintaining those high-quality structures obtained by means of evolution. Specifically, we consider a variation of the well-known uniform crossover [14] which uses the structural information of each individual, providing better descentants [48]. Given the three individuals involved in the combination, p^t, l^t, g^t, if they have the same value for a particular gene, the descendant individual inherits that value. Otherwise, the gene value is randomly selected according to the objective function value of each combined individual. In mathematical terms, the k-th gene value in the next generation is computed as:

$$p^{t+1}[k] = \begin{cases} p^t[k] & \text{if} \quad 0 \leq r < \psi_1 \\ l^t[k] & \text{if} \quad \psi_1 \leq r < \psi_2 \\ g^t[k] & \text{if} \quad \psi_2 \leq r \leq \psi_3 \end{cases} \tag{11}$$

where r is a random number in the interval $[0, \psi_3]$ and ψ_1, ψ_2, ψ_3 are defined as:

$$\psi_1 = \varphi_1 \cdot |p^t| \tag{12}$$

$$\psi_2 = \varphi_1 \cdot |p^t| + \varphi_2 \cdot |l^t| \tag{13}$$

$$\psi_3 = \varphi_1 \cdot |p^t| + \varphi_2 \cdot |l^t| + \varphi_3 \cdot |g^t| \tag{14}$$

being $\varphi_1, \varphi_2, \varphi_3$ weight factors that control the influence of each crossovered individual.

Let us focus on the particular pair $[i_k, j_k] \in \mathcal{T}$. In this case, if $r < \psi_1$, the corresponding pair keeps its previous status (either non-connected or connected with the same disjoint path as in $p^t[k]$). On the other hand, if $\psi_1 \leq r < \psi_2$, then $p^{t+1}[k]$ is set to $l^t[k]$, which might produce an infeasible solution (for example, if $l^t[k] \neq \emptyset$ and $l^t[k] \neq p^t[k]$). If this situation happens, we eliminate as many paths as necessary to guarantee the feasibility of p^{t+1}. Similarly if $\psi_2 \leq r \leq \psi_3$ the assignment of $g^t[k]$ to $p^{t+1}[k]$ could again produce an unfeasible solution. As before, we remove as many paths as necessary to keep the feasibility.

The crossover probability is directly set to 100%, which means that all the individuals of the population are selected for crossover. We have experimentally tested that this approach drives to significantly better results. It is worth mentioning that pairs of terminals are scanned at random with the goal of favoring the diversification of the search.

4.3. Post-processing improvement

Mutation strategies are usually introduced in EAs with the objective of favoring the diversification of the search. This mechanism is traditionally implemented by changing the value of a given allele at random. In the case of the EDP problem, the representation of a solution is quite complex. Therefore, an straightforward implementation of a mutation operator drives to either non-feasible solutions or low-quality solutions. We then propose a different approach. Specifically, instead of looking for diversification, we replace the traditional mutation operator with a post-processing improvement strategy, whose main objective is to reinforce the intensification of the search. Specifically, the proposed algorithm checks whether non-connected terminals could be reconnected with a disjoint path or not. In particular, let $[i_k, j_k]$ be a non-connected path in p^{t+1}. We then scan paths in the previously constructed set of shortest paths $SP(k)$. This set is scanned from the shortest, $SP(k, 1)$, to the largest, $SP(k, \lambda)$, verifying whether the inclusion of a particular path produces a feasible solution (it does not share any edge with the paths already incorporated in p^{t+1}) or not.

All the individuals of the population at the current iteration t are subjected to this procedure. In order to avoid biasing the search, individuals are scanned at random. When this method ends, the EA procedure is ready to start a new iteration. Finally, note that the final result returned to the output will be the best solution found g^{t+1}.

5. Two-stage solution procedure

Exact algorithms guarantee the optimality of the solution for any combinatorial optimization problem (ignoring a time limit). Some well-known exact methods are, for example, Branch-and-Bound, Branch-and-Cut, Branch-and-Price, Lagrangean Relaxation, or Dynamic Programming. With the development of general-purpose solvers, such as GUROBI or CPLEX, exact algorithms are now able to solve some difficult problems with moderate size. In fact, if the algorithm succeeds (i.e., it is stopped before completion time) optimal solutions are obtained. Otherwise, at least, information about upper/lower bounds to the optimal solution can be obtained. Finally, exact methods allow to prune parts of the search space in which optimal solutions cannot be located. However, many interesting problems in Science and Engineering are often very difficult to solve to optimality due to the huge size of the search space. As it is described in [15], other additional problems may arise. For example: the memory required by an exact algorithm may lead to the early abortion of a general-purpose solver; high performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change; and for many combinatorial problems

the best performing exact algorithms are highly problem dependent and thus, they require large execution times.

Metaheuristics (MHs) are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial optimization problems arising in human activities, such as economics, industry, or engineering. MHs can be seen as general algorithmic frameworks that require relatively few modifications to be adapted to tackle a specific problem. They constitute a very diverse family of optimization algorithms including methods such as simulated annealing (SA), tabu search (TS), genetic algorithms (GA), ant colony optimization (ACO), or variable neighborhood search (VNS). These approximate algorithms mainly differ in how each one constructs, combines and/or improves a solution (trajectory-based MHs) or a set of solutions (population-based MHs). Although these methods find high-quality solutions in reduced computing times, they do not guarantee the optimality. In fact, it is not possible to determine the distance between the MH solution and the optimal one.

