An efficient evolutionary algorithm for the orienteering problem

Gorka Kobeaga[a,d,*], María Merino[b,d], Jose A. Lozano[a,c,e]

[a]*BCAM - Basque Center for Applied Mathematics, Spain.*
[b]*Dep. of Applied Mathematics and Statistics and Operations Research, University of the Basque Country UPV/EHU, Spain.*
[c]*Dep. of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, Spain.*
[d]*Stochastic Optimization Group, University of the Basque Country UPV/EHU, Spain.*
[e]*Intelligent Systems Group, University of the Basque Country UPV/EHU, Spain.*

Abstract

This paper deals with the Orienteering Problem, which is a routing problem. In the Orienteering Problem each node has a profit assigned and the goal is to find the route that maximizes the total collected profit subject to a limitation on the total route distance. To solve this problem, we propose an evolutionary algorithm, whose key characteristic is to maintain unfeasible solutions during the search. Furthermore, it includes a novel solution codification for the Orienteering Problem, a novel heuristic for node inclusion in the route, an adaptation of the Edge Recombination crossover developed for the Travelling Salesperson Problem, specific operators to recover the feasibility of solutions when required, and the use of the Lin-Kernighan heuristic to improve the route lengths. We compare our algorithm with three state-of-the-art algorithms for the problem on 344 benchmark instances, with up to 7397 nodes. The results show a competitive behavior of our approach in instances of low-medium dimensionality, and outstanding results in the large dimensionality instances reaching new best known solutions with lower computational time than the state-of-the-art algorithms.

Keywords: Orienteering Problem, Travelling Salesperson Problem, Evolutionary Algorithm, Combinatorial Optimization

1. Introduction

The Orienteering Problem (OP), also called the Selective Travelling Salesperson Problem or the Maximum Collection Problem, is a well known NP-hard combinatorial optimisation problem [13]. The origin of the problem is a sports game, where the participants are given a topographical map with detailed control points with rewards. The participants try to visit the points in order to maximise the total prize obtained; however, there is a time limitation. Therefore, the OP is basically

*Corresponding author: Gorka Kobeaga.
Tel.: +34946567842.
Address: Mazarredo, 14. 48009 Bilbao, Basque Country, Spain.

Email addresses: `gkobeaga@bcamath.org` (Gorka Kobeaga), `maria.merino@ehu.es` (María Merino), `ja.lozano@ehu.eus` (Jose A. Lozano)

a routing problem with profits and it can be seen as a combination between the Knapsack Problem and the Travelling Salesperson Problem (TSP) [26]. Many practical problems have been modeled as an OP, some examples are the following: TSP without enough time to visit all the cities [25], the home fuel delivery problem [13], and the most recent tourist trip design problem [20, 21, 28, 30], or the mobile-crowdsourcing problem [33]. In addition, many variants of the OP have been analysed: the classical ones being the team OP, the OP with time windows or the time dependent OP; more recently new variants have been introduced, such as the stochastic OP, the generalized OP, the arc OP, the multi-agent OP or the clustered OP, among others. See [2, 22, 24, 27] and [14] for a recent survey.

Since the first publications appeared, many algorithms have been developed to solve the basic OP. A classification of the exact algorithms is presented in [8], where the branch-and-cut [10] should be highlighted. Plenty of heuristic approaches have also been proposed, such as tabu search [12], artificial neural networks [29], ant colony optimisation [16], variable neighbourhood search [3], two-parameter iterative algorithm [20] or a greedy randomised adaptive search procedure with and without path relinking [6].

There are several Evolutionary Algorithms (EA) proposed in the literature to solve OP and TOP [5, 9, 17, 18, 23], among others. In all of these approaches the solutions are initialized with constructive methods which add a new node to the route while the distance limitation constraint is satisfied and codified based on the visiting sequence of nodes. The tour lengths are improved using the 2-opt heuristic and general purpose genetic operators are adapted for the evolutionary part. Particularly, all of them use an adaptation of the single-point crossover or its generalization, the n-point crossover. Approaches [5, 9, 17] and [23], have been tested in the benchmark instances proposed by [7] (40 instances involving up to 66 nodes) and [23] (49 instances involving up to 33 nodes). Approach [18] has been tested in 90 TSPLib-based instances and 15 VRP-based instances proposed by [10].

The OP can be defined by a 5-tuple $\langle G, d, s, v_1, d_0 \rangle$, where $G = \langle V, E \rangle$ is a graph with node set $V = \{v_1, \ldots, v_n\}$ (vertices, cities, customers, locations) and edge set E; $d = (d_{i,j})_{i,j=1}^{n}$ is the non-negative distance (time or weight) matrix between vertices, i.e., $d_{i,j}$ is the distance from node v_i to node v_j; and $s = (s_1, \ldots, s_n)$ is a non-negative vector that represents scores, i.e., s_i is the score of node v_i. The OP goal is to determine a route starting and ending at v_1 (depot) under the distance limitation, d_0, that maximises the sum of the scores of the visited nodes. The mathematical formulation of the OP can be found in [14].

The remainder of the paper is organised as follows. In Section 2, the new evolutionary algorithm is described. The experimental results with a comparison with the most competitive heuristics and the exact algorithms in the literature are presented in Section 3. Conclusions and the future work are presented in Section 4. Appendix A shows the detailed results for the instances and algorithms tested.

2. Evolutionary Algorithm for the Orienteering Problem

In this paper, we propose a population-based evolutionary optimisation technique whose main characteristic is to maintain unfeasible solutions during the search process. Essentially the algorithm follows the steady-state genetic algorithm schema [31] with the difference that, at some generations, we perform a tour-improving procedure followed by node dropping and adding strategies, for feasibility conversion and path tightening respectively. The pseudocode is described in Algorithm 1.

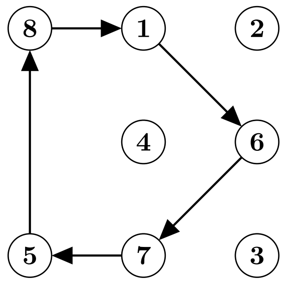
Our approach, in addition to the common parameters of any genetic algorithm (population size, mutation probability), uses a specific parameter, $d2d$, that controls the frequency of the feasibility and improving phase.

Algorithm 1 Evolutionary Algorithm

```
Inputs: $G$, $d$, $s$, $v_1$, $d_0$.

Build initial population (2.2);
Tour improvement (2.4);
Drop operator (2.5);
Add operator (2.6);

i=0;
while not ( stopping criteria are satisfied and mod(i,d2d) = 0 ) do
  i=i+1;
  if $mod(i, d2d) \neq 0$ then
    Select two parents (2.3.1);
    Crossover (2.3.2);
    Mutation (2.3.3);
    if child better than worst in the population then
      Insert the child in the place of the worst individual;
    end if
  else
    Tour improvement (2.4);
    Drop operator (2.5);
    Add operator (2.6);
  end if
end while

Output: A route starting and finishing in $v_1$.
```

2.1. Individual codification

A solution to the problem can be seen as a sequence defined by a subset of nodes (route). In order to codify that solution, a permutation of the whole set of nodes has been considered, $\pi = (\pi_1, \ldots, \pi_n)$. In this permutation, π_i represents the next node visited after vertex i in the route. The nodes in the route form a cycle in the permutation and a node which is not in the route is codified as a fixed point, i.e, $\pi(i) = i$. Figure 1 shows a solution of an OP whose associated codification is the following permutation, $\pi = (6, 2, 3, 4, 8, 7, 5, 1)$. Note that the nodes in the solution route $\{1, 6, 7, 5, 8, 1\}$ form a cycle in the permutation π, while those nodes off the route $\{2, 3, 4\}$ are fixed points in the permutation.



$$\pi = (6, 2, 3, 4, 8, 7, 5, 1)$$

Figure 1: Example of a solution in an eight-node graph and its corresponding codification as a permutation.

Note that not every possible permutation is a valid solution of the problem: first, the route length limitation constraint may not be satisfied; secondly, sub-routes may also appear. However, this route codification has been chosen for implementation reasons. On one hand, a fixed length codification was desirable; on the other hand, some operations over the solutions, such as checking if a node is contained in the route, can be efficiently implemented using this codification. A similar codification was previously proposed for the Prize Collecting TSP [4].

Notice that v_1 is always included on the route. When v_1 is the only node on the route, this is a special case whose solution is codified by the identity permutation.

2.2. Initial population

Algorithm 2 shows a pseudocode for generating *npop* individuals for the initial population. An individual is generated in two steps. In the first step, a subset of nodes is randomly chosen, where each node is sampled with probability p. In the second step, a route passing through the subset of nodes is randomly created and codified as described in Section 2.1.

Algorithm 2 Initial population

```
for $i = 1$ to $npop$ do
  $v_1$ node is included in the subset of nodes;
  for $j = 2$ to $n$ do
    $v_j$ node is included in the subset of nodes with probability $p$;
  end for
  Construct a tour by randomly ordering the selected nodes;
end for
```

The probability p is a parameter of the algorithm, where $n \cdot p$ determines the expected number of visited nodes of each generated individual. In addition, note that the obtained initial individuals could be unfeasible.

2.3. Genetic Operators

In this section we will describe the genetic operators - parent selection, crossover and mutation - that are used to evolve the population. While the chosen parent selection operator is a general purpose selection procedure, the crossover and mutation operators have been specifically developed or adapted for the OP problem.

2.3.1. Parents selection

Our selection operator is a kind of hybridisation between tournament and roulette wheel selection, see Algorithm 3. In the first step, *ncand* candidates are uniformly at random selected from the population. In the second step, the roulette wheel selection is carried out, based on the individual fitness (i.e., its objective function or total score), where a correction is performed (subtraction of the minimum fitness) in order to point out the fitness quality differences between candidates.

Algorithm 3 Parents selection

```
Select uniformly at random $ncand$ candidates from the population, $C = \{I_1, \ldots, I_{ncand}\} \subset \{1, \ldots, npop\}$;
Compute $m := min_{I_i \in C}(f_{I_i})$, where $f_{I_i}$ is the objective function value of $I_i \in C$;
Compute $r_i := f_{I_i} - m + 1$, $i = 1, \ldots, ncand$;
Compute $p_i := \frac{r_i}{\sum r_i}$, $I_i \in C$;
Sample twice with the distribution $(p_1, \ldots, p_{ncand})$ to obtain two parents;
```

2.3.2. Crossover operator

The crossover produces a new child solution from a given pair of parents solutions by using an adaptation of the well-known Edge Recombination operator [32]. In the OP, we are interested in inheriting two main characteristics from the parents related with the nodes and the edges. Regarding the visited nodes, we want to maintain all the nodes that are common to both parent solutions, to include, with a probability, the nodes that belong to only one parent, and to exclude the nodes that do not belong to any parent solution. Regarding the route length, we want to use as many edges of the parents as possible in order to pass on the maximum amount of information and decrease length quality losses in the new child solution.

The original ER crossover [32] was designed for problems where the solution space consists of Hamiltonian cycles; now we have extended it for a larger set of sequencing/ordering problems, where the solution space consists of simple cycles which do not necessarily contain all the nodes. This generalisation does not use the information of the associated cost of the edges and, therefore, it is possible to produce unfeasible solutions for the OP.

The operator uses the so-called edge map, which is a summary of parental information, to guide the procedure. The **edge map** contains, for each common node of the parental graph, its degree, connected nodes and intermediate paths. Representing the route of the first and second parent as the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ respectively, the **parental graph** consists of all vertices and edges arising in the solutions of the two parents, i.e., $PG = (V_1 \cup V_2, E_1 \cup E_2)$. A node u is a **common node** of the parental graph if that vertex belongs simultaneously to both parents, $u \in V_1 \cap V_2$. A node u is a **connected node** of a node v if u and v are common nodes and there exists a path over the parental graph connecting both nodes which does not contain a third common node. The **degree** of a common node is the number of nodes which are connected to it. An **intermediate path** between two common nodes u and v is any path from u to v, which is inside the parental graph with no more common nodes.

The ER crossover operator builds the child route as follows (see Algorithm 4): first, the edge map is constructed, and the starting node v_1 is assigned to be the **current node**. Each time the current node is reassigned, it is removed from the edge map, and the degree of each non-visited

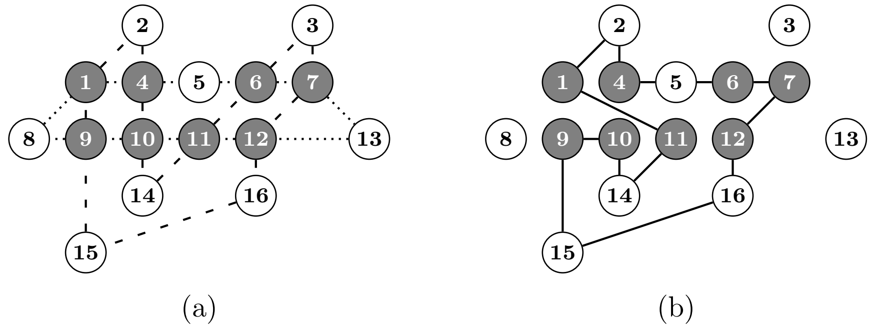


Figure 2: Example of a crossover. (a) Parental graph. The route of the first parent is represented by dotted line, the route of the second parent with dashed line. The common nodes are filled in gray. (b) Child after the crossover.

common node is recomputed. At each step we will decide which the next common node to visit is by selecting from the set of the non-visited connected nodes of the current node the one that has the lowest degree, where ties are broken randomly. If we reach a node whose all connected common nodes are already visited, we will choose the next node randomly from the set of non-visited common nodes. A intermediate path between the current node and next node is randomly chosen and its nodes incorporated to the route. The process finishes when there are no more common nodes left to visit.

Note that the operator does not make sense when there is a unique common node, v_1, or when the solution routes are equal. In any of these cases, the crossover procedure is skipped, and one of the parent solutions is cloned.

In Figure 2, two parents solutions are shown (a) and the child (b) produced after the ER crossover application. Table 1 shows the associated edge-map and Figure 3 shows some illustrative steps of the operator. The algorithm starts at common starting node 1. Both of its connected nodes, 4 and 9, have degree two - we have already removed the node 1 from the edge map-, so the algorithm will make a random choice. Assume that the common node 4 is chosen, and again randomly we choose one of the possible paths to reach the node 4, in this case we assume that the path chosen is $(1, 2, 4)$, see first step in Figure 3. The candidates for the next common node are 6 and 10. Both have degree 2 so randomly choose one, assume that 6 is chosen. There is a unique path to choose that goes from 4 to 6, the one that passes through node 5, see second step. In the last step, all the common nodes have been visited so the algorithm will join node 11 and 1.

