Grouping memetic search for the colored traveling salesmen problem

Pengfei He [a], Jin-Kao Hao [a,*], Qinghua Wu [b]

[a]*LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[b]*School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China*

Accepted to Information Sciences, April 2021

Abstract

The colored traveling salesmen problem is a node routing problem with multiple salesmen, where the cities are divided into m exclusive city sets and one shared city set. The objective is to minimize the total traveling distance of m Hamiltonian circuits (routes) under the following constraints: each exclusive city is to be visited by the corresponding salesman, while each shared city can be visited by any salesman. In this work, we present the first grouping memetic algorithm for solving this challenging problem. The algorithm includes three main components: (i) a greedy randomized heuristic for population initialization; (ii) a dedicated local search procedure for local optima exploration; (iii) a backbone-based crossover operator for solution recombination. We show computational results on three sets of 65 popular benchmark instances to demonstrate the competitiveness of our algorithm. We especially report improved upper bounds for 38 instances (for more than 58% cases). We also present first computational results with the general CPLEX solver, including 10 proven optimal solutions. Finally, we shed lights on the impacts of the key components of the algorithm.

Keywords: Memetic algorithm; Local search; Colored traveling salesmen problem, TSP and Multiple TSP.

1 Introduction

The colored traveling salesmen problem (CTSP) can be stated as follows [21]. Let G=(V, A) be a complete undirected graph, where $V = \{0, 1, 2, \cdots, n\}$ is

* Corresponding author.
Email addresses: `pengfeihe606@gmail.com` (Pengfei He), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `qinghuawu1005@gmail.com` (Qinghua Wu).

the set of nodes (or cities) and $A = \{\{i,j\} : i,j \in V, i \neq j\}$ is the set of edges. Each edge $\{i,j\} \in A$ has a non-negative weight c_{ij} representing the traveling distance between cities i and j. All cities are divided into $m+1$ disjoint sets: m exclusive city sets $\{C_1, C_2, \cdots, C_m\}$, and one shared city set S such that $\cup_{k=1}^{m} C_k \cup S = V$ and $\cap_{k=1}^{m} C_i \cap S = \emptyset$. The cities of an exclusive set C_k $(k = 1, 2, \cdots, m)$ are to be visited by salesman k only, while the shared cities can be visited by any of the m salesmen. Besides, city 0 (the depot) belongs to the shared city set S and is visited by all salesmen. CTSP is to determine m shortest Hamiltonian tours (routes) starting from the depot and ending at the depot such that each exclusive city in C_k is visited exactly once by salesman k and each shared city is visited exactly once by one of the m salesmen. A mathematical model of CTSP is provided in Appendix A.

CTSP generalizes a variant of the classical traveling salesman problem (TSP), known as the multiple traveling salesmen problem (MTSP) where all cities are shared [3,4]. Besides, if there is only one salesman ($m = 1$), CTSP becomes TSP [1]. Given that CTSP generalizes these NP-hard problems, solving CTSP is computationally challenging.

CTSP is a useful model for a number of practical problems [21]. For instance, Li et al. [20] presented typical applications concerning multi-bridge machining systems, i.e., a dual-bridge waterjet machining center and a dual-manipulator hull welding system. In these systems, there are two independent individual machines (cutting machines and manipulators) with their individual workspaces (exclusive cities) and a shared workspace (shared cities). Finding a collision-free scheduling of these machines corresponds to solving a CTSP with two salesmen. The rice harvesters problem studied in He et al. [15,16] can be considered from the CTSP perspective. Rices from moist fields and non-moist fields need to be harvested by harvesters (salesmen) such that moist fields (exclusive cities) can only be visited by crawler-harvesters, while non-moist fields (shared cities) can be visited by both crawler-harvesters and wheelers-harvesters. Scheduling the harvesters comes down to solving the CTSP problem.

Heuristics and metaheuristics are natural approaches for finding sub-optimal solutions of difficult optimization problems that cannot be solved exactly. Unlike the related MTSP for which numerous heuristics are available [31,37,4,30,36], only six metaheuristic-based algorithms were studied for CTSP: genetic algorithm [21], variable neighborhood search [24], artificial bee colony (ABC) [28,9], ant colony optimization [8], and iterated two phase local search [14]. These algorithms have reported valuable computational results on various benchmark instances. Meanwhile, it is observed that they lack robustness and stability in particular when they are applied to solve large scale instances.

In this work, we investigate for the first time the powerful memetic algorithm (MA) framework for solving CTSP and present a competitive grouping memetic algorithm (GMA) dedicated to the problem. Indeed, effective MAs have been proposed to solve the related MTSP [36,18,19,23] and several vehicle routing problems [25,35,5,29]. However, most of these MAs are based on the so-called giant tours and split algorithms, which are not suitable for CTSP due to the presence of exclusive cities. We consider CTSP from the perspective of grouping problems [10] and introduce an effective grouping memetic algorithm (GMA). The proposed algorithm integrates two complementary key components: an original local optima exploration procedure (to find high quality local optima, Section 3.3) and a dedicated backbone-based crossover operator (to generate promising new offspring, Section 3.4). As demonstrated by the computational results shown in Section 4, the proposed algorithm competes very favorably with the state-of-the-art CTSP algorithms on three sets of benchmark instances.

The rest of this paper is organized as follows. Section 2 presents a literature review and related works. The proposed grouping memetic algorithm is presented in Section 3. Computational results and comparisons with state-of-the-art algorithms are presented in Section 4. In Section 5, the impacts of key components of the algorithm are discussed. Section 6 presents conclusions and future research directions.

2 Literature review and related works

Given the theoretical and practical significance, CTSP has attracted considerable attention in recent years and several heuristic methods have been presented. In this section, we review the existing solution approaches for CTSP and related works.

In 2014, Li et al. [21] introduced the colored traveling salesmen problem to optimize routes of a dual-bridge waterjet cutting machine tool. As solution methods, they presented four genetic algorithms (basic GA, GA with greedy initialization, hill-climbing GA and simulated annealing GA), where the *dual-chromosome* encoding was used to represent the candidate solutions. The first chromosome is a permutation of all cities except depot 0, while the second chromosome assigns a salesman to each of the shared and exclusive cities in the corresponding position of the first chromosome. They presented computational results on 20 small scale benchmarks created from existing symmetric TSP instances (with up to 101 vertices). They showed that the hybrid algorithm combining simulated annealing and GA dominated the three other algorithms and their algorithms performed better than the general mixed integer programming tool Lingo.

Then, in 2017, Meng et al. [24] proposed a variable neighborhood search (VNS) which employs a *direct-route* encoding to represent the solutions. VNS consists of two phases. The first phase perturbs the current solution by two shaking operations (*Interchange* and *Relocation*), while the second phase improves the perturbed solution by applying a local search based on two search operations (neighborhood change and 2-opt). Compared with the four GAs [21], VNS showed its competitiveness on the 20 benchmark instances.

Later, in 2018, Pandiri and Singh [28] presented an artificial bee colony (ABC) based on the *m-tour* encoding. This encoding uses m arrays, and each array includes all the cities visited by the corresponding salesman. They provided a proof that the size of the solution space of CTSP with the *m-tour* encoding is smaller than that of the *dual-chromosome* encoding. They showed that ABC could match or update the best results reported in [21,24] on the 20 small scale benchmark instances with very short cutoff times. Besides, they introduced 8 new medium scale instances (with 229-666 cities) and reported computational results.

Also in 2018, Dong et al. [8] employed an ant colony optimization (ACO) with multi-tasks learning for solving CTSP. The multi-task cooperative learning was proposed to improve the efficiency of ACO. To assess their algorithm, they introduced 6 medium (with 202-431 cities) and 5 large instances (with 1002 cities) and showed the competitiveness of ACO compared with the four GAs [21]. Nevertheless, this algorithm did not compete well with VNS [24] on the 20 small scale instances in terms of the best and average results.

In 2019, Dong et al. [9] presented another artificial bee colony algorithm (ABC) and reported computational results on 26 new large instances (with 2461-7397 cities). These new large scale instances could be used by subsequent studies to evaluate their algorithms. However, this ABC algorithm performed worse than the ABC algorithm of [28] on the 20 small scale instances.

Finally, He and Hao [14] proposed an iterated two-phase local search (ITPLS), which is based on a new *adjacency representation* of the candidate solutions. This representation relies on an array $A[m, n + 1]$ such that for each route r ($r = 1, \ldots, m$), $A[r, i] = j$ ($i, j = 0, \ldots, n, i \neq j$) if and only if the route goes from city i to city j. ITPLS applies jointly inter-route optimization and intra-route optimization for solution improvement, reinforced by a probabilistic greedy perturbation strategy to diversify the search. Extensive computational results were reported on all the benchmark instances available in the literature (a total of 65 instances), including 29 improved best known results.

According to the computational results reported in the literature, we identify ABC [28] and ITPLS [14] as the current state-of-the-art CTSP algorithms.

CTSP generalizes the popular multiple traveling salesmen problem (MTSP), which has attracted much interest in the last decades. For instance, Wang et al. [36] introduced a memetic algorithm for solving MTSP, which includes a variable neighborhood descent to search local optima. Another evolutionary algorithm was proposed by Kashan et al. [18] for solving MTSP from the perspective of grouping problems. Other representative studies were reported in [31,37,4,30]. However, these methods are not suitable for CTSP, because of the presence of exclusive cities.

In this work, we are interested in designing a practically effective algorithm for solving CTSP with the memetic framework. This is motivated by two considerations. First, one notices that the route of each salesman can be considered as a TSP solution. Therefore, the optimization of each individual route can naturally benefit from existing powerful TSP methods. Second, we can consider CTSP from the perspective of grouping problems in the sense that the shared cities are to be dispatched into m groups (m being the number of salesmen). As such, the population-based memetic framework with a meaningful crossover represents an attractive approach given that it has been applied with great success to several difficult grouping problems (e.g., [10,11,39]).

3 Grouping memetic algorithm for CTSP

Given a CTSP instance, the search space explored by CTSP is a multi-route problem whose candidate solutions consist of m tours where the k-th tour includes city 0, the exclusive cities of C_k and some shared cities of S.

In this section, we present the grouping memetic algorithm (GMA) for solving CTSP. For a CTSP instance, GMA explores a search space Ω composed of all candidate feasible solutions, where a candidate solution φ consists of m tours $\{r_1, r_2, \ldots, r_m\}$ with r_k ($k = 1, 2, \ldots, m$) being the k-th route visited by the k-th salesman. Given a solution $\varphi \in \Omega$, its objective value $f(\varphi)$ is given by the total distance of its m routes. The goal of GMA is thus to find a solution with the smallest objective value as possible.

