Iterated two-phase local search for the colored traveling salesmen problem

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

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

[b] Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France

***Engineering Applications of Artificial Intelligence*, 97:104018, Jan. 2021**
https://doi.org/10.1016/j.engappai.2020.104018

Abstract

The colored traveling salesmen problem (CTSP) is a generalization of the popular traveling salesman problem with multiple salesmen. In CTSP, the cities are divided into m exclusive city sets (m is the number of salesmen) and one shared city set. The goal of CTSP is to determine a shortest Hamiltonian circuit (also called route or tour) for each of the m salesmen satisfying that 1) each route includes all cities of an exclusive city set and some (or all) cities of the shared city set, and 2) each city of the shared city set is included in one unique route. CTSP is a relevant model for a number of practical applications and is known to be computationally challenging. We present the first iterated two-phase local search algorithm for this important problem which combines a local optima exploration phase and a local optima escaping phase. We show computational results on 65 common benchmark instances to demonstrate its effectiveness and especially report 22 improved upper bounds. We make the source code of the algorithm publicly available to facilitate its use in future research and real applications.

Keywords: colored traveling salesman problem; routing; combinatorial optimization; heuristics; local search.

1 Introduction

The colored traveling salesmen problem (CTSP), introduced by Li et al. [19], is a generalization of the popular traveling salesman problem with multiple

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

salesmen. In CTSP, the set V of n cities are divided into m exclusive city sets (m is the number of salesmen) and one shared city set S. The goal of CTSP is to determine a shortest Hamiltonian circuit (also called route or tour) for each of the m salesmen satisfying that 1) each route includes all cities of an exclusive city set and some (or all) cities of the shared city set, and 2) each city of the shared city set is included in one unique route. One observes that when we have only one salesman and the shared city set (i.e., $m = 1$ and $S = V$), CTSP degenerates to the very popular symmetric traveling salesman problem (TSP) [1]. On the other hand, if all cities are shared (i.e., $m > 1$ and $S = V$), then CTSP becomes the multiple traveling salesmen problem (MTSP) [2,11,24], which is a classical TSP variant. Finally, it is worth noting that CTSP is related to, but different from the site-dependent vehicle routing problem (SDVRP) [4,29] due to the absence of capacity constraint in CTSP.

Like other TSP models, CTSP has a number of practical applications [19], such as collision-free scheduling of multi-bridge machining systems [18] and rice harvesting schedules [14]. However, as a generalization of the NP-hard TSP, CTSP is computationally challenging, especially when one needs to solve large scale problem instances. Given its theoretical and practical significance, a number of studies have been reported in recent years. As the literature review in Section 2.2 shows, several algorithms have been developed for solving CTSP, including genetic algorithms [19], artificial bee colony [8,27], ant colony optimization (ACO) [7] and variable neighborhood search [23]. We notice that existing studies except [23] concern population-based algorithms. No study has investigated the conceptually simpler iterated local search approach, which is known to be very successful for numerous optimization problems including routing problems [3,25,30] and other NP-hard problems [9,16,37]. This work fills this gap by introducing the first iterated two-phase local search (ITPLS) algorithm for CTSP. We summarize the work as follows.

First, the proposed algorithm relies on an iterated two-phase process to explore the search space. The local optima exploration phase aims to examine various local optimal solutions of increasing quality within a limited search regions. This is achieved by alternating between an inter-route optimization procedure and an intra-route optimization procedure. When this search phase is observed to get trapped in a deep local optimum, the local optima escaping phase is triggered to help the algorithm out of the trap and guide the search to an unvisited region, from where the local optima exploration phase resumes. These two phases are thus repeated until a stopping condition is met.

Second, we report results of extensive computational experiments on three sets of 65 benchmark instances from the literature and show comparisons with existing reference algorithms. In particular, we present improved best-known results (new upper bounds) for 22 instances, which are useful for future research on CTSP.

Third, we make the source code of our algorithm publicly available, which can be used by researchers and practitioners to solve other problems that can be modeled by CTSP.

The rest of this paper is organized as follows. In Section 2, we formulate the problem and present a literature review of existing studies on CTSP. In Section 3, we introduce the general framework of the proposed algorithm and its components. In Section 4, we show computational results on benchmark instances and comparisons with the literature. In Section 5, we summarize the findings of the work and present research perspectives.

2 Problem Definition and Literature Review

In this section, we first introduce the colored traveling salesmen problem and then present the related works in the literature.