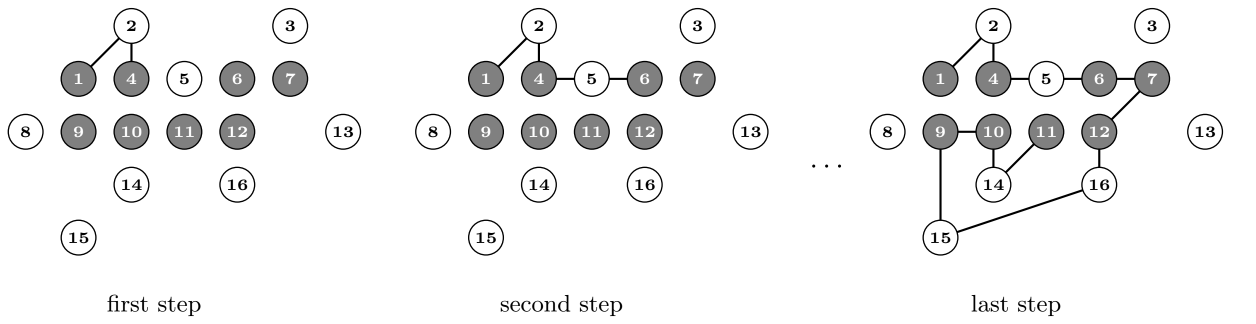


Figure 3: Illustration of the crossover operator. The left and center figures show the results of two consecutive steps of the crossover algorithm, while the right figure represents the last step before closing the route.

Table 1: Example of an edge map

Common Node	Connected Nodes	Degree	Intermediate paths
1	4	2	(1,4), (1,2,4)
	9		(1,9), (1,8,9)
4	1	3	(4,1), (4,2,1)
	6		(4,5,6)
	10		(4,10)
6	4	3	(6,5,4)
	7		(6,7), (6,3,7)
	11		(6,11)
7	6	2	(7,6), (7,3,6)
	12		(7,12), (7,13,12)
9	1	3	(9,1), (9,8,1)
	10		(9,10)
	12		(9,15,16,12)
10	4	3	(10,4)
	9		(10,9)
	11		(10,11), (10,14,11)
11	6	3	(11,6)
	10		(11,10), (11,14,10)
	12		(11,12)
12	7	3	(12,7), (12,13,7)
	9		(12,16,15,9)
	11		(12,11)

Algorithm 4 ER crossover operator

```
Initialize *current node* to $v_1$;
while there are non-visited common nodes do
  Remove all the occurrences of *current node* from the connected nodes of edge map;
  if at least one connected node of the current node is not visited then
    Update *next node* as the connected node with the smallest degree, ties are broken randomly;

    Choose randomly an intermediate path between *current node* and *next node*. Insert the path after the *current node*;
    Rename the *next node* as the *current node*;
  else
    if there are non-visited common nodes then
      Select randomly a non-visited common node and insert it on the route after the *current node*;
      Call it the *current node*;
    end if
  end if
end while
```

2.3.3. Mutation operator

At each generation, after the crossover operator has been applied, a mutation is performed, see Algorithm 5. To perform the mutation, we will choose a node uniformly at random from $\{v_2, \dots, v_n\}$. If the node is on the route, the node is dropped and its adjacent nodes are connected. If the node is not on the route, it is inserted in the best place - using the same heuristic explained later in the add operator 2.6.

Algorithm 5 Mutation operator

```
Select uniformly at random a node from $\{v_2, \dots, v_n\}$;
if the node is on the route then
  Remove the node from the route and connect the adjacent nodes;
else
  Insert the node on the route, using the heuristic explained in Section 2.6;
end if
```

2.4. Tour improvement operator

The feasibility of a solution closely depends on the order of the visiting nodes. A set of nodes could belong to a feasible or an unfeasible solution, depending just on the ordering of them on the route. The aim of this operator is to decrease the length of the routes as much as possible. In this manner, first, unfeasible solutions are attempted to convert to feasible solutions, second, the lengths of the feasible solutions are decreased in order to insert new nodes during the add operator 2.6.

Finding the shortest route for a subset of nodes is equivalent to solving a TSP for that set of nodes. In the extensive literature that has the TSP, there is a vast quantity of heuristic approaches that can be used for the OP. We are particularly interested in those local search techniques that provide a high quality solution in a reasonable time due to the fact that the tour improvement is applied many times during the algorithm. In this paper we have considered three alternatives, the 2-opt, 3-opt and Lin-Kernighan [1, 15].

2.5. Drop operator

Improving the tour length might not be enough to convert an unfeasible solution to a feasible one, it could still continue violating the route length limitation constraint. In this case, in order to obtain a feasible solution, it is necessary to delete nodes from the solution until it fits the distance limitation.

To that end, we sort the nodes contained in the route considering both the cost in terms of length and the fitness gain for visiting each node. Namely, we want to drop the nodes that concurrently have a low contribution to the fitness and are costly to visit. Let us define the value for sorting the visited nodes as $drop(v_i) = \frac{s_i}{d_{i_p,i}+d_{i,i_n}-d_{i_p,i_n}}$ where v_{i_p} and v_{i_n} nodes are the previous and next nodes to v_i, respectively (see the drop operator in Algorithm 6 and the example in Figure 4).

Thereby, at each step of the drop operator the node with the lowest *drop* value is removed from the solution. The algorithm stops once it obtains a feasible solution.

$$\begin{aligned}
drop(v_5) &= 3/(2+1-\sqrt{5}) & &\sim 3.93\\
drop(v_6) &= 3/(\sqrt{2}+\sqrt{2}-2) & &\sim 3.62\\
drop(v_7) &= 4/(\sqrt{2}+1-\sqrt{5}) & &\sim 22.45\\
drop(v_8) &= 2/(1+2-\sqrt{5}) & &\sim 2.618
\end{aligned}$$

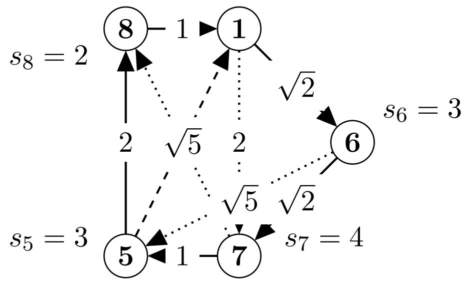


Figure 4: Drop operator example. After evaluating the *drop* value of each node, the node 8 is removed from the tour.

Algorithm 6 Drop operator

```
while not distance limitation constraint is satisfied do
    Order nodes according to drop index and remove the one with the lowest value;
    Update route length and fitness;
end while
```

2.6. Add operator

Once the individual has been made feasible, we apply an improvement mechanism to it. It consists of the addition of new nodes to the current route. This operator is applied for node inclusion while the distance limitation constraint is satisfied, see Algorithm 7.

When dealing with node insertion, we have to set some criteria in order to select the most suitable node to add to the route and, then, to decide where the insertion should be made.

Before defining the insertion criteria, let us define an associated value, $addcost(v_i)$, for each non-visited node, v_i, that approximates the increase of the route length when inserting it to the route. A common heuristic that appears in the literature in order to calculate the *addcost* value is to evaluate the cost of each possible insertion in the route and to take the minimum value [6, 20]. If m represents the number of visited nodes, then the computational cost of selecting the candidate to insert at each step of this approach is $O((n-m)\cdot m) \leq O(n^2)$. Using the information calculated in the first step, i.e., the insertion position and the *addcost* of each non-visited candidate, it is possible to decrease the computational cost of selecting a candidate for the rest of the steps. This way we have $O((n-m)\cdot m)$ for the first step and $O(n-m)$ for the rest of the steps.

Although the previous method is quadratic, a faster node insertion method is still desirable, since a large amount of queries of this type are made during the algorithm. Therefore, we propose a new heuristic method for node insertion, one that speeds up the process at the expense of the quality of the *addcost* approximation.

To evaluate the inserting cost of a non-visited node, we start by finding the three nearest visited nodes. If two of these three nearest nodes are adjacent in the route, the *addcost* value is the cost of inserting the candidate node between these two nodes (see Figure 5). When there are more than two pairwise adjacent nodes in the 3-nearest set, the *addcost* value is determined by the choice that minimises the adding cost. Otherwise, if none of the three nearest nodes are adjacent to each other, calculate the cost of inserting the candidate node between the contiguous nodes of the three nearest nodes. There are six different options, so we choose the one with the minimum value for the *addcost*.

Because of the design of the proposed minimum cost insertion heuristic, when the distance matrix is given by spatial points, the computational cost can be decreased using a data structure to accelerate the proximity queries. In our case, we have used a k-d tree, which is built once in the whole algorithm.

Finally, the *addvalue* is defined to set the inserting preference of a non-visited node using the *addcost* and the score of the node. The inserting preference of a non-visited node depends whether the insertion is feasible or not. When the insertion is feasible, i.e., the current length plus the *addcost* value is not greater than the route length limit, the inserting preference is defined as $addvalue(v_i) = s_i/addcost(v_i)$. When the insertion is not feasible, the inserting preference of the node is set to 0. If the maximum value of *addvalue* is positive, the node which maximizes the *addvalue* is inserted in the route, and the process is repeated. The add operator stops when adding any of the non-visited nodes leads to an unfeasible solution, i.e., when all the *addvalue*s are 0.

With this new heuristic, the computational cost of selecting the candidate for the first iteration of each application of the *addcost* is $O((n-m)\cdot log(m))$ when nodes are spatial points, and $O((n-m)\cdot m)$ otherwise. Moreover, for the rest of the iterations, as we only need to compare if the last inserted node is closer than the previous three nearest nodes, the computational cost of selecting a candidate would be $O(n-m)$ regardless of whether the nodes are spatial points or not.

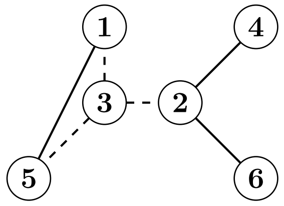


Figure 5: Example of an evaluation of the cost of inserting a node in the route. Node v_3 is the node to evaluate and the rest of nodes are part of the route (solid line). In this case, the best position for the node v_3 in the route is between the adjacent nodes v_1 and v_5.

In Figure 5 we represent the calculation of the heuristic to obtain the *addcost* of the non-visited node v_3. First, we search the three nearest nodes from v_3 on the route, in this case v_1, v_2 and v_5. Given that v_1 and v_5 are adjacent in the route, we assign to v_3 the increase of the distance route if v_3 is added between v_1 and v_5, i.e., $addcost(v_3) = d_{1,3} + d_{3,5} - d_{1,5}$.

Algorithm 7 Add operator

```
while not stop do
  for node $v_i$ not in route do
    Get the three nearest nodes in the route for $v_i$, $V_3^i = \{v_1^i, v_2^i, v_3^i\}$;
    if at least two nodes of $V_3$ are adjacent in the route then
      Find the pair $(v_{prev}, v_{next})$ where $v_{prev}, v_{next} \in V_3$ that are adjacent in the route that minimizes $d_{prev,i} + d_{i,next} - d_{prev,next}$;
    else
      Let us define:
        $V_3^* = \{(v_1^i, v_{1_{next}}^i), (v_2^i, v_{2_{next}}^i), (v_3^i, v_{3_{next}}^i)\} \cup \{(v_{1_{prev}}^i, v_1^i), (v_{2_{prev}}^i, v_2^i), (v_{3_{prev}}^i, v_3^i)\}$
      Find the pair $(v_{prev}, v_{next}) \in V_3^*$ that minimizes $d_{prev,i} + d_{i,next} - d_{prev,next}$;
    end if
    $addcost(v_i) = d_{prev,i} + d_{i,next} - d_{prev,next}$;
    if route length + $addcost(v_i) \leq d_0$ then
      $addvalue(v_i) = s_i / addcost(v_i)$;
    else
      $addvalue(v_i) = 0$;
    end if
  end for

  Select the node, $v_i$, which maximizes addvalue;
  if $addvalue(v_i) > 0$ then
    Include the selected node in the route;
    Update route length and fitness;
  else
    Stop;
  end if
end while
```

2.7. Stopping criteria

There are two main stopping criteria for our evolutionary algorithm. The first one is based on the distribution of the population fitness. Specifically, the algorithm stops when a certain proportion of the solutions has the same fitness as the best solution of the population. The second one is a limitation on the execution time.

These criteria are evaluated after the feasibility of the solutions is checked and the add operator is performed, particularly, when the generation number is a multiple of $d2d$.

3. Computational Experiments

This section presents the results of the computational experiments carried out for testing the evolutionary algorithm explained in the previous section. The proposed approach has been compared with the exact branch-and-cut algorithm (B&C) [10] and three state-of-the-art heuristics: GRASP with Path Relinking (GRASP-PR) [6], tabu search (TS) [12] and the two-parameter interactive algorithm (2-P IA) [20]. Results for TS are not reported because they were not competitive compared with the rest of the approaches, but they are available upon request from the authors. The benchmark instances have been generated from those obtained from TSPLib repository. We have split up the instances into two groups: medium-sized instances, up to 400 nodes and large-sized ones, up to 7397 nodes. As detailed in the literature, three generations are classified according to the definition of scores. A fourth generation has been created with the most difficult instances for the exact methodology.

The solution quality and the computational cost have been analysed. The solutions have been measured in terms of the quality gap (*gap*), defined as the relative difference in percentage between the best known or optimal solution (*opt*) and the solution of the corresponding algorithm (*best*), i.e., $gap = 100 \cdot \frac{opt-best}{opt}$. The computational cost in seconds is measured via time consumption.

The computational experiments are reported as follows: Section 3.1 presents the instances we have experimented with. Section 3.2 details the parameter selection. The validation of the proposed algorithm components is carried out in Section 3.3. Section 3.4 shows the performance of the evolutionary algorithm versus the exact B&C and two state-of-art heuristics: GRASP-PR and 2-P IA. In Section 3.5 we discuss the contributions of the proposed algorithm. The detailed numerical results are shown in Appendix A. Supplementary results and extended material of the experiments can be obtained from `https://github.com/bcamath-ds/paper-sm-ea4op2017`.

3.1. Benchmark instances

We have considered the TSPLib-based test instances proposed in [10]. In that paper, the authors described three methods of generating scores for OP instances from TSPLib. For all of these three generations the distance limitation is set as a half of the TSP solution, $d_0 = \lceil 0.5 \cdot v(TSP) \rceil$.