The proposed procedure tries to combine the two aforementioned algorithmic techniques in an effective way. The two-stage approach proposed in this paper is focused on calculating high quality solutions for the EDP. It is composed of an exact algorithm, in order to solve the Integer Linear Programming (ILP) formulation, combined with an Evolutionary Algorithm (EA).

We show in Algorithm 1 the pseudo-code of the proposed procedure. It has 7 input parameters ($time_{max}, \alpha, \lambda, \varphi_1, \varphi_2, \varphi_3, size$). The total time of the whole procedure is named as $time_{max}$. The parameter α determines the percentage of the total time used to execute the ILP procedure. Then, $(1-\alpha) \cdot t_{max}$ is the maximum allowed time for the EA. The parameter λ determines the number of shortest paths constructed between each pair of terminal nodes (see Section 4.1). The crossover operator uses a roulette wheel strategy to combine three individuals of the population (see Section 4.2), where the weight of each individual is controlled with parameters φ_1, φ_2, and φ_3. The last parameter, $size$, controls the number of individuals in the population.

Our method starts by solving the ILP formulation of the EDP problem (see Section 3) with CPLEX. The solver is executed for $ILP_{time} = \alpha\ time_{max}$ seconds (step 2). The exact algorithm tries to compute an optimal solution within a limited computing time. If it succeeds (i.e, $opt =$ `true`), the procedure stops and returns the optimal solution found (step 4). Otherwise, the suboptimal solution is used by the EA algorithm as a seed to guide the search strategy (steps 6 to 23).

The EA stage updates relevant variables first (steps 6 to 7). Then, the procedure constructs the initial population in steps 8 to 12 with the procedure described in Section 4.1. Considering that there is no previous information for each individual, the historical solution is set to the current one (step 10). Before starting the evolution of the initial population, the best global solution is updated in step 13.

The evolution of the population is executed for $(1-\alpha) \cdot time_{max}$ seconds. In each iteration, solutions stored in individuals are generated as follows. First, a temporary individual is constructed with the procedure described in Section 4.2 (step 17). Then, that individual is improved with the method introduced in Section 4.3 (step 18). After that, for each individual the best historical solution is updated (if necessary) (step 19). Once all individuals are combined and improved, the so far best individual is updated (if required) in step 21 and the generation counter is increased (step 22). When the running time reaches the maximum allowed time, the evolution ends returning the best individual (solution) found so far (step 25).

As we will see, the ILP and the EA algorithms combined in our two-stage approach outperform both procedures when used separately. From the EA point of view, the ILP solution gives a good starting seed for combining solutions and, from the ILP point of view, the EA approach rapidly improves suboptimal ILP solutions. Therefore, both strategies can be considered complementary and their hybridization can yield effective methods.

The rationale behind the choice of an unusual design of a EA is based on the following two empirical observations. First, in the context of EDP, we have observed that exploration (diversification) is as important as exploitation (intensification). For that reason, we select from the whole catalog of MHs a population-based one, where the diversity is maintained by considering a relatively large set of solutions. Second, we have also observed that classical recombination and mutation operators do not usually yield promising outcomes in short computing times.

Algorithm 1: `Two-stage approach`

`Input` :$time_{max}, \alpha, \lambda, \varphi_1, \varphi_2, \varphi_3, size$
`Output` :g^t

{start ILP Phase}
$(g^0, opt) \leftarrow$ `solveWithCPLEX`$(\alpha \cdot time_{max})$
if $(opt =$ `true`$)$ **then**
  return g^0
else
  {start EA Phase}
  $\mathcal{P} \leftarrow \emptyset$
  for $(1 \leq m \leq size)$ **do**
    $p_m^1 \leftarrow$ `createIndividual`(λ)
    $l_m^1 \leftarrow p_m^1$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_m^1\}$
  end for
  $g^1 \leftarrow$ `updateBest`$(g^0, \mathcal{P})$
  $t \leftarrow 1$
  while (`terminationCondition`$((1-\alpha) \cdot$ `time`$_{\texttt{max}}$)) **do**
    for $1 \leq m \leq size$ **do**
      $temp \leftarrow$ `crossover`$(p_m^t, l_m^t, g_m^t, \varphi_1, \varphi_2, \varphi_3)$
      $p_m^{t+1} \leftarrow$ `postProcessing`$(temp)$
      $l_m^{t+1} \leftarrow$ `updateLocal`(p_m^{t+1}, l_m^t)
    end for
    $g^{t+1} \leftarrow$ `updateBest`$(g^t, \mathcal{P})$
    $t \leftarrow t + 1$
  end while
end if
return g^t

6. Experimental results

This section reports the computational experiments that we have performed for testing the efficiency of the proposed two-stage procedure for solving the EDP problem. The ILP was solved with CPLEX (version 12.7) while the EA was implemented in Java 8. All the experiments were conducted on a Pentium(R) Dual-Core CPU at 3.20GHz, and 4GB of RAM memory. We have considered two sets of instances previously used in this problem.