2.1 Problem Definition

Given a complete undirected graph $G = (V, E)$ with a set of vertices (or cities) $V = \{0, 1, 2, \ldots, n-1\}$ and a set of weighted edges E where each vertex represents a city and each non-negative edge weight c_{ij} represents the traveling distance between cities i and j ($c_{ij} = c_{ji}$). The city set V is divided into $m+1$ disjoint sets: m exclusive city sets $C_1, C_2, \ldots, C_m$, and one shared city set S such that $\cup_{k=1}^{m} C_k \cup S = V$ and $\cap_{k=1}^{k} C_i \cap S = \emptyset$. The cities of each exclusive set C_k ($k = 1, 2, \ldots, m$) are to be visited by the salesmen k and each city from the shared city set S is to be visited by one of the m salesmen. City 0 (the depot) belongs to the shared set S and is visited by all salesmen. CTSP is to find a group of m Hamiltonian circuits (also called routes or tours) starting from the depot and ending at the depot for the m salesmen to minimize the total traveling distance of the m routes, where the exclusive cities of a set C_k are visited exactly once by the salesman k and the shared cities are visited exactly once by any of the m salesmen. Formally, CTSP can be described by the following mathematical model [19], where $M = \{1, 2, \ldots, m\}$ represents the set of the m salesmen.

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

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

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

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

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

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

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

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

2.2 Literature Review

CTSP was introduced in [19] to optimize routes of a dual-bridge waterjet cutting machine tool. The tool consists of two independent bridge machines with an overlapping workspace for both machines to enter and two exclusive workspaces at both ends of an overlapping workspace for each machine only. Besides, CTSP could also be formulated several practical problems arising in agricultural engineering. For example, He et al. [14] scheduled combine-harvesters to visit geographically dispersed fields under constraints of moist fields, that is moist fields could only be visited by crawler-harvesters and however non-moist fields could be visited by any harvester. In this model, moist fields can be considered as exclusive cities and non-moist fields are shared cities. Similar problems could be also found in [36].

Given its interest, the CTSP model has received increasing attention and several solution algorithms have been developed for solving the problem. In [19], Li et al. presented four genetic algorithms (basic GA, GA with greedy initialization, hill-climbing GA and simulated annealing GA), and introduced the first set of 20 small scale benchmarks based on existing symmetric TSP instances (with up to 101 vertices). They demonstrated that their algorithms performed better than the general mixed integer programming tool Lingo. Meng et al. [23] employed variable neighborhood search and reported improved results on the instances introduced in [19]. Later, Pandiri and Singh [27] presented an artificial bee colony algorithm (ABC). In their work, they not only reported better results on the 20 small instances compared to the previous algorithms [19,23], but also presented the first results for 8 new medium scale instances (with 229 to 666 vertices). At the same time, Dong et al. [7] employed ant colony optimization (ACO) with multi-tasks learning. They showed that their ACO algorithm did not compete well with the ABC algorithm [27] on the set of 20 small instances. This study also provided 6 medium (with 202 to 431 vertices) and 5 large instances (with 1002 vertices). Finally, Dong et al. [8] implemented another ABC algorithm and reported computational results on 26 new large instances (with 2461 to 7397 vertices).

The above studies have greatly contributed to advancing the state-of-the-art of practically solving CTSP and reported interesting computational results on benchmark instances. Meanwhile, one notices that most existing algorithms are based on bio-inspired approaches, which rely on a population of candidate solutions to explore the search space. In this work, we are interested in investigating the conceptually simpler single trajectory iterated local search approach [21] for solving CTSP. The proposed algorithm employes an iterated two-phase procedure to examine candidate solutions by performing local optimization. As shown in Section 4, the algorithm is able to compete favorably with the best CTSP algorithms on the benchmark instances from the literature.

3 An Iterated Two-Phase Local Search

We now present the iterated two-phase local search algorithm for solving CTPS. After introducing the solution representation, we show the general framework and its composing ingredients.

3.1 Solution Representation and Search Space

As a multi-route problem, CTSP can benefit from the solution representations of MTSP including the *m-tour* encoding [31], dual-chromosome encoding [28] and one-chromosome encoding [33]. For instance, the *m-tour* and dual-chromosome representations were used in [26] and [19] for CTSP, respectively. Besides, Pandiri and Singh [27] showed that the *m-tour* encoding was more space efficient than the dual-chromosome representation. In this work, we adapted the *adjacency representation* introduced in [13] for TSP (see Fig. 1) to the case of CTSP. Specifically, a solution is composed of m routes where each route is represented by an array such that city j of the route occupies position i in the array if the route goes from city i to city j. For the cities which are not on the route, the corresponding positions are filled by -1. Fig. 1 illustrates a solution with 2 routes: 0-1-3-2-7 and 0-4-5-6-8-10-9. Compared to other representations such as the *m-tour* encoding used in [27], our representation has the advantage of easing the insert operation between two routes. For example, if city 8 is deleted from route 2 and inserted behind city 1 of route 1, the time complexity for this operation is $O(1)$ with our representation because it is unnecessary to displace other cities. This is to be contrasted to the time complexity $O(|S| + |C_m|)$ of the *m-tour* encoding, because cities 3, 2, 7 need to move back one position in route 1 while in route 2 cities 10 and 9 need to move forward one position.