In the literature, three common methods were used to represent solutions of CTSP: dual chromosome encoding [21], *m-tour* encoding [28] and *adjacency representation* encoding [14]. In this work, we adopt the *adjacency representation* encoding, which has the advantage of encoding each route (group of cities) independently to facilitate inter-routes operations. The interested reader is referred to [14] for more details and an illustrative example.

3.1 General scheme

The proposed GMA algorithm consists of four main components: population initialization, local optima exploration, backbone-based crossover and population updating. GMA starts with an initial population P of p solutions generated by the population initialization procedure (Section 3.2). It then repeats a number of generations during which new candidate solutions are sampled.

At each generation, the backbone-based crossover combines two randomly and uniformly selected parent solutions to generate a promising offspring solution (Section 3.4). The local optima exploration (LOE) is then applied to improve the offspring solution (Section 3.3), followed by population update (Section 3.5). This evolution process is terminated when a predefined stopping condition (e.g., an allowed number of generations, an allotted cutoff time limit) is reached. In this work, we use a cutoff time limit. The pseudo-code of GMA is shown in Algorithm 1.

Algorithm 1: Pseudo-code of GMA algorithm

```
Input: Instance $I$, population size $p$, number of the nearest cities $N_n$, parameter $\alpha$
Output: The best solution $\varphi^*$ found
1  begin
2      $P = \{\varphi_1, \varphi_2, \cdots, \varphi_p\} \leftarrow$ PopInitilize $(I, p)$; /* Build an initial population of $p$ elite solutions, Section 3.2 */
3      $\varphi^* \leftarrow \arg\min \{f(\varphi_i) : i = 1, 2, \cdots, p\}$;
4      while Stopping condition is not met do
5          randomly and uniformly select two parents $\varphi_F$ and $\varphi_M$ from $P$;
6          $\varphi_O \leftarrow Backbone\_Crossover(\varphi_F, \varphi_M)$; /* Generate an offspring solution by backbone-based crossover, Section 3.4 */
7          $\varphi_O \leftarrow$ LOE $(\varphi_O, N_n, \alpha)$; /* Improve the new solution by local optima exploration, Section 3.3 */
8          if $f(\varphi_O) < f(\varphi^*)$ then
9              $\varphi^* \leftarrow \varphi_O$;
10         end
11         $P \leftarrow PoolUpdating(P, \varphi_O)$; /* Update the population with the new solution, Section 3.5 */
12     end
13     return $\varphi^*$;
14 end
```

3.2 Population initialization

The GMA starts its search with an initial population P of p high-quality (elite) solutions. To construct a population, we use a greedy randomized heuristic to generate a feasible solution, which is further improved by LOE described in Section 3.3. The improved solution is then inserted into P if the solution is different from any existing solution in P; otherwise, this solution is discarded. This process is repeated until p different solutions are generated. Thanks to the greedy randomized heuristic and subsequent LOE improvement step, we obtain a diverse and high-quality population.

A feasible solution is constructed by the greedy randomized heuristic according

Algorithm 2: Pseudo-code of the greedy randomized heuristic

```
Input: Instance $I$ (exclusive city sets $\{C_1, C_2, \cdots, C_m\}$, shared city set
        $S$) and distance matrix
Output: A feasible solution $\varphi$
1  begin
2      $\varphi \leftarrow \emptyset$; /* First step: build $m$ partial routes with
        exclusive cities                                                  */
3      for $k = 1$ to $m$ do
4          $r_k \leftarrow \{0\}$; /* Initiate the route with the city 0   */
5          while $C_k \neq \emptyset$ do
6              Select randomly and uniformly a city $i$ from set $C_k$;
7              Insert city $i$ in route $r_k$ such that the route distance increase is
                minimal;
8              Remove city $i$ from set $C_k$;
9          end
10         $\varphi \leftarrow \varphi \cup \{r_k\}$;
11     end
       /* Second step: dispatch the shared cities $S \setminus \{0\}$ among $m$
        partial routes                                                    */
12     $S' \leftarrow S \setminus \{0\}$;
13     while $S' \neq \emptyset$ do
14         Select randomly and uniformly a city $j$ from set $S'$;
15         Insert city $j$ into a route of $\varphi$ such that the total distance increase
            is minimal;
16         Remove city $j$ from set $S'$;
17     end
18     return $\varphi$;
19 end
```

to the following steps: 1) build a partial route for each of the m salesmen by using the corresponding exclusive cities; 2) dispatch the shared cities among the m partial routes to obtain a complete solution. The pseudo-code of the greedy randomized heuristic is shown in Algorithm 2. During the first step (lines 4-12), to create the k-th partial route r_k, one first initiates the route with the city 0. Then, the exclusive cities in C_k are selected randomly and uniformly, and inserted one by one into r_k such that the insertion gives the smallest increase of the route distance. When all exclusive cities of every salesman are inserted into the corresponding route, the first step stops, leading to a partial solution φ composed of m partial routes. During the second step (lines 13-18), the shared cities j from $S \setminus \{0\}$ are processed randomly and uniformly, and inserted one by one into a route of the partial solution φ such that its total distance increase is minimal. When all shared cities are inserted, an initial solution is obtained. The first step has a time complexity of $O(|C_m|^2 \times m)$, while the second step is bounded by $O(|S| \times n)$. Therefore, the time complexity of the greedy randomized heuristic is $O(|S| \times n)$.

3.3 Local optima exploration

Local optimization plays a key role in a memetic algorithm and constitutes one of the driving forces for finding solutions of increasing quality. For an effective examination of local optima, GMA employes a specific strategy that combines an inter-route optimization and an intra-route optimization procedure heuristic. Specifically, our local optima exploration procedure (LOE) iterates two complementary search components: the constrained cross-exchange operator (CCE) (Section 3.3.1) and a TSP heuristic called Edge Assembly Crossover (EAX) [26] (Section 3.3.2). CCE improves solutions by exchanging two substrings (sub-routes) from two routes. The routes modified by CCE are indicated by a binary vector RT of length m ($RT[i] = 1$ if route i is changed by CCE, $RT[i] = 0$ otherwise). Then each modified route is further optimized by EAX. CCE and EAX are repeated until the current solution φ cannot be further improved. Algorithm 3 shows the pseudo-code of the local optima exploration procedure integrating the CCE operator and the EAX heuristic.

Algorithm 3: Pseudo-code of local optima exploration

```
Input: Solution $\varphi$, number of the nearest cities $N_n$, parameter $\alpha$
Output: Improved solution $\varphi_B$
1  begin
2      $\varphi_B \leftarrow \varphi$;
3      $Flag \leftarrow$ true;
4      $RT[k] \leftarrow$ false $\forall k \in \{1, \ldots, m\}$; /* $RT$ is a binary vector,
         indicating the routes modified by CCE                          */
5      while $Flag$ do
6          $< \varphi, Flag, RT > \leftarrow CCE(\varphi, N_n, \alpha)$; /* Solution improvement
             by CCE, Section 3.3.1                                       */
7          for $k = 1, \ldots, m$ do
8              if $RT[k] = true$ then
9                  $\varphi \leftarrow \varphi \setminus \{r_k\}$;
10                 $r_k \leftarrow EAX(r_k)$; /* Intra-route improvement by EAX,
                     Section 3.3.2                                       */
11                 $\varphi \leftarrow \varphi \cup \{r_k\}$;
12             end
13         end
14         if $f(\varphi) < f(\varphi_B)$ then
15             $\varphi_B \leftarrow \varphi$;
16         end
17     end
18     return $\varphi_B$;
19 end
```

3.3.1 Constrained cross-exchange

The conventional cross-exchange was initially designed for vehicle routing problems [2,33,6]. It is a generic local search operator which performs exchanges of two consecutive substrings (sub-routes) $\hat{r_i}$ and $\hat{r_j}$ from two different routes r_i and r_j. However, given the particularity of exclusive cities in CTSP, the cross-exchange cannot be used directly in our context. For this reason, we propose a constrained cross-exchange (CCE) in this work. Moreover, it is known that the cross-exchange has a high time complexity [2,33]. CCE uses a suitable pruning technique to reduce this complexity.

The evaluation of a CCE move for CTSP is summarized in two steps. The first step is to determine the start of two substrings and the second step is to identify the suitable length of both substrings ($\hat{r_{k_1}}$ and $\hat{r_{k_2}}$). For the start of substring $\hat{r_{k_1}}$, we first need to find an edge which will break route r_{k_1}. Suppose the edge is $\{I_1, I_2\}$. Then, a suitable new neighbor of city I_1 needs to be determined. To limit the number of candidate moves, CCE uses the following heuristic pruning technique that only considers the neighbors among the N_n nearest cities. Suppose that city J_3 is such a neighbor, and city J_3 belongs to route r_{k_2}. If edge $\{I_1, J_3\}$ is added as a new edge, edge $\{J_2, J_3\}$ or edge $\{J_3, J_4\}$ should be removed. Once the starts of two substrings (I_2 and J_3) are determined, we need to identify the length of each substring. It is worth noting that each substring should not include any exclusive cities because these cities are only visited by the corresponding salesman.

Because the number of cities of each substring can vary from 0 to α (a parameter), all feasible combinations of the two substrings with their given starts can be listed, and the move gain δ for each combination can also be calculated. There are at most $(\alpha + 1)^2$ combinations of two substrings. When a substring is empty and the other is non-empty ($\hat{r_{k_1}} = \emptyset$ or $\hat{r_{k_2}} = \emptyset$), these two cases are Or-opt [27,33]. However, both substrings cannot be empty simultaneously. Therefore, at most $(\alpha + 1)^2 - 1$ combinations of two substrings could be listed for two given starts. Then, we need to identify the best move (i.e., with the largest gain δ_l) in these combinations. So far, a CCE move $< \hat{r_{k_1}}, \hat{r_{k_2}} >$ is acquired and the lengths of two substrings are determined. For all combinations of the two starts, the global minimal move gain δ_b can be identified. If $\delta_b < 0$, $Flag \leftarrow true$, $RT[k_1] \leftarrow 1$ and $RT[k_2] \leftarrow 1$; then, solution φ is updated by swapping two substrings ($\hat{r_{k_1}}$ and $\hat{r_{k_2}}$); otherwise, solution φ, $Flag$ and matrix RT are returned, because the stopping condition ($\delta_b \geq 0$) of CCE is met. As for the time complexity of CCE, there are $O(|S| \times (\alpha + 1))$ ways to select the first substring in any given route, while $O(N_n \times (\alpha + 1))$ ways exist to select the second substring in another route. Therefore, the time complexity of CCE is $O(|S| \times N_n \times ((\alpha + 1)^2 - 1))$.