In addition to that, we have extended the benchmark in two directions. On the one hand, we have defined a new generation method (generation 4) involving instances with $d_0 \neq \lceil 0.5 \cdot v(TSP) \rceil$. With that in mind, we have considered the instances with scores of generation 2 and created all the cases with $d_0 = \lceil \alpha \cdot v(TSP) \rceil$, where $\alpha \in \{0.05,\ 0.10,\ \ldots,\ 0.45,\ 0.55,\ \ldots,\ 0.95\}$. From these 18 cases we have chosen the most difficult instance for the B&C. When all the problems finish before the time limitation for the B&C, we choose the α whose solution takes the longest time. Otherwise, when at least one problem reaches the time limitation, we choose the α whose solution takes the longest separation time at the end of the time limitation for the B&C .

On the other hand, we have extended the set of benchmark instances to larger size problems. So far, instances up to 400 nodes of the TSPLib had been evaluated in the OP literature; we have considered also the ones involving up to 7397 nodes. Altogether, the experiments have been made on a set of 344 instances. The benchmark instances set is summarized in Table 2.

Table 2: Generations for instances based on TSPLib.

Generation	Score for the ith node, $i \in [n]$	α	# medium-sized $n \leq 400$	# large-sized $n > 400$
Gen1	1	0.5	45	41
Gen2	$1 + (7141 \cdot (i-1) + 73) \mod 100$	0.5	45	41
Gen3	$1 + \lfloor 99 \cdot d_{1,i} / \max_{j \in [n]} d_{1,j} \rfloor$	0.5	45	41
Gen4	$1 + (7141 \cdot (i-1) + 73) \mod 100$	α^*	45	41

α^* : α of the hardest instance for the B&C.

All the instances used for the computational experiments, including the best routes obtained by the EA4OP, are available in `https://www.github.com/bcamath-ds/OPLib`. The original TSPLib instance set proposed by [19] can be obtained in `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`. In both instance sets the distances between the nodes are integers.

3.2. Parameter and heuristic selection

In order to perform the parameter and heuristic selection, we have selected five medium-sized instances of generation 2, involving the largest amount of nodes without repeating the "family" (gil262, a280, lin318, pr299 and rd400). We have chosen instances from generation 2 precisely because, contrary to the other generations, here the scores are pseudo-randomly generated.

3.2.1. Solution initialization parameters

As explained in Section 2.2, an initial solution is generated in two steps: the first one consists of randomly selecting a subset of nodes to be visited; the second one consists of constructing the tour involving the selected nodes, i.e., giving an order in the selected subset of nodes. It is desirable that the average number of selected nodes in a solution, $n \cdot p$, is close to the number of nodes in the optimal solution. A straightforward choice is to select p as the proportion between the distance limit and the TSP solution, i.e., $p = d_0/v(TSP)$. However, the results achieved by the B&C in [10] show that the optimal solution tends to visit a higher number of nodes than expected. Therefore, we have decided to overestimate the number of initial nodes using $p = \sqrt{d_0/v(TSP)}$. To approximate the TSP value, Lin-Kerninghan heuristic has been used and this computational time has been considered in the global time of the algorithm.

In order to get an idea of the influence of the parameter p on the population initialization and on EA4OP, we have tested three different choices of p: α^2, α and $\sqrt{\alpha}$ where $\alpha = d_0/v(TSP)$. The rest of the parameters have been set as detailed in Section 3.2.2. The experiments show that the best mean gap either for the initialization or for the EA4OP is obtained using $p = \sqrt{\alpha}$. Furthermore, in the initialization, the closest solutions to the optimum in terms of visited number of nodes are obtained using the parameter value mentioned. However, it is interesting to note that the higher the value of p used, the longer the time that is needed to initialize the population, due to the higher amount of nodes included in TSP problems that are solved during the initialization, see details in supplementary material.

3.2.2. Genetic operator parameters

The parameters value selection for the algorithm (*ncand*, *npop*, *d2d* and *pmut*) has been performed using non-parametric statistical tests: Friedman test for multiple (more than two) mean comparisons, Wilcoxon signed-rank test for two mean comparisons and Finner post-hoc test for pairwise comparisons [11]. For all of these tests, 0.05 has been used as significance level.

Taking into account that, depending on the target (gap or time), the selected values for the parameters might differ, gap has been prioritized over time. Therefore, the analysis is performed in two steps: in the first step, a Friedman test on the gap is carried out for each parameter. If it does not find significant differences for the parameter values, all of the values are considered for the next step. Otherwise if it finds significant differences between the achieved gaps by the different values, we continue with Finner post-hoc tests to select the values that obtain the best gaps. Those values which are not significantly different from the best gap are considered for the second step. If all values have significant differences with the best, this is the value chosen and the procedure finishes here for that parameter. In the second step, previously selected values are considered and the procedure detailed for the gap is repeated now for the time. In the case that there are several parameter values with no significant differences with the value that obtains the best mean time, the value with the lowest mean gap is chosen.

For each parameter, the following set of values has been considered: $ncand \in \{5,\ 7,\ 10\}$, $npop \in \{10,\ 20,\ 50,\ 100\}$, $pmut \in \{0.01,\ 0.05,\ 0.1\}$ and $d2d \in \{5,\ 10,\ 20,\ 50\}$ where $d2d < npop$. For each parameter a univariate analysis has been conducted, except for *npop* and *d2d* - for which a bivariate analysis has been carried out. The values *(npop, d2d)*=(100, 5) and (100, 10) have been excluded from the analysis because those configurations require an excessive amount of time. Each combination of the parameters has been run 10 times.

Table 3 details the mean gaps, the mean times and the p-values of the tests obtained during the selection procedure. For instance, based on this information, we have set *pmut* parameter to 0.01. In the first step, using Friedman test we obtained that there are no significant differences in terms of gap between the values of *pmut*, therefore, all the values were considered for the next step. Regarding the time, Friedman test gave that there exist significant differences between the *pmut* values, so we proceed with the Finner post-hoc test. Finally, parameter value 0.01 is selected since comparing the gap between *pmut* values with no significant differences in terms of computation times it obtains the lowest gap. After the statistical tests, the following parameter values were chosen for the computational experiments: $ncand = 10$, $npop = 100$, $d2d = 50$ and $pmut = 0.01$.

3.2.3. Tour improvement operator

Preliminary experiments in the 5 instances of the previous training set with 2-opt, 3-opt and Lin-Kernighan approaches showed that the Lin-Kernighan technique as TSP local search was the most suitable. We appreciated that the solutions obtained for the OP using this method were better than with the rest of the techniques, while the time needed to accomplish the search was not

Table 3: Statistical hypothesis testing for parameter selection

		Gap			Time		
Parameter	Value	Mean	Friedman p-value	Post Hoc corrected p-value	Mean	Friedman p-value*	Post Hoc corrected p-value
	10	5.184		-	5.287		
ncand	7	5.521	$< 2 \cdot 10^{-16}$	$6.5 \cdot 10^{-04}$	5.941		
	5	6.209		$< 2 \cdot 10^{-16}$	6.262		
	(100, 20)	2.559		-	6.910	$5.7 \cdot 10^{-14}$	
	(100, 50)	2.732		0.280	4.403		
	(50, 5)	4.281		$8 \cdot 10^{-07}$	5.146		
(npop, d2d)	(50, 10)	4.326	$< 2 \cdot 10^{-16}$	$4 \cdot 10^{-08}$	2.863		
	(50, 20)	4.390		$1 \cdot 10^{-09}$	1.949		
	(20, 5)	8.243		$< 2 \cdot 10^{-16}$	0.811		
	(20, 10)	8.577		$< 2 \cdot 10^{-16}$	0.485		
	(10, 5)	16.56		$< 2 \cdot 10^{-16}$	0.157		
	0.01	5.583			5.692		-
pmut	0.05	5.725	0.57		5.8702	$4.4 \cdot 10^{-03}$	0.583
	0.1	5.606			5.9279		$6 \cdot 10^{-03}$

*: Wilcoxon signed-rank test has been used to compare (100,20) and (100,50).

substantially larger.

3.2.4. Stopping criteria

As explained in Section 2.7, there are two stopping criteria. The first one, which is based on the distribution of the fitness, stops the algorithm when the first quartile of the population's fitness is the same as the best solution fitness. The second one, which is a time limitation, stops the algorithm when the execution time exceeds 5 hours.

3.3. EA4OP components validation

In this section, we verify that all the components in the EA4OP are necessary to obtain high quality solutions. We have implemented three algorithms in order to evaluate the contribution of the components in the EA4OP algorithm.

- Algorithm 3.3.1: This algorithm builds a large random population and applies the drop and add operators to each individual. We have considered the average number of solutions used by the EA4OP to set the size of the population, *npop*, for Algorithm 3.3.1, for each instance and generation. As we are using a steady-state algorithm, the amount of solutions used in a run of the EA4OP is equal to the initial population size plus the number of iterations. Algorithm 3.3.1 is used to evaluate the contribution of the evolution process of our algorithm.

- Algorithms 3.3.2 and Algorithm 3.3.3: In these algorithms, we consider a EA4OP but without the crossover. Instead of selecting two parents and crossing them, we select only one parent using the procedure of the parents selection operator and applying the mutation operator. Two different versions of this algorithm have been tested, both of which differ in the relaxation of the stopping criteria. Algorithm 3.3.2 stops when all the solutions of the population have the same fitness. Since Algorithm 3.3.2 obtains lower computation times than EA4OP, we have also considered Algorithm 3.3.3, which is similar to Algorithm 3.3.2 but stops when the computation time reaches the mean time used by EA4OP. Algorithms 3.3.2 and 3.3.3 are used to evaluate the contribution of the ER crossover operator.

In order to perform the comparison of these algorithms, all of them have been configured with the same parameters used in EA4OP ($ncand = 10$, $npop = 100$, $d2d = 50$ and $pmut = 0.01$), except the parameter *npop* for Algorithm 3.3.1, which has been explained above. These algorithms have been run in five medium-sized and five large-sized instances of each generation, which have been selected using the same criteria as in Section 3.2.

Table 4: Comparison between the results of Algorithm 3.3.1, Algorithm 3.3.2, Algorithm 3.3.3 and EA4OP.

		Algorithm 3.3.1		Algorithm 3.3.2		Algorithm 3.3.3		EA4OP	
Generation	Size	Gap	Time	Gap	Time	Gap	Time	Gap	Time
Gen1	Medium	13.75	10.40	10.71	1.81	9.63	4.64	1.76	4.54
	Large	15.19	15468.30	10.07	1527.96	7.44	5501.40	0.00	5500.71
Gen2	Medium	13.20	11.96	9.01	1.75	8.43	5.05	1.21	4.93
	Large	16.33	16887.87	10.80	1721.30	5.93	6397.23	0.00	6397.06
Gen3	Medium	14.58	12.46	11.32	1.95	10.08	5.45	3.69	5.04
	Large	17.15	17635.65	10.94	1719.08	7.95	6241.82	0.00	6241.36
Gen4	Medium	2.16	6.89	1.28	1.79	1.24	5.66	0.07	5.14
	Large	16.75	17095.59	10.82	1490.95	7.23	3498.83	0.00	3498.29

The results are summarized in Table 4. They show that building a large random population needs a large amount of time while it does not obtain competitive results in terms of solution quality. This large amount of time is due to the requirements for making a random population feasible. It can be concluded that the proposed evolution speeds up the generation of individuals. Furthermore, it is essential to obtain high quality solutions.

Table 4 also shows that in most of the instances Algorithm 3.3.3 improves the gap results of Algorithm 3.3.2, however they are still not competitive with those obtained by EA4OP. Therefore, it can be assumed that the proposed adaptation of the ER crossover operator has an important contribution in the EA4OP.

In view of these results, we assume that the contribution of the evolutionary part and, specifically, the proposed adaptation of the ER crossover are essential in the overall algorithm.

3.4. Comparison with state-of-the-art algorithms

The experiments have been run on a workstation with Intel(R) Xeon(R) CPU E5-2609 v3 @ 1.90GHz processor using a single thread and a maximum of 4 GB RAM.

The evolutionary algorithm for OP (EA4OP) was implemented in C language. We have reused the code from the Concorde TSP solver for the routines related to dynamic k-d trees and the Lin-Kernighan TSP method. The source code has been published with a GPLv3 license, except the third-party code mentioned above, which has an academic license. The code is available at `https://github.com/bcamath-ds/compass`.

For comparison purposes, the following algorithms have been tested: the exact B&C algorithm from [10] and two heuristics: GRASP-PR [6] and 2-P IA [20]. For each heuristic, 10 runs have been performed at each instance, while the exact algorithm has been run once. All the experiments have been performed under the same conditions: the same machine, the same language (C) and the same compiler (gcc 4.8.5) with the same flags (-O3).

For a fair comparison in terms of computational time, the results of the B&C algorithm have been obtained with CPLEX 12.5.0 instead of the original LP solver CPLEX 3.0. Note that the papers [6] and [20] considered the results published in [10].

New optimal solutions have been obtained with the updated execution of the B&C algorithm for all the instances that stopped after 5 hours in [10]: two in generation 1 (ts225, pr226), four in generation 2 (pr266, pr299, lin318, rd400) and four in generation 3 (pr144, pr299, lin318, rd400). The optimal solution for score generation 2 are 6662, 9182, 10923 and 13652, respectively, while in the paper, the mentioned solutions after 5 hours of computation were 6615, 9161, 10900 and 13648, respectively. However, the bounds published for score generation 1 and 3 in the original paper are higher than the values obtained with the updated software. We conjecture that, incidentally, upper bounds were published instead of the best known solutions. For the instances of generation 1, results 125 and 134 appeared in the original paper, and solutions 124 and 126 are now reported, respectively. For the ones of generation 3, old results 3809, 10358, 10382 and 13229 are different from new results 3745, 10343, 10368 and 13223.

The parameters used in the compared heuristic algorithms where those reported by default in the respective papers. However, we have increased two B&C parameters to take advantage of the resources of the current machines. We have experimentally checked that the updated parameters improve the results of the originals parameters. The parameters considered in the runs are as follows:

- B&C: In the cutting plane phase, 200 variables (instead of 100) can be added at each round of pricing up. Additionally, we resort to branching whenever the upper bound did not improve by at least 0.001 in the last 20 (instead of 10) cutting-plane iterations of the current branching node.
- 2-P IA: number of iterations without improvement before termination is 4500, number of nodes to choose from each iteration of route initialization and the number of nodes removed from each iterative change are 4.
- GRASP-PR: greediness parameter is 0.2, number of solutions is 100, constructive methods combine C1 and C2.