	0	1	2	3	4	5	6	7	8	9	10
Route 1	1	3	7	2	-1	-1	-1	0	-1	-1	-1
Route 2	4	-1	-1	-1	5	6	8	-1	10	0	9

Fig. 1. Illustrative example of the adjacency representation for a solution with 2 routes

For a solution $s = (s_1, s_2, \ldots, s_m)$, where s_k $(k = 1, 2, \ldots, m)$ represents the kth route which includes the cities visited by the kth salesman, its objective value $F(s)$ is given by the total traveling distance calculated as follows.

$$F(s) = \sum_{k=1}^{m} \left(\sum_{i=1}^{|s_k|-1} c_{s_k(i)s_k(i-1)} + c_{s_k(0)s_k(|s_k|-1)} \right) \tag{8}$$

where $|s_k|$ indicates the number of cities in route s_k.

3.2 General Procedure

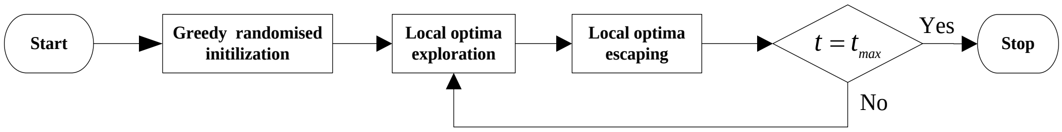

Fig. 2. Flow chart of the general ITPLS procedure

The proposed iterated two-phase local search (ITPLS) for CTSP relies on the iterated local search framework [21], which has been applied with success to a number of routing problems [3,25,30]. Generally, ITPLS iterates a local optima exploration phase and a local optima escaping phase (see the illustrative flow chart in Fig. 2). As shown in Algorithm 1, before entering the first main 'while' loop, a greedy randomized heuristic is employed to construct an initial solution (line 2, Sections 3.3). Then, the algorithm repeats a number of 'while' iterations to find solutions of increasing quality. At each iteration, the local optima exploration phase is first performed to investigate different local optimal solutions (line 5, Section 3.4) by alternating an intra-route optimization of the m routes and an inter-route optimization between two routes. The best solution s_b found during this phase is used to update the recorded best solution s^* if needed (lines 6-8). When the local optima exploration phase terminates, the search is considered to be stagnating. The algorithm then switches to the local optima escaping phase to guide the search process to a new region from where the local optima exploration phase resumes (line 9, Sections 3.5). This process is repeated until a stopping condition is met, which is typically an allowable cutoff time (t_{max}) or maximum number of iterations.

Algorithm 1: General procedure of ITPLS for CTSP

Input: Instance I, probability P_i, search depth of SbTS O_{max}, probability P_s, parameter α, probability P_a

Output: The best solution s^* found

1 **begin**
    `/* Solution initialization, Sections 3.3 */`
2     $s \leftarrow Greedy\_randomized\_heuristic(I, P_i)$
3     $s^* \leftarrow s$
4   **while** *a Stopping condition is not met* **do**
        `/* Stage 1: local optima exploration, Section 3.4 */`
5         $s_b \leftarrow Local\_optima\_exploration(s, O_{max}, P_s, \alpha)$
6   **if** $F(s_b) < F(s^*)$ **then**
7             $s^* \leftarrow s_b$ `/* update the best solution ever found */`
8   **end**
        `/* Stage 2: local optima escaping, Sections 3.5 */`
9         $s \leftarrow Local\_optima\_escaping(s, P_a)$
10   **end**
11   **return** s^*
12 **end**

3.3 Greedy Randomized Initialization

The initial solution of the ITPLS is generated by a greedy randomized initialization procedure which includes two steps. The first step builds a partial route for each of the m salesmen by using its exclusive set of cities. The second step dispatches the shared cities among the m partial routes to obtain a complete solution. To build the kth ($k = 1, \ldots, m$) partial route s_k, a random city i in C_k is used to initiate the greedy construction. Then the remaining cities of C_k are considered in a random order and greedily inserted into s_k to minimize the distance of the route. The first step terminates when all the cities of each exclusive set C_k are included in one partial route. Then, the second step follows to insert greedily and probabilistically the cities of the shared set into the m routes as follows. For each city in S (except the depot 0 which is the starting city of all routes), it is inserted into the best position among all the m routes if the greedy probability P_i is verified, otherwise, it is inserted into a random position of a random route. With this greedy randomized initialization procedure, we can obtain multiple diverse initial solutions. Indeed, with $P_i = 0$, we have a pure greedy procedure. By varying P_i, we control the acceptance of random insertions. Finally, 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 $O(|S| \times n)$.