For example, Fig. 1 illustrates two cases of determining the starts of two substrings. Then two complete CCE moves ($\hat{r_{k_1}} = \{I_2\}$ and $\hat{r_{k_2}} = \{J_3, J_4\}$ or $\hat{r_{k_2}} = \{J_3, J_2\}$) are illustrated in Fig. 2, where cities $\{I_2, J_2, J_3\}$ are shared. If edge $\{J_2, J_3\}$ is broken in the first step, the substring $\hat{r_{k_2}} = \{J_3, J_4\}$ is serial and in order. However, if edge $\{J_3, J_4\}$ is broken in the first step, the substring $\hat{r_{k_2}} = \{J_3, J_2\}$ is serial and reverse.

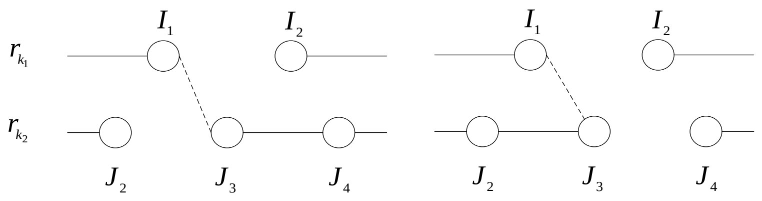


Fig. 1. Illustrative example of starts for a CCE move.

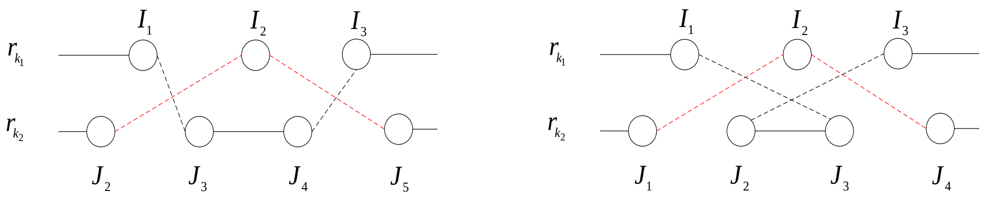


Fig. 2. Illustrative example of complete CCE moves.

One may note the following differences between CCE and cross-exchange [2]. First, the cross-exchange operator used in [2] does not limit the length of the substrings to be exchanged; however, in CCE, the length of the substrings must be less than or equal to the value fixed by the parameter α. Second, in CCE, exclusive cities are constrained to stay in a route and cannot be moved to other routes. Therefore, the two substrings to be exchanged should not include any exclusive cities. Finally, unlike vehicle routing for which cross-exchange was designed, there is no capacity limitation for each salesman in CTSP. So CCE does not consider this capacity constraint.

3.3.2 Edge assembly crossover (EAX) for TSP

For the optimization of each individual route, the constraint of exclusive cities can be ignored. Thus optimizing each route corresponds to solving a TSP. There are several sophisticated and powerful heuristics designed for solving TSP. For example, the well-known fast 2-opt heuristic or LK algorithm can be used to improve each route [17,2,31]. In this work, we adopt the EAX heuristic [26] [1], which is among the best TSP heuristics. In our case, EAX helps to keep each route to being optimal or near-optimal in the iterative process of LOE.

[1] The code of EAX is available at: https://github.com/sugia/GA-for-TSP

3.4 Backbone-based crossover

Crossover is another important ingredient of a memetic algorithm and should be designed with care in order to favor transmissions of useful information from parents to offspring [13], while respecting the problem specific structure. One popular way of designing meaningful crossover for grouping problems such as CTSP is to explore the so-called backbone information, which typically corresponds to solution attributes shared by elite solutions [32,12,11,39]. In this work, we follow this idea and design a dedicated backbone crossover for CTSP.

Let φ_F and φ_M be two parent solutions in the population. Based on φ_F and φ_M, we divide the set of shared cities except the depot $(S \setminus \{0\})$ into two categories, i.e., common elements and non-common elements.

Definition 1: Given two parent solutions $\varphi_F = \{r_1^F, r_2^F, \ldots, r_m^F\}$ and $\varphi_M = \{r_1^M, r_2^M, \ldots, r_m^M\}$, a city $i \in S \setminus \{0\}$ with respect to φ_F and φ_M is a common element if there exists a $k \in \{1, \ldots, m\}$ such that i appears in both r_k^F and r_k^M (i.e., $i \in r_k^F \cap r_k^M$). If i appears in r_k^F and r_l^M $(k \neq l)$, city i is a non-common element.

Then, an offspring solution φ_O is constructed in two steps. In the first step, a donor parent is first chosen randomly and uniformly between φ_F and φ_M. A partial offspring solution φ_O is then created by inheriting all m routes of the donor parent without the shared cities. In the second step, for each city $i \in S \setminus \{0\}$, if it is a common element appearing in r_k^F and r_k^m, then city i is greedily inserted into route r_k^O of the partial offspring solution. If city i is a non-common element $(i \in r_k^F, i \in r_l^M$ and $k \neq l)$, we randomly and uniformly select one route of the partial offspring solution and then greedily insert i into the selected route such that the insertion leads to the smallest increase of the total distance.

$$\varphi_F = \{r_1^F=\{0,1,2,3,8,9\};\ r_2^F=\{0,4,5,6,7,10\}\}$$
$$\varphi_M = \{r_1^M=\{0,2,1,3,9,10\};\ r_2^M=\{0,6,4,5,7,8\}\}$$
$$\Big\Downarrow \text{The first step}$$
$$\varphi_O = \{r_1^O=\{0,2,1,3\};\ r_2^O=\{0,6,4,5\}\}$$
$$\Big\Downarrow \text{The second step}$$
$$\varphi_O = \{r_1^O=\{0,2,1,3,9,8\};\ r_2^O=\{0,6,7,4,5,10\}\}$$

Fig. 3. Illustrative example of the backbone-based crossover

Fig. 3 shows an example of the crossover operator for a CTSP instance with 11 cities {0,1,. . . ,10} and $m = 2$ salesmen with their sets of exclusive cites $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6\}$, and the set of shared cities $S \setminus \{0\} = \{7, 8, 9, 10\}$

(marked in red color). Let $\varphi_F = \{r_1^F = \{0,1,2,3,8,9\}; r_2^F = \{0,4,5,6,7,10\}\}$ and $\varphi_M = \{r_1^M = \{0,2,1,3,9,10\}; r_2^M = \{0,6,4,5,7,8\}\}$ be the parent solutions. By *Definition* 1, cities 7 and 9 are common elements, while 8 and 9 are non-common elements. First, suppose that φ_M is the donor parent. Then offspring φ_O inherits the routes r_1^M and r_1^M by deleting the four shared cities, leading to $\varphi_O \leftarrow \{r_1^O = \{0,2,1,3\}; r_2^O = \{0,6,4,5\}\}$. Then the shared cities $\{7,8,9,10\}$ are successively considered until they are all inserted. City 7 is a common element of the second routes of the parent solutions, it is thus greedily inserted into the partial route r_2^O, supposing this is the cheapest insertion that increases the least the total distance. City 8 is a non-common element, it is greedily inserted into the partial route r_1^O or r_2^O with equal probability. Suppose that route r_1^O is selected, and city 8 is inserted into route r_1^O at the cheapest place leading to the smallest increase of the route distance. Cites 9 and 10 are processed in the same way. When all shared cities $\{7,8,9,10\}$ are inserted into φ_O, a feasible offspring solution is constructed successfully, which is then submitted to LOE for further improvement.

The time complexity of the crossover can be estimated as follows. The first step needs to scan all the cities of the donor parent to allow its m routes to be partially inherited. This is achieved in $O(n)$ time. In the second step, the shared cities in $S \setminus \{0\}$ are inserted into the partial offspring at the most suitable places, while the time complexity of evaluating each move gain is $O(1)$. The second step can be performed in $O(|S| \times n)$ time. As the result, the time complexity of the crossover is $O(|S| \times n)$.

3.5 Pool updating strategy

For each new offspring solution φ_O improved by LOE in Section 3.3, the pool updating strategy uses φ_O to update the population P as follows. If the offspring φ_O is different from any existing solutions and better than the worst solution in P, φ_O replaces the worst solution; otherwise φ_O is discarded.

4 Experimental results and comparisons

This section presents a performance assessment of the GMA algorithm. We show computational studies on well-known benchmark instances from the literature, and comparisons with existing state-of-the-art algorithms for CTSP.

4.1 Benchmark and experimental protocol

We employ three sets of 65 benchmark instances, which were commonly used in previous studies on CTSP. The first set (Set I) contains 20 small instances which were introduced in [21], and the number of cities is between 21 to 101 while the number of salesmen m is between 2 and 7. The second set (Set II), introduced in [28,8], contains 14 medium size instances with 202, 229, 431, 666 cities, and $10-40$ salesmen. The last set (Set III), presented in [9,8], contains 31 large instances with $1002-7397$ cities and $3-60$ salesmen.

GMA was coded in C++ and complied with a g++ compiler with the -O3 option [2]. All experiments were conducted on a computer with an AMD-6134 processor (2.3 GHz and 2 GB RAM) running Linux.

To assess the performance of GMA, we show comparisons with the following algorithms: artificial bee colony (ABC) [28] (2018), ant colony optimization (ACO) [8] (2018) and iterated two phase local search (ITPLS) [14] (2021). Indeed, computational results reported in the literature indicate that these three algorithms represent the state-of-the-art of solving the above benchmark instances, while ABC [28] and ITPLS [14] are clearly two dominating algorithms. So we use ABC (source code unavailable) and ITPLS (source code available) as the main reference algorithms and cite ACO (source code unavailable) when it is appropriate.

To make the comparisons as fair as possible, we faithfully re-implemented the best ABC algorithm of [28] [3]. We verified that our implementation (denoted as re-ABC) was able to reproduce the results reported in [28] (and in fact, our ABC implementation even obtained some better results than those reported in [28]). To ensure a fair comparison, we ran the compared algorithms on our computer under the same cutoff limits: 1, 10 and 60 minutes for Sets I, II and III, with the exception of 240 minutes for the largest instances with at least 7000 cities of Set III. Given the stochastic nature of the compared algorithms, we ran each algorithm 20 times independently to solve each instance with the above time limits.

In order to assess the gaps between the heuristic solutions (from GMA and the reference algorithms) and the optimal solutions, we also investigated the general mixed integer programming solver CPLEX (version 12.7) based on the binary linear programming model from [21] (see Appendix A). Our experiment indicated that CPLEX with this model can only solve optimally 10 smallest instances of Set I within 7200 seconds, but fails to solve any instance of Sets II and III due to memory overflow.