- Set 1 (120 instances) [9]. This set contains the following families of graphs: `graph3` (164 vertices and 370 edges) and `graph4` (434 vertices and 981 edges), whose structure resembles parts of the communication network of the Deutsche Telekom AG in Germany; `AS.BA.R-Wax.v100e217` (100 vertices and 217 edges) and `bl-wr2-wht2.10-50` (500 vertices and 1020 edges) generated with BRITE [43] by considering different topology properties (degree, diameter, clustering coefficient, etc.); and `mesh 25x25` (625 vertices and 1200 edges). Each family contains 20 instances and the pair of terminals ranges from $0.10 \cdot V$ to $0.40 \cdot V$ and the weight of each edge is 1.
- Set 2 (54 instances) [16]. This set contains the following families of graphs: two internet-like topologies (`blrand` and `blsdeg` with 500 vertices and 1020 edges) generated with BRITE [43] with the parameters recommended in [9]; two mesh graphs of sizes denoted as `mesh15x15` (225 vertices and 420 edges) and `mesh25x25` (625 vertices and 1200 edges); two families of graphs derived from the Steiner problem (`steinb` and `steinc` where the number of vertices and edges ranges from 100 to 500 and from 100 to 12500, respectively); and planar graphs, denoted as `plan`, where the number of vertices and edges ranges from 100 to 50 and from 135 to 1477, respectively. Each family contains 12 instances (with the exception of `steinb` and `steinc` that contain 9 instances each one) and the number of pair terminals are $0.10 \cdot V$, $0.25 \cdot V$, and $0.40 \cdot V$. Finally, the weight of each edge is set to 1.

We have divided this section into two different parts: preliminary experimentation and final experimentation. The first one is performed to adjust the values of the key search parameters of our method and to show the merit of the proposed strategies. To avoid over-fitting and allow us to extract conclusions about robustness and effectiveness of the proposed strategies, we conduct the preliminary experimentation with the Set 2 of 54 instances.

6.1. Preliminary experimentation

In order to have a fair comparison among all variants, the maximum running time is configured to be the same for all of them. In particular, it is set to $t_{max} = \rho \cdot V \cdot K$, where V and K are the number of vertices and pairs of terminals, respectively. We consider an extra parameter ρ, that allows us to have computing times similar to previous approaches. Specifically, if we set ρ to 0.0024, our procedure presents, on average, shorter computing time than previous procedures.

Parameter	Type	Domain	Default value
α	`Real`	[0.1,0.9]	0.5
λ	`Integer`	[10,120]	100
$size$	`Integer`	[10,150]	100
φ_1	`Real`	[0.1,0.9]	0.2
φ_2	`Real`	[0.1,0.9]	0.4
φ_3	`Real`	[0.1,0.9]	0.5

Table 2: Parameter adjustment.

With the objective of finding the most appropriate parameter values for the proposed method, we have used iRace [39]. This software implements an iterated racing procedure able to find the configuration that obtains the best results. In other words, its goal is to automatically configure a given optimization algorithm

by finding the most appropriate settings for a particular algorithm executed over a set of instances of an optimization problem.

Given a set of instances, for each one, iRace computes a set of elite algorithm configurations and builds a sampling model around these. These distributions are centered around the best value of the parameter. After running iRace, the empirical sampling distribution can be shown (as given in Figure 3), which gives an indication of the range of good values for a parameter. See [39] for further details.

Our two-stage procedure has 6 parameters described in Section 5. In Table 2, we list these parameters, their type, and ranges in which they are modified. In order to reduce the computing time associated to this preliminary experimentation, we start the process by considering a standard initialization (fourth column of Table 2).

In Figure 3, we show the empirical probability distribution of the best parameter value for α, λ, $size$, φ_1, φ_2 and φ_3. These probability distributions have been obtained by iRace (version 2.2). As we can observe, these distributions present a clear maximum for all parameters with the only exception of the parameter *alpha*. In this case, there are two different modes. This behavior can be partially explained by the fact that the two-stage approach prefers to execute mainly the ILP (large values of α) in small and medium instances since it obtains better results than EA. In large instances, EA obtains better results, so the two-stage procedure tries to increase the presence of EA (small values of α).

The best configuration of our procedure was obtained with $\alpha = 0.35$, $\lambda = 110$, $size = 58$, $\varphi_1 = 0.33$, $\varphi_2 = 0.42$, and $\varphi_3 = 0.25$. Therefore, we consider for the remaining experiments that the parameters of our procedure take the aforementioned values.

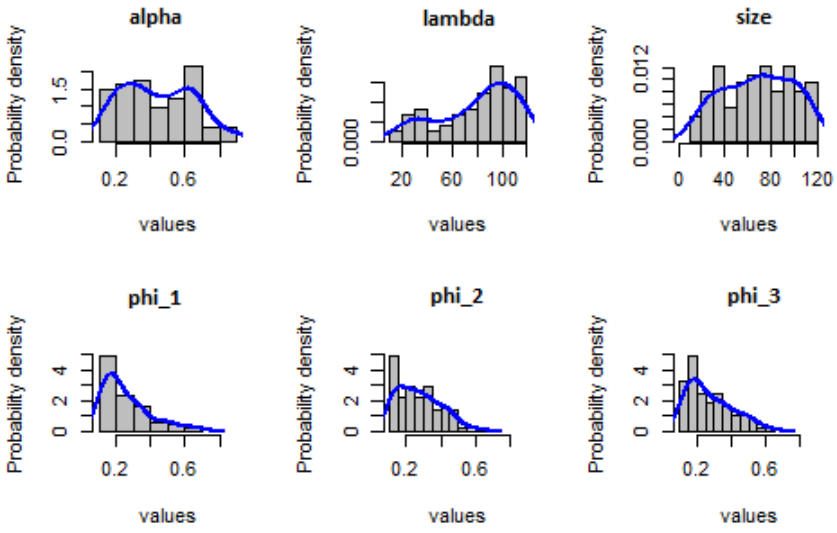

Figure 3: Empirical probability distribution of the parameter value for each parameter of the two-stage method.