3.4 Local Optima Exploration

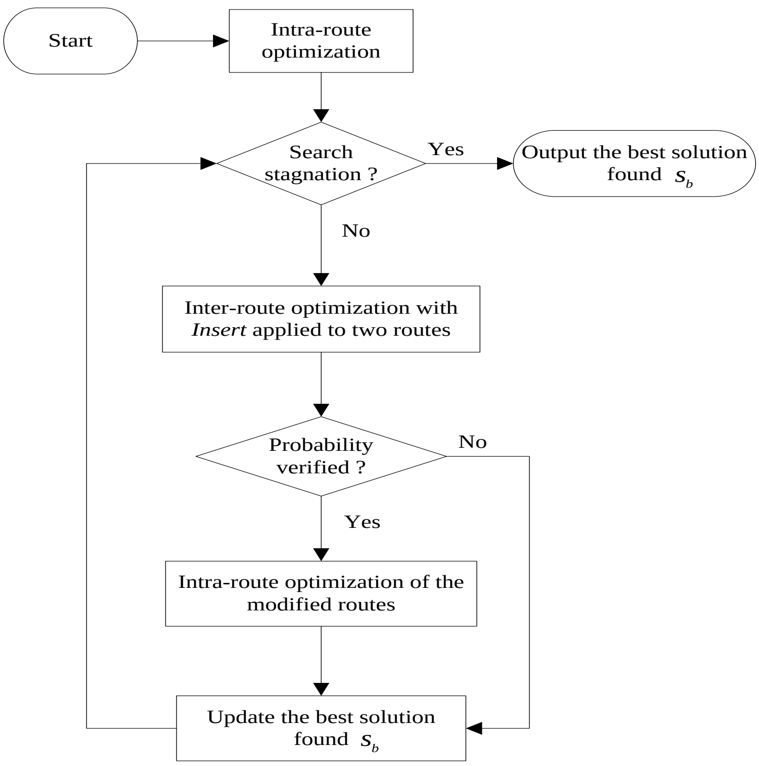

Fig. 3. Flow chart of the local optima exploration phase

The local optima exploration phase (LOEP) is the key search component of the proposed algorithm and combines a (global) *inter-routing optimization* procedure and a (local) *intra-route optimization* procedure to explore various local optimal solutions. The inter-routing optimization aims to find better solutions

Algorithm 2: The framework of local optima exploration

Function *Local_optima_exploration*$(s, O_{max}, P_s, \alpha)$
Input: Input solution s, search depth O_{max}, parameter α
Output: The best solution s_b found
begin
  $R \leftarrow s$
  $s \leftarrow intra-route\ optimization(R)$ `/* intra-routing optimization to improve all routes */`
  $R \leftarrow \emptyset$
  $s_b \leftarrow s$
  for $i \leftarrow 0$ ***to*** $L-1$ **do**
    $H_1[i] \leftarrow 0$; $H_2[i] \leftarrow 0$; `/* initialization of hash vectors */`
  end
  $N_i \leftarrow 0$
  `/* Main search */`
  while $N_i \leq O_{max}$ **do**
    `/* Enter inter-routing optimization */`
    $\delta \leftarrow F(s \oplus Insert(\cdot)) - F(s)$ `/* calculate the move gain, Sections 3.4.1 and 3.4.2 */`
    $s \leftarrow s \oplus Insert(k_1, i_1, k_2, i_2)$ `/* perform the best non-tabu move, Sections 3.4.1 and 3.4.2 */`
    $F(s) \leftarrow F(s) + \delta(k_1, i_1, k_2, i_2)$
    $H_1[h_1(s)] \leftarrow 1; H_2[h_2(s)] \leftarrow 1$ `/* enter the solution into tabu list */`
    Update matrix δ `/* update move gains with the fast computation technique, Appendix A */`
    $R \leftarrow R \cup \{s_{k_1}, s_{k_2}\}$ `/* record the two modified routes` s_{k_1} `and` s_{k_2} `involved in the performed move` $Insert(k_1, i_1, k_2, i_2)$ `*/`
    if *Probability*(m, α) *is verified* **then**
      $s \leftarrow intra-route\ optimization(R, P_s,)$ `/* intra-route optimization to improve the two modified routes, Section 3.4.3 */`
      $R \leftarrow \emptyset$
    end
    if $F(s) < F(s_b)$ **then**
      $s_b \leftarrow s$
      $N_i \leftarrow 0$
    else
      $N_i \leftarrow N_i + 1$
    end
  end
  return s_b
end