4.2 Parameter tuning

GMA requires 3 parameters: population size p, number of nearest cities N_n and parameter α. In order to identify a set of suitable parameters, we used the

[2] The code of our algorithm will be available at http://www.info.univ-angers.fr/pub/hao/CTSP.html

[3] Our implementation of ABC [28] is available from the link given in footnote 2.

popular 'IRACE' package [22] for automatic parameters tuning. The tuning was performed on 8 benchmark instances with 202 to 5397 cities. For the experiment, the tuning budget was set to 500 runs, each with a time limit of half of the cutoff time. The studied and final values (suggested by IRACE) of these parameters are shown in Table 1.

Table 1
Parameters tuning results

Parameters	Section	Description	Considered values	Final value
p	3.1	population size	{10,15,20,25,30}	20
N_n	3.3.1	number of nearest cities	{30,40,50,60,70,80,90}	50
α	3.3.1	maximum length of substring	{1,2,3,4,5,6,7}	7

4.3 Computational results and comparisons with existing algorithms

Computational results of GMA and the reference algorithms on set I are shown in Table 2. For CPLEX, we report for each instance the upper bound (UB), the lower bound (LB) and the *Gap* given by $(UB-LB)/LB\times 100$. So $Gap = 0$ implies that an optimal solution is found. Columns $6-17$ report respectively the results of re-ABC, ITPLS and GMA in terms of the best objective value f_{best} (over 20 runs), the average objective value f_{avg} , standard deviation σ and the average time in seconds to reach the best objective value (Time(s)). For the f_{best} and f_{avg} indicators, equally best values are shown in italic font.

Given that both the upper bounds and lower bounds are available for these instances, we include the geometric mean of each algorithm for a global assessment (row Geomean). For CPLEX, the geometric mean is calculated with the gaps between UB and LB by $(\prod_{i=1}^{h}\frac{UP_i}{LB_i})^{\frac{1}{h}}$ where UP_i and LB_i are the upper and lower bound of the ith instance, respectively. Similarly, for the other algorithms (re-ABC, ITPLS, and GMA), we calculate their geometric means for the best and average objective values by replacing UP_i with the f_{best} and f_{avg} values, respectively.

Finally, to assess the statistically significant difference between GMA and each main compared algorithm, Table 5 shows the *p-values* from the Wilcoxon signed-rank test. With a confidence level of 95%, a *p-value* smaller than 0.05 indicates a statistically significant difference between the pair of compared results.

From Table 2 on the 20 small instances of Set I, the following observations can be made. First, CPLEX is able to solve optimally the 10 smallest instances with $21-51$ cities and $2-4$ salesmen. For the remaining instances, the gap between UP and LP remains reasonable and tends to increase with the size of the instance. For the three heuristic algorithms, they perform equally well in terms of solution quality by reaching their best solutions consistently including the 10 optimal values. The geometric means indicates that the three heuristic algorithms can reach the same results in terms of both the best and average results. Meanwhile, the heuristic algorithms have smaller geometric means compared with CPLEX and thus perform better for this set of instances. In terms of computational efficiency, GMA and re-ABC perform better than ITPLS since they require significantly less computation times to reach the same results. It is worth mentioning that none of the other algorithms in the literature, such as GA [21] (2014), VNS [24][4] (2017), ACO [8] (2018), and ABC [9] (2019) can reach such a performance (they report worse results for some instances or their best results cannot be reached consistently).

Table 3 presents the results of the compared algorithms (re-ABC, ITPLS and GMA) on the 14 medium instances of Set II with $202-666$ cities and $10-40$ salesmen. In addition to the main reference algorithms re-ABC and ITPLS, we also include in this comparison ACO [8] for indicative purposes, which only reported results for six instances. For each algorithm except ACO, we show the best and average objective values (f_{best} and f_{avg}), the standard deviation (σ) and the average time to reach the best objective value (Time(s)). Equally best values are indicated in italic font, while strictly best values are highlighted in boldface. Moreover, the last column $Imp(\%)$ provides the percentage improvement of GMA's best result f_{best} over the best objective value f_{bk} of the reference algorithms, computed as $(f_{best} - f_{bk})/f_{bk} \times 100$. Thus a negative $Imp(\%)$ value indicates that GMA improved the best results of the reference algorithms. For Set II, we ignored the geometric means given that the lower bounds needed for their calculations are unavailable. In fact, we tried to obtain LB for these instances by solving, with CPLEX, the linear programming relaxation of the model presented in the Appendix. But CPLEX terminates abnormally due to memory overflow without proving any results or bounds.

Table 3 indicates that GMA finds better results for 7 out of the 14 instances, and matches the best results of the reference algorithms for 3 other instances. The Wilcoxon signed-rank test on the f_{best} and f_{avg} values in Table 5 also confirms that GMA significantly outperforms the two main reference algorithms. We do not insist on computation time because the main compared algorithms report solutions of different quality. Nevertheless, the three main compared algorithms (re-ABC, ITPLS and GMA) require comparable computation times to reach their best solutions. Note that the results of ACO [8] are somewhat inconsistent. Among the six instances tested by ACO, even if it reports three better results than the other algorithms (indicated with a star), its results for the three other instances as well as for most of the 20 small instances of Set I are considerably worse than algorithms, such as ABC [28] and ITPLS [14].

Table 4 presents the computational results of the compared algorithms for the

[4] VNS reports a wrong result of 465.28 for eil51-2 because this result is smaller than the proven optimal value of 478.08 from CPLEX.

Table 2. Comparative results of GMA and reference algorithms on Set I. The equally best values are indicated in italic.

	CPLEX				re-ABC				ITPLS				GMA (this work)			
Instance	UB	LB	t(s)	*Gap*(%)	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)
eil21-2	*144.92*	144.92	1	0.00	*144.92*	144.92	0.00	1.00	*144.92*	144.92	0.0	18.32	*144.92*	144.92	0.0	1.00
eil21-3	*157.48*	157.48	1	0.00	*157.48*	157.48	0.00	1.00	*157.48*	157.48	0.0	13.15	*157.48*	157.48	0.0	1.00
eil31-2	*259.36*	259.36	2	0.00	*259.36*	259.36	0.00	1.00	*259.36*	259.36	0.0	12.70	*259.36*	259.36	0.0	1.00
eil31-3	*295.31*	295.31	20	0.00	*295.31*	295.31	0.00	1.00	*295.31*	295.31	0.0	12.85	*295.31*	295.31	0.0	1.00
eil31-4	*315.97*	315.97	61	0.00	*315.97*	315.97	0.00	1.00	*315.97*	315.97	0.0	16.90	*315.97*	315.97	0.0	1.00
eil41-2	*346.24*	346.24	7	0.00	*346.24*	346.24	0.00	1.00	*346.24*	346.24	0.0	14.45	*346.24*	346.24	0.0	1.00
eil41-3	*367.84*	367.84	46	0.00	*367.84*	367.84	0.00	1.00	*367.84*	367.84	0.0	22.05	*367.84*	367.84	0.0	1.00
eil41-4	*392.14*	392.14	120	0.00	*392.14*	392.14	0.00	1.00	*392.14*	392.14	0.0	11.55	*392.14*	392.14	0.0	1.00
eil51-2	*478.08*	478.08	126	0.00	*478.08*	478.08	0.00	1.05	*478.08*	478.08	0.0	21.55	*478.08*	478.08	0.0	1.00
eil51-3	*469.50*	469.50	773	0.00	*469.50*	469.50	0.00	1.00	*469.50*	469.50	0.0	20.40	*469.50*	469.50	0.0	1.00
eil51-4	*489.99*	485.88	7201	0.85	*489.99*	489.99	0.00	1.00	*489.99*	489.99	0.0	28.55	*489.99*	489.99	0.0	1.40
eil51-5	*525.98*	503.84	7212	4.39	*525.98*	525.98	0.00	1.10	*525.98*	525.98	0.0	14.35	*525.98*	525.98	0.0	1.00
eil76-3	596.18	583.41	7211	2.19	*593.28*	593.28	0.00	1.00	*593.28*	593.28	0.0	16.40	*593.28*	593.28	0.0	1.00
eil76-4	*603.79*	585.69	7202	3.09	*603.79*	603.79	0.00	1.50	*603.79*	603.79	0.0	17.75	*603.79*	603.79	0.0	1.60
eil76-5	656.56	620.25	7206	5.85	*651.99*	651.99	0.00	1.00	*651.99*	651.99	0.0	6.75	*651.99*	651.99	0.0	1.00
eil76-6	687.43	624.25	7202	10.12	*672.73*	672.73	0.00	2.10	*672.73*	672.73	0.0	31.40	*672.73*	672.73	0.0	1.00
eil101-4	746.93	697.83	7204	7.04	*726.82*	726.82	0.00	1.30	*726.82*	726.82	0.0	18.45	*726.82*	726.82	0.0	1.00
eil101-5	854.23	750.92	7203	13.76	*779.15*	779.15	0.00	1.05	*779.15*	779.15	0.0	10.80	*779.15*	779.15	0.0	1.00
eil101-6	783.08	706.47	7209	10.84	*759.55*	759.55	0.00	1.25	*759.55*	759.55	0.0	12.75	*759.55*	759.55	0.0	1.70
eil101-7	840.60	729.92	7201	15.16	*798.85*	798.85	0.00	1.20	*798.85*	798.85	0.0	12.56	*798.85*	798.85	0.0	1.80
Geomean	1.0335	-	-	-	1.0235	1.0235	-	-	1.0235	1.0235	-	-	1.0235	1.0235	-	-

Table 3. Comparative results of GMA and reference algorithms on Set II. Equally best values are indicated in italic. The strictly best values are indicated in boldface.