We now test whether the combination of the exact algorithm with the metaheuristic produces better results than the execution of both algorithms separately. Specifically, we compare the objective function value of the solutions obtained by the metaheuristic (EA) and by the two-stage method executed separately over the Set 2, by using the same computing time in both cases. Figure 4 (left) shows a scatter plot that compares the performance of the two-stage approach versus EA. Each point in this plot has as x-coordinate the value obtained by EA and as y-coordinate the value obtained by the hybrid approach. If a point is above the bisection line (represented with a dotted line), it implies that the two-stage approach obtained a better objective function value than the EA approach. Notice that we have used a logarithmic scale to

better observe differences close to the origin. Analogously, Figure 4 (right) shows the corresponding scatter plot for the two-stage procedure versus ILP.

Results reported in these plots suggest the superiority of the two-stage approach. Specifically, ILP+EA achieves, in all instances, better (above the bisection line) or equal (in the bisection line) results than EA. These conclusions can be also extended when comparing with the ILP approach, where the hybrid method obtains a worse result only in a single instance.

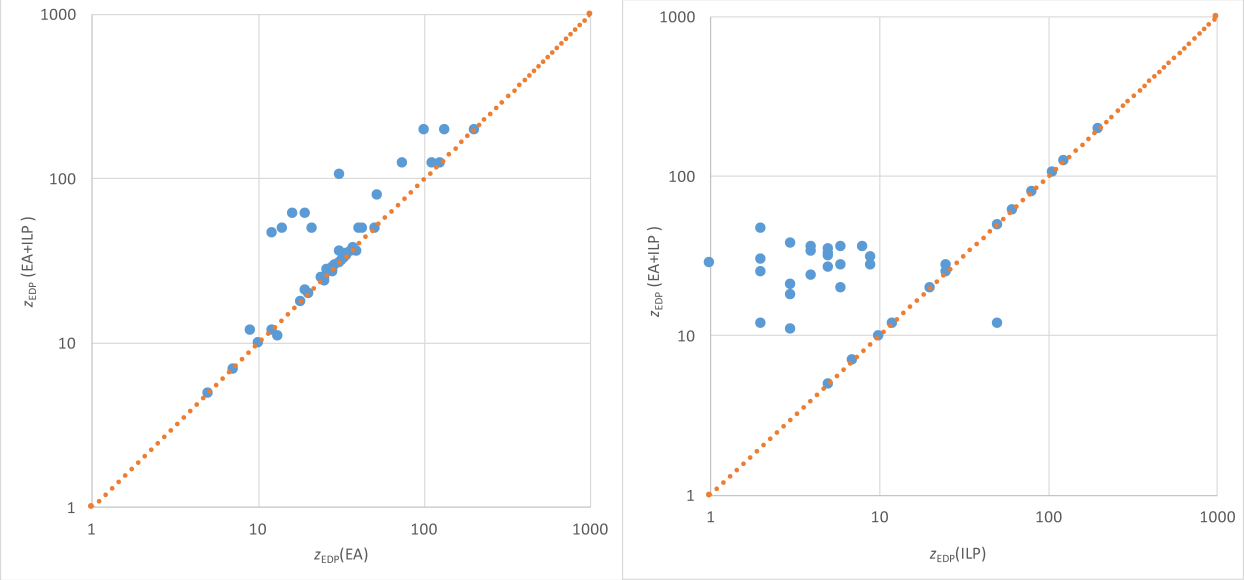

Figure 4: It is worthy to combine the ILP and EA approaches in a two-stage ILP+EA approach.

6.2. Final experimentation

Once we have adjusted the key search parameters and showed the suitability of the hybridization of the exact and heuristic procedures, we compare our proposal, ILP+EA, with the best previous methods identified in the state-of-the-art. Algorithmic comparison is always a very difficult task since it depends on qualitative aspects such as programming language, skills of the programmer, hardware platform, etc. In order to overcome this situation, we directly take the results from the original source. As far as we know, there is not an unified benchmark to compare algorithms for the EDP problem. Therefore, we separate our final experimentation into two different comparisons. In the first one, we compare the two-stage method proposed in this paper (parametrized with the configuration described above) with the Extended Artificial Ant Colony (ACO), the Multi-start Simple Greedy (MSGA) (both described in [9]), and the Genetic Algorithm (GA) proposed in [30]. For the sake of completeness, we also include the ILP and the EA executed in isolation for a similar time than the ILP+EA. Table 3 summarizes average results for Set 1 (120 instances)[1]. Each main row reports for each algorithm: z_{EDP}, average of the number of edge-disjoint paths over the set of considered instances; Dev. (%), average percent deviation with respect to the best solution found in the comparison by any of the methods; and Time (s), average computing time in seconds required by the procedure. In the heading of each column we indicate the name of the graph family (Graph), the number of terminal pairs per instance (#T) and the amount of instances in each family (#inst.).