by moving cities between two routes while the intra-route optimization focuses on the distance minimization of each individual route. Both inter-routing optimization and intra-route optimization are based on the tabu search metaheuristic [12]. Specifically, inter-route optimization uses the so-called solution based tabu search (SbTS) [17,34,35] while intra-route optimization mixes the 2-opt heuristic [5,15,20] and a simple tabu search heuristic.

The pseudo-code of the optima exploration phase is shown in Algorithm 2 (see also the illustrative flow chart in Fig. 3), where s is the current solution composed of m routes, s_b records the current best solution found during the local optima exploration phase and, R stores the set of routes modified by inter-route optimization and H_i ($i = 1, 2$) are hash tables used as the tabu list of SbTS and explained in Section 3.4.2. After the preparatory operations including a first intra-route optimization and initialization of the hash tables (lines 3-9), the procedure enters the main 'while' loop to repeat inter-routing optimization and intra-route optimization.

At each 'while' loop in Algorithm 2, LOEP first performs inter-route optimization (lines 12-17). For this, LOEP uses a $|S| \times n$ matrix δ to store the distance variation (called move gain) of inserting a city taken from a route to another route (see Section 3.4.1). Based on δ, the best $Insert(k_1, i_1, k_2, i_2)$ move (i.e., city i_2 of route s_{k_2} is inserted after city i_1 on route s_{k_1}) is selected and performed to obtain a neighbor solution (line 13). The matrix δ and tabu list are updated accordingly (lines 15-16, see Section 3.4.2). The two modified routes s_{k_1} and s_{k_2} are collected in R (line 17). After that, one decides whether the modified routes in R are to be further improved by the intra-route optimization procedure (lines 18-21). In principle, it would be desirable to re-optimize the modified routes s_{k_1} and s_{k_2}. However, intra-route optimization is time consuming and running this procedure too often could be counterproductive. As a compromise, the intra-route optimization procedure is performed according to a probability $P(m, \alpha)$, which depends on the number of routes m and parameter α. The idea is that short routes have more chances to be further optimized by intra-route optimization than long routes considering that optimizing short routes is less time consuming. Finally, during the LOEP phase, the best solution s_b is updated each time an improved best solution is discovered. If the best solution s_b cannot be updated for O_{max} (a parameter called search depth) consecutive LOEP iterations, the local optima exploration phase is considered to be trapped in a deep local optimum. As a result, this LOEP phase terminates and returns the best recorded solution s_b. To go beyond the local optimum trap, the algorithm triggers the local optima escaping phase (Section 3.5), which applies a destruction-reconstruction procedure to generate a new starting solution for the next round of LOEP.

The ingredients of the local optima exploration phase, including the neighborhood, tabu strategy, and intra-route optimization, are explained in the following subsections.

3.4.1 Inter-route optimization

Inter-routing optimization focuses on moving cities between routes. For this, we adopt the popular *Insert* operator to define a neighborhood which is explored by solution-based tabu search. Specifically, $Insert(k_1, i_1, k_2, i_2)$ denotes the operation that deletes city i_2 from route s_{k_2} and inserts i_2 after city i_1 of route s_{k_1}. To ensure that each *Insert* operation generates a feasible solution, the displaced city i_2 must be a shared city of set S (excluding the depot). Thus given the current solution s, applying *Insert* to s generates the following set $N(s)$ of neighbor solutions.

$$N(s) = \{s' \leftarrow s \oplus Insert(k_1, i_1, k_2, i_2) : k_1 \in M, k_2 \in M, i_1 \in V, i_2 \in S \backslash \{0\}\} \tag{9}$$

where $s' \leftarrow s \oplus Insert(k_1, i_1, k_2, i_2)$ denotes the neighbor solution of s given by applying $Insert(k_1, i_1, k_2, i_2)$. It is clear that the size of this neighborhood is bounded by $O(|S| \times n)$.

Given this neighborhood, the inter-route optimization procedure identifies the best eligible neighbor solution identified by the best $Insert(k_1, i_1, k_2, i_2)$ with the largest move gain according to the δ matrix. A neighbor solution is eligible if it is not forbidden by the tabu list (see Section 3.4.2).