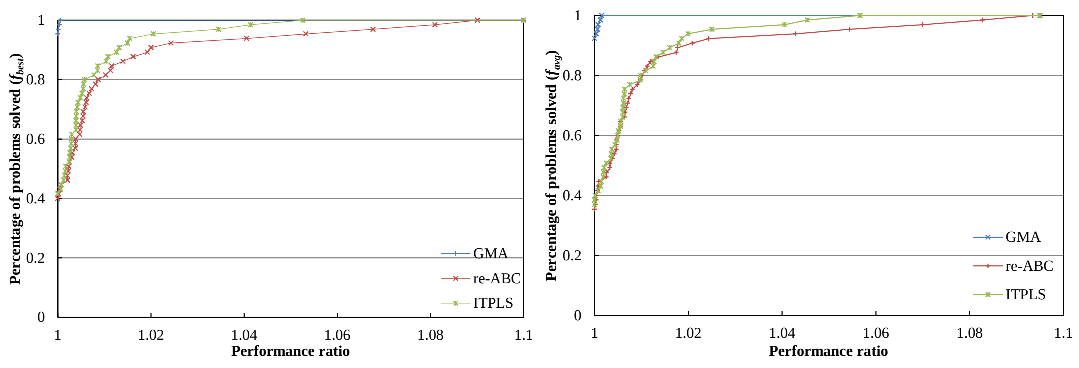
		re-ABC [14]				ITPLS[14]				GMA (this work)				
Instance	ACO [8]	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)	Imp(%)
gr202-12	**71924.00***	99871.00	100033.20	110.54	329.45	99871.00	100009.50	112.58	93.55	99871.00	100162.50	185.46	396.65	39.00
gr202-25	**99606.00***	173547.00	173596.80	54.01	348.85	173418.00	173523.80	46.77	141.25	173477.00	173594.65	75.72	211.80	74.00
gr202-35	**118495.00***	233749.00	233817.85	70.16	316.15	233749.00	233857.80	73.17	156.85	233871.00	234003.35	72.81	93.85	97.00
gr229-10	-	*222167.00*	222354.85	164.08	244.60	*222167.00*	222347.65	103.50	226.70	*222167.00*	222173.75	30.19	201.90	0.00
gr229-15	-	*264146.00*	264146.00	0.00	69.60	*264146.00*	264146.00	0.00	67.00	*264146.00*	264146.00	0.00	154.85	0.00
gr229-20	-	*319669.00*	319669.00	0.00	303.05	*319669.00*	319671.90	12.97	128.20	*319669.00*	319880.15	547.77	83.20	0.00
gr229-30	-	*406664.00*	407194.85	375.21	301.35	*406664.00*	406884.00	225.72	186.20	406701.00	407389.75	279.37	73.10	0.01
gr431-12	330554.00	249031.00	249682.25	293.07	300.25	249421.00	250036.95	613.23	221.45	**248447.00**	248447.00	0.00	29.50	-0.23
gr431-25	464298.00	348056.00	348431.10	203.82	333.30	348181.00	349238.10	417.38	253.70	**347335.00**	347559.80	420.13	394.95	-0.21
gr431-40	483977.00	416189.00	416758.40	249.58	355.20	416552.00	417963.75	958.14	296.90	**415314.00**	415387.45	88.31	159.45	-0.21
gr666-10	-	390188.00	392234.00	971.38	159.45	389583.00	396841.55	2716.00	485.90	**387562.00**	389594.80	3417.34	473.55	-0.52
gr666-15	-	448604.00	449997.35	716.97	248.15	448257.00	449635.25	800.17	223.60	**446475.00**	447123.60	328.49	506.20	-0.40
gr666-20	-	522157.00	523583.15	937.90	177.55	521149.00	522650.90	1006.57	249.50	**519121.00**	519773.45	397.47	512.25	-0.39
gr666-30	-	652587.00	654001.50	633.57	224.80	651801.00	653318.10	927.19	255.05	**650116.00**	650974.90	417.87	535.70	-0.26

Table 4. Comparative results of GMA and reference algorithms on Set III. The strictly best values are indicated in boldface.

	re-ABC [14]				ITPLS [14]				GMA (this work)				
Instance	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)	f_{best}	f_{avg}	σ	Time(s)	$Imp(\%)$
pr1002-5	316437.00	317425.40	479.59	1121.95	318587.00	320348.80	1058.72	2053.45	**313885.00**	314083.20	128.79	1868.40	-0.81
pr1002-10	382201.00	382844.90	423.48	843.40	383112.00	384908.55	936.35	1615.60	**379846.00**	379911.00	162.49	1361.65	-0.62
pr1002-20	516256.00	517481.55	452.81	1606.00	517917.00	519664.85	794.31	1683.05	**514968.00**	515784.55	498.49	1815.70	-0.25
pr1002-30	664648.00	665676.30	559.24	2213.60	664308.00	666702.20	929.80	1753.65	**661540.00**	662613.10	958.81	1792.00	-0.42
pr1002-40	806022.00	807838.65	786.79	1757.50	805967.00	808503.35	1444.57	1946.35	**803624.00**	803642.45	74.24	953.10	-0.29
fnl2461-3	114188.00	114509.80	145.77	3562.50	110007.00	110553.50	413.84	2773.75	**105637.00**	105754.90	43.78	1092.55	-3.97
fnl2461-6	122312.00	122612.30	188.81	3593.50	118513.00	119199.15	387.44	2273.70	**116128.00**	116287.05	77.36	1882.75	-2.01
fnl2461-12	145800.00	146374.85	220.68	3597.65	145023.00	145688.60	284.37	3128.30	**143477.00**	143866.10	214.32	3410.40	-0.99
fnl2461-24	222465.00	223335.30	456.26	2231.65	221494.00	221739.80	163.09	3039.95	**221116.00**	221317.35	121.79	3376.65	-0.17
fnl2461-30	268431.00	269140.30	535.88	2188.65	267355.00	267593.85	169.31	2842.00	**267017.00**	267249.45	116.69	3206.00	-0.13
fnl3461-3	162335.00	162909.20	177.47	3504.05	156753.00	157420.50	391.84	2900.75	**148917.00**	148979.55	33.13	1744.75	-5.00
fnl3461-12	170762.00	171243.80	238.70	3612.85	165455.00	166525.25	512.43	3034.60	**159934.00**	160040.70	63.09	1928.10	-3.34
fnl3461-12	192874.00	193582.75	336.98	3612.90	188223.00	188963.25	371.47	3293.00	**185363.00**	185621.60	143.67	3188.80	-1.52
fnl3461-24	266686.00	267130.75	187.32	3504.60	265078.00	265672.70	342.83	3045.55	**263631.00**	263980.40	177.39	3405.60	-0.53
fnl3461-30	308742.00	308963.10	95.74	2526.10	307562.00	308018.40	208.49	3014.70	**307071.00**	307252.30	113.62	3268.00	-0.17
fnl3461-40	385443.00	385727.50	120.94	2313.95	385122.00	385296.60	113.73	2827.00	**384573.00**	384722.95	77.96	3310.50	-0.14
pla5397-20	38335000	38392330	26980.76	3504.15	38331500	38494950	73265.21	3483.15	**38006100**	38018450	25354.72	3059.70	-0.85
pla5397-30	51299400	51340355	22848.87	3205.35	51339600	51451470	82834.36	3403.30	**51138000**	51143260	3180.10	3174.00	-0.31
pla5397-40	64408200	64476755	30612.77	3480.90	64285900	64404060	61216.52	3437.45	**64097900**	64121760	18498.09	2854.40	-0.29
pla5397-50	74008700	74019335	5148.66	2359.50	74051200	74145910	44659.82	3407.50	**73993600**	73993610	30.78	2709.10	-0.02
pla5397-60	85303100	85324645	10454.99	2161.00	85323100	85424400	60048.45	3046.95	**85266200**	85266750	235.08	3116.55	-0.04
pla6397-20	36672000	36748165	37220.64	3608.95	36404600	36575675	103990.10	3337.10	**35951800**	35997920	19571.59	3062.95	-1.24
pla6397-30	47689800	47750055	24229.16	3457.40	47551800	47832460	98829.28	3175.25	**47346400**	47368155	12197.91	3141.80	-0.43
pla6397-40	56948400	57022520	31690.54	3400.30	56860500	56945530	54061.04	3040.75	**56638000**	56653280	11345.84	3096.10	-0.40
pla6397-50	67415000	67485965	27014.35	3077.95	67347700	67419380	40122.95	2983.45	**67161500**	67171190	7667.49	3018.50	-0.28
pla6397-60	75077200	75118385	16341.50	3314.70	74983600	75063660	52339.27	3099.85	**74791200**	74803075	10330.12	3235.45	-0.26
pla7397-20	42262900	42432435	78855.53	14431.45	41804200	42027405	138563.86	13750.30	**41260500**	41422195	70887.70	12204.65	-1.11
pla7397-30	53648400	53717345	45595.94	14017.35	53183700	53358655	113523.08	13778.95	**52636900**	52780890	88332.83	12932.47	-0.93
pla7397-40	65847100	65919250	42106.65	14072.90	65441600	65662845	142232.68	13643.15	**64937200**	65029520	58826.59	12906.25	-0.77
pla7397-50	77194500	77265730	38981.29	14064.05	76701700	76784335	74134.09	13748.85	**76331100**	76406770	47628.33	12635.55	-0.44
pla7397-60	87041500	87103321	35045.55	13562.42	86628200	86749490	77747.32	13763.90	**86153700**	86224380	48429.67	13182.35	-0.58

31 large instances of Set III ($1002 - 7397$ cities and $3 - 60$ salesmen) with the same information as in Table 3. These results clearly show the dominance of the proposed GMA algorithm over the reference algorithms for these large instances, by systematically reporting better results in terms of the best and the average objective values. Moreover, GMA requires the shortest computation times to reach its solutions for this set of large instances, demonstrating its remarkable search capacity and high computational efficiency. According to the *p-values* (less than 0.05) from the Wilcoxon signed-rank test shown in Table 5, the difference between GMA and each compared algorithm is statistically significant.

Tables 2-4 demonstrate the high competitiveness of the proposed GMA algorithm compared with the state-of-the-art algorithms for solving the existing CTSP benchmark instances. Its superiority becomes more evident when medium and large instances are solved. GMA reports improved best-known results (new upper bounds) for 7 medium instances of Set II and all 31 large instances of Set III, which are useful for future research on CTSP.



(a) (b)

Fig. 4. Performance profiles of GMA and two reference algorithms on the 65 instances of sets I, II, and III. The left part corresponds to the best results while the right part is for the average results.

For a more intuitive illustration of the performance assessment of the algorithms, we use the *performance profile* [7], which is a popular benchmarking tool for rigorous comparison of different algorithms. In general, *performance profiles* adopt a specific performance metric (in our case, we use f_{best} and f_{avg}) on all sets of instances. To compare a set of algorithms X over a set of problems Q, the performance ratio is defined by $r_{x,q} = \frac{f_{x,q}}{min\{f_{x,q:x \in X, q \in Q}\}}$. If an algorithm x does not report result for a problem q, $r_{x,q} = +\infty$. The performance function of an algorithm x is computed by $Q_x(\tau) = \frac{|q \in Q| r_{x,q} \leq \tau|}{|Q|}$. The value $Q_x(\tau)$ computes the fraction of problems that the algorithm x can solve with at most τ many times the cost of the best algorithm. For example, $Q_x(1)$ equals the number of problems that algorithm x solved better than, or as good as the other algorithms in Q. Similarly, the value $Q_x(r_f)$ is the maximum number of problems that algorithm x solved. Therefore, $Q_x(1)$ and $Q_x(r_f)$ represent the efficiency and robustness of algorithm x, respectively.

Fig. 4 shows the performance profiles of GMA, ITPLS and re-ABC. We observe that GMA dominates the reference algorithms in terms of the best and average values. Indeed, GMA has a much higher $Q_x(1)$ value compared with the reference algorithms, indicating that GMA can find better or equal results for all instances. Furthermore, GMA reaches $Q_x(r_f)$ firstly, indicating a high robustness of our algorithm.