The average number of disjoint paths and the average computing time are directly copied from [9] and [30]. The computer used to execute MSGA and Ext. ACO is an Intel(R) Pentium(R) 4 processor at 3.06 GHz and 900 Mb of memory running a Linux operating system with 1.98 processor load, implemented in

[1]Individual results per instance are available at www.optsicom.es/edp/edp.zip

C++, and compiled using GCC 2.95.2 with the -O3 option. The GA was implemented in MATLAB and the experiments were executed on a Intel(R) i7 CPU at 1.6 GHz and 4 GB of memory running a Windows 7 operating system. Our experiments were conducted on a Pentium(R) Dual-Core CPU at 3.20GHz, and 4GB of RAM memory. Our experimental platform can be considered equivalent to the one used by [30]. On the other hand, the experimental platform used by [9] is considerably older. However, all tested algorithms are sequentially executed, being frequency of the processor (similar in all cases) one of the most relevant parameters (directly related to the amount of instructions executed per second). RAM memory could also influence in the performance of procedure. In our case, the Java Virtual Machine has limited the amount of memory to 1Gb, which is similar to 900Mb used in [9]. For these reasons, the computing time must be considered as an indicative value in this comparison.

Results shown in Table 3 indicate that the three proposed procedures present competitive results when comparing with the state-of-the-art methods. As expected, ILP presents the best results in small and medium size instances. However, for larger problems, its performance notably decreases. On the other hand, EA exhibits a really robust behavior across all instances. Finally, ILP+EA combines the best of the two algorithmic strategies emerging as the best performing method among all compared algorithms.

Regarding the comparison of our best method and previous approaches, we observe that ILP+EA obtains on average the best results in terms of objective function (41.66) and deviation (6.33%). However, it is important to remark MSGA and GA achieve the best results in subset `AS.BA.R-Wax.v100e217` with $K = 10$ an $K = 25$, respectively. Nevertheless, notice that our two-stage procedure achieves these results in a fifth of the computing time used by the other methods. Nonetheless, as we mentioned above, it should be considered as a qualitative comparison since algorithms were executed in different computers, programming languages, etc.

We apply the non-parametric Friedman test for multiple correlated samples to the best solutions obtained by the six compared methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 1 is assigned to the best method and rank 6 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p-value or significance will be small. The resulting p-value lower than 0.0001 obtained in this experiment clearly indicates that there are statistically significant differences among the six methods tested (considering a level of significance of 0.05). Specifically, the rank values produced by this test are 1.75 (ILP+EA), 2.58 (ILP), 3.69 (ACO), 3.75 (EA), 4.00 (GA),and 4.23 (MSGA).

In the second experiment, we compare the three methods proposed in this paper with the Extended Artificial Ant Colony (ACO) [9]; the local search method combined with the simple greedy algorithm (LS-SGA) and the recursive local search method (LS-R), both introduced [16]; and two versions of the message passing algorithm (MP and MP with reinforcement) described in [2]. Table 4 summarizes average results for Set 2 (54 instances)[2]. In this set, the performance of ILP seems to be worst. In fact, in subset `steinc16` it does not find even a feasible solution. On the other hand, EA presents again a robust behavior across all instances. Regarding the comparison with previous approaches, we observe that MP with reinforcement obtains very competitive results. In particular, it obtains the best results in 6 (out of 13) subsets of instances, while our method obtains the best results in 9 subsets. Considering that the computing time is not included in [2], we run our method without a time limit, denoted as ILP+EA*, to determine whether our method is able to outperform MP with reinforcement. In particular ILP+EA* reaches the objective function values of its competitor when it is executed for 217.15 s (`mesh25x25`) and 186.76 s (`planar200`). It is important to remark that our method does not match the best result in instance `steinb10.40` in 1000 seconds. In this particular instance, MP with reinforcement finds 29 disjoint paths while ILP+EA* obtains 28.

