Document downloaded from:

http://hdl.handle.net/10251/36442

This paper must be cited as:

Yuichi Nagata; Soler Fernández, D. (2012). A new genetic algorithm for the asymmetric traveling salesman problem. Expert Systems with Applications. 39(10):8947-8953. doi:10.1016/j.eswa.2012.02.029.

The final publication is available at

http://dx.doi.org/10.1016/j.eswa.2012.02.029

Copyright Elsevier

A new genetic algorithm for the asymmetric traveling salesman problem

Yuichi Nagata[a,*], David Soler[b]

[a] *Tokyo Institute of Technology, Kanagawa, Japan*

[b] *Institut Universitari de Matemàtica Pura i Aplicada,*

Universitat Politècnica de València, València, Spain

Abstract

The asymmetric traveling salesman problem (ATSP) is one of the most important combinatorial optimization problems. It allows us to solve, either directly or through a transformation, many real-world problems. We present in this paper a new competitive genetic algorithm to solve this problem. This algorithm has been checked on a set of 153 benchmark instances with known optimal solution and it outperforms the results obtained with previous ATSP heuristic methods.

Keywords: Asymmetric traveling salesman problem, genetic algorithm, crossover operator, metaheuristics

1 Introduction

The asymmetric traveling salesman problem (ATSP) is one of the most important and well-known problems in combinatorial optimization, and it can be stated as follows:

*Correspondence: Y. Nagata, Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, 4259 Nagatsuta Midori-ku Yokohama, Kanagawa 226-8502, Japan. E-mail: nagata@fe.dis.titech.ac.jp.

Given a complete directed graph $G = (V, A)$, V being the vertex set and A being the arc set, with nonnegative costs associated with its arcs, find a minimum cost circuit in G passing through each vertex exactly once.

The ATSP has important applications to real-world problems, especially in sequencing, in distribution, and in vehicle routing problems. As a result, dozens of papers and several surveys have been written about ATSP in the past few decades. To be brief, we only mention the classical survey by Fischetti *et al.* (2002), detailing different exact procedures for the ATSP; the most recent exact procedure by Corberan *et al.* (2005), capable of solving large-size ATSP instances; the latest competitive heuristic approach for solving the ATSP by Xing *et al.* (2008), whose article also serves as a good survey on metaheuristic approaches for solving the ATSP; the work by Oncan *et al.* (2009), in which different formulations for the ATSP are compared; and the very recent concise guide by Laporte (2010) and survey on local search algorithms for this problem by Rego *et al.* (2011).

In addition to its direct applications to real-world problems, a review of the literature shows that many single-vehicle routing problems, which actually cannot be modeled as an ATSP, can be solved through a polynomial transformation into an ATSP. The main advantage of this transformation is that efficient algorithms developed for the ATSP can be directly applied to these problems without any modification.

Among the problems that can be solved through a transformation into an ATSP are several classes of arc and/or node routing problems, defined on undirected, directed, or mixed graphs, some of which include turn penalties, forbidden turns, time-dependent costs, or delivery time windows. In this area, we cite the works by Laporte (1997), Clossey *et al.* (2001), Corberán *et al.* (2002), Blais and Laporte (2003), Soler *et al.* (2008), and Albiach *et al.* (2008). In most of these articles, the transformations they propose consist of two steps. They first transform their problem into another combinatorial optimization problem, namely the asymmetric generalized traveling salesman problem (AGTSP), which in turn can be transformed into an ATSP (see, e.g., Noon and Bean (1993) or Ben-Arieh *et al.* (2003)). The AGTSP is actually a generalization of the ATSP in which each customer has several alternative locations and only one of them has to be selected for service. It can be briefly defined as follows:

Given a directed graph $G = (V, A)$ with nonnegative costs associated with its arcs, such that V is partitioned into k nonempty subsets $\{S_i\}_{i=1}^k$, find a minimum cost circuit passing through exactly one vertex of each subset S_i $\forall i \in \{1, \ldots, k\}$.

It is worth commenting here that the idea of solving a single-vehicle routing problem by transforming it into an ATSP through an AGTSP has been recently generalized to solve different hard multivehicle routing problems, as we can see through the papers by Soler *et al.* (2009), Baldacci *et al.* (2010), Bräysy *et al.* (2011), and Micó and Soler (2011).

All these facts have encouraged us to present a powerful new approximation algorithm to solve the ATSP, with the aim of improving results given by previous ATSP heuristics to help experts solve real-world problems (such as those cited above) or to measure the efficiency of actual or future proposed heuristics that directly address these problems. Our solution approach is based on genetic algorithms (GAs). The GA is a population-based approach for heuristic search in optimization problems, and it has been applied to a wide variety of optimization problems. In particular, many GAs or memetic algorithms (MAs) have been developed for the symmetric TSP (STSP) up to the present date (see, e.g., the very recent papers by Albayrak and Allahverdi (2011) and Chen and Chien (2011)). Here, the MA is also a population-based heuristic search that combines a GA with a local search to organize a more intensive local search. However, applications of the GA or MA to the ATSP are rather limited. Here we refer the reader to the three latest GAs and MAs proposed for the ATSP, all of which have shown good performance (see Choi *et al.* (2003), Buriol *et al.* (2004), and Xing *et al.* (2008)).

The main feature of our GA is the use of an edge assembly crossover (EAX) operator, which was originally proposed by Nagata and Kobayashi (1997) for both the STSP and the ATSP. The EAX generates offspring solutions by combining (undirected) edges or arcs from two parent solutions and adding relatively few short edges or arcs. The main difference between the EAX for the ATSP and the EAX for the STSP is that the first one combines parents' arcs (directed edges) without changing their orientation whereas the second one combines parents' (undirected) edges without respect to their orientation. Since a GA using an EAX was originally proposed, it has been significantly enhanced to improve the performance for solving the STSP; the capability of the EAX for generating good offspring solutions has been enhanced by Nagata (2004, 2006b) and the capability of the GA framework for maintaining the population diversity has been also enhanced by Nagata (2006a). Because of the growing importance of the ATSP, in this paper we investigate the performance of the GA using the EAX for the ATSP by applying similar enhancements to those developed for the STSP and introducing an additional enhancement to further improve the performance.