So each inter-routing optimization application with the $Insert(\cdot)$ operator can be performed in $O(|S| \times n)$ time. In the Appendix A, we present a streamlined technique for fast computation and update of move gains in the δ matrix, which reduces the time complexity to $O(n)$.

3.4.2 Tabu strategy

With tabu search [12], each visited candidate solution is recorded in a data structure called tabu list to avoid revisiting the same solution during subsequent search. In this work, we adopt the so-called solution-based tabu search [17,34,35], where the tabu list is implemented with hash tables. It is worth mentioning that to our knowledge, this is the first application of this approach to a routing problem.

Specifically, the tabu list relies on two hash vectors H_1 and H_2 of length L (L is a large number, set to be 10^8 in this work) associated to two hash functions

h_1 and h_2 defined by Eqs. (10) and (11).

$$h_1(s) = \sum_{k=1}^{m}(k \times \sum_{i=1}^{|s_k|-1} s_k(i)) \tag{10}$$

$$h_2(s) = \sum_{k=1}^{m} \sum_{i=2}^{|s_k|-1} s_k(i-1) * s_k(i) \tag{11}$$

where $s_k(i)$ represents the ith city in route s_k and $|s_k|$ is the number of cities in route s_k.

Given a candidate solution s, it is forbidden by the tabu list (i.e., excluded for consideration) if $H_1(h_1(s) \; mod \; L) \wedge H_2(h_2(s) \; mod \; L) = 1$; and otherwise, this solution is eligible for consideration.

Let $s' \leftarrow s \oplus Insert(k_1, i_1, k_2, i_2)$ be a neighbor solution. The two hash values for s' can be calculated by Eqs. (12,13)

$$h_1(s') = h_1(s) + i_2 * k_1 - i_2 * k_2 \tag{12}$$

$$h_2(s') = h_2(s) + i_2 * i_1 + i_2 * i_1^n - i_1 * i_1^n + *i_2^n - i_2^p * i_2 - i_2 * i_2^n \tag{13}$$

where i_1^n is the next city after i_1 in route s_{k_1}, i_2^p and i_2^n are the previous and next city to i_2 in route s_{k_2}, respectively. Therefore, the time complexity of determining the tabu status for a neighbor solution s' is $O(1)$ based on Eqs. (12,13).

3.4.3 Intra-route Optimization

Since each route could be regarded as a case of TSP, the well-known fast 2-opt heuristic for TSP [5,15,20] is a natural choice for intra-route optimization. Basically the 2-opt heuristic iteratively reduces the tour distance by performing edge exchanges as follows: disconnect the current tour by removing 2 edges and reconnect the tour by 2 other edges in such a way that the new tour has a shorter distance. This process continues until no improving edge exchange exists. The 2-opt heuristic has the advantages of be simple and very fast. For this reason, several previous studies on CTSP such as [26,31,32] used the 2-opt heuristic for individual route optimization. However, given that 2-opt follows the strict descent principle, it can be easily trapped in local optima.

In this work, our intra-route optimization procedure adopts an enhanced strat-

Algorithm 3: Intra-route optimization

```
Input: Set of routes to be optimized $R$, probability $P_s$
Output: Set of improved routes $R_b$
begin
    $R_b \leftarrow \emptyset$
    for each route $s_k$ in $R$ do
        $s_k^* \leftarrow s_k$
        if $rand() > P_s$ then
            /* Route-optimization with 2-opt */
            $\Delta \leftarrow F(s_k) - F(s_k \oplus 2 - opt)$
            while there exist improving 2-opt move ($\Delta < 0$) do
                $s_k \leftarrow s_k \oplus 2 - opt$ /* perform the best improving 2-opt move */
                $F(s_k) \leftarrow F(s_k) - \Delta$
                $\Delta \leftarrow F(s_k) - F(s_k \oplus 2 - opt)$
            end
            $s_k^* \leftarrow s_k$
        else
            /* Route-optimization with simple tabu search */
            $n_i \leftarrow 0$
            while $n_i < |s_k|$ do
                $\Delta \leftarrow F(s_k) - F(s_k \oplus 2 - opt)$ /* perform the best eligible 2-opt move */
                $F(s_k) \leftarrow F(s_k) - \Delta$
                $s_k \leftarrow s_k \oplus 2 - opt$
                Update the tabu list
                if $F(s_k^*) > F(s_k)$ then
                    $s_k^* \leftarrow s_k$
                    $n_i \leftarrow 0$
                else
                    $n_i \leftarrow n_i + 1$
                end
            end
        end
        $R_b \leftarrow R_b \cup s_k^*$
    end
    return $R_b$
end
```

egy, which applies the 2-opt heuristic and a simple tabu search (STS) heuristic in a probabilistic way (See Algorithm 3). Specifically, With probability P_s, we perform STS, and with probability $1 - P_s$, we apply 2-opt. The STS heuristic used in the intra-route optimization follows the conventional attribute-based tabu approach [12]. STS relies on the same edge exchange operation as for the