Next, the summary results and a comparative analysis is shown for medium-sized instances and large-sized instances. The detailed numerical results can be seen in Appendix A.

3.4.1. Comparison for medium-sized instances

The TSPLib instances of medium dimensionality contain 45 problems with 48 to 400 nodes. The Table 5 summarises the average quality gap (Gap) and time consumption (Time) for the four generations according to the size ranges, the best results between heuristics are highlighted in bold. See also performance by generation in Figure 6.

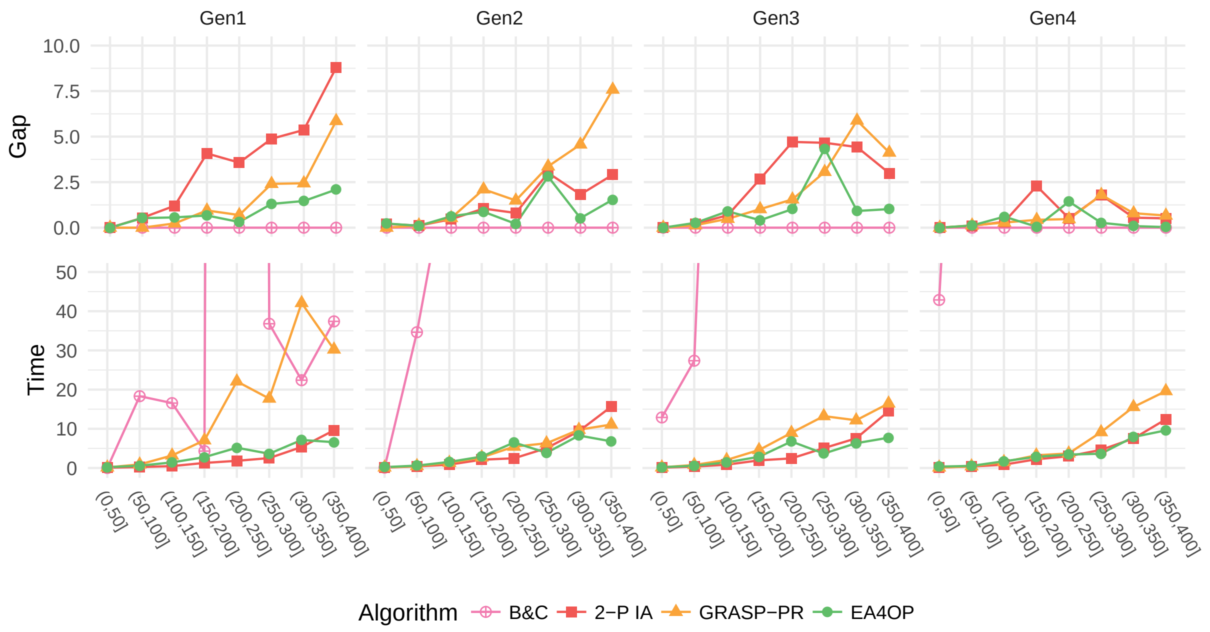


Figure 6: Average gap and time by range in medium-size instances.

Table 5: Algorithms comparison by range in medium-sized instances.

		B&C		2-P IA		GRASP-PR		EA4OP	
Range	#	Gap	Time	Gap	Time	Gap	Time	Gap	Time
(0,50]	12	*	13.97	0.05	**0.10**	*****	0.16	0.06	0.25
(50,100]	56	*	67.24	0.24	**0.36**	**0.10**	0.66	0.25	0.58
(100,150]	44	*	297.23	0.67	**0.77**	**0.38**	2.03	0.67	1.54
(150,200]	24	*	213.34	2.52	**1.90**	1.12	4.40	**0.50**	2.85
(200,250]	20	*	897.83	2.40	**2.45**	1.05	10.06	**0.74**	5.47
(250,300]	16	*	730.80	3.58	4.33	2.66	11.61	**2.18**	**3.71**
(300,350]	4	*	4854.90	3.04	7.46	3.42	19.90	**0.75**	**7.42**
(350,400]	4	*	1429.30	3.80	13.05	4.56	19.35	**1.17**	**7.68**
All	180	*	427.32	1.31	**1.67**	0.80	4.32	**0.63**	2.23

*: optimal solution achieved

Note that all the instances can be solved up to optimality by B&C. However, the execution time is extremely high for this exact approach. In terms of gap, GRASP-PR performed better in generations 1, 3 and 4, while 2-P IA obtained better averages in generation 2, as reported in Tables A.1, A.3, A.5 and A.7. Taking into account all the medium-sized instances, GRASP-PR obtains the best average gap. In terms of Time, 2-P IA obtains the best results in all the generations. However, EA4OP shows competitive results, obtaining similar execution times to those of 2-P IA in the smallest instances and better time results in the biggest instances.

Table 6 shows the performance of EA4OP versus 2-P IA and GRASP-PR. The table summarizes the following information: *Gap*, number of instances in which an algorithm's solution is higher than the other one's; *Time*, number of instances in which an algorithm's execution time is lower than the other one's; *Pareto*, number of instances in which an algorithm dominates the other algorithm. Pareto efficiency criterion states that a solution dominates the other one if it obtains better results in at least one of the objectives while not degrading any of the others (in our case the objectives are gap and time). Ties are computed in an additional column.

Table 6: Comparison against state-of-the-art heuristics in terms of quality, time and Pareto efficiency in medium-sized instances.

	Gen1			Gen2			Gen3			Gen4		
	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP
Gap	4	21	20	14	9	22	15	9	21	8	11	26
Time	39	0	6	26	0	19	29	0	16	31	0	14
Pareto	24	0	6	16	0	15	17	0	10	15	0	12
	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP
Gap	6	30	9	13	10	22	15	8	22	9	10	26
Time	11	0	34	23	0	22	13	0	32	24	0	21
Pareto	11	0	29	13	0	16	7	0	20	11	0	16

In terms of Gap, EA4OP obtained better solutions than 2-P IA in all four generations, whereas in terms of Time and Pareto, 2-P IA obtains better solutions in all four generations. When we compare EA4OP with GRASP-PR, in terms of Gap and Pareto, EA4OP is better than GRASP-PR in all four generations. In terms of Time, EA4OP obtains better results in generations 1 and 3, and little worse results in generations 2 and 4.

3.4.2. Comparison for large-sized instances

The TSPLib instances of large dimensionality contain 41 problems within 417 and 7397 nodes. Table 7 summarises the quality of solutions (Gap) and execution time (Time) for the four generations according to the size ranges. The number of solved instances is detailed in an extra column for B&C and GRASP-PR. The average gap was calculated excluding the missing solutions, whereas the average times were calculated considering 18000 seconds for problems in which the time limit was reached. The best results between heuristics are highlighted in bold. See also performance by generation in Figure 7.

Most of these instances (130 of 164) can not be solved up to optimality by B&C. Furthermore, B&C finished unexpectedly for 52 of the instances, not obtaining any solution. Globally, EA4OP obtained better solutions than B&C in 96 of the 164 cases. Compared with the rest of heuristic algorithms, EA4OP obtained better quality solutions in all the generations. Additionally, in this large-sized instance set, EA4OP shows the best performance in execution time compared with the

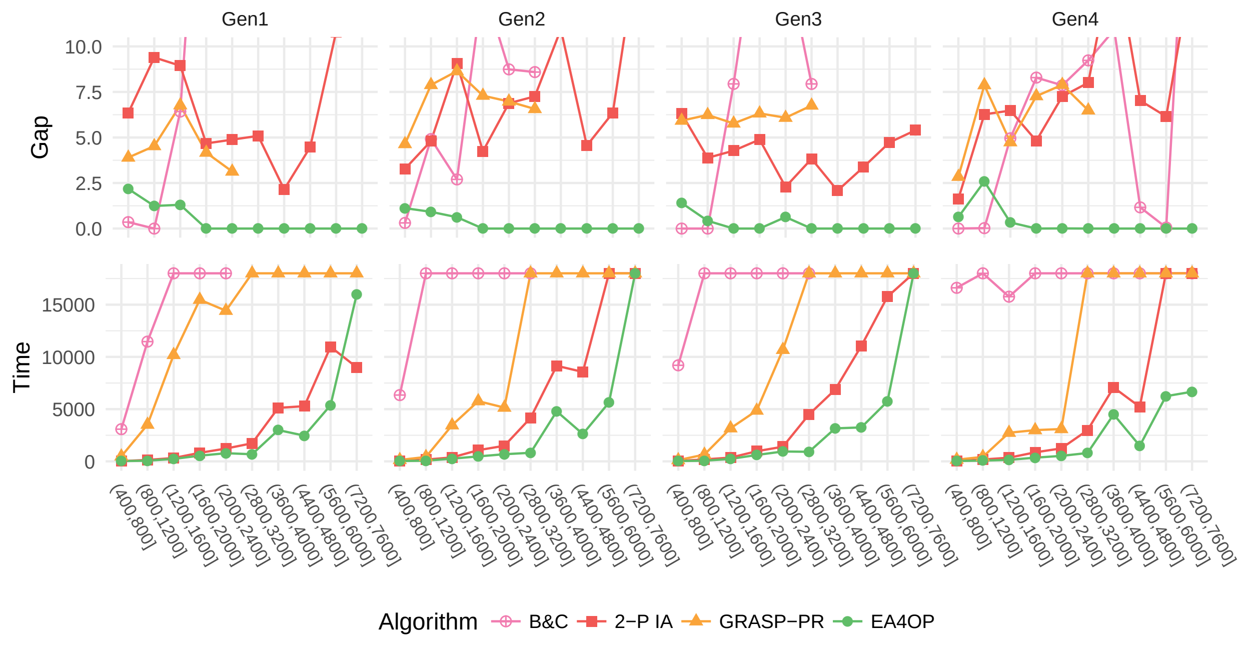


Figure 7: Average gap and time by problem size in large-sized instances.

Table 7: Comparison of algorithms by range in large-sized instances.

		B&C			2-P IA		GRASP-PR			EA4OP	
Range	#	#	Gap	Time	Gap	Time	#	Gap	Time	Gap	Time
(400,800]	60	55	0.16	9040.89	4.39	**29.15**	60	4.33	224.37	**1.33**	35.95
(800,1200]	20	12	1.24	16910.62	6.09	169.50	20	6.63	1254.55	**1.29**	**67.61**
(1200,1600]	28	15	5.65	16955.71	7.19	352.83	28	6.48	4889.35	**0.57**	**219.35**
(1600,2000]	16	11	15.50	-	4.66	931.49	16	6.27	7271.88	*	**496.68**
(2000,2400]	16	12	10.79	-	5.33	1341.13	15	6.21	8332.97	**0.16**	**733.63**
(2800,3200]	4	3	8.59	-	6.05	3339.27	3	6.60	-	*	**804.96**
(3600,4000]	4	1	10.92	-	7.61	7052.71	0	NA	-	*	**3859.71**
(4400,4800]	4	1	1.16	-	4.86	7527.78	0	NA	-	*	**2455.06**
(5600,6000]	8	1	0.06	-	7.01	15681.39	0	NA	-	*	**5745.60**
(7200,7600]	4	1	25.47	-	15.94	15750.76	0	NA	-	*	**14662.28**
All	164	112	4.21	13343.85	5.73	1899.47	142	5.54	5226.42	**0.76**	**990.42**

*: best known solution achieved
-: time limit of 5 hours exceeded
NA: solution not available after time limit exceeded

rest of the heuristics in all the generations. Note that GRASP-PR was not able to return any solution in 22 instances after the execution time was exceeded.

Table 8 shows that in large-sized instances EA4OP obtains much better solutions in terms of quality, time and Pareto efficiency, compared with 2-P IA and GRASP-PR for all the generations.

Table 8: Comparison against state-of-the-art heuristics in terms of quality, time and Pareto efficiency in large-sized instances.

	Gen1			Gen2			Gen3			Gen4		
	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP	2-P IA	tie	EA4OP
Gap	3	1	37	1	0	40	3	0	38	2	0	39
Time	16	0	25	8	1	32	9	1	31	9	0	32
Pareto	3	0	25	1	0	33	0	0	29	1	0	31
	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP	GRASP-PR	tie	EA4OP
Gap	4	0	37	0	1	40	1	0	40	1	0	40
Time	0	0	41	0	1	40	0	1	40	0	0	41
Pareto	0	0	37	0	0	41	0	0	40	0	0	40

3.5. EA4OP contributions

The computational experiments have shown that several characteristics are essential in the effectiveness of the EA4OP. Probably the most relevant feature is the use of unfeasible solutions during the search process. It allows us to obtain high quality solutions without being penalized in terms of computational time, as shown in Section 3.2.2. Furthermore, the parameter $d2d$ helps to strike a balance between the solution quality and the computational time.

To our knowledge, the initialization technique of the solutions used in the EA4OP is also novel for the OP. In the proposed initialization, the solutions are built based on the relation between the distance limit and the TSP value of the whole set of nodes (the Lin-Kerninghan approximation of this value). This relation is used to estimate the amount of nodes in the optimal solution and then the solutions are built randomly based on this information. This initialization might be useful, mainly, for population-based algorithms for the variations of the OP to provide diversity to the initial population.

We consider the adaptation of the ER crossover as a contribution to the solution of the OP and routing problems in general. In addition to the problems that consist of permutations, this adaptation also allows us to deal with a wider range of problems whose solution space consists of simple cycles. Moreover, as shown in Section 3.3, the proposed crossover turns out to be an effective technique to mix solutions in the OP.

Another contribution that we find remarkable for routing problems is the proposed approach to find the minimum cost insertion in the add operator. When the distance matrix is given by spatial points, its design allows the use of a data structure, i.e., k-d tree, that strongly reduces the computational cost, improving the overall results.

All in all, the EA4OP proves to be an efficient algorithm for the OP. Not only does the EA4OP obtain competitive results in medium-sized instances in comparison to the state-of-the-art algorithms, but it also achieves outstanding results in terms of quality in an even lower execution time.

4. Conclusions and future work

We have presented an efficient evolutionary algorithm for the OP. Essentially, the algorithm follows the steady-state genetic algorithm schema. It differs in that the proposed method maintains unfeasible solutions during the search and considers specific operators to recover it when required. An Edge Recombination crossover has been adapted for the OP, a novel method for node inclusion has been proposed and the Lin-Kerninghan heuristic has been used to improve route lengths.