To check the efficiency of our GA, we have tested it on 153 benchmark ATSP instances, all of which have known optimal solutions. These instances correspond to two different sets: the 27 well-known instances from TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/) and 126 instances with up to 315 vertices from the work by Soler *et al.* (2008).

The rest of the paper is organized as follows. Section 2 presents the genetic algorithm for the ATSP, Section 3 shows the computational results obtained for the set of 153 instances, and Section 4 gives some conclusions about this work.

2 The genetic algorithm

In this section, we describe the GA framework and the EAX developed for the ATSP. In Section 2.1 we first present the EAX, which includes adapted enhancements developed for the STSP. The GA framework in which the EAX is integrated is then presented in Section 2.2. We present in Section 2.3 the local search procedure used for generating an initial population for the proposed GA.

2.1 EAX for the ATSP

The EAX for the ATSP consists of five steps as outlined below and illustrated in Figure 1. If more than two offspring solutions are generated, Steps 3–5 are repeated.

[Basic algorithm]

Step 1 Given parent solutions denoted as p_A and p_B, let G_{AB} be the directed graph defined as $G_{AB} = (V, E_A \cup E_B \backslash E_A \cap E_B)$, where E_A and E_B are defined as sets of arcs consisting of p_A and p_B, respectively.

Step 2 Partition all edges of G_{AB} into *AB-cycles*. An AB-cycle is defined as a cycle in G_{AB} such that arcs from E_A and E_B are alternately linked in the opposite orientation. Note that partition of the arcs into *AB-cycles* is always possible and uniquely determined (which is easy to prove).

Step 3 Construct an *E-set* by selecting *AB-cycles* according to a given rule, where an *E-set* is defined as the union of *AB-cycles*.

Step 4 Generate an intermediate solution from p_A by removing the arcs of E_A and adding the arcs of E_B in the *E-set*. As a result, an intermediate solution consists of one or more subtours as illustrated in Figure 1.

Step 5 Connect subtours into a tour to generate an offspring solution. More precisely, subtours are connected one by one, each time connecting a subtour consisting of the least number of arcs to one of the other subtours so that the sum of the cost of the arcs is minimized. Here, two subtours are connected by deleting one arc from each of the subtours and adding two arcs to connect them in such a way that all arcs in the resulting subtour have the same orientation.

In Step 3, an *E-set* of any combination of *AB-cycles* can be constructed, and the EAX can generate various intermediate solutions. One simple strategy, which was proposed in the original EAX (Nagata and Kobayashi, 1997), is to select a number of *AB-cycles* randomly, each with a probability of 0.5, to construct an *E-set*. An EAX with this selection strategy is called EAX-Rand and it typically forms an intermediate solution that contains arcs of E_A and arcs of E_B equally. An alternative strategy for selecting *AB-cycles* was proposed by Nagata (2004a) to enhance the EAX for the STSP. This strategy randomly selects a single *AB-cycle* without overlapping the previous selection. Therefore, an EAX with this strategy can generate at most the same number of offspring solutions as the number of *AB-cycles* generated. An EAX with this selection strategy is called EAX-1AB and typically forms an intermediate solution similar to p_A because it is generated from p_A by replacing a relatively small number of arcs with the same number of arcs of p_B. The advantage of EAX-1AB is its ability to better maintain the population diversity when used in an appropriate GA framework, which is presented in Algorithm 1 (see Section 2.2), resulting in a significant improvement in the solution quality. Another selection strategy (called EAX-Block) for selecting *AB-cycles* was also proposed by Nagata (2006b), but we do not use this variant of the EAX in this paper, because we could not find a significant improvement by using this strategy in our preliminary experiments.

2.2 Main framework

The main GA framework is shown in Algorithm 1. The search is initiated by generating N_{pop} initial solutions (line 1). Here, we use a local search procedure using a variant of the 3-opt neighborhood to generate each of the initial population members. The detailed description of the local search procedure is presented in Section 2.3.

For each generation of the GA (lines 3–15), each population member is selected once, as both parent p_A and parent p_B in random order (lines 3 and 5). For each pair of parents, EAX-1AB generates offspring solutions, where n_{ch} refers to the number of offspring solutions generated (line 6). Let n_{ch}^{max} be a parameter that specifies the maximum number of offspring solutions for each pair of parents. Therefore, n_{ch} will be equal to the number of *AB-cycles* if the number of *AB-cycles* is less than n_{ch}^{max}. Then, the best solution among the generated offspring solutions, denoted as c_{best}, is selected according to a given evaluation function (line 7). If the tour cost of c_{best} is better than that of p_A (line 8), it will replace one of the population members selected as p_A or p_B (line 10 and 12). If c_{best} is more similar to p_A than to p_B, the population member selected as p_A is replaced (lines 9 and 10), where $\text{dist}(x, y)$ returns the number of different arcs between the two tours. In contrast, if c_{best} is more similar to p_B than to p_A and the tour cost of c_{best} is better than those of p_B and p_A, the population member selected as p_B is replaced (lines 11 and 12). Iterations of the generation are repeated until a termination condition is met (line 16). Finally, the best solution in the population is returned (line 17).