2-opt heuristic and uses a tabu list to record the exchanged edges. As such, each time an edge is exchanged (removed or added) in the current route s_k, it will not be considered by STS for the next tl consecutive iterations where tl is called the tabu tenure fixed to be $T_l * |s_k|$ ($T_l = 0.3$ in this work). STS terminates if the best route s_k^* is not updated within $|s_k|$ steps.

Finally, one notices that intra-route optimization is applied at two places of the local optima exploration phase: to improve all routes of the input solution s (line 3, Algorithm 2) and to improve the two modified routes after each inter-route optimization step (line 19, Algorithm 2).

3.5 Local Optima Escaping

When the local optima exploration phase terminates, the search is considered to be trapped in some deep local optimum. To get rid of the trap, the local optima escaping phase is launched. Our local optima escaping procedure is composed of two steps. The first step destroys the input solution s by deleting some shared cities while the second step re-inserts these deleted cities into different routes. In the first step, each shared city is deleted according to a destruction probability P_d defined by $P_d = 1 - \frac{e^{-\beta/T}}{2}$, where β is the number of non-improvement iterations in Algorithm 1 and T is a parameter. The second step is similar to the second step of the greedy randomized initialization heuristic in Section 3.3. Each deleted city is inserted to the position which minimizes the distance if the greedy probability P_a is verified; Otherwise this position is discarded. After all deleted shared cities are inserted, a new solution is obtained, which serves as the new starting solution of the next round of the local optima exploration phase. The probabilities P_d and P_a control the diversification degree of the algorithm. We show a sensitive analysis of these parameters in Section 4.3. The time complexity of the local optima escaping procedure is $O(|S| \times n)$.

3.6 Computational Complexity of ITPLS

As shown in Algorithm 1, each iteration of ITPLS performs two subroutines: local optima exploration and local optima escaping. The local optima exploration part includes intra-route optimization and inter-route optimization. The time complexity of each iteration of intra-route optimization and inter-route optimization is $O((|S + C_k|)^2)$ and $O(|S| \times n)$, respectively. Furthermore, the time complexity of local optima escaping is $O(|S| \times n)$.

4 Experimental Results and Comparisons

In this section, we report computational experiments on three sets of 65 benchmark instances from the literature. The benchmark instances, the experiment protocol and parameters, and computational results are presented in the following subsections.

4.1 Benchmark Instances

For CTSP, three sets of 65 benchmark instances are available in the literature.

Set I: this set contains small 20 instances, generated from six graphs by varying the number of routes and exclusive cities in each instance. The number of cities is between 21 to 101, while the number of salesmen m is between 2 and 7. These instances were first introduced by Li et al. [19], and tested in [7,8,19,23,27].

Set II: this set contains medium 14 instances, generated from four graphs by varying the number of routes and exclusive cities in each instance. The number of cities n is between 202 and 666, and the number of salesmen m is between 10 and 40. The 6 instances related to the two graphs with 202 and 431 cities were proposed by Dong et al.[7], while the remaining instances were proposed by Pandiri and Singh [27].

Set III: this set includes large 31 instances, generated from five graphs by varying the number of routes and exclusive cities in each instance. The number of cities n in this set is between 1002 and 7397, and the number of salesmen m is between 3 and 60. The 5 instances related to the first graph were presented by Dong et al. [7], and the remaining instances were proposed by Dong et al. [8].

4.2 Experimental Protocol

The ITPLS algorithm was coded in C++, and complied by g++ with the -O3 option [1]. Our computational experiments were conducted on a computer with an AMD-6134 processor (2.3GHz and 2G RAM) under Linux.

Reference algorithms. There are five heuristic algorithms for CTSP reported in the literature.