We have tested the EA4OP in 344 instances based on TSPLib. We have found the EA4OP competitive in medium-sized instances (up to 400 nodes). Comparing the EA4OP in terms of Pareto efficiency, we have found that from the 180 instances of the medium-sized set, EA4OP gets 43 Pareto optimums while 2-P IA does so for 72 instances. Also, EA4OP obtains 81 Pareto optimums, while the GRASP-PR does so for 42 instances. As for the medium-sized instances, B&C has been run again with an updated LP solver in a modern machine, and 10 new optimal solutions were found: two in generation 1, four in generation 2 and four in generation 3 (for these instances, execution in [10] stopped because the time limit was reached).

The computational results on large-sized instances (up to 7397 nodes) are excellent for the EA4OP in terms of quality and time. Moreover, the EA4OP algorithm found higher solutions than the ones returned by the exact approach after five hours of computation. Additionally, the execution time is lower than the ones of the rest of the compared techniques. Particularly, from the 164 instances of the large-sized set, EA4OP obtained the Pareto optimum in 118 instances, while the 2-P IA, which turns out to be the most competitive heuristic algorithm, did it for 5 instances.

Ordering the algorithms in terms of average quality gap, we have obtained the following results: for medium-sized instances, B&C (0.00%), EA4OP (0.63%), GRASP-PR (0.80%) and 2-P IA (1.31%); and for large-sized instances, EA4OP (0.76%), B&C (4.21%), 2-P IA (5.73%) and GRASP-PR (5.54%).

Ordering the algorithms in terms of average time consumption, we have obtained the following results: for medium-sized instances, 2-P IA (1.67 sec), EA4OP (2.23 sec), GRASP-PR (4.32 sec) and B&C (427.32 sec); and for large-sized instances, EA4OP (990.42 sec), 2-P IA (1899.47 sec), GRASP-PR (5226.42 sec) and B&C (13343.85 sec).

In order to obtain better quality solutions or decrease time consumption, it would be interesting to advance developing new operators or adapt the ones developed for other routing problems. Additionally, it could be revelant to build better quality initial populations. Giving a different a priori probability to each node might contribute to this aim. Furthermore, it would be challenging to consider the very large-sized instances, in particular, 26 TSPLib instances left with nodes from 11849 to 85900. Another point of particular interest would be the application of the EA4OP to solve classical variants of OP (such as the team OP, the OP with time windows or the time dependent OP) as well as recent ones (such as the stochastic OP, the generalized OP, the arc OP, the multi-agent OP or the clustered OP).

Acknowledgements

This research has been partially supported by Spanish Ministry of Economy and Competitiveness MINECO through the BCAM Severo Ochoa excellence accreditation SEV-2013-0323 and Project I+D Excellence MTM2015-65317-P; by Spanish Ministry of Science and Innovation through the project TIN2016-78365-R; by the Basque Government through the BERC 2014-2017, IT609-13 and IT-928-16 programs; and by the University of the Basque Country UPV/EHU through the UFI BETS 2011 programme.

The authors thank Dr. A. Duarte, Dr. J.J. Salazar-Gonzalez and Dr. J. Silberholz, who, for comparison purposes, provided us with the codes used in [6], [10] and [20] respectively, and also the two anonymous reviewers, who significantly improved the manuscript with their comments.

Appendix A. Detailed results

In this Appendix the numerical results are detailed for the four algorithms (B&C, 2-P IA, GRASP-PR and EA4OP) and the full classification, that is, eight tables. Table A.1 shows the results for generation 1 and medium-sized instances, Table A.2 for generation 1 and large-sized instances, Table A.3 for generation 2 and medium-sized instances, Table A.4 for generation 2 and large-sized instances, Table A.5 for generation 3 and medium-sized instances, Table A.6 for generation 3 and large-sized instances, Table A.7 for generation 4 and medium-sized instances and Table A.8 for generation 4 and large-sized instances. The headings are as follows: *instance*, name codification of the instance; *best*, best known solution of the corresponding algorithm; *gap*, quality gap with respect to the global best known solution; *time*, average time (in seconds) of 10 runs. In the last row, average summary for gap and time are shown. The symbols mean the following: $*$, best known solution achieved (or optimum solution achieved for instances in which B&C finishes before time limit); $-$, execution stopped because 5-hour time limit was exceeded; NA, solution not available after time limit exceeded; “ . ”, the code finished unexpectedly. The best results for each instance among heuristics are in **bold**, in terms of quality solution and time. In the last row of the tables, average gap and average time are computed, considering 18000 seconds for problems that did not finish in that time. The averages are calculated excluding missing values.

Table A.1: Generation 1, $n \leq 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
att48	5314	0.50	31	31	*	0.00	**31**	*	**0.07**	**31**	*	0.15	**31**	*	0.25
gr48	2523	0.50	31	31	*	0.00	**31**	*	**0.07**	**31**	*	0.13	**31**	*	0.13
hk48	5731	0.50	30	30	*	0.00	**30**	*	**0.10**	**30**	*	0.19	**30**	*	0.24
eil51	213	0.50	29	29	*	0.00	**29**	*	0.10	**29**	*	**0.06**	**29**	*	0.24
berlin52	3771	0.50	37	37	*	0.00	**37**	*	**0.09**	**37**	*	0.22	**37**	*	0.30
brazil58	12698	0.50	46	46	*	0.00	**46**	*	**0.10**	**46**	*	0.32	**46**	*	1.00
st70	338	0.50	43	43	*	0.10	**43**	*	**0.19**	**43**	*	0.29	**43**	*	0.32
eil76	269	0.50	47	47	*	0.10	46	2.13	**0.19**	**47**	*	0.20	46	2.13	0.32
pr76	54080	0.50	49	49	*	0.10	**49**	*	**0.19**	**49**	*	0.94	**49**	*	0.61
gr96	27605	0.50	64	64	*	0.10	**64**	*	**0.37**	**64**	*	1.91	**64**	*	1.44
rat99	606	0.50	52	52	*	0.40	51	1.92	**0.22**	**52**	*	0.48	**52**	*	0.66
kroA100	10641	0.50	56	56	*	0.40	**56**	*	**0.27**	**56**	*	1.33	55	1.79	0.34
kroB100	11071	0.50	58	58	*	95.40	**58**	*	**0.35**	**58**	*	1.57	57	1.72	0.63
kroC100	10375	0.50	56	56	*	0.40	**56**	*	**0.41**	**56**	*	1.12	**56**	*	0.48
kroD100	10647	0.50	59	59	*	0.10	**59**	*	**0.30**	**59**	*	1.56	58	1.69	0.65
kroE100	11034	0.50	57	57	*	159.20	55	3.51	**0.28**	**57**	*	1.42	**57**	*	0.50
rd100	3955	0.50	61	61	*	0.20	**61**	*	**0.38**	**61**	*	1.32	**61**	*	0.74
eil101	315	0.50	64	64	*	0.10	63	1.56	**0.37**	**64**	*	0.48	**64**	*	0.79
lin105	7190	0.50	66	66	*	0.30	**66**	*	**0.26**	**66**	*	2.77	**66**	*	1.42
pr107	22152	0.50	54	54	*	0.30	**54**	*	**0.21**	**54**	*	0.73	**54**	*	0.93
gr120	3471	0.50	75	75	*	0.10	74	1.33	**0.48**	**75**	*	2.29	74	1.33	1.20
pr124	29515	0.50	75	75	*	0.30	**75**	*	**0.31**	**75**	*	5.20	**75**	*	1.11
bier127	59141	0.50	103	103	*	0.30	**103**	*	**0.44**	**103**	*	3.85	**103**	*	1.18
pr136	48386	0.50	71	71	*	1.40	69	2.82	**0.51**	70	1.41	1.31	**71**	*	0.96
gr137	34927	0.50	81	81	*	1.50	**81**	*	**0.61**	**81**	*	7.10	78	3.70	3.44
pr144	29269	0.50	77	77	*	1.30	73	5.19	**0.42**	**77**	*	5.93	**77**	*	2.61
kroA150	13262	0.50	86	86	*	175.40	85	1.16	**0.90**	**86**	*	2.64	**86**	*	1.17
kroB150	13065	0.50	87	87	*	1.20	**86**	1.15	**0.94**	**86**	1.15	2.90	**86**	1.15	1.00
pr152	36841	0.50	77	77	*	1.40	76	1.30	**0.72**	**77**	*	8.78	**77**	*	3.64
u159	21040	0.50	93	93	*	3.40	82	11.83	**0.86**	**92**	1.08	5.20	**92**	1.08	1.11
rat195	1162	0.50	102	102	*	2.60	**99**	2.94	**1.01**	**99**	2.94	2.62	**99**	2.94	1.78
d198	7890	0.50	123	123	*	3.20	120	2.44	**1.46**	122	0.81	11.62	**123**	*	6.68
kroA200	14684	0.50	117	117	*	1.20	112	4.27	2.04	**117**	*	6.53	**117**	*	**1.74**
kroB200	14719	0.50	119	119	*	14.10	117	1.68	1.68	118	0.84	7.73	**119**	*	**1.66**
gr202	20080	0.50	145	145	*	12.70	140	3.45	**2.17**	**145**	*	11.84	**145**	*	6.89
ts225	63322	0.50	124	124	*	10216.30	**124**	*	1.45	**124**	*	6.08	**124**	*	**1.28**
tsp225	1958	0.50	129	129	*	94.40	117	9.30	**1.61**	126	2.33	7.76	**127**	1.55	2.29
pr226	40185	0.50	126	126	*	166.20	121	3.97	**1.20**	**126**	*	21.55	**126**	*	6.61
gr229	67301	0.50	176	176	*	0.90	174	1.14	**2.76**	174	1.14	63.24	**176**	*	8.81
gil262	1189	0.50	158	158	*	0.90	150	5.06	3.06	151	4.43	9.56	**156**	1.27	**2.84**
pr264	24568	0.50	132	132	*	21.20	**132**	*	**1.16**	**132**	*	24.77	**132**	*	5.62
a280	1290	0.50	147	147	*	13.60	133	9.52	**2.20**	**143**	2.72	11.19	**143**	2.72	3.00
pr299	24096	0.50	162	162	*	111.50	154	4.94	3.71	158	2.47	25.32	**160**	1.23	**3.12**
lin318	21015	0.50	205	205	*	22.40	194	5.37	**5.36**	200	2.44	42.09	**202**	1.46	7.15
rd400	7641	0.50	239	239	*	37.40	218	8.79	9.62	225	5.86	30.23	**234**	2.09	**6.59**
average					*	248.05		2.15	**1.14**		0.66	7.66		**0.62**	2.12

Table A.2: Generation 1, $n > 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
fl417	5931	0.50	228	228	*	-	**227**	0.44	**5.67**	226	0.88	308.91	224	1.75	11.84
gr431	85707	0.50	350	350	*	139.90	343	2.00	**12.32**	346	1.14	479.25	**349**	0.29	32.84
pr439	53609	0.50	313	313	*	833.30	**310**	0.96	14.48	305	2.56	626.66	**310**	0.96	**9.92**
pcb442	25389	0.50	251	251	*	14.90	235	6.37	8.37	235	6.37	44.02	**244**	2.79	**6.93**
d493	17501	0.50	320	320	*	347.30	297	7.19	**18.88**	312	2.50	539.47	**315**	1.56	19.10
att532	13843	0.50	363	363	*	593.00	340	6.34	**16.23**	**351**	3.31	597.00	347	4.41	23.14
ali535	101170	0.50	424	.	.	.	416	1.89	**23.60**	417	1.65	793.08	**424**	*	73.03
pa561	1382	0.50	356	356	*	2103.60	340	4.49	**17.74**	330	7.30	134.50	**348**	2.25	23.18
u574	18453	0.50	354	354	*	61.40	316	10.73	**17.53**	332	6.21	205.07	**344**	2.82	17.93
rat575	3387	0.50	322	322	*	59.50	293	9.01	19.89	302	6.21	103.68	**309**	4.04	**13.76**
p654	17322	0.50	344	327	4.94	-	**344**	*	**26.68**	343	0.29	1611.23	336	2.33	28.89
d657	24456	0.50	386	386	*	715.70	344	10.88	**20.41**	367	4.92	240.25	**377**	2.33	23.24
gr666	147179	0.50	503	503	*	634.20	474	5.77	**29.49**	490	2.58	855.40	**497**	1.19	109.54
u724	20955	0.50	439	439	*	1077.10	358	18.45	35.04	415	5.47	374.33	**429**	2.28	**27.77**
rat783	4403	0.50	438	438	*	594.30	391	10.73	43.91	407	7.08	273.84	**422**	3.65	**34.59**
dsj1000	9329844	0.50	632	.	.	.	562	11.08	136.57	604	4.43	4123.41	**632**	*	**81.20**
pr1002	129523	0.50	604	604	*	-	516	14.57	118.87	558	7.62	1864.39	**572**	5.30	**45.92**
u1060	112047	0.50	627	.	.	.	577	7.97	**85.97**	607	3.19	3026.01	**627**	*	90.04
vm1084	119649	0.50	777	777	*	4927.40	726	6.56	170.98	744	4.25	6309.60	**770**	0.90	**56.29**
pcb1173	28446	0.50	633	.	.	.	590	6.79	174.26	613	3.16	2244.38	**633**	*	**60.65**
d1291	25401	0.50	684	.	.	.	**684**	*	**173.44**	670	2.05	-	646	5.56	434.87
rl1304	126474	0.50	766	.	.	.	627	18.15	228.75	694	9.40	4790.58	**766**	*	**102.45**
rl1323	135100	0.50	811	811	*	-	674	16.89	327.32	706	12.95	6345.53	**782**	3.58	**89.68**
nrw1379	28319	0.50	771	.	.	.	740	4.02	351.37	747	3.11	4934.80	**771**	*	**106.97**
fl1400	10064	0.50	1043	909	12.85	-	935	10.35	**477.51**	950	8.92	-	**1043**	*	518.25
u1432	76485	0.50	738	.	.	.	697	5.56	249.66	693	6.10	1252.50	**738**	*	**121.46**
fl1577	11125	0.50	880	.	.	.	813	7.61	442.40	837	4.89	-	**880**	*	**286.47**
d1655	31064	0.50	846	.	.	.	820	3.07	**566.47**	800	5.44	7889.11	**846**	*	757.70
vm1748	168278	0.50	1246	873	29.94	-	1195	4.09	647.09	1219	2.17	-	**1246**	*	**178.50**
u1817	28601	0.50	879	.	.	.	865	1.59	**771.89**	864	1.71	-	**879**	*	975.58
rl1889	158268	0.50	1167	890	23.74	-	1051	9.94	1291.33	1081	7.37	-	**1167**	*	**269.81**
d2103	40225	0.50	1069	.	.	.	1018	4.77	993.63	1000	6.45	-	**1069**	*	**951.27**
u2152	32127	0.50	1048	.	.	.	1021	2.58	1721.31	1019	2.77	-	**1048**	*	**1350.23**
u2319	117128	0.50	1167	.	.	.	1145	1.89	730.20	1165	0.17	3669.11	**1167**	*	**423.26**
pr2392	189016	0.50	1292	1140	11.76	-	1159	10.29	1461.82	NA	NA	-	**1292**	*	**402.29**
pcb3038	68847	0.50	1572	.	.	.	1492	5.09	1732.09	NA	NA	-	**1572**	*	**681.94**
fl3795	14386	0.50	1815	.	.	.	1776	2.15	5120.78	NA	NA	-	**1815**	*	**2994.90**
fnl4461	91283	0.50	2350	.	.	.	2245	4.47	5286.82	NA	NA	-	**2350**	*	**2462.65**
rl5915	282765	0.50	3358	.	.	.	2868	14.59	12111.74	NA	NA	-	**3358**	*	**5361.54**
rl5934	278023	0.50	3145	.	.	.	2925	7.00	9815.39	NA	NA	-	**3145**	*	**5382.25**
pla7397	11630364	0.50	5141	.	.	.	3692	28.19	**9003.06**	NA	NA	-	**5141**	*	15981.78
average					3.96	7433.41		7.43	1329.29		4.55	7893.56		**1.17**	**990.82**