[2]Individual results per instance are available at www.optsicom.es/edp/edp.zip

	Graph	graph3	graph3	graph3	graph4	[1]v100e217	[1]v100e217	[1]v100e217	[2]10-50.sdg	[2]10-50.sdg	[2]10-50.sdg	mesh25x25	mesh25x25	mesh25x25	Avg.
	#T	16	41	65	43	10	25	40	50	125	200	62	156	250	
	#inst.	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	(20 Inst.)	
MSGA	z_{EDP}	15.70	32.00	37.60	42.05	8.05	13.60	17.00	22.55	38.85	51.20	45.50	59.95	67.45	34.73
	Dev.(%)	0.63%	15.01%	13.56%	2.10%	0.00%	4.23%	21.66%	54.21%	44.18%	14.67%	19.40%	21.47%	27.20%	18.33%
	Time(s)	0.96	23.24	49.27	95.74	0.43	4.33	9.83	208.49	534.62	1306.13	398.37	1639.32	2807.36	544.47
ACO	z_{EDP}	15.70	31.80	40.30	41.45	7.88	13.83	17.80	24.15	42.25	55.70	43.96	69.25	87.55	37.82
	Dev.(%)	0.63%	15.54%	7.36%	3.49%	2.11%	2.61%	17.97%	50.96%	39.30%	7.17%	22.13%	9.29%	5.50%	14.16%
	Time(s)	0.46	27.95	57.90	168.87	0.16	1.82	2.21	48.06	190.06	798.97	679.55	1845.70	3165.17	537.45
GA	z_{EDP}	15.00	32.20	36.40	42.00	7.00	12.60	21.70	25.00	41.50	60.00	*	*	*	29.34
	Dev.(%)	5.06%	14.48%	16.32%	2.21%	13.04%	11.27%	0.00%	49.24%	40.37%	0.00%	*	*	*	15.20%
	Time(s)	32.00	93.00	153.00	291.00	15.00	54.00	93.00	768.00	1661.00	3592.00	*	*	*	675.20
ILP	z_{EDP}	14.45	37.65	38.15	42.95	8.05	14.20	18.15	49.25	69.60	13.75	56.00	6.80	10.05	29.16
	Dev.(%)	8.54%	0.00%	12.30%	0.00%	0.00%	0.00%	16.36%	0.00%	0.00%	77.08%	0.80%	91.09%	89.15%	22.72%
	Time(s)	2.75	10.41	18.74	3.99	2.70	1.67	2.51	24.37	138.82	247.03	49.31	241.79	393.00	87.47
EA	z_{EDP}	15.70	32.40	41.60	38.05	7.60	13.90	17.64	21.40	40.45	53.25	45.75	76.34	92.65	38.21
	Dev.(%)	0.63%	13.94%	4.37%	11.41%	5.59%	2.11%	18.71%	56.55%	41.88%	11.25%	18.95%	0.00%	0.00%	14.26%
	Time(s)	6.30	16.12	25.60	44.78	2.40	6.00	9.61	60.01	150.04	240.07	93.01	234.13	375.10	97.17
ILP+EA	z_{EDP}	15.80	35.55	43.50	42.95	8.05	14.15	18.00	47.70	43.35	53.50	56.45	73.40	89.15	41.66
	Dev.(%)	0.00%	5.58%	0.00%	0.00%	0.00%	0.35%	17.05%	3.15%	37.72%	10.83%	0.00%	3.85%	3.78%	6.33%
	Time(s)	3.72	11.25	24.20	4.66	0.50	2.06	3.03	26.53	152.87	245.45	57.28	239.01	384.85	88.88

Table 3: Summary of results. Best average values are highlighted with bold fonts.

[1]AS-BA-R-wax

[2]bl-wr2wht2

* We have not included results of GA over `mesh25x25` subset since the authors did not provide them in [30].

	Graph	bl.rand.10	bl.sdeg.10	mesh.15.10	mesh.25.10	steinb10	steinb16	steinc6	steinc11	steinc16	planar.50	planar.100	planar.200	planar.500	Avg.
	#T	50,125,200	50,125,200	22,56,90	62,156,250	7,18,30	10,25,40	50,125,200	50,125,200	50,125,200	5,12,20	10,25,40	20,50,80	50,125,200	
	#inst.	(6 Inst.)	(6 Inst.)	(6 Inst.)	(6 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	(3 Inst.)	
ACO	z_{EDP}	30.67	29.50	29.00	48.00	17.00	23.00	87.00	124.33	125.00	12.33	23.67	38.00	66.00	50.27
	Dev.(%)	4.17%	40.60%	18.54%	30.27%	5.56%	4.17%	30.40%	0.53%	13.04%	0.00%	2.74%	24.00%	47.20%	17.02%
	Time(s)	149.04	204.25	673.44	966.49	112.85	341.16	741.87	346.68	23.11	12.19	233.69	564.39	1096.63	420.45
LS-SGA	z_{EDP}	30.67	31.17	30.33	54.17	17.33	23.67	89.33	125.00	125.00	12.33	24.00	41.00	75.33	52.26
	Dev.(%)	4.17%	37.25%	14.79%	21.31%	3.70%	1.39%	28.53%	0.00%	13.04%	0.00%	1.37%	18.00%	39.73%	14.10%
	Time(s)	449.89	646.14	564.79	971.15	232.54	280.78	630.24	188.74	205.39	8.98	273.92	675.61	1065.62	476.44
LS-R	z_{EDP}	30.67	29.67	31.33	57.33	17.67	24.00	94.33	125.00	125.00	12.33	24.00	42.00	77.33	53.13
	Dev.(%)	4.17%	40.27%	11.99%	16.71%	1.85%	0.00%	24.53%	0.00%	13.04%	0.00%	1.37%	16.00%	38.13%	12.93%
	Time(s)	239.39	493.36	713.63	1101.05	170.53	218.86	994.67	257.94	114.42	10.98	235.09	586.72	936.23	467.14
MP	z_{EDP}	30.67	29.67	35.60	*	17.67	24.00	107.33	125.00	50.00	12.33	24.00	20.00	87.00	46.94
	Dev.(%)	4.17%	40.27%	0.00%	*	1.85%	0.00%	14.13%	0.00%	65.22%	0.00%	1.37%	60.00%	30.40%	18.12%
	Time(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
MP	z_{EDP}	30.67	29.67	33.00	68.83	18.00	23.67	105.33	125.00	125.00	12.33	24.33	44.00	79.67	55.35
(reim=0.002)	Dev.(%)	4.17%	40.27%	7.30%	0.00%	0.00%	1.39%	15.73%	0.00%	13.04%	0.00%	0.00%	12.00%	36.27%	10.01%
	Time(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
MSG	z_{EDP}	29.17	26.83	27.67	50.33	17.33	23.00	90.83	125.00	125.00	12.33	24.00	40.83	75.00	51.33
	Dev.(%)	8.85%	45.97%	22.28%	26.88%	3.70%	4.17%	27.33%	0.00%	13.04%	0.00%	1.37%	18.35%	40.00%	16.30%
	Time(s)	*	*	*	*	*	*	*	*	*	*	*	*	*	*
ILP	z_{EDP}	24.17	29.50	3.83	22.33	5.33	6.67	125.00	125.00	0.00	10.67	13.33	50.00	125.00	41.60
	Dev.(%)	24.48%	40.60%	89.23%	67.55%	70.37%	72.22%	0.00%	0.00%	100.00%	13.51%	45.21%	0.00%	0.00%	40.24%
	Time(s)	105.51	135.39	30.77	197.23	2.78	5.59	13.82	34.99	786.44	0.95	4.22	6.85	18.71	103.33
EA	z_{EDP}	25.83	25.17	19.00	28.00	17.33	23.67	97.67	125.00	125.00	12.33	22.33	37.33	72.00	48.51
	Dev.(%)	19.27%	49.33%	46.63%	59.32%	3.70%	1.39%	21.87%	0.00%	13.04%	0.00%	8.22%	25.33%	42.40%	22.35%
	Time(s)	150.13	150.14	30.29	210.64	3.31	6.01	150.02	150.04	930.82	1.49	6.01	24.02	146.70	150.74
ILP +	z_{EDP}	32.00	49.67	19.17	42.83	17.67	23.67	125.00	125.00	125.00	12.33	22.67	50.00	125.00	60.67
EA	Dev.(%)	0.00%	0.00%	46.16%	37.77%	1.85%	1.39%	0.00%	0.00%	0.00%	0.00%	6.85%	0.00%	0.00%	7.23%
	Time(s)	147.15	124.69	30.67	222.86	3.34	31.09	14.00	33.65	958.39	1.86	4.53	6.95	18.60	122.91