In the latest version of the GA using the EAX for the STSP by Nagata (2006b), the best offspring solution (c_{best}) replaces only parent p_A rather than both parents (i.e., lines

Algorithm 1 : Procedure GA()

1: $\{x_1, \ldots, x_{N_{pop}}\}$:= Generate_Initial_Population();
2: **repeat**
3: Let $r(\cdot)$ be a random permutation of $1, \ldots, N_{pop}$;
4: **for** $i := 1$ to N_{pop} **do**
5: $p_A := x_{r(i)}$, $p_B := x_{r(i+1)}$; $(r(N_{pop}+1) = r(1))$
6: $\{c_1, \ldots, c_{n_{ch}}\}$:= Crossover(p_A, p_B);
7: c_{best} := Select_Best($c_1, \ldots, c_{n_{ch}}$);
8: **if** $\text{cost}(c_{best}) < \text{cost}(p_A)$ **then**
9: **if** $\text{dist}(c_{best}, p_A) < \text{dist}(c_{best}, p_B)$ **then**
10: $x_{r(i)} := c_{best}$;
11: **else if** $\text{dist}(c_{best}, p_A) > \text{dist}(c_{best}, p_B)$ and $\text{cost}(c_{best}) < \text{cost}(p_B)$ **then**
12: $x_{r(i+1)} := c_{best}$;
13: **end if**
14: **end if**
15: **end for**
16: **until** a termination condition is satisfied
17: **return** the best individual in the population;

10, 11, and 13 are ignored). This selection strategy was introduced to better maintain the population diversity because EAX-1AB frequently generates offspring solutions that are more similar to p_A than to p_B. We call this selection strategy "SEL1" in this paper. EAX-1AB adapted to the ATSP has the same property, although not to the same extent as EAX-1AB for the STSP. If c_{best} is more similar to p_B than to p_A and c_{best} replaces p_A, the population diversity will be fairly reduced because two similar solutions remain in the population. Therefore, we add the additional selection mechanism that replaces p_B rather than p_A if c_{best} is more similar to p_B than to p_A. We call this selection strategy "SEL2" in this paper.

Although the most straightforward evaluation function for evaluating offspring solutions would be the tour cost, an alternative evaluation function was used in the latest version of the GA for the STSP to maintain the population diversity in a positive manner. The use of this evaluation function, however, did not improve the performance significantly in our preliminary experiments on ATSP instances. Therefore, we evaluate offspring solutions simply by the tour cost.

2.3 Local search

The local search procedure used for generating the initial population is a simple hill-climbing method using a variant of the 3-opt neighborhood. For information on classical local search algorithms for the ATSP, we refer the reader to the work by Kanellakis and Papadimitriou (1980). The initial population consisting of N_{pop} tours is generated by performing the local search procedure N_{pop} times.

The 3-opt neighborhood used in this paper is defined as the tours that can be obtained from the current tour by replacing three arcs in other possible ways so that all arcs in the resulting tour have the same orientation. More precisely, let (v_1, v_2), (v_3, v_4), and (v_5, v_6) denote three different arcs to be removed, and assume that v_3 appears after v_1 and before v_5 in the current tour. The three arcs to be added must be (v_1, v_4), (v_3, v_6), and (v_5, v_2) to make the arcs in the resulting tour have the same orientation. To speed up the local search procedure, we use a variant of the "don't look bits" strategy (Bentley, 1990). Let $\mathcal{N}(v)$ be a subset of the 3-opt neighborhood that requires $v = v_1$, and we call it a subneighborhood. To reduce the neighborhood size of a subneighborhood, v_4 is restricted to the vertices that are at most within the ten nearest from v_1. When using the don't look bits strategy, each of the vertices is assigned a flag having value *true* or *false*.

The local search procedure begins by generating a random tour and setting all flags to *false*. At each iteration, a tour that improves the current tour is searched in subneighborhoods whose associated flags are *false*. If an improving tour is found, the current tour is immediately moved to the new tour, and the flags of the three starting points of the added three edges are set to *false*. However, a flag of a vertex is set to *true* whenever no improving move is found in the corresponding subneighborhood. Iterations are repeated until all flags become *true*.

3 Computational experiments

In this section, we first perform the GA with several different configurations on the well-known 27 instances included in TSPLIB to detect the best one. The results obtained with the best configuration are then compared for this set of 27 instances with those of the latest genetic algorithms proposed for the ATSP. Finally, we also run our GA on a set of 126 instances created in the work by Soler *et al.* (2008).

The GA was implemented in C++ and was executed in a virtual machine environment (i.e., each job is executed on a single core, but multiple jobs may be executed in the same node) on a cluster with Intel Xeon 2.93-GHz nodes.

3.1 Analysis of algorithm components

We performed the proposed GA with four different configurations on the 27 instances in TSPLIB to detect the best configuration. Two parameters of Algorithm 1 are set as follows: $N_{pop} = 100$ and $n_{ch}^{max} = 30$. Each run is terminated when the best solution is not improved over the last 20 generations. Four different configurations of the GA are summarized below.

SEL1+EAX-Rand Selection strategy SEL1 is used in Algorithm 1. EAX-Rand is used as the crossover operator. For each pair of parents, the number of offspring solutions is set to the number of *AB-cycles* if the number of *AB-cycles* generated is less than n_{ch}^{max}. We use this restriction to compare EAX-Rand with EAX-1AB under the same condition.

SEL2+EAX-Rand Selection strategy SEL2 is used in Algorithm 1. EAX-Rand is used as the crossover operator. The number of offspring solutions is restricted as in the case of SEL1+EAX-Rand.

SEL1+EAX-1AB Selection strategy SEL1 is used in Algorithm 1. EAX-1AB is used as the crossover operator.

SEL2+EAX-1AB Selection strategy SEL2 is used in Algorithm 1. EAX-1AB is used as the crossover operator.

We performed the GA 100 times on each instance with each configuration. Table 1 shows results. The first column lists the instance names, where the number of vertices is indicated by the name except for kro124p (the number of vertices being 100 in this instance). For each of the GAs with four different configurations, we list the number of runs that succeeded in finding the optimal solution over 100 runs (Suc.), the average percentage excess with respect to the optimal solutions (a-err), and the average computation time per single run in seconds (a-T). Average results over all instances are also listed at the bottom of the table.