Table A.3: Generation 2, $n \leq 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
att48	5314	0.50	1717	1717	*	0.00	**1717**	*	**0.08**	**1717**	*	0.16	**1717**	*	0.32
gr48	2523	0.50	1761	1761	*	0.20	1750	0.62	**0.13**	**1761**	*	0.15	1749	0.68	0.20
hk48	5731	0.50	1614	1614	*	0.10	**1614**	*	**0.11**	**1614**	*	0.14	**1614**	*	0.16
eil51	213	0.50	1674	1674	*	0.40	**1674**	*	0.20	**1674**	*	**0.14**	1668	0.36	0.18
berlin52	3771	0.50	1897	1897	*	93.40	**1897**	*	**0.10**	**1897**	*	0.19	**1897**	*	0.35
brazil58	12698	0.50	2220	2220	*	0.10	**2220**	*	**0.17**	**2220**	*	0.31	2218	0.09	1.52
st70	338	0.50	2286	2286	*	19.40	2285	0.04	0.35	**2286**	*	0.35	2285	0.04	**0.31**
eil76	269	0.50	2550	2550	*	0.10	2540	0.39	**0.31**	**2550**	*	0.35	**2550**	*	0.43
pr76	54080	0.50	2708	2708	*	0.40	**2708**	*	**0.32**	**2708**	*	0.52	**2708**	*	0.48
gr96	27605	0.50	3396	3396	*	1.70	3394	0.06	**0.51**	**3396**	*	0.72	3394	0.06	1.44
rat99	606	0.50	2944	2944	*	0.90	2932	0.41	0.55	**2944**	*	**0.47**	**2944**	*	0.49
kroA100	10641	0.50	3212	3212	*	0.90	**3212**	*	0.69	**3212**	*	0.69	**3212**	*	**0.57**
kroB100	11071	0.50	3241	3241	*	6.70	3239	0.06	**0.50**	**3241**	*	0.62	3238	0.09	0.52
kroC100	10375	0.50	2947	2947	*	85.60	**2947**	*	**0.53**	2909	1.29	0.59	2931	0.54	0.60
kroD100	10647	0.50	3307	3307	*	45.00	3295	0.36	**0.56**	**3307**	*	0.72	**3307**	*	0.65
kroE100	11034	0.50	3090	3090	*	230.10	**3090**	*	0.57	3082	0.26	0.64	3082	0.26	**0.50**
rd100	3955	0.50	3359	3359	*	0.20	3351	0.24	0.52	3351	0.24	0.70	**3359**	*	**0.50**
eil101	315	0.50	3655	3655	*	153.00	3636	0.52	**0.50**	3643	0.33	0.62	**3655**	*	0.82
lin105	7190	0.50	3544	3544	*	67.30	3536	0.23	**0.56**	**3544**	*	1.12	3530	0.40	1.10
pr107	22152	0.50	2667	2667	*	0.60	**2667**	*	**0.40**	**2667**	*	0.55	**2667**	*	1.05
gr120	3471	0.50	4371	4371	*	35.80	4358	0.30	**0.74**	**4371**	*	0.84	4356	0.34	1.37
pr124	29515	0.50	3917	3917	*	0.50	**3917**	*	**0.54**	3901	0.41	1.96	3899	0.46	1.34
bier127	59141	0.50	5383	5383	*	58.80	5328	1.02	**1.08**	5331	0.97	2.12	**5381**	0.04	1.71
pr136	48386	0.50	4309	4309	*	2.10	4244	1.51	**1.02**	4228	1.88	1.03	**4309**	*	1.15
gr137	34927	0.50	4286	4286	*	196.90	**4281**	0.12	**0.87**	4270	0.37	1.79	4099	4.36	3.09
pr144	29269	0.50	4003	4003	*	90.40	3963	1.00	**0.74**	**4003**	*	2.54	3965	0.95	3.02
kroA150	13262	0.50	4918	4918	*	241.40	**4913**	0.10	1.80	4842	1.55	**1.05**	4902	0.33	1.26
kroB150	13065	0.50	4869	4869	*	24.80	4853	0.33	1.32	4853	0.33	**1.08**	**4869**	*	1.19
pr152	36841	0.50	4279	4279	*	2.20	**4269**	0.23	**1.38**	4227	1.22	3.47	4245	0.79	3.47
u159	21040	0.50	4960	4960	*	192.20	4938	0.44	1.90	4889	1.43	1.83	**4941**	0.38	**1.44**
rat195	1162	0.50	5791	5791	*	128.80	5666	2.16	1.99	5612	3.09	1.64	**5703**	1.52	**1.55**
d198	7890	0.50	6670	6670	*	74.20	6622	0.72	**2.10**	6625	0.67	4.39	**6660**	0.15	7.33
kroA200	14684	0.50	6547	6547	*	68.70	6461	1.31	2.67	6279	4.09	2.76	**6534**	0.20	**1.71**
kroB200	14719	0.50	6419	6419	*	34.70	**6328**	1.42	2.69	6282	2.13	2.16	6278	2.20	**1.97**
gr202	20080	0.50	7789	7789	*	85.70	7703	1.10	**2.13**	7659	1.67	4.99	**7789**	*	8.77
ts225	63322	0.50	6834	6834	*	6.60	6749	1.24	1.68	6743	1.33	2.75	**6819**	0.22	**1.47**
tsp225	1958	0.50	6987	6987	*	174.50	**6936**	0.73	2.60	6818	2.42	2.94	**6936**	0.73	**1.87**
pr226	40185	0.50	6662	6662	*	74.10	6646	0.24	**2.67**	6621	0.62	4.96	**6658**	0.06	7.29
gr229	67301	0.50	9177	9177	*	182.60	9111	0.72	**3.33**	9046	1.43	11.59	**9174**	0.03	13.19
gil262	1189	0.50	8321	8321	*	89.60	8100	2.66	5.20	7907	4.98	**3.15**	**8175**	1.75	3.47
pr264	24568	0.50	6654	6654	*	23.00	6244	6.16	**4.01**	**6654**	*	8.14	6173	7.23	5.94
a280	1290	0.50	8428	8428	*	103.80	8269	1.89	5.61	8021	4.83	4.60	**8304**	1.47	**2.85**
pr299	24096	0.50	9182	9182	*	426.50	9060	1.33	5.45	8846	3.66	9.35	**9112**	0.76	**3.23**
lin318	21015	0.50	10923	10923	*	862.40	10724	1.82	9.39	10424	4.57	9.77	**10866**	0.52	**8.29**
rd400	7641	0.50	13652	13652	*	293.50	13255	2.91	15.66	12617	7.58	11.09	**13442**	1.54	**6.80**
average					*	92.89		0.76	**1.92**		1.19	2.48		**0.63**	2.38

Table A.4: Generation 2, $n > 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
fl417	5931	0.50	11894	11894	*	-	**11873**	0.18	**9.40**	11787	0.90	105.45	11787	0.90	16.73
gr431	85707	0.50	18318	18318	*	969.50	18112	1.12	**20.07**	17908	2.24	174.86	**18287**	0.17	51.38
pr439	53609	0.50	16171	16171	*	1298.30	15505	4.12	24.62	15698	2.92	230.64	**16085**	0.53	**11.77**
pcb442	25389	0.50	14484	14484	*	6259.10	13895	4.07	17.77	13595	6.14	15.03	**14273**	1.46	**6.83**
d493	17501	0.50	16729	.	.	.	16450	1.67	19.50	16355	2.24	103.18	**16729**	*	**17.15**
att532	13843	0.50	19598	19598	*	-	18755	4.30	36.95	18903	3.55	109.26	**19265**	1.70	**23.43**
ali535	101170	0.50	21954	21954	*	2099.70	21394	2.55	**37.79**	21202	3.43	175.15	**21910**	0.20	95.05
pa561	1382	0.50	19576	19576	*	1487.10	18279	6.63	29.36	17904	8.54	32.39	**18894**	3.48	**23.45**
u574	18453	0.50	19351	19351	*	612.50	18809	2.80	36.86	17785	8.09	36.94	**18966**	1.99	**16.33**
rat575	3387	0.50	18251	18251	*	931.10	17670	3.18	30.94	17293	5.25	32.20	**17705**	2.99	**14.97**
p654	17322	0.50	17821	17160	3.71	-	17182	3.59	**37.39**	17358	2.60	457.42	**17821**	*	42.82
d657	24456	0.50	21503	21503	*	2682.40	19969	7.13	43.01	19253	10.46	49.17	**21162**	1.59	**22.90**
gr666	147179	0.50	26336	.	.	.	26064	1.03	**52.93**	25657	2.58	351.18	**26336**	*	136.48
u724	20955	0.50	24223	24223	*	5830.50	23311	3.77	53.95	22852	5.66	69.00	**23793**	1.78	**28.71**
rat783	4403	0.50	24861	.	.	.	24098	3.07	62.44	23617	5.00	88.40	**24861**	*	**32.36**
dsj1000	9329844	0.50	35772	35772	*	-	33354	6.76	142.94	32630	8.78	409.18	**34463**	3.66	**83.34**
pr1002	129523	0.50	31746	27066	14.74	-	30440	4.11	95.47	29416	7.34	219.21	**31746**	*	**46.19**
u1060	112047	0.50	35110	.	.	.	34061	2.99	138.50	32184	8.33	367.08	**35110**	*	**77.78**
vm1084	119649	0.50	40687	40687	*	-	38642	5.03	267.85	37699	7.34	758.46	**40308**	0.93	**55.67**
pcb1173	28446	0.50	35826	.	.	.	33992	5.12	204.88	33096	7.62	370.54	**35826**	*	**69.94**
d1291	25401	0.50	35153	.	.	.	31880	9.31	304.48	33781	3.90	1735.53	**35153**	*	**289.25**
rl1304	126474	0.50	40561	.	.	.	38654	4.70	422.77	35268	13.05	732.29	**40561**	*	**97.68**
rl1323	135100	0.50	43347	43347	*	-	37905	12.55	326.45	35908	17.16	799.79	**41459**	4.36	**89.78**
nrw1379	28319	0.50	45602	.	.	.	42693	6.38	273.21	42690	6.39	694.02	**45602**	*	**117.51**
fl1400	10064	0.50	56258	53222	5.40	-	53329	5.21	**567.33**	49614	11.81	4025.94	**56258**	*	794.15
u1432	76485	0.50	44810	.	.	.	41791	6.74	248.94	41956	6.37	716.32	**44810**	*	**100.91**
fl1577	11125	0.50	45505	.	.	.	37061	18.56	510.23	44675	1.82	15494.72	**45505**	*	**334.28**
d1655	31064	0.50	47211	.	.	.	44895	4.91	**551.08**	44080	6.63	1548.97	**47211**	*	683.17
vm1748	168278	0.50	66685	.	.	.	65106	2.37	1536.33	62818	5.80	15203.11	**66685**	*	**195.85**
u1817	28601	0.50	50366	.	.	.	47606	5.48	**589.37**	47196	6.29	2025.62	**50366**	*	734.39
rl1889	158268	0.50	60084	52047	13.38	-	57552	4.21	1612.05	53798	10.46	4273.16	**60084**	*	**286.07**
d2103	40225	0.50	57202	.	.	.	50715	11.34	1038.20	53128	7.12	6370.31	**57202**	*	**682.28**
u2152	32127	0.50	60211	53976	10.36	-	56556	6.07	1184.77	55250	8.24	4063.42	**60211**	*	**1164.38**
u2319	117128	0.50	78102	72790	6.80	-	73848	5.45	1915.20	74799	4.23	3624.98	**78102**	*	**447.06**
pr2392	189016	0.50	71018	64577	9.07	-	67711	4.66	1828.25	65111	8.32	6519.53	**71018**	*	**440.57**
pcb3038	68847	0.50	91842	83951	8.59	-	85176	7.26	4158.91	85813	6.56	-	**91842**	*	**820.37**
fl3795	14386	0.50	103397	.	.	.	92086	10.94	9148.20	NA	NA	-	**103397**	*	**4788.96**
fnl4461	91283	0.50	140424	.	.	.	134030	4.55	8559.91	NA	NA	-	**140424**	*	**2618.15**
rl5915	282765	0.50	176678	.	.	.	164345	6.98	-	NA	NA	-	**176678**	*	**5512.40**
rl5934	278023	0.50	171649	.	.	.	161816	5.73	-	NA	NA	-	**171649**	*	**5757.80**
pla7397	11630364	0.50	272452	.	.	.	229543	15.75	-	NA	NA	-	**272452**	*	-
average					3.27	11644.10		5.67	2198.50		6.48	4389.82		**0.63**	**1093.37**