Table 4: Summary of results. Best average values are highlighted with bold fonts.

* These authors do not report results for these subset of instances

In order to validate these results we conduct the non-parametric Friedman test. The resulting p-value lower than 0.0001 obtained in this experiment clearly indicates that there are statistically significant differences among the nine methods tested (considering a level of significance of 0.05). Specifically, the rank values produced by this test are 1.08 (ILP+EA*), 2.62 (MP reinforcement), 2.92 (ILP+EA), 3.31 (LS-R), LS-SGA (4.00), MP (4.00), MGS (5.54), EA (6.23), and ILP (6.92).

7. Conclusions

In this paper, we propose a two-stage procedure based on the hybridization between an Integer Linear Programming (ILP) technique and an Evolutionary Algorithm (EA) to deal with the edge-disjoint paths (EDP) problem. We provide an experimental comparison between our procedure and the best previous methods for this problem in the state of the art. We performed an extensive computational testing over 174 instances that have previously been used. Experimental results show that the proposed algorithm outperforms the best identified methods. We also performed statistical tests to confirm the significance of the obtained results, thus, identifying the two-stage approach here proposed as the best algorithm in terms of quality and computing time.

Of particular interest in our work has been testing the combination of exact and approximate procedures in a two-stage method. Through extensive experimentation, we have been able to determine the benefits of this combination. We believe that our findings can be translated to other combinatorial problems and it will help in the development of more elaborated methods.

Acknowledgements

We are grateful to referees and associate editors for their constructive and stimulating suggestions. This research has been partially supported by the Spanish Ministry of "Economía y Competitividad", with grants ref. TIN2015-65460-C2-2-P, MTM2015-63710-P, TIN 2014-57458-R and "Comunidad de Madrid" with grant ref. S2013/ICE-2894.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
[2] F. Altarelli, A. Braunstein, L. DallAsta, C. de Bacco, and S. Franz. The edge-disjoint path problem on random graphs by message-passing. *PLoS One*, 10(12), 2015.
[3] B. Awerbuch, R. Gawlick, F.T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS94)*, 1994.
[4] J. Bang-Jensen and G. Guting. *Digraphs: Theory, Algorithms and Applications*. Springer Monographs in Mathematics. Springer, 2001.
[5] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
[6] A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, 2000.
[7] M. Bazargan. *Airline Operations and Scheduling*. Ashgate, 2007.
[8] M.J. Blesa and C. Blum. Ant colony optimization for the maximum edge-disjoint paths problem. In *Raidl G.R. et al. (eds) Applications of Evolutionary Computing. EvoWorkshops 2004. Springer, Berlin, Heidelberg*, volume Lecture Notes in Computer Science 3005, pages 160–169, 2004.
[9] M.J. Blesa and C. Blum. Finding edge-disjoint paths in networks by means of artificial ant colonies. *Journal of Mathematical Modeling and Algorithms*, 6(3):361–391, 2007.
[10] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
[11] W.T. Chan, F.Y.L. Chin, and H.F. Ting. Escaping a grid by edge-disjoint paths. *Algorithmica*, 36(4):343–359, 2003.
[12] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 628–637, 2003.
[13] C. de Bacco, S. Franz, D. Saad, and C.H. Yeung. Shortest node-disjoint paths on random graphs. *Journal of Statistical Mechanics: Theory and Experiment*, 2014(7), 2014.
[14] O. Dolezal, T. Hofmeister, and H. Lefmann. A comparison of approximation algorithms for the maxcut-problem. Technical report, Reihe CI 57/99, SFB 531, Universität Dortmund, 1999.