Table 5
Summary of comparative results between GMA and two reference algorithms

Algorithm pair	Set/Instance	Indicator	Better	Equal	Worse	$p - value$
GMA vs. ITPLS	I/20	f_{best}	0	20	0	0.00E+00
		f_{avg}	0	20	0	0.00E+00
	II/14	f_{best}	7	4	3	2.44E-04
		f_{avg}	8	1	5	4.80E-02
	III/31	f_{best}	31	0	0	1.17E-06
		f_{avg}	31	0	0	1.17E-06
GMA vs. re-ABC	I/20	f_{best}	0	20	0	0.00E+00
		f_{avg}	0	20	0	0.00E+00
	II/14	f_{best}	8	4	2	1.37E-02
		f_{avg}	9	1	4	4.79E-02
	III/31	f_{best}	31	0	0	1.17E-06
		f_{avg}	31	0	0	1.17E-06

Finally, Table 5 summarizes the results reported by the compared algorithms on the three sets of 65 instances. Column 2 gives the set name and the number of instances in the set. Column 3 shows the quality indicators (f_{best} and f_{avg}). Columns 4-6 count the number of instances for which GMA achieves a better, equal or worse value compared with each reference algorithm. The last column presents the *p-values* from the Wilcoxon signed-rank test. Table 5 reveals large performance gaps between GMA and each reference algorithm on Sets II and III. We conclude that GMA is very competitive for solving CTSP and this is particularly true for large instances.

5 Discussion and analysis

5.1 Benefit of the key components

In this section, we justify the design choices behind the proposed GMA algorithm. For this, we investigate the impacts of its key components: Constrained Cross-exchange, EAX as well as backbone-based crossover. For our experiments, we used the 45 instances of Sets II and III and ignored the instances of Set I. Indeed, for the instances of Set I, their best-known results can be consistently reached by the state-of-the-art algorithms including ABC, ITPLS and GMA within a very short time. As such, these instances are too easy to be used to compare algorithm variants.

Table 6
Comparative results on Set II between GMA and its three variants. Strictly best values are shown in boldface.

Instance	GMA		GMA$_0$		GMA$_1$		GMA$_2$	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
gr202-12	**99871.00**	100162.50	100292.00	100722.85	100196.00	100573.25	**99871.00**	100217.90
gr202-25	173477.00	173594.65	173782.00	173828.55	**173394.00**	173681.40	173511.00	173643.90
gr202-35	233871.00	234003.35	234126.00	234226.30	233907.00	234074.20	**233749.00**	233948.20
gr229-10	**222167.00**	222173.75	**222167.00**	222255.75	223266.00	224262.85	**222167.00**	222167.00
gr229-15	**264146.00**	264146.00	264224.00	265715.20	265537.00	266727.75	264183.00	264186.90
gr229-20	**319669.00**	319880.15	320976.00	322435.20	**319669.00**	320910.90	**319669.00**	320424.30
gr229-30	**406701.00**	407389.75	407692.00	408434.25	407962.00	408942.80	407226.00	407648.55
gr431-12	**248447.00**	248447.00	**248447.00**	248462.10	252253.00	254230.25	**248447.00**	248447.00
gr431-25	**347335.00**	347559.80	347545.00	348599.25	350446.00	351721.50	347459.00	347800.20
gr431-40	415314.00	415387.45	**415280.00**	415560.35	416983.00	419148.30	415280.00	415342.00
gr666-10	387562.00	389594.80	392586.00	395300.20	399415.00	404852.60	**387321.00**	388644.95
gr666-15	**446475.00**	447123.60	449908.00	452081.65	453684.00	459695.45	446839.00	447329.35
gr666-20	519121.00	519773.45	523090.00	525733.00	523178.00	526787.50	**519071.00**	520190.70
gr666-30	**650116.00**	650974.90	653524.00	656015.75	652608.00	655150.05	651330.00	652005.05

Table 7
Comparative results on Set III between GMA and its three variants. Strictly best values are indicated in boldface.

Instance	GMA		GMA$_0$		GMA$_1$		GMA$_2$	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
pr1002-5	313885.00	314083.20	314065.00	314495.30	324126.00	327673.20	**313867.00**	313946.15
pr1002-10	**379846.00**	379911.00	380489.00	380656.65	388221.00	391193.95	**379846.00**	379920.85
pr1002-20	514968.00	515784.55	515927.00	516825.70	522573.00	524484.30	**513814.00**	515288.60
pr1002-30	**661540.00**	662613.10	663314.00	664050.80	666270.00	669233.55	**661540.00**	662284.40
pr1002-40	**803624.00**	803642.45	804971.00	805937.95	808207.00	810596.10	**803624.00**	804251.20
fnl2461-3	**105637.00**	105754.90	105793.00	105943.85	112896.00	113202.90	**105637.00**	105753.75
fnl2461-6	**116128.00**	116287.05	116531.00	116761.30	120621.00	121369.65	116173.00	116273.20
fnl2461-12	**143477.00**	143866.10	146060.00	146259.00	145953.00	146626.45	143739.00	144100.30
fnl2461-24	**221116.00**	221317.35	225527.00	226078.10	222280.00	222706.05	221167.00	221439.10
fnl2461-30	**267017.00**	267249.45	271199.00	271586.55	267799.00	268144.30	267296.00	267441.70
fnl3461-3	**148917.00**	148979.55	148957.00	149065.90	160427.00	161053.00	**148917.00**	148978.25
fnl3461-6	159934.00	160040.70	160181.00	160340.95	169106.00	169983.65	**159906.00**	160052.60
fnl3461-12	**185363.00**	185621.60	186394.00	186910.25	192212.00	192988.60	185652.00	185814.80
fnl3461-24	**263631.00**	263980.40	267515.00	267723.05	267007.00	267684.65	263763.00	264050.00
fnl3461-30	307071.00	307252.30	310275.00	310787.45	308936.00	309526.70	**306991.00**	307148.20
fnl3461-40	**384573.00**	384722.95	387335.00	387810.05	385794.00	385981.80	384611.00	384752.45
pla5397-20	38006100	38018450	38049400	38084660	38527700	38625835	**38006000**	38013320
pla5397-30	**51138000**	51143260	51297400	51353685	51294600	51385805	51141700	51148700
pla5397-40	**64097900**	64121760	64337200	64420905	64192200	64258245	64100100	64148815
pla5397-50	**73993600**	73993610	73993700	73993870	74048200	74119855	**73993600**	73993615
pla5397-60	**85266200**	85266750	85269600	85281395	85347400	85397905	85266900	85267150
pla6397-20	35951800	35997920	36298800	36344680	36502300	36650005	**35945200**	36020685
pla6397-30	**47346400**	47368155	47646200	47680965	47555300	47699290	47390500	47418730
pla6397-40	**56638000**	56653280	56881800	56921780	56828400	56884080	56661500	56688900
pla6397-50	**67161500**	67171190	67297700	67370130	67293800	67359825	67181100	67207485
pla6397-60	**74791200**	74803075	74941900	74988450	74935800	74986940	74814700	74845785
pla7397-20	**41260500**	41422195	42063000	42217895	41653000	41817975	41446600	41501520
pla7397-30	**52636900**	52780890	53655100	53801700	52953600	53082005	52810400	52893300
pla7397-40	**64937200**	65029520	66045400	66163565	65151400	65267450	64993500	65070450
pla7397-50	**76331100**	76406770	77166800	77274005	76467400	76569390	76344200	76422895
pla7397-60	**86153700**	86224380	87017600	87097055	86243600	86382840	86171600	86233840

5.1.1 Benefit of constrained cross-exchange

To highlight the benefit of the constrained cross-exchange (CCE, Section 3.3.1), we compared GMA with a variant GMA$_0$, where CCE is removed from

LOE. In other words, only EAX is employed in GMA$_0$ in the local optima exploration component.

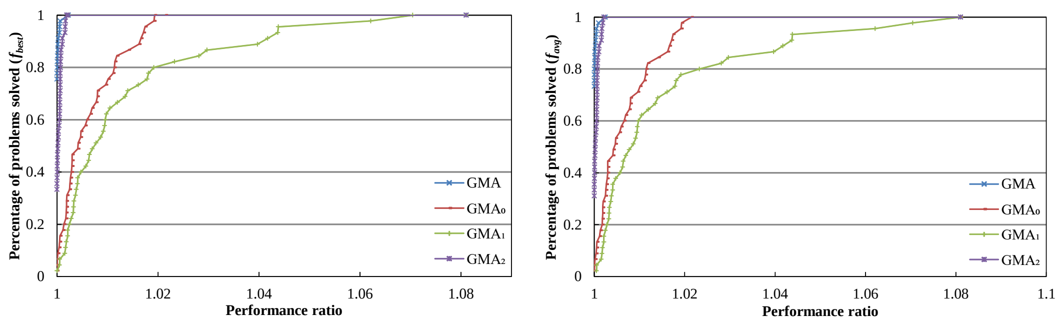
Computational results of GMA and GMA$_0$ are shown in Tables 6 and 7 and summarized in Table 8 and Fig. 5. The results indicate that GMA performs significantly better than GMA$_0$ in terms of f_{best} and f_{avg}. For f_{best}, GMA dominates GMA$_0$ by getting 42 better results out of the 45 tested instances and reporting only one worse result. Furthermore, the statistically significant difference between GMA and GMA$_0$ is verified by the Wilcoxon singed-rank test with a 95% level of confidence in Table 8. Therefore, this experiment confirms the usefulness of CCE for the GMA algorithm.

5.1.2 Benefit of EAX

To assess the benefit of EAX in LOE, we created another variant GMA$_1$ in which EAX is replaced by 2-opt [17] for individual route optimization. GMA$_1$ shares the other ingredients of GMA.

From the results in Tables 6 and 7, we observe that GMA significantly outperforms GMA$_1$ on all instances, except gr202-25. For gr202-25, the best result of GMA$_1$ is slightly better than GMA. Furthermore, the small *p-value* (less than 0.05) in Table 8 from the Wilcoxon singed-rank test attests the significant difference between GMA and GMA$_1$. Moreover, the performance profiles of Fig. 5 indicate that GMA surpasses GMA$_1$ in terms of f_{best} and f_{avg}. Indeed, GMA arrives at $Q_x(r_f)$ firstly, much earlier than GMA$_1$. These observations illustrate the benefit of EAX in GMA.

5.1.3 Benefit of backbone-based crossover



(a) (b)

Fig. 5. Performance profiles of GMA and its three variants on the set of 45 selected instances. The left part corresponds to the best results while the right part is for the average results.