First, we compare the results of the GAs with two configurations, SEL1+EAX-Rand and SEL1+EAX-1AB. The table shows that the solution quality of the GA with EAX-1AB is better than that of the GA with EAX-Rand when selection strategy SEL1 is used. A major factor in this difference is that the population diversity is better maintained by replacing a population member (selected as p_A) with a relatively similar offspring solution and EAX-1AB usually generates offspring solutions similar to p_A. Next, we focus on the difference between selection strategies SEL1 and SEL2. The table shows that the use of selection strategy SEL2 improves the solution quality regardless of the crossover type. The main reason for this improvement is also that the population diversity is better maintained by replacing a population member with a relatively similar offspring solution and selection strategy SEL2 enhances this property. Here, one should note that SEL2+EAX-Rand is comparable to SEL2+EAX-1AB with respect to the solution quality, but SEL2+EAX-Rand requires more computation time than SEL2+EAX-1AB. This is because EAX-1AB can generate offspring solutions more efficiently than EAX-Rand, which is another advantage of EAX-1AB. By comparing the results listed in the table, we can see that SEL2-EAX-1AB seems to be a good configuration for the GA presented in Algorithm 1.

3.2 Comparison with other algorithms

We compare the results of our GA with configuration SEL2+EAX-1AB with those of the latest GAs or MAs proposed in the literature to solve the ATSP: the heuristics proposed by Choi *et al.* (2003), Buriol *et al.* (2004), and Xing *et al.* (2008). We make comparisons for the 27 well-known instances in TSPLIB. Table 2 lists the results; the algorithms are denoted as the reference on the first line of the table; this is followed by the type of method, the computer specifications, and the number of runs. For each

Table 1: Results of the GAs with four different configurations.

	SEL1+EAX-Rand			SEL2+EAX-Rand			SEL1+EAX-1AB			SEL2+EAX-1AB		
Instance	Suc.	a-err	a-T	Suc.	a-err	a-T	Suc.	a-err	a-T	Suc.	a-err	a-T
br17	100	0.0000	0.00	100	0.0000	0.00	100	0.0000	0.00	100	0.0000	0.00
p43	92	0.0014	0.01	92	0.0014	0.04	91	0.0016	0.03	94	0.0011	0.02
ry48p	100	0.0000	0.01	100	0.0000	0.04	100	0.0000	0.02	100	0.0000	0.02
ft53	75	0.0275	0.02	93	0.0101	0.05	64	0.0408	0.02	95	0.0064	0.03
ftv33	100	0.0000	0.01	100	0.0000	0.02	100	0.0000	0.01	100	0.0000	0.01
ftv35	97	0.0041	0.01	100	0.0000	0.03	99	0.0014	0.01	100	0.0000	0.02
ftv38	92	0.0105	0.01	100	0.0000	0.03	99	0.0013	0.01	100	0.0000	0.02
ftv44	49	0.3162	0.02	62	0.2356	0.03	56	0.2728	0.02	66	0.2108	0.02
ftv47	100	0.0000	0.02	100	0.0000	0.03	100	0.0000	0.02	100	0.0000	0.02
ftv55	100	0.0000	0.02	100	0.0000	0.05	100	0.0000	0.03	100	0.0000	0.03
ftv64	100	0.0000	0.03	100	0.0000	0.05	100	0.0000	0.03	100	0.0000	0.04
ft70	67	0.0017	0.04	88	0.0006	0.08	69	0.0016	0.04	85	0.0008	0.04
ftv70	100	0.0000	0.03	100	0.0000	0.07	100	0.0000	0.04	100	0.0000	0.04
ftv90	75	0.0399	0.05	91	0.0114	0.10	85	0.0222	0.05	96	0.0057	0.05
ftv100	79	0.0296	0.07	90	0.0112	0.11	87	0.0162	0.08	97	0.0034	0.06
kro124p	93	0.0039	0.05	96	0.0012	0.16	96	0.0012	0.07	97	0.0009	0.08
ftv110	86	0.0179	0.09	87	0.0148	0.15	94	0.0072	0.10	93	0.0072	0.08
ftv120	79	0.0263	0.11	93	0.0069	0.22	92	0.0088	0.12	87	0.0157	0.10
ftv130	79	0.0243	0.08	87	0.0130	0.20	88	0.0130	0.13	89	0.0104	0.11
ftv140	75	0.0285	0.11	86	0.0124	0.18	84	0.0202	0.15	89	0.0116	0.12
ftv150	82	0.0146	0.11	88	0.0100	0.32	93	0.0065	0.18	91	0.0073	0.15
ftv160	92	0.0101	0.15	97	0.0041	0.26	96	0.0056	0.18	99	0.0011	0.17
ftv170	96	0.0044	0.16	96	0.0044	0.43	98	0.0022	0.24	98	0.0022	0.19
rbg323	100	0.0000	1.16	100	0.0000	3.59	100	0.0000	1.44	100	0.0000	0.96
rbg358	100	0.0000	1.65	100	0.0000	4.22	100	0.0000	1.82	100	0.0000	1.32
rbg403	100	0.0000	1.82	100	0.0000	4.68	100	0.0000	1.86	100	0.0000	1.61
rbg443	100	0.0000	1.76	100	0.0000	4.12	100	0.0000	1.77	100	0.0000	1.70
Average	89.19	0.021	0.28	94.30	0.012	0.71	92.26	0.016	0.31	95.41	0.011	0.26

algorithm, we list the number of runs that succeeded in finding the optimal solution (Suc.) if presented in the literature, the average percentage excess with respect to the optimal solutions (a-err), and the average computation time per single run in seconds (a-T). Results for the blank cells are not presented in the literature.

Table 2 shows that our GA is superior to all three compared algorithms in terms of average solution quality; only the procedure by Xing *et al.* seems to find similar solutions to ours (but they present results for only 16 out of the 27 instances), but their average results are slightly worse than ours. For the computation time, it is very difficult to make a fair comparison because the algorithms in the table were executed using different computers (and languages). Nonetheless, our GA would not be considerably slower than the compared heuristics even if the difference in computer speed is

Table 2: Comparisons with other algorithms.