[15] I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In *EvoWorkshops'03 Proceedings of the 2003 international conference on Applications of evolutionary computing. Springer-Verlag Berlin, Heidelberg*, pages 211–223, 2003.
[16] P.Q. Dung, Y. Deville, and P. van Hentenryck. A constraint-based local search for constraint optimization on trees and paths. *Constraints*, 17(4):357–408, 2012.
[17] R.C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, IEEE Press*, pages 39–43, 1995.
[18] T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. In *26th International Symposium of Mathematical Foundations of Computer Science (MFCS 2001). Springer-Verlag*, volume 2136 of LNCS, pages 351–362, 2001.
[19] T. Erlebach. Approximation: Theory and algorithms. edge-disjoint paths and unsplittable flow. Technical report, ETH Zürich. Winter 2003/2004. Available: *https://www.ti.inf.ethz.ch/ew/courses/ApproxAlgs0304/problem_sets/edpsurvey.pdf*, 2003.
[20] T. Erlebach. Approximation algorithms for edge-disjoint paths and unsplittable flow. In *E. Bampis et al. (Eds.): Efficient Approximation and Online Algorithms. Springer-Verlag*, volume 3484 of LNCS, pages 97–134, 2006.
[21] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14 (3):326–355, 2001.
[22] T. Erlebach and D. Vukadinovic. New results for path problems in generalized stars, complete graphs, and brick wall graphs. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001)*, volume 2138 of LNCS, pages 483–494, 2001.
[23] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science. North-Holland Publishing Company*, 10(2):111–121, 1980.
[24] A. Frank. Packing paths, circuits, and cuts. a survey. In *B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, Paths, Flows, and VLSI-Layout. Springer-Verlag*, pages 47–100, 1990.
[25] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.
[26] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
[27] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.
[28] U.I. Gupta, D. T. Lee, and J. Y. T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.
[29] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Nearoptimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC99)*, pages 19–28, 1999.
[30] C.C. Hsu and H.J. Cho. A genetic algorithm for the maximum edge-disjoint paths problem. *Neurocomputing*, 148(1):17–22, 2015.
[31] C.C. Hsu, H.J. Cho, and S.C. Fang. Routing and wavelength assignment in optical networks from maximum edge-disjoint paths. In *Pan JS., Krmer P., Snel V. (eds) Genetic and Evolutionary Computing. Advances in Intelligent Systems and Computing, Springer, Cham*, volume 238, pages 95–103, 2014.
[32] R.M. Karp. Reducibility among combinatorial problems. In *Miller R. E. and Thatcher J. W., eds. Complexity of Computer Computations, Plenum, New York*, pages 85–103, 1972.
[33] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, Piscataway, NJ, IEEE Press*, pages 4104–4109, 1997.
[34] J. Kleinberg and E. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *In Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC95)*, pages 26–35, 1995.
[35] J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS95)*, pages 52–61, 1995.
[36] S.G. Kolliopoulos. Edge-disjoint paths and unsplittable flow. *Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC*, 2007.
[37] S.G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO VI)*, volume 1412 of LNCS, pages 153–168, 1998.
[38] M. Kramer and J. van Leeuwen. The complexity of wire routing and finding the minimum area layouts for arbitrary vlsi circuits. In *F. P. Preparata, editor, Advances in Computing Research; VLSI Theory. JAI Press Inc., Greenwich, CT-London*, volume 2, pages 129–146, 1984.
[39] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez-Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:46–58, 2016.
[40] L. Lovasz and D. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. American Mathematical Society, 2009.
[41] E.Q.V. Martins and M.M.B. Pascoal. A new implementation of yens ranking loopless paths algorithm. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1 (2):121–133, 2002.
[42] D. Marx. Eulerian disjoint paths problem in grid graphs is np-complete. *Discrete Applied Mathematics*, 143(1-3):336–341, 2004.
[43] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: Boston University representative internet topology generator. Technical report, Computer Science Department. Boston University, 2001.

[44] M. Mezard and A. Montanari. *Information, physics, and computation*. Oxford University Press, 2009.
[45] M. Mezard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002.
[46] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint path problem. *Combinatorica*, 13(1):97–107, 1993.
[47] T. Nishizeki, J. Vygen, and X. Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Applied Mathematics*, 115(1-3):177–186, 2001.
[48] P. Moscato P and C. Cotta. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristic. F. Glover and G. A. Kochenberger editors, Kluwer, Norwell, Massachusetts, USA*, 2003.
[49] K. Price, R.M. Storn, and J.A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer, 2005.
[50] N. Robertson and P. Seymour. Graph minors. xiii. the disjoint path problem. *Journal of Combinatorial Theory Series B*, 63:65–110, 1995.
[51] R. Storn. On the usage of differential evolution for function optimization. In *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523, 1996.
[52] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.
[53] J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90, 1995.
[54] J.Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27:526–530, 1970.