Table A.5: Generation 3, $n \leq 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
att48	5314	0.50	1049	1049	*	38.50	**1049**	*	**0.13**	**1049**	*	0.18	**1049**	*	0.26
gr48	2523	0.50	1480	1480	*	0.20	**1480**	*	**0.07**	**1480**	*	0.20	**1480**	*	0.13
hk48	5731	0.50	1764	1764	*	0.00	**1764**	*	**0.09**	**1764**	*	0.14	**1764**	*	0.22
eil51	213	0.50	1399	1399	*	0.20	**1399**	*	**0.12**	**1399**	*	0.17	1398	0.07	0.22
berlin52	3771	0.50	1036	1036	*	124.70	**1036**	*	**0.19**	**1036**	*	0.30	1034	0.19	0.64
brazil58	12698	0.50	1702	1702	*	0.00	**1702**	*	**0.13**	**1702**	*	0.33	**1702**	*	0.71
st70	338	0.50	2108	2108	*	0.40	**2108**	*	**0.24**	**2108**	*	0.37	**2108**	*	0.31
eil76	269	0.50	2467	2467	*	0.40	2461	0.24	**0.30**	2462	0.20	0.44	**2467**	*	0.36
pr76	54080	0.50	2430	2430	*	0.20	**2430**	*	**0.26**	**2430**	*	0.56	**2430**	*	0.57
gr96	27605	0.50	3170	3170	*	61.50	**3170**	*	**0.39**	3153	0.54	1.07	3166	0.13	1.41
rat99	606	0.50	2908	2908	*	4.90	**2896**	0.41	**0.47**	2881	0.93	0.80	2886	0.76	0.78
kroA100	10641	0.50	3211	3211	*	63.30	**3211**	*	**0.30**	**3211**	*	1.16	3180	0.97	0.38
kroB100	11071	0.50	2804	2804	*	0.60	**2804**	*	**0.46**	**2804**	*	1.34	2785	0.68	0.51
kroC100	10375	0.50	3155	3155	*	1.50	**3155**	*	**0.38**	3149	0.19	0.86	**3155**	*	0.44
kroD100	10647	0.50	3167	3167	*	10.70	3123	1.39	0.65	**3167**	*	1.18	3141	0.82	**0.58**
kroE100	11034	0.50	3049	3049	*	1.50	3027	0.72	0.56	**3049**	*	1.48	**3049**	*	**0.47**
rd100	3955	0.50	2926	2926	*	113.20	**2924**	0.07	0.62	**2924**	0.07	0.90	2923	0.10	**0.48**
eil101	315	0.50	3345	3345	*	29.80	3333	0.36	**0.46**	3322	0.69	0.76	**3345**	*	0.56
lin105	7190	0.50	2986	2986	*	51.90	**2986**	*	**0.54**	**2986**	*	1.89	2973	0.44	2.09
pr107	22152	0.50	1877	1877	*	660.90	**1877**	*	**0.29**	**1877**	*	1.15	1802	4.00	0.82
gr120	3471	0.50	3779	3779	*	1.50	3736	1.14	**0.96**	3745	0.90	1.15	**3748**	0.82	1.36
pr124	29515	0.50	3557	3557	*	1021.50	3517	1.12	**0.62**	**3549**	0.22	2.41	3455	2.87	0.88
bier127	59141	0.50	2365	2365	*	79.90	2356	0.38	**1.08**	2332	1.40	2.07	**2361**	0.17	2.62
pr136	48386	0.50	4390	4390	*	86.70	**4390**	*	**0.93**	4380	0.23	2.56	**4390**	*	1.13
gr137	34927	0.50	3954	3954	*	8.60	3928	0.66	**1.13**	3926	0.71	1.89	**3954**	*	1.88
pr144	29269	0.50	3745	3745	*	112.60	3633	2.99	**0.77**	**3745**	*	3.36	3700	1.20	2.41
kroA150	13262	0.50	5039	5039	*	330.70	**5037**	0.04	1.26	5018	0.42	3.06	5019	0.40	**1.07**
kroB150	13065	0.50	5314	5314	*	107.60	5267	0.88	1.31	5272	0.79	2.31	**5314**	*	**1.04**
pr152	36841	0.50	3905	3905	*	1122.40	3557	8.91	**0.80**	**3905**	*	4.07	3902	0.08	3.62
u159	21040	0.50	5272	5272	*	52.20	**5272**	*	1.33	**5272**	*	4.46	**5272**	*	**0.94**
rat195	1162	0.50	6195	6195	*	49.90	**6174**	0.34	2.22	6086	1.76	3.06	6139	0.90	**2.00**
d198	7890	0.50	6320	6320	*	286.10	5985	5.30	**1.86**	6162	2.50	5.86	**6290**	0.47	7.14
kroA200	14684	0.50	6123	6123	*	122.30	6048	1.22	2.73	6084	0.64	4.64	**6114**	0.15	**1.72**
kroB200	14719	0.50	6266	6266	*	40.10	**6251**	0.24	2.79	6190	1.21	5.46	6213	0.85	**1.77**
gr202	20080	0.50	8616	8616	*	224.80	8111	5.86	**2.05**	8419	2.29	9.12	**8605**	0.13	10.45
ts225	63322	0.50	7575	7575	*	171.20	7149	5.62	1.47	7510	0.86	6.15	**7575**	*	**1.14**
tsp225	1958	0.50	7740	7740	*	150.30	7353	5.00	**2.38**	**7565**	2.26	5.04	7488	3.26	2.58
pr226	40185	0.50	6993	6993	*	32.60	6652	4.88	**1.97**	**6964**	0.41	15.50	6908	1.22	8.01
gr229	67301	0.50	6328	6328	*	10.20	6190	2.18	**4.42**	6205	1.94	9.03	**6297**	0.49	11.65
gil262	1189	0.50	9246	9246	*	133.40	8915	3.58	5.68	8922	3.50	6.07	**9094**	1.64	**3.94**
pr264	24568	0.50	8137	8137	*	20.70	7820	3.90	3.98	7959	2.19	17.88	**8068**	0.85	**3.62**
a280	1290	0.50	9774	9774	*	213.30	8719	10.79	4.53	**9426**	3.56	9.42	8684	11.15	**3.22**
pr299	24096	0.50	10343	10343	*	363.60	**10305**	0.37	6.07	10033	3.00	19.61	9959	3.71	**3.95**
lin318	21015	0.50	10368	10368	*	534.80	9909	4.43	7.57	9758	5.88	12.18	**10273**	0.92	**6.33**
rd400	7641	0.50	13223	13223	*	293.20	12828	2.99	14.49	12678	4.12	16.46	**13088**	1.02	**7.74**
average					*	149.66		1.69	**1.80**		0.96	4.18		**0.90**	2.31

Table A.6: Generation 3, $n > 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
fl417	5931	0.50	14220	14220	*	6227.60	12792	10.04	12.49	13709	3.59	73.99	**14186**	0.24	**12.45**
gr431	85707	0.50	10911	10911	*	1046.90	10735	1.61	**17.18**	10500	3.77	103.88	**10817**	0.86	54.50
pr439	53609	0.50	15160	15160	*	-	13006	14.21	15.35	14694	3.07	153.19	**15097**	0.42	**10.96**
pcb442	25389	0.50	14819	14819	*	-	14446	2.52	11.57	14206	4.14	31.56	**14522**	2.00	**6.58**
d493	17501	0.50	25167	25167	*	-	21458	14.74	**15.15**	23362	7.17	197.43	**24981**	0.74	19.18
att532	13843	0.50	15498	15498	*	933.20	15178	2.06	23.23	14573	5.97	75.56	**15342**	1.01	**22.75**
ali535	101170	0.50	9328	.	.	.	8884	4.76	**26.04**	8672	7.03	162.88	**9328**	*	94.09
pa561	1382	0.50	14482	14482	*	10543.80	13662	5.66	35.44	13271	8.36	36.99	**14034**	3.09	**21.35**
u574	18453	0.50	20064	20064	*	1409.30	19368	3.47	34.05	18747	6.56	44.60	**19691**	1.86	**19.77**
rat575	3387	0.50	20109	20109	*	1426.50	19669	2.19	33.87	19007	5.48	47.82	**19879**	1.14	**18.03**
p654	17322	0.50	24492	24492	*	-	22303	8.94	30.94	**24221**	1.11	284.87	24130	1.48	**18.54**
d657	24456	0.50	24562	24562	*	4053.30	22401	8.80	32.94	21893	10.87	69.62	**23772**	3.22	**21.89**
gr666	147179	0.50	17020	17020	*	-	15561	8.57	**46.77**	15545	8.67	227.61	**16902**	0.69	143.87
u724	20955	0.50	28348	28348	*	5870.60	27072	4.50	58.19	26665	5.94	150.49	**27932**	1.47	**29.26**
rat783	4403	0.50	27566	27566	*	7232.30	**26870**	2.52	80.85	25591	7.16	153.06	26797	2.79	**30.64**
dsj1000	9329844	0.50	30943	.	.	.	30043	2.91	183.75	28822	6.85	781.25	**30943**	*	**79.18**
pr1002	129523	0.50	39449	39449	*	-	37244	5.59	115.38	35808	9.23	485.46	**38762**	1.74	**47.30**
u1060	112047	0.50	36570	.	.	.	35649	2.52	179.48	34873	4.64	689.68	**36570**	*	**75.88**
vm1084	119649	0.50	37653	37653	*	-	36170	3.94	167.56	36121	4.07	813.15	**37508**	0.39	**54.21**
pcb1173	28446	0.50	40069	.	.	.	38284	4.45	301.94	37506	6.40	477.29	**40069**	*	**66.16**
d1291	25401	0.50	38132	30106	21.05	-	36419	4.49	**212.23**	36063	5.43	1288.60	**38132**	*	299.87
rl1304	126474	0.50	41214	40478	1.79	-	37562	8.86	362.55	37859	8.14	1000.74	**41214**	*	**81.11**
rl1323	135100	0.50	46641	44458	4.68	-	43029	7.74	323.11	42990	7.83	904.51	**46641**	*	**93.53**
nrw1379	28319	0.50	43972	.	.	.	42412	3.55	418.50	40170	8.65	870.77	**43972**	*	**124.75**
fl1400	10064	0.50	57226	54792	4.25	-	57131	0.17	**471.99**	55269	3.42	7075.68	**57226**	*	599.81
u1432	76485	0.50	46657	.	.	.	45806	1.82	305.25	45084	3.37	1291.16	**46657**	*	**138.02**
fl1577	11125	0.50	45692	.	.	.	44188	3.29	428.11	44062	3.57	9751.32	**45692**	*	**295.62**
d1655	31064	0.50	58728	51168	12.87	-	55771	5.04	**600.91**	54121	7.84	3487.68	**58728**	*	674.25
vm1748	168278	0.50	70958	68979	2.79	-	67785	4.47	1280.00	68976	2.79	6251.95	**70958**	*	**225.29**
u1817	28601	0.50	63639	52186	18.00	-	60751	4.54	**738.77**	59783	6.06	4171.11	**63639**	*	1302.35
rl1889	158268	0.50	68422	43374	36.61	-	64660	5.50	1260.33	62538	8.60	5535.04	**68422**	*	**244.97**
d2103	40225	0.50	78084	76035	2.62	-	**78084**	*	1585.02	73034	6.47	-	77333	0.96	**1168.90**
u2152	32127	0.50	73400	52091	29.03	-	71469	2.63	**1326.63**	68152	7.15	9579.31	**73400**	*	1619.61
u2319	117128	0.50	79351	79351	*	-	**78319**	1.30	1210.42	76250	3.91	6496.30	78113	1.56	**569.76**
pr2392	189016	0.50	84094	60225	28.38	-	79704	5.22	1496.23	78364	6.81	8624.02	**84094**	*	**422.73**
pcb3038	68847	0.50	104667	96356	7.94	-	100660	3.83	4491.30	97596	6.76	-	**104667**	*	**917.39**
fl3795	14386	0.50	97707	.	.	.	95675	2.08	6867.61	NA	NA	-	**97707**	*	**3158.89**
fnl4461	91283	0.50	164201	.	.	.	158654	3.38	11047.56	NA	NA	-	**164201**	*	**3248.64**
rl5915	282765	0.50	199336	.	.	.	189096	5.14	15139.79	NA	NA	-	**199336**	*	**5593.23**
rl5934	278023	0.50	207385	.	.	.	198428	4.32	16384.22	NA	NA	-	**207385**	*	**5881.87**
pla7397	11630364	0.50	320744	.	.	.	303425	5.40	-	NA	NA	-	**320744**	*	-
average					5.86	13749.78		4.80	2082.26		6.02	4814.36		**0.63**	**1109.93**