Author	Choi at al. (2003)		Buriol *et al.* (2004)			Xing *et al.* (2008)		Our method $N_{pop} = 100$			Our method $N_{pop} = 300$		
Method	GA		MA			MA		GA			GA		
Computer	Pentium 550 MHz		Pentium 1.7 GHz			Pentium 1.6 GHz		Xeon 2.93 GHz			Xeon 2.93 GHz		
# of Runs	3		20			100		100			100		
Instance	a-err	a-T	Suc.	a-err	a-T	a-err	a- T	Suc.	a-err	a-T	Suc.	a-err	a-T
br17	0.00	0	20	0.00	0.05	0.000	1.9	100	0.000	0.00	100	0.0000	0.00
ftv33	0.00	4	20	0.00	0.05			100	0.000	0.01	100	0.0000	0.06
ftv35	0.14	1	20	0.00	0.08	0.000	15.6	100	0.000	0.02	100	0.0000	0.05
ftv38	0.13	2	5	0.10	0.26			100	0.000	0.02	100	0.0000	0.08
p43	0.00	1	11	0.01	0.35	0.000	4.2	94	0.001	0.02	100	0.0000	0.10
ftv44	1.30	5	7	0.44	0.36			66	0.211	0.02	94	0.0372	0.07
ftv47	0.00	3	20	0.00	0.13	0.000	90.3	100	0.000	0.02	100	0.0000	0.07
ry48p	0.33	5	17	0.03	0.32	0.001	53.4	100	0.000	0.02	100	0.0000	0.07
ft53	0.00	8	20	0.00	0.2	0.000	41.2	95	0.006	0.03	100	0.0000	0.09
ftv55	0.00	5	20	0.00	0.16	0.000	125.6	100	0.000	0.03	100	0.0000	0.10
ftv64	0.00	7	20	0.00	0.24	0.000	101.2	100	0.000	0.04	100	0.0000	0.17
ft70	0.21	20	8	0.03	0.86	0.000	83.6	85	0.001	0.04	100	0.0000	0.14
ftv70	0.00	12	19	0.01	0.38	0.000	43.8	100	0.000	0.04	100	0.0000	0.19
ftv90	0.00	20	20	0.00	0.28			96	0.006	0.05	100	0.0000	0.15
ftv100	0.00	62	20	0.00	0.40			97	0.003	0.06	100	0.0000	0.29
kro124p	0.52	67	18	0.01	0.74	0.001	28.9	97	0.001	0.08	100	0.0000	0.33
ftv110	0.54	61	18	0.02	0.98			93	0.007	0.08	100	0.0000	0.36
ftv120	0.29	89	7	0.14	2.00			87	0.016	0.10	100	0.0000	0.37
ftv130	0.59	97	18	0.01	1.30			89	0.010	0.11	100	0.0000	0.42
ftv140	0.32	81	14	0.08	2.11			89	0.012	0.12	100	0.0000	0.53
ftv150	0.55	88	18	0.01	1.54			91	0.007	0.15	100	0.0000	0.56
ftv160	0.12	101	16	0.02	2.04			99	0.001	0.17	100	0.0000	0.65
ftv170	0.22	94	15	0.05	2.78	0.020	68.3	98	0.002	0.19	100	0.0000	0.69
rbg323	0.00	2	20	0.00	0.07	0.000	110.4	100	0.000	0.96	100	0.0000	4.25
rbg358	0.00	3	20	0.00	0.08	0.000	58.4	100	0.000	1.32	100	0.0000	5.63
rbg403	0.00	2	20	0.00	0.08	0.000	33.1	100	0.000	1.61	100	0.0000	6.63
rbg443	0.00	4	20	0.00	0.09	0.000	144.2	100	0.000	1.70	100	0.0000	6.74

considered, and it seems that the MA by Xing *et al.* would require longer computation time. We additionally performed our GA with a population size of 300 to improve the solution quality. Our GA with a population size of 300 found optimal solutions in all 100 runs for 26 out of the 27 TSPLIB instances (94 on ftv44). The computation time was increased by about a factor of 3, so our GA with a population size of 300 would still be faster than the MA by Xing *et al.*, but in this case with a much greater difference in our favor for the average deviations.

3.3 Other results

As mentioned in Section 1, many single-vehicle routing problems that actually cannot be modeled as an ATSP can be solved through a polynomial transformation into an ATSP. This is the case of the problem presented by Soler *et al.* (2008). They generated 128 instances for that problem and they transformed them into ATSP instances. To solve the transformed instances, they run the ATSP exact procedure by Corberan *et al.* (2005) on a PC with 1.8-GHz Pentium IV processor. Most of the instances were optimally solved in less than 2 h of running time, some instances needed more than 10 h to obtain the optimal solution (with one of them taking more than 17 h), and in only two instances was the execution aborted owing to the long period of time spent without finding the optimal cost.

Thus, we have a set of 126 ATSP instances with known optimal solution obtained from the aforementioned work, and with a number of vertices between 64 and 315. We have uploaded all data corresponding to this set of instances to the website http://www.iumpa.upv.es/arxivDSoler/, to make them easily available to any researcher.

This set has been recently used by Bräysy *et al.* (2011) as a benchmark set to test the efficiency of a memetic algorithm for a capacitated vehicle routing problem. Although this algorithm was not specifically designed to solve problems with a single vehicle, it was able to find the optimal solution in 86 out of the 126 instances.

In Table 3, we show the results on this set of ATSP instances obtained with our GA with configuration SEL2+EAX-1AB where the population size was set to 300. In this table, the 126 instances are partitioned into 24 groups, separated by horizontal lines, according to the features of the original instances from which they come. For each instance the table lists the name of the original instance (Name); the number of vertices ($|V|$); the optimal cost (Opt-cost); the time in seconds to obtain the optimal solution with the exact procedure (Opt-T); the best solution found in 100 runs of the GA (Best), with Opt shown if this solution coincides with the optimal one; the average cost over the 100 runs (a-cost), with Opt meaning that the GA has reached the optimal solution in the 100 runs; and finally the average computation time per single run in seconds (a-T).

From Table 3 we can see that our GA was able to find the optimal solution in 121 out of the 126 instances. The worst case was instance D143042, where the percentage excess of the best solution obtained with our GA over the optimal cost is only 0.0021%, and the second worst case was D183841b, where the percentage excess of the best solution obtained with our GA over the optimal cost is only 0.0004%. Moreover, with respect to the instances where the GA did not find the optimal solution in all 100 runs, the worst average deviation of the 100 runs with respect to the optimal cost also corresponds to instance D143042 (0.0024%).