To study the effectiveness of the backbone-based crossover, we compared GMA with a third variant GMA$_2$. In GMA$_2$, the backbone-based crossover is replaced by a crossover proposed by Singh and Baghel [30], which was designed for the related MTSP problem. This crossover selects one of the two parents uniformly at random and copies, from the parent to the offspring, the most promising route (i.e., the route having the smallest ratio of route length to the number of cities in that route). Then all the cities belonging to the route are deleted from both parents by connecting the predecessor of each city to its successor, and the length of the route is updated accordingly.

From the comparative results (f_{best} and f_{avg}) of GMA and GMA$_2$ in Tables 6 and 7, we observe that in terms of f_{best}, GMA dominates GMA$_2$ by acquiring 25 better results, 10 equal results and 10 worse results. The Wilcoxon signed-rank test, shown in Table 8, also confirms that GMA outperforms significantly GMA$_2$ on the large instances (set III). This experiment demonstrates that the backbone-based crossover operator contributes positively to the performance of GMA, in particular for solving large instances.

Finally, Fig. 5 provides other useful information for the importance of each ingredient of GMA. For example, GMA$_1$ performs the worst because it has the worst (smallest) $Q_x(1)$ value and reaches $Q_x(r_f)$ lastly. Therefore, we can summarize that EAX is the most important component of GMA, followed by CCE, finally the backbone-based crossover operator.

Table 8
Summary of comparative results between GMA and and its three variants

Algorithm pair	Set/Instance	Indicator	Better	Equal	Worse	$p - value$
GMA vs. GMA$_0$	II/14	f_{best}	11	2	1	9.77E-04
		f_{avg}	14	0	0	1.22E-04
	III/31	f_{best}	31	0	0	1.17E-06
		f_{avg}	31	0	0	1.17E-06
GMA vs. GMA$_1$	II/14	f_{best}	12	1	1	7.32E-04
		f_{avg}	14	0	0	1.22E-04
	III/31	f_{best}	31	0	0	1.17E-06
		f_{avg}	31	0	0	1.17E-06
GMA vs. GMA$_2$	II/14	f_{best}	6	4	4	3.34E-01
		f_{avg}	9	1	4	9.42E-02
	III/31	f_{best}	19	6	6	1.60E-03
		f_{avg}	23	0	8	9.94E-04

5.2 Influences of selection, pool updating and mutation

In addition to the local optimization and crossover components, the performance of a memetic algorithm such as GMA could be influenced by other factors such as parent selection, pool updating and mutation. According to our experiments with the roulette-wheel selection strategy and the rank-pool updating strategy [39], no significant changes were observed regarding the performance of the GMA algorithm. In this subsection, we focus on studying the influence of mutation. Specifically, when the best solution φ^* is not improved for $maxNoImpor$ consecutive iterations (we empirically set $maxNoImpor = 50$), the search is judged to be stagnating. Then a mutation operator is triggered to modify one third of the solutions in the population (i.e., each solution is mutated with an equal probability of 1/3). The mutation consists of displacing a certain number of randomly and uniformly chosen shared cities. To be specific, for a solution to be mutated, $|S| \times 0.3$ shared cities are first removed, leading to a partial solution. Then these removed shared cities are inserted into the partial solution one by one, using the second step of the greedy randomized heuristic to minimize the distance increase. After that, each mutated solution is optimized by the local optima exploration procedure of Section 3.3. Comparative results of GMA and the GMA variant extended with the mutation (called GMA$_3$) are shown in Tables 9 and 10.

Table 9
Comparative results on Set II between GMA and GMA$_3$ (with mutation). Strictly best values are indicated in boldface.

	GMA			GMA$_3$		
Instance	f_{best}	f_{avg}	σ	f_{best}	f_{avg}	σ
gr202-12	99871.00	100162.50	185.46	99871.00	100136.95	201.03
gr202-25	173477.00	173594.65	75.72	**173358.00**	173569.15	100.05
gr202-35	233871.00	234003.35	72.81	233871.00	233973.20	91.05
gr229-10	222167.00	222173.75	30.19	222167.00	222167.00	0.00
gr229-15	264146.00	264146.00	0.00	264146.00	264147.85	8.27
gr229-20	319669.00	319880.15	547.77	319669.00	319776.90	332.11
gr229-30	406701.00	407389.75	279.37	**406664.00**	407333.55	320.53
gr431-12	248447.00	248447.00	0.00	248447.00	248447.00	0.00
gr431-25	347335.00	347559.80	420.13	347335.00	347565.85	441.71
gr431-40	415314.00	415387.45	88.31	415314.00	415364.05	73.31
gr666-10	387562.00	389594.80	3417.34	**387459.00**	389350.40	2939.49
gr666-15	446475.00	447123.60	328.49	**446322.00**	447109.50	345.89
gr666-20	519121.00	519773.45	397.47	519121.00	519664.20	360.33
gr666-30	650116.00	650974.90	417.87	650116.00	650894.35	369.63
Best/All	0/14	2/14	-	**4/14**	**11/14**	-
p-value	-	-	-	1.25E-01	1.20E-03	-

The results indicate that in terms of f_{best}, GMA$_3$ outperforms GMA by obtaining 13 better results, 28 equal results and 4 worse results. However, the Wilcoxon signed-rank test shows that there is no statistically significant difference. On the contrary, in terms of f_{avg}, GMA$_3$ dominates GMA by acquiring 34 better results, 2 equal results and 9 worse results. The Wilcoxon signed-rank test confirms that GMA$_3$ significantly outperforms GMA. This experiment indicates that the mutation strategy can indeed improve the performance of GMA. Especially, it significantly reinforces the stability of the algorithm.

5.3 Convergence analysis

Finally, we investigate the convergence behaviors of GMA (and the GMA$_3$ variant with mutation) and two key reference algorithms (re-ABC and ITPLS). For this study, we acquired the running profiles of these algorithms on two representative instances of Set II (gr431-25, gr666-30). We ran each algorithm 20 times with the cutoff time of 600 seconds per run and recorded the best objective values during the process. The results of this experiment are shown in Fig. 6.

Table 10
Comparative results on Set III between GMA and GMA$_3$ (with mutation). Strictly best values are indicated in boldface.

	GMA			GMA$_3$		
Instance	f_{best}	f_{avg}	σ	f_{best}	f_{avg}	σ
pr1002-5	313885.00	314083.20	128.79	313885.00	314106.55	127.60
pr1002-10	379846.00	379911.00	162.49	379846.00	379902.00	140.98
pr1002-20	514968.00	515784.55	498.49	**514244.00**	515655.65	586.90
pr1002-30	661540.00	662613.10	958.81	661540.00	662422.80	710.68
pr1002-40	803624.00	803642.45	74.24	803624.00	803624.00	0.00
fnl2461-3	105637.00	105754.90	43.78	105637.00	105755.90	43.10
fnl2461-6	116128.00	116287.05	77.36	116128.00	116278.25	75.96
fnl2461-12	143477.00	143866.10	214.32	143477.00	143811.60	173.59
fnl2461-24	221116.00	221317.35	121.79	**221105.00**	221299.40	110.59
fnl2461-30	267017.00	267249.45	116.69	267017.00	267230.20	102.23
fnl3461-3	148917.00	148979.55	33.13	148917.00	148979.55	33.13
fnl3461-12	159934.00	160040.70	63.09	159934.00	160035.25	65.67
fnl3461-12	185363.00	185621.60	143.67	185363.00	185605.90	136.94
fnl3461-24	**263631.00**	263980.40	177.39	263672.00	263972.35	162.72
fnl3461-30	307071.00	307252.30	113.62	**307026.00**	307233.95	122.99
fnl3461-40	384573.00	384722.95	77.96	384573.00	384720.40	79.81
pla5397-20	38006100	38018450	25354.72	38006100	38018355	25426.04
pla5397-30	51138000	51143260	3180.10	51138000	51142525	2790.75
pla5397-40	64097900	64121760	18498.09	64097900	64120030	16710.07
pla5397-50	73993600	73993610	30.78	73993600	73993605	22.36
pla5397-60	**85266200**	85266750	235.08	85266300	85266710	202.35
pla6397-20	35951800	35997920	19571.59	35951800	36000990	20031.03
pla6397-30	47346400	47368155	12197.91	47346400	47370840	12919.44
pla6397-40	56638000	56653280	11345.84	**56635600**	56653405	12491.62
pla6397-50	67161500	67171190	7667.49	**67158800**	67170200	7372.00
pla6397-60	74791200	74803075	10330.12	**74788500**	74801030	9630.71
pla7397-20	**41260500**	41422195	70887.70	41311800	41425215	57057.91
pla7397-30	**52636900**	52780890	88332.83	52672800	52781955	71085.17
pla7397-40	64937200	65029520	58826.59	**64926700**	65019290	57437.00
pla7397-50	76331100	76406770	47628.33	**76306200**	76391380	47713.59
pla7397-60	86153700	86224380	48429.67	**86121900**	86210057.89	53419.75
Best/All	4/31	7/31	-	**9/31**	**23/31**	-
p-value	-	-	-	3.31E-01	3.68E-02	-

One notices first that the curves of the population-based GMA and GMA$_3$ do not start at time 0. This is because that these algorithms spent a non-negligible portion of the time on generating the initial population (around 60 and 100 seconds for gr431-25 and gr666-30, respectively). From Fig. 6, one observes that re-ABC and ITPLS improve their solution quality quickly at the beginning of the search, and slow down or even stagnate as the time going. For GMA and GMA$_3$, the population initialization step allowed them to start the search with high-quality solutions. The best solution in the population is continually updated when the time goes on, implying that GMA and GMA$_3$ can better benefit from the allowed time to improve their solutions.