Table A.7: Generation 4, $n \leq 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
att48	6909	0.65	1870	1870	*	106.00	**1870**	*	**0.12**	**1870**	*	0.16	**1870**	*	0.52
gr48	4037	0.80	2264	2264	*	22.40	**2264**	*	**0.14**	**2264**	*	0.15	**2264**	*	0.40
hk48	9169	0.80	2177	2177	*	0.20	**2177**	*	0.14	**2177**	*	**0.13**	**2177**	*	0.15
eil51	384	0.90	2490	2490	*	82.10	2481	0.36	**0.13**	2486	0.16	0.14	**2490**	*	0.24
berlin52	4526	0.60	2089	2089	*	115.00	2085	0.19	**0.13**	**2089**	*	0.23	2085	0.19	0.48
brazil58	11428	0.45	2070	2070	*	132.00	**2070**	*	**0.18**	**2070**	*	0.29	2060	0.48	1.08
st70	574	0.85	3316	3316	*	127.70	**3314**	0.06	**0.33**	**3314**	0.06	**0.33**	**3314**	0.06	0.42
eil76	458	0.85	3646	3646	*	45.10	3638	0.22	0.37	3632	0.38	**0.33**	**3646**	*	0.52
pr76	75712	0.70	3361	3361	*	1047.70	3358	0.09	**0.31**	3358	0.09	0.52	**3361**	*	0.62
gr96	52449	0.95	4851	4851	*	212.30	**4851**	*	0.36	**4851**	*	**0.33**	**4851**	*	0.37
rat99	727	0.60	3502	3502	*	16.00	3488	0.40	0.72	3488	0.40	**0.50**	**3502**	*	0.60
kroA100	20218	0.95	4999	4999	*	187.10	**4999**	*	0.54	**4999**	*	0.60	**4999**	*	**0.36**
kroB100	9964	0.45	2935	2935	*	34.40	**2935**	*	**0.53**	**2935**	*	0.62	**2935**	*	0.61
kroC100	7263	0.35	1962	1962	*	261.60	**1962**	*	0.60	1950	0.61	**0.44**	1955	0.36	0.46
kroD100	4259	0.20	1212	1212	*	11.80	**1212**	*	0.30	**1212**	*	**0.23**	**1212**	*	0.41
kroE100	17655	0.80	4635	4635	*	203.40	4631	0.09	**0.54**	**4635**	*	0.82	4616	0.41	0.69
rd100	4747	0.60	3815	3815	*	164.60	**3815**	*	**0.57**	**3815**	*	0.76	3808	0.18	0.75
eil101	409	0.65	4308	4308	*	90.80	4294	0.32	0.70	4300	0.19	**0.59**	**4306**	0.05	0.83
lin105	5033	0.35	2455	2455	*	1020.60	**2455**	*	**0.38**	**2455**	*	1.00	2453	0.08	0.81
pr107	13291	0.30	2072	2072	*	159.00	**2072**	*	**0.35**	**2072**	*	0.92	**2072**	*	1.95
gr120	5901	0.85	5830	5830	*	236.70	5817	0.22	**0.81**	5814	0.27	1.14	**5830**	*	1.25
pr124	17710	0.30	2036	2036	*	163.80	**2036**	*	**0.37**	**2036**	*	0.78	1937	4.86	1.18
bier127	53227	0.45	5068	5068	*	278.40	5045	0.45	**1.19**	5046	0.43	2.28	**5067**	0.02	2.28
pr136	33871	0.35	2860	2860	*	6303.60	2831	1.01	0.87	**2833**	0.94	0.90	2820	1.40	**0.74**
gr137	55883	0.80	6523	6523	*	203.10	6513	0.15	**1.04**	6500	0.35	2.17	**6516**	0.11	2.52
pr144	40976	0.70	5641	5641	*	357.90	5611	0.53	**0.95**	5624	0.30	2.71	**5639**	0.04	4.53
kroA150	19893	0.75	6858	6858	*	415.90	6835	0.34	**1.28**	6828	0.44	2.17	**6855**	0.04	1.69
kroB150	20904	0.80	7023	7023	*	303.00	6987	0.51	1.50	7011	0.17	2.20	**7020**	0.04	**1.16**
pr152	51578	0.70	5823	5823	*	483.60	5201	10.68	**1.30**	5819	0.07	2.64	**5820**	0.05	5.21
u159	14729	0.35	3147	3147	*	1145.20	**3147**	*	1.41	**3147**	*	1.81	**3147**	*	**0.92**
rat195	2207	0.95	9753	9753	*	205.40	9724	0.30	3.73	9630	1.26	2.36	**9750**	0.03	**1.69**
d198	6312	0.40	4661	4661	*	492.70	4589	1.54	**1.91**	4642	0.41	3.40	**4654**	0.15	4.95
kroA200	26432	0.90	9892	9892	*	340.30	9829	0.64	**2.52**	9862	0.30	4.89	**9892**	*	2.73
kroB200	26494	0.90	9849	9849	*	253.20	9796	0.54	2.50	9796	0.54	4.33	**9842**	0.07	**1.62**
gr202	6025	0.15	1071	1071	*	376.10	**1071**	*	0.63	**1071**	*	**0.47**	995	7.10	1.47
ts225	113979	0.90	11002	11002	*	3524.60	10827	1.59	2.74	10885	1.06	3.90	**11002**	*	**1.87**
tsp225	3525	0.90	10972	10972	*	706.70	10952	0.18	4.70	10912	0.55	4.40	**10972**	*	**2.52**
pr226	32148	0.40	4893	4893	*	1183.10	4868	0.51	**2.16**	4879	0.29	5.81	**4890**	0.06	4.83
gr229	121142	0.90	11482	11482	*	563.10	11451	0.27	4.79	11430	0.45	**4.02**	**11482**	*	6.46
gil262	357	0.15	2031	2031	*	1770.50	2029	0.10	1.88	**2031**	*	2.88	2030	0.05	**1.35**
pr264	34395	0.70	10253	10253	*	277.50	9808	4.34	**4.13**	9858	3.85	13.97	**10166**	0.85	6.42
a280	1935	0.75	12064	12064	*	351.80	11810	2.11	5.83	11731	2.76	7.12	**12048**	0.13	**3.39**
pr299	45782	0.95	14986	14986	*	7771.90	14894	0.61	6.71	14898	0.59	12.67	**14980**	0.04	**3.46**
lin318	35725	0.85	15132	15132	*	-	15049	0.55	**7.52**	15012	0.79	15.57	**15119**	0.09	7.91
rd400	14517	0.95	20107	20107	*	5093.10	20004	0.51	12.42	19973	0.67	19.62	**20101**	0.03	**9.61**
average					*	1218.69		0.65	**1.83**		0.41	2.96		**0.38**	2.09

Table A.8: Generation 4, $n > 400$

				Branch-&-Cut			2-Parameter IA			GRASP with PR			EA4OP		
instance	d0	α	opt	best	gap	time	best	gap	time	best	gap	time	best	gap	time
fl417	10082	0.85	20496	20496	*	-	20438	0.28	**11.82**	20366	0.63	82.41	**20494**	0.01	39.61
gr431	51425	0.30	13976	13976	*	-	13652	2.32	**18.13**	13499	3.41	165.40	**13969**	0.05	50.29
pr439	75052	0.70	19613	19613	*	3936.10	19435	0.91	23.59	19139	2.42	177.81	**19510**	0.53	**13.61**
pcb442	10156	0.20	5839	5839	*	-	**5749**	1.54	9.47	5600	4.09	9.50	5650	3.24	**3.40**
d493	24502	0.70	21740	21740	*	-	21357	1.76	28.72	20856	4.07	74.96	**21674**	0.30	**21.00**
att532	26302	0.95	26728	26728	*	-	26570	0.59	19.12	26526	0.76	73.04	**26728**	*	**17.20**
ali535	40468	0.20	13520	13520	*	15739.60	13393	0.94	**29.43**	13102	3.09	406.85	**13442**	0.58	73.07
pa561	2487	0.90	27719	27712	0.03	-	27228	1.77	31.85	26822	3.24	40.43	**27719**	*	**24.14**
u574	35060	0.95	28823	28823	*	-	28545	0.96	**24.81**	28446	1.31	70.59	**28822**	0.00	26.03
rat575	6096	0.90	28364	28364	*	-	27995	1.30	43.23	27664	2.47	39.55	**28334**	0.11	**24.68**
p654	27715	0.80	31814	31814	*	-	31657	0.49	**40.11**	31383	1.35	514.38	**31717**	0.30	123.82
d657	44021	0.90	32548	32548	*	13485.10	32059	1.50	52.01	31927	1.91	123.96	**32534**	0.04	**33.00**
gr666	103026	0.35	21013	21013	*	-	20369	3.06	**39.13**	20412	2.86	511.53	**20901**	0.53	132.65
u724	35624	0.85	34988	34988	*	-	34169	2.34	57.31	33923	3.04	108.86	**34921**	0.19	**40.93**
rat783	1321	0.15	7829	7829	*	-	7459	4.73	23.26	7203	8.00	32.23	**7548**	3.59	**13.35**
dsj1000	6530891	0.35	27357	27357	*	-	24840	9.20	127.16	25113	8.20	573.23	**25352**	7.33	**48.13**
pr1002	90666	0.35	23527	23527	*	-	21029	10.62	180.07	20224	14.04	127.34	**22482**	4.44	**35.67**
u1060	190480	0.85	51775	51768	0.01	-	50921	1.65	190.31	50302	2.85	430.51	**51775**	*	**150.58**
vm1084	107684	0.45	38678	38678	*	-	36455	5.75	201.55	35535	8.13	551.57	**38228**	1.16	**50.34**
pcb1173	48359	0.85	56010	55954	0.10	-	53687	4.15	206.44	52559	6.16	469.21	**56010**	*	**77.73**
d1291	5081	0.10	4029	4029	*	2335.60	**4029**	*	**35.22**	**4029**	*	71.53	4024	0.12	45.07
rl1304	189711	0.75	57782	57782	*	-	53775	6.93	470.54	51258	11.29	1163.85	**57545**	0.41	**112.18**
rl1323	243180	0.90	65664	65476	0.29	-	63666	3.04	492.15	61632	6.14	1070.12	**65664**	*	**99.81**
nrw1379	53807	0.95	69214	69119	0.14	-	68510	1.02	385.51	68046	1.69	974.63	**69214**	*	**152.00**
fl1400	18115	0.90	70488	70476	0.02	-	70444	0.06	**177.57**	70449	0.06	4573.17	**70488**	*	287.75
u1432	91783	0.60	54540	54540	*	-	49738	8.80	565.02	50150	8.05	910.86	**53550**	1.82	**127.79**
fl1577	7788	0.35	33754	22191	34.26	-	25157	25.47	327.70	31740	5.97	10432.87	**33754**	*	**200.71**
d1655	21745	0.35	31880	29920	6.15	-	30024	5.82	**300.33**	30040	5.77	781.57	**31880**	*	371.31
vm1748	252417	0.75	82126	81778	0.42	-	80623	1.83	776.22	79540	3.15	5491.77	**82126**	*	**265.55**
u1817	20021	0.35	36416	31800	12.68	-	33428	8.21	845.38	32142	11.74	891.10	**36416**	*	**418.80**
rl1889	237402	0.75	83081	71527	13.91	-	80240	3.42	1536.30	76078	8.43	4799.95	**83081**	*	**363.35**
d2103	24136	0.30	34192	31045	9.20	-	29811	12.81	805.24	31176	8.82	1291.98	**34192**	*	**465.36**
u2152	28914	0.45	54744	48472	11.46	-	50467	7.81	1587.17	49688	9.24	3227.28	**54744**	*	**906.84**
u2319	187405	0.80	110995	110995	*	-	108463	2.28	1218.39	107764	2.91	4687.71	**110960**	0.03	**438.26**
pr2392	132312	0.35	50902	45407	10.80	-	47791	6.11	1355.53	45506	10.60	3173.63	**50902**	*	**285.26**
pcb3038	75732	0.55	101173	91831	9.23	-	93070	8.01	2974.79	94607	6.49	-	**101173**	*	**800.13**
fl3795	11509	0.40	80069	71328	10.92	-	67850	15.26	7074.26	NA	NA	-	**80069**	*	**4496.09**
fnl4461	54770	0.30	85088	84098	1.16	-	79110	7.03	5216.83	NA	NA	-	**85088**	*	**1490.80**
rl5915	480701	0.85	279277	279116	0.06	-	264269	5.37	-	NA	NA	-	**279277**	*	**8438.60**
rl5934	222418	0.40	137838	.	.	.	128287	6.93	-	NA	NA	-	**137838**	*	**4037.07**
pla7397	5815182	0.25	142399	106131	25.47	-	121860	14.42	-	NA	NA	-	**142399**	*	**6667.36**
average					3.66	17087.41		5.04	1987.85		5.07	3807.94		**0.60**	**767.54**

References

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[2] C. Archetti, A. Corberán, I. Plana, J. M. Sanchis, and M. G. Speranza. A branch-and-cut algorithm for the orienteering arc routing problem. *Computers & Operations Research*, 66(C):95–104, 2016.

[3] C. Archetti, A. Hertz, and M. Grazia Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2007.

[4] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[5] H. Bouly, D.-C. Dang, and A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR-A Quarterly Journal of Operations Research*, 8(1):49–70, 2010.

[6] V. Campos, R. Martí, J. Sánchez-Oro, and A. Duarte. Grasp with path relinking for the orienteering problem. *Journal of the Operational Research Society*, 65(12):1800–1813, 2014.

[7] I. Chao, B. L. Golden, and E. A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475 – 489, 1996.

[8] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[9] J. Ferreira, A. Quintas, J. A. Oliveira, G. A. B. Pereira, and L. Dias. *Solving the Team Orienteering Problem: Developing a Solution Tool Using a Genetic Algorithm Approach*, pages 365–375. Springer International Publishing, Cham, 2014.

[10] M. Fischetti, J. J. Salazar-González, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.

[11] S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044 – 2064, 2010. Special Issue on Intelligent Distributed Information Systems.

[12] M. Gendreau, G. Laporte, and F. Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2):539–545, 1998.

[13] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[14] A. Gunawan, H. C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332, 2016.

[15] D. S. Johnson. *Local optimization and the Traveling Salesman Problem*, pages 446–461. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[16] Y.-C. Liang, S. Kulturel-Konak, and A. Smith. Meta heuristics for the orienteering problem. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress*, pages 384–389. IEEE, 2002.

[17] Y. Marinakis, M. Politis, M. Marinaki, and N. Matsatsinis. *A Memetic-GRASP Algorithm for the Solution of the Orienteering Problem*, pages 105–116. Springer International Publishing, Cham, 2015.

[18] K. Ostrowski, J. Karbowska-Chilinska, J. Koszelew, and P. Zabielski. Evolution-inspired local improvement algorithm solving orienteering problem. *Annals of Operations Research*, 253(1):519–543, 2017.

[19] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[20] J. Silberholz and B. Golden. The effective application of a new approach to the generalized orienteering problem. *Journal of Heuristics*, 16(3):393–415, 2010.

[21] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. V. Berghe, and D. V. Oudheusden. A personalized tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence archive*, 22:964–985, 2008.

[22] H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32(6):1379–1407, 2005.

[23] M. F. Tasgetiren. A genetic algorithm with an adaptive penalty function for the orienteering problem. *Journal of Economic and Social Research*, 4(2):1–26, 2001.

[24] F. Tricoire, M. Romauch, K. F. Doerner, and R. F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010.

[25] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35:797–809, 1984.

[26] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[27] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009.

[28] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *OR Insight*, 20(3):21–27, 2007.

[29] Q. Wang, X. Sun, B. L. Golden, and J. Jia. Using artificial neural networks to solve the orienteering problem. *Annals of Operations Research*, 61(1):111–120, 1995.

[30] X. Wang, B. L. Golden, and E. A. Wasil. *Using a Genetic Algorithm to Solve the Generalized Orienteering Problem*, pages 263–274. Springer US, Boston, MA, 2008.

[31] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[32] D. L. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and travelling salesman: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 133–140, San Mateo, CA, 1989. Morgan Kaufmann.

[33] M. Yuen, I. King, and K. Leung. A survey of crowdsourcing systems. In *In IEEE Socialcom*, 2011.