Table 3: Computational results for instances from Soler *et al.* (2008).

Name	$\|V\|$	Opt-cost	Opt-T	Best	a-cost	a-T
D41140	64	417880	6.31	Opt	Opt	0.2
D61440	77	512674	14.22	Opt	Opt	0.3
D4940m	51	336325	2.69	Opt	Opt	0.1
D81740m	102	664613	5.65	Opt	Opt	0.4
D4940	67	422508	6.26	Opt	Opt	0.2
D81740	129	809706	51.74	Opt	Opt	0.6
D82040m	103	689971	12.19	Opt	Opt	0.5
D102240m	121	803595	34.54	Opt	Opt	0.6
D82040	122	790971	8.57	Opt	Opt	0.7
D102240	163	1027751	204.87	Opt	Opt	1.0
D102640m	118	815096	151.26	Opt	Opt	1.3
D122640m	130	880335	21.7	Opt	Opt	1.3
D122940m	148	997772	765.55	Opt	Opt	1.1
D102640	139	926142	330.59	Opt	Opt	1.8
D122640	150	990448	69.15	Opt	Opt	1.1
D122940	209	1316922	1018.59	Opt	1316922.2	2.6
D143040a	161	1072673	105.4	Opt	Opt	1.5
D143340	230	1452339	767.64	Opt	Opt	4.7
D143340m	162	1100177	585.72	Opt	Opt	1.4
D143040	236	1472328	97.22	Opt	1472332.6	3.3
D143040b	186	1210175	185.7	Opt	Opt	3.2
D163440a	169	1145914	181.96	Opt	Opt	1.4
D163440	226	1452914	1519.4	Opt	Opt	0.9
D163440b	200	1310767	197.4	Opt	1310767.3	2.4
D183840a	184	1255781	305.49	Opt	Opt	2.9
D163740	226	1468709	2585.73	Opt	Opt	4.3
D183840	258	1652091	1171.17	Opt	Opt	5.9
D183840b	220	1449912	565.84	Opt	Opt	4.3
D204240a	213	1438448	108.65	Opt	1438453.5	2.5
D184040a	196	1331558	3467.28	Opt	Opt	2.4
D204240	315	1980807	408.64	Opt	1980814.5	8.3
D184040	262	1686092	3119.43	Opt	Opt	6.7
D184040b	226	1493759	1037.65	Opt	1493776.2	4.5
D204240b	275	1764636	1126.68	Opt	Opt	5.1
D41141	68	454381	28.73	Opt	Opt	0.3

Table 3: Computational results for instances from Soler *et al.* (2008).

Name	$\|V\|$	Opt-cost	Opt-T	Best	a-cost	a-T
D4941	70	452101	12.97	Opt	452107.4	0.3
D61441	84	561657	17.52	Opt	Opt	0.4
D61641	95	633760	72.12	Opt	Opt	0.5
D81741	112	730893	67.99	Opt	730893.9	0.8
D82041m	122	804695	999.92	Opt	Opt	0.7
D102241m	128	854290	151.98	Opt	Opt	0.8
D82041	136	880751	104.35	Opt	Opt	0.9
D102241	161	1029382	658.94	Opt	Opt	1.9
D122641a	152	1011650	444.4	Opt	Opt	1.4
D102641m	152	1009032	633.89	Opt	Opt	1.2
D122941m	157	1058436	212.95	Opt	Opt	1.3
D122641	207	1306865	378.99	Opt	Opt	3.1
D102641	188	1199118	1006.29	Opt	1199121.2	2.3
D122941	192	1243493	388.99	Opt	Opt	1.9
D122641b	179	1156769	367.12	Opt	Opt	2.0
D143041a	164	1103917	166.81	Opt	1103917.2	2.5
D143341a	178	1201011	40.43	Opt	1201016.0	2.3
D143341b	211	1376011	3313.77	Opt	1376020.6	2.6
D143041	217	1389294	81.79	1389296	1389296.0	4.3
D143341	251	1584011	2085.68	Opt	1584022.4	5.0
D143041b	185	1219063	213.49	Opt	Opt	3.1
D163441a	198	1313082	853.43	Opt	Opt	2.8
D163441	282	1761528	886.66	Opt	1761554.6	8.7
D163441b	242	1547424	1253.51	Opt	Opt	2.8
D163741m	200	1343095	2447.26	Opt	Opt	2.6
D163741	235	1530323	928.57	Opt	1530329.7	7.1
D183841a	206	1385166	1724.55	Opt	Opt	3.8
D183841	278	1769642	970.48	1769648	1769648.0	7.5
D183841b	237	1552379	977.39	1552385	1552385.0	5.1
D184041b	242	1591521	1054.9	Opt	Opt	5.3
D184041a	224	1491653	1180.46	Opt	Opt	5.0
D184041	282	1803657	315.32	Opt	Opt	7.4
D204241a	224	1510820	503.23	Opt	1510820.9	4.1
D204441a	230	1556414	460.77	Opt	Opt	4.1
D204241	291	1872362	5613.99	Opt	1872363.1	6.4
D204441	315	2007744	5536.11	Opt	2007778.5	9.2
D204241b	258	1693166	6030.21	Opt	1693169.9	5.4
D204441b	273	1785585	1125.2	Opt	1785588.3	7.3
D4942	80	517768	12.91	Opt	Opt	0.4
D61642	138	875072	98.32	Opt	Opt	1.1
D61442	120	768117	145.17	Opt	Opt	0.7
D81742	127	828368	252.99	Opt	Opt	1.0
D82042	128	849394	190.2	Opt	Opt	0.9
D102242	146	963636	1941.23	Opt	963636.6	1.5
D122642m	162	1077557	945.37	Opt	1077562.4	1.6
D122942m	172	1149670	1966.17	Opt	1149674.2	2.2
D102642	160	1065324	275.34	Opt	1065326.2	1.9
D122642	176	1153625	3036.56	Opt	1153629.6	2.2
D122942	188	1235878	623.02	Opt	1235882.4	4.2
D143042m	184	1224644	942.36	Opt	1224664.1	3.6
D143342m	200	1329993	184.33	Opt	1329995.5	3.4

Table 3: Computational results for instances from Soler *et al.* (2008).