6 Conclusions

The colored traveling salesmen problem is a relevant variant of the popular traveling salesmen problem and generalizes the well-known multiple traveling salesmen problem. In this work, we presented the first grouping memetic al-

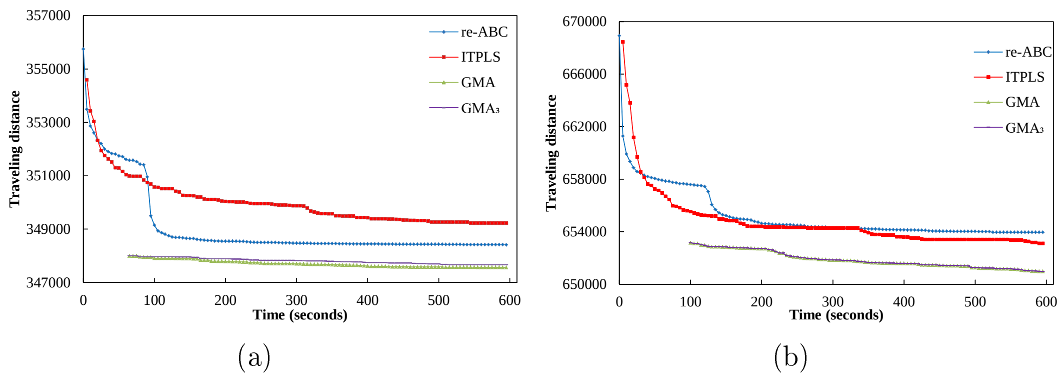


Fig. 6. Convergence charts (running profiles) of re-ABC, ITPLS, GMA and GMA$_3$ for solving two representative instances (gr431-25 and gr666-30). The results were obtained from 20 independent executions of each compared algorithms

gorithm for solving CTSP. The algorithm relies on a specific backbone-based crossover to generate promising offspring solutions by solution recombination and a powerful local optima exploration for offspring improvement. Extensive computational results on three sets of 65 benchmark instances in the literature indicate that our algorithm is very competitive compared with existing leading algorithms. In particular, it reports 38 new upper bounds while matching 24 best-known results. We also investigated the interest of CPLEX for solving CTSP and reported 10 proven optimal solutions for the first time.

For future work, there are several perspectives. First, the cross-exchange operator used for inter-route optimization has a high time complexity. This implies that the local optimization component of the proposed algorithm is time consuming. As such, for a given time unit (e.g., a short cutoff time), the algorithm will not be able to sample many candidate solutions, limiting thus its performance. To cope with this problem, one possible way is to reduce the number of candidate solutions considered by CCE. To this end, it is interesting to investigate the idea of neighborhood pruning that proves to be successful for vehicle routing [2,34] and TSP [17]. Second, recent research on using learning technique to reinforce optimization algorithm showed interesting results (e.g. [38]). As such, it would be useful to study in depth hybrid approaches that combine learning strategies and GMA. Third, CTSP is strongly related to MTSP and TSP, for which powerful algorithms exist. Ideas of these algorithms could be useful for solving CTSP. Finally, to the best of our knowledge, no dedicated exact algorithm exists for CTSP. Efforts are needed to fill the gap.

Acknowledgments

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. We thank Prof. Jun Li and Prof. Alok Singh for providing their test instances. Support from the China Scholarship Council (CSC, No. 201906850087) for the first author is also acknowledged.

References

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, The Traveling Salesman Problem: a Computational Study, Princeton University Press, 2006.

[2] F. Arnold, K. Sörensen, Knowledge-guided local search for the vehicle routing problem, Computers & Operations Research 105 (2019) 32–46.

[3] T. Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures, Omega 34 (3) (2006) 209–219.

[4] A. E. Carter, C. T. Ragsdale, A new approach to solving the multiple traveling salesperson problem using genetic algorithms, European Journal of Operational Research 175 (1) (2006) 246–257.

[5] D. Cattaruzza, N. Absi, D. Feillet, T. Vidal, A memetic algorithm for the multi trip vehicle routing problem, European Journal of Operational Research 236 (3) (2014) 833–848.

[6] P. Chen, H.-K. Huang, X.-Y. Dong, Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem, Expert Systems with Applications 37 (2) (2010) 1620–1627.

[7] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2) (2002) 201–213.

[8] X. Dong, W. Dong, Y. Cai, Ant colony optimisation for coloured travelling salesman problem by multi-task learning, IET Intelligent Transport Systems 12 (8) (2018) 774–782.

[9] X. Dong, Q. Lin, M. Xu, Y. Cai, Artificial bee colony algorithm with generating neighbourhood solution for large scale coloured traveling salesman problem, IET Intelligent Transport Systems 13 (10) (2019) 1483–1491.

[10] E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Sons, Inc., USA, 1998.

[11] P. Galinier, Z. Boujbel, M. C. Fernandes, An efficient memetic algorithm for the graph partitioning problem, Annals of Operations Research 191 (1) (2011) 1–22.

[12] P. Galinier, J.-K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (4) (1999) 379–397.

[13] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of memetic algorithms, Springer, 2012, pp. 73–94.

[14] P. He, J.-K. Hao, Iterated two-phase local search for the colored traveling salesmen problem, Engineering Applications of Artificial Intelligence 97 (2021) 104018.

[15] P. He, J. Li, H. Qin, Z. He, R. He, Fields distinguished by edges and middles visited by heterogeneous vehicles to minimize non-working distances, Computers and Electronics in Agriculture 170 (2020) 105273.

[16] P. He, J. Li, D. Zhang, S. Wan, Optimisation of the harvesting time of rice in moist and non-moist dispersed fields, Biosystems Engineering 170 (2018) 12–23.

[17] K. Helsgaun, An effective implementation of the lin–kernighan traveling salesman heuristic, European Journal of Operational Research 126 (1) (2000) 106–130.

[18] A. H. Kashan, A. A. Akbari, B. Ostadi, Grouping evolution strategies: An effective approach for grouping problems, Applied Mathematical Modelling 39 (9) (2015) 2703–2720.

[19] L. T. Kóczy, P. Földesi, B. Tüű-Szabó, Enhanced discrete bacterial memetic evolutionary algorithm-an efficacious metaheuristic for the traveling salesman optimization, Information Sciences 460 (2018) 389–400.

[20] J. Li, X. Meng, X. Dai, Collision-free scheduling of multi-bridge machining systems: a colored traveling salesman problem-based approach, IEEE/CAA Journal of Automatica Sinica 5 (1) (2017) 139–147.

[21] J. Li, M. Zhou, Q. Sun, X. Dai, X. Yu, Colored traveling salesman problem, IEEE Transactions on Cybernetics 45 (11) (2014) 2390–2401.

[22] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[23] Y. Lu, J.-K. Hao, Q. Wu, Hybrid evolutionary search for the traveling repairman problem with profits, Information Sciences 502 (2019) 91–108.

[24] X. Meng, J. Li, X. Dai, J. Dou, Variable neighborhood search for a colored traveling salesman problem, IEEE Transactions on Intelligent Transportation Systems 19 (4) (2017) 1018–1026.

[25] Y. Nagata, O. Bräysy, Edge assembly-based memetic algorithm for the capacitated vehicle routing problem, Networks: An International Journal 54 (4) (2009) 205–215.

[26] Y. Nagata, S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, INFORMS Journal on Computing 25 (2) (2013) 346–363.

[27] I. Or, Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking, PhD thesis (Department of Industrial Engineering and Management Science, Northwestern University).

[28] V. Pandiri, A. Singh, A swarm intelligence approach for the colored traveling salesman problem, Applied Intelligence 48 (11) (2018) 4412–4428.

[29] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, Computers & Operations Research 31 (12) (2004) 1985–2002.

[30] A. Singh, A. S. Baghel, A new grouping genetic algorithm approach to the multiple traveling salesperson problem, Soft Computing 13 (1) (2009) 95–101.

[31] B. Soylu, A general variable neighborhood search heuristic for multiple traveling salesmen problem, Computers & Industrial Engineering 90 (2015) 390–401.

[32] W. Sun, J.-K. Hao, W. Wang, Q. Wu, Memetic search for the equitable coloring problem, Knowledge-Based Systems 188 (2020) 105000.

[33] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, A tabu search heuristic for the vehicle routing problem with soft time windows, Transportation Science 31 (2) (1997) 170–186.

[34] P. Toth, D. Vigo, The granular tabu search and its application to the vehicle-routing problem, INFORMS Journal on Computing 15 (4) (2003) 333–346.

[35] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, W. Rei, A hybrid genetic algorithm for multidepot and periodic vehicle routing problems, Operations Research 60 (3) (2012) 611–624.

[36] Y. Wang, Y. Chen, Y. Lin, Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem, Computers & Industrial Engineering 106 (2017) 105–122.

[37] S. Yuan, B. Skinner, S. Huang, D. Liu, A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms, European Journal of Operational Research 228 (1) (2013) 72–82.

[38] Y. Zhou, B. Duval, J.-K. Hao, Improving probability learning based local search for graph coloring, Applied Soft Computing 65 (2018) 542–553.

[39] Y. Zhou, J.-K. Hao, F. Glover, Memetic search for identifying critical nodes in sparse graphs, IEEE Transactions on Cybernetics 49 (10) (2018) 3699–3712.

A Appendix

In this appendix, we show the mathematical model that we used to report the results of the general ILP solver CPLEX in Section 4. The model is based on [14] and [21].

$$Min\ F = \sum_{k=1}^{m} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} x_{ijk} \tag{A.1}$$

$$\sum_{i=1}^{n-1} x_{0ik} = 1, \forall k \in M \tag{A.2}$$

$$\sum_{i=1}^{n-1} x_{i0k} = 1, \forall k \in M \tag{A.3}$$

$$\sum_{i}\sum_{j} x_{ijk} = 0, i \in (C_k \cup S), j \in V \backslash (C_k \cup S), \forall k \in M \tag{A.4}$$

$$\sum_{j=0}^{n-1}\sum_{k=1}^{m} x_{jik} = 1, j \neq i, i \in V \backslash \{0\} \tag{A.5}$$

$$\sum_{l} x_{jlk} = \sum_{i} x_{ijk}, i \neq j \neq l, j, i, l \in C_k \cup S, \forall k \in M \tag{A.6}$$

$$u_{ik} - u_{jk} + n \times x_{ijk} \leq n - 1, j \neq i, i, j \in V \backslash \{0\}, \forall k \in M \tag{A.7}$$

The binary variable $x_{ijk} = 1$ indicates that the k-th salesman passes through edge $\{i, j\}$, and otherwise $x_{ijk} = 0$. u_{ik} is the number of cities visited on the k-th route from the depot up to city i. The objective function of CTSP is given by Eq. (A.1) and Eqs. (A.2-A.7) are the constraints of the problem. Eqs. (A.2) and (A.3) require that each salesman starts from the depot and returns to the depot. Eq. (A.4) indicates that each salesman can only visit its own exclusive cities and some shared cities. Eq. (A.5) means that each city except the depot can only be visited exactly once. Eq. (A.6) indicates that a salesman can only arrive at its exclusive and shared cities to continue its route. Eqs. (A.6) and (A.7) are employed to eliminate the subtours for each salesman.

This ILP model has two variables for every edge (for both directions) and uses the Miller-Tucker-Zemlin subtour elimination constraints. Given that this model includes more variables than necessary and the Miller-Tucker-Zemlin subtour elimination constraints are known to be very slow for TSP from practical point of view, it would be interesting to investigate other formulations. A possible way would be to define an undirected model using, e.g., the subtour elimination constraints of Dantzig-Fulkerson-Johnson. Even if such a formulation makes the use of ILP solvers impossible, it can form a basis for designing dedicated branch-and-cut algorithms, which can be expected to outperform the approach based on general ILP solvers.