Name	$\|V\|$	Opt-cost	Opt-T	Best	a-cost	a-T
D143042	211	1369730	298.3	1369759	1369763.5	4.8
D143342	237	1525120	641.75	1525122	1525122.0	4.4
D163442	253	1622974	593.09	Opt	1622984.5	5.3
D163442a	206	1367607	512.45	Opt	Opt	3.1
D163742a	214	1432892	8394.27	Opt	1432921.6	6.6
D163442b	219	1442855	4401.89	Opt	1442857.0	3.6
D163742b	246	1602916	3442.84	Opt	1602952.5	4.7
D163742	301	1888149	5267.52	Opt	1888173.4	8.8
D183842m	213	1434387	586.55	Opt	1434407.1	4.9
D183842	233	1542727	251.83	Opt	1542740.9	7.2
D184042m	229	1534827	6919.67	Opt	Opt	3.8
D184042	270	1753907	6398.55	Opt	Opt	5.8
D204242a	236	1587087	1284.93	Opt	1587089.3	4.6
D204242	289	1872624	5832.32	Opt	1872626.1	10.8
D204242b	265	1742305	3821.49	Opt	1742312.0	6.9
D41143	108	695196	402.88	Opt	Opt	0.7
D61443	128	822217	240.52	Opt	822217.9	1.1
D61643	134	869353	2003.02	Opt	Opt	1.5
D81743	147	948708	741.11	Opt	948719.1	2.1
D82043m	154	1003444	1497.6	Opt	1003444.2	1.6
D82043	168	1079720	133.91	Opt	Opt	2.1
D102643	168	1121130	2091.13	Opt	Opt	2.2
D122643	186	1218825	131.72	Opt	Opt	2.2
D122943m	210	1365993	554.48	Opt	Opt	1.2
D122943	240	1526097	2901.27	Opt	Opt	7.1
D4944	120	758785	5141.3	Opt	758791.0	1.0
D61444	152	964730	2109.14	Opt	964740.4	1.8
D61644	153	984935	1924.15	Opt	984937.0	1.9
D81744	154	995860	13342.23	Opt	995865.8	2.2
D102244m	194	1245433	1997.75	Opt	1245436.6	3.7
D82044	168	1091866	1016.54	Opt	Opt	2.5
D122644m	198	1295659	63854.52	Opt	1295663.4	3.8
D122944m	210	1380447	6368.56	Opt	1380447.9	6.4
D122944	224	1456564	14789.84	Opt	1456573.8	8.4
D102644	214	1379555	35721.07	Opt	1379562.2	3.8
D143044m	219	1438672	225.13	Opt	1438704.1	9.9
D143044	245	1578725	4017.8	Opt	1578759.7	9.4
D143344	236	1549308	39896.66	Opt	Opt	6.4
D163744m	254	1671485	37142.93	Opt	Opt	13.8
D163744	278	1801619	50808.44	Opt	1801621.8	16.7

4 Conclusions

We have presented a new GA to solve the ATSP. We have checked its efficiency on a set of 153 benchmark ATSP instances with known optimal solution. From the obtained results with the GA, we believe that our procedure is very competitive. In fact, it outperforms the results obtained with previous ATSP heuristics in the well-known set of ATSP instances from TSPLIB. Because the ATSP can be used to solve many real-world problems, either directly or through a transformation, our aim is to show that the procedure presented here can be useful for solving these problems or for measuring the efficiency of actual or future proposed heuristics that directly address these problems. Moreover, to strengthen this aim, through the website http://www.iumpa.upv.es/arxivDSoler/ we make easily available to any researcher, a detailed information on the set of 126 ATSP instances used here and in previous papers and not coming from TSPLIB.

Acknowledgements

This work has been partially supported by the Ministerio de Educación y Ciencia of Spain (Project No. TIN2008-06441-C02-01).

References

Albayrak, M., & Allahverdi, N. (2011). Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications*, 38, 1313–1320.

Albiach, J., Sanchis, J.M., & Soler, D. (2008). An asymmetric TSP with time windows and with time-dependent travel times and costs: An exact solution through a graph transformation. *European Journal of Operational Research*, 189, 789–802.

Baldacci, R., Bartolini, E., & Laporte, G. (2010). Some applications of the generalized vehicle routing problem. *Journal of the Operational Research Society*, 61, 1072–1077.

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., & Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters*, 31, 357–365.

Bentley, J.L. (1990). Experiments on traveling salesman heuristics, Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 91–99.

Blais, M., & Laporte, G. (2003). Exact solution of the generalized routing problem through graph transformations. *Journal of the Operational Research Society*, 54, 906–910.

Bräysy, O., Martínez, E., Nagata, Y., & Soler, D. (2011). The mixed capacitated general routing problem with turn penalties. *Expert Systems with Applications*, 30, 12954–12966.

Buriol, L., França, P.M., & Moscato, P. (2004). A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10, 483–506.

Chen, S.M., & Chien, C.Y. (2011). Parallelized genetic ant colony systems for solving the traveling salesman problem. *Expert Systems with Application*, 38, 3873–3883.

Choi, I.C., Kim, S.I., & Kim, H.S. (2003). A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem. *Computers & Operations Research*, 30, 773–786.

Clossey, J., Laporte, G., & Soriano, P. (2001). Solving arc routing problems with turn penalties. *Journal of the Operational Research Society*, 52, 433–439.

Corberán, A., Martí, R., Martínez, E., & Soler, D. (2002). The rural postman problem on mixed graphs with turn penalties. *Computers & Operations Research*, 29, 887–903.

Corberán, A., Mejía, G., & Sanchis, J.M. (2005). New results on the mixed general routing problem. *Operations Research*, 53, 363–376.

Fischetti, M., Lodi, A., & Toth, P. (2002). Exact methods for the asymmetric traveling salesman problem. In: G. Gutin & A.P. Punnen (Eds.), *The Traveling Salesman Problem and Its Variations*, pp. 168–205 (Boston: Kluwer).

Kanellakis, P.C., & Papadimitriou, C.H. (1980). Local search for the asymmetric traveling salesman problem. *Operations Research*, 28, 1086–1099.

Laporte, G. (1997). Modeling and solving several classes of arc routing problems as traveling salesman problems. *Computers & Operations Research*, 24, 1057–1061.

Laporte, G. (2010). A concise guide to the traveling salesman problem. *Journal of the Operational Research Society*, 61, 35–40.

Micó, J.C., & Soler, D. (2011). The capacitated general windy routing problem with turn penalties. *Operations Research Letters*, 39, 265–271.

Nagata, Y., & Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. Proceedings of the 7th International Conference on Genetic Algorithms, pp. 450–457.

Nagata, Y. (2004). The EAX algorithm considering diversity loss. Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, *Lecture Notes in Computer Science*, 3242, 332–341.

Nagata, Y. (2006a). Fast EAX algorithm considering population diversity for traveling salesman problems. Proceedings of the 6th International Conference on Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, 3906, 171–182.

Nagata, Y. (2006b). New EAX crossover for large TSP instances. Proceedings of the 9th International Conference on Parallel Problem Solving from Nature, *Lecture Notes in Computer Science*, 4193, 372–381.

Noon, C.E., & Bean, J.C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31, 39–44.

Oncan, T., Altinel, I.K., & Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, 36, 637–654.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leadings methods, implementations and latest advances. *European Journal of Operational Research*, 211, 427–441.

Soler, D., Albiach, J., & Martínez, E. (2009). A way to optimally solve a time-dependent vehicle routing problem with time windows. *Operations Research Letters*, 37, 37–42.

Soler, D., Martínez, E., & Micó, J.C. (2008). A transformation for the mixed general routing problem with turn penalties. *Journal of the Operational Research Society*, 59, 540–547.

Xing, L.N., Chen, Y.W., Yang, K.W., How, F., Shen, X.S., & Cai, H.P. (2008). A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem. *Engineering Applications of Artificial Intelligence*, 21, 1370–1380.

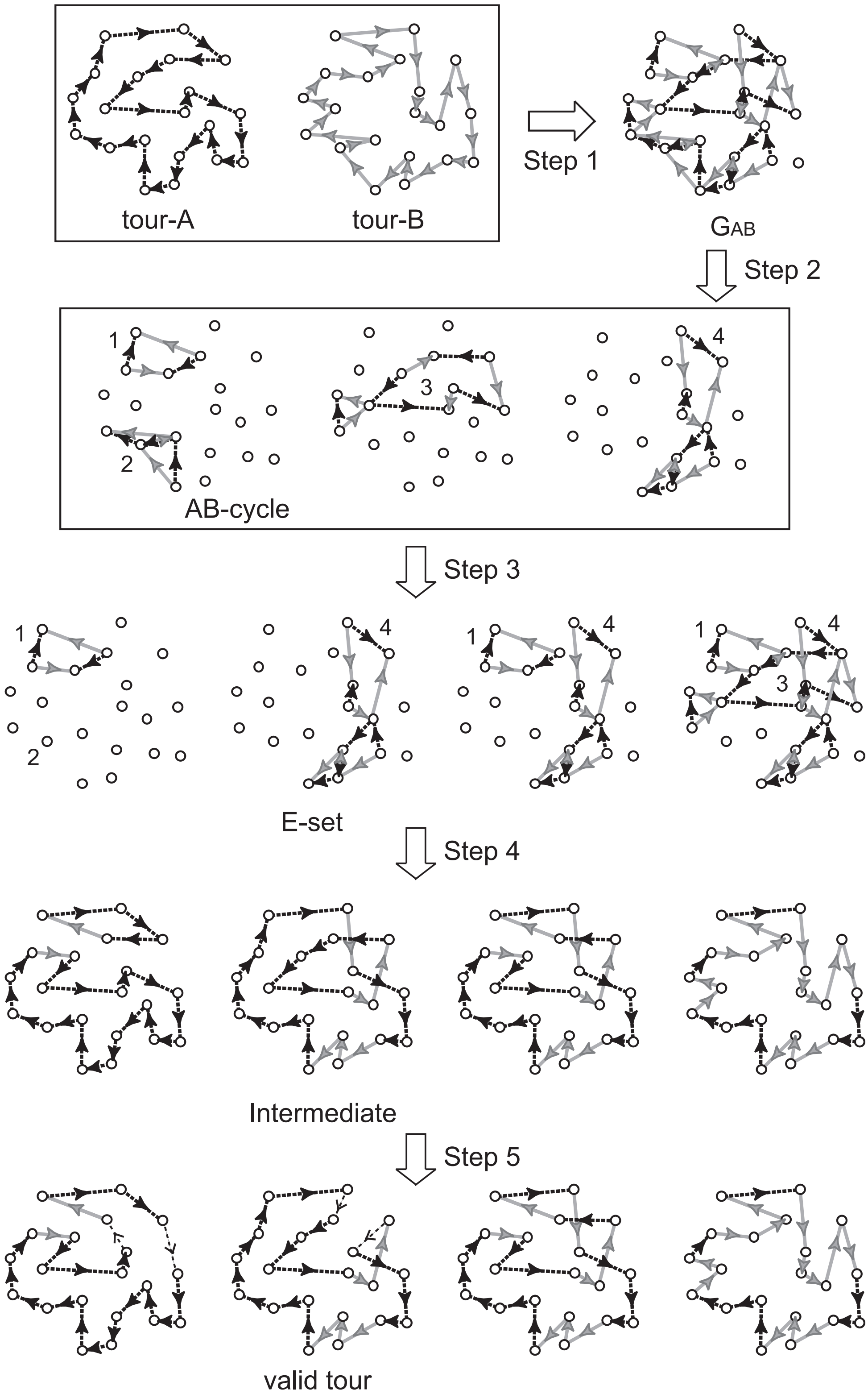

Figure 1: Illustration of the EAX for the ATSP.