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

- Genetic algorithms (GAs) [19] (2014), which report results on Set I only. Their experiments were performed on a computer with a 3.3GHz processor and under the stopping condition of10 minutes.
- Variable neighborhood search (VNS) [23] (2017), which reports results on Set I only. Their experiments were performed on a computer with a 3.4GHz processor and under the stopping condition of a maximum of 10000 epochs.
- Artificial bee colony (ABC) [27] (2018), which reports results on Set I and 8 out of 14 instances of Set II. Their experiments were performed on a computer with a 3.4GHz processor. The stopping condition for instances with $21 - 41$ cities, $51 - 101$ cities and $229 - 666$ cities was 1 second, 5 seconds, and 60 seconds, respectively.
- Ant colony optimization (ACO) [7] (2018), which reports results on Set I, 6 out of 14 instances of Set II and 5 out of 31 instances of Set III. Their experiments were performed on a computer with a 3.01GHz processor and the stopping condition was not indicated.
- Artificial bee colony (ABC) [8] (2019), which reports results on 26 out of 31 instances of Set III. They used a computer with a 3.4GHz processor, and the stopping condition is the maximum non-updated iteration number, which was fixed to 60.

From the results reported in these studies, we identify ABC by Pandiri and Singh [27] as the current best algorithm for CTSP and use it as our principal reference algorithm for our comparative study. Since the source code of this algorithm (and the other reference algorithms) is unavailable, we faithfully re-implemented the ABC algorithm of [27] [2]. We verified that our implementation was able to reproduce the results reported in [27] (and in fact, our ABC implementation even obtained some better results than those in [27]). To ensure a fair comparison, we ran our algorithm and ABC on our computer under the same cutoff limits. Specifically, we ran ITPLS 20 times with the parameter setting of Table 1 and ABC 20 times with the parameter setting given in [27] on each instance. The cutoff time t_{max} per run was set to be 1, 10 and 60 minutes for sets I, II and III, respectively, except $t_{max} = 240$ minutes for the large instances with at least 7000 cities.

For the other algorithms (VNS [23], ABC [27], ACO [7], ABC [8]), we replicate the published results, while excluding GAs of [19] given that they are fully dominated by the other algorithms. Since the reference algorithms did not report results on all benchmark sets, their results are included only for indicative purposes.

[2] Our implementation of the ABC algorithm [27] is available from the page given in footnote 1.

4.3 Parameter Calibration

We now conduct a sensitivity analysis of the parameters of the ITPLS algorithm. For this study, we first identified a rough value range for each parameter and analyzed one parameter at a time. Specifically, we varied the values of the studied parameter in its range while keeping the other parameters to their default values as shown in Table 1. The value ranges of the parameters are: $P_i = \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}, O_{max} = \{20, 50, 80, 120, 150, 200\}, P_s = \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}, T_l = \{0.1, 0.3, 0.5, 0.7, 0.9\}, T = \{10, 30, 50, 70, 100, 200\}$, $P_a = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. This experiment was based on 8 representative instances (gr229-30, gr431-25, gr666-20, pr1002-10, fnl2461-3, fnl3461-12, pla5397-50, pla6397-30) covering both medium and larger instances. To assess a parameter setting (PS), we measure the gaps between the results by ITPLS with this particular setting and the results of ITPLS with its default parameter setting in terms of the best and average values, calculated as follows.

$$Avg_{gap} = \sum \frac{F_{PS\_avg} - F_{ITPLS\_avg}}{F_{ITPLS\_avg}} \tag{14}$$

$$Best_{gap} = \sum \frac{F_{PS\_best} - F_{ITPLS\_best}}{F_{ITPLS\_best}} \tag{15}$$

For each instance, we ran 20 times ITPLS with each parameter setting and the results are shown in Fig. 4, where the Avg_{gap} and $Best_{gap}$ (the smaller, the better) are defined in Eqs. (14) and (15).

Fig. 4(a) indicates that the probability P_i in the greedy randomized heuristic does not influence much the results. As for O_{max} (the depth of SbTS), Fig. 4(b) shows that this parameter impacts the performance of ITPLS slightly. Fig. 4(c) reveals that the simple tabu search (STS) plays an important role for intra-route optimization in ITPLS. Indeed, if only 2-opt is used ($P_s = 0$), the results are the worst. When tabu search is also employed ($P_s > 0$), the performances are improved considerably, with $P_s = 0.3$ leading to the best performance (defined as the default value for ITPLS). Fig. 4(d) indicates that the tabu tenure of STS also influences the performances of ITPLS, with $T_l = 0.3$ being the best value. As shown in Fig. 4(e), the number of the deleted cities in the local optima escaping phase impacts slightly the performance of ITPLS, with $T = 50$ being a suitable value. Finally, the probability P_a used in the same phase influences the performance of ITPLS and $P_a = 0.4$ (Fig. 4(f)) is identified as the best value and used as the default value for ITPLS.

The values of Table 1 can be considered to define the default setting of ITPLS. And this setting was consistently used to conduct all the experiments reported below, except $P_a = 0.1$ (instead of its default value of 0.4) was used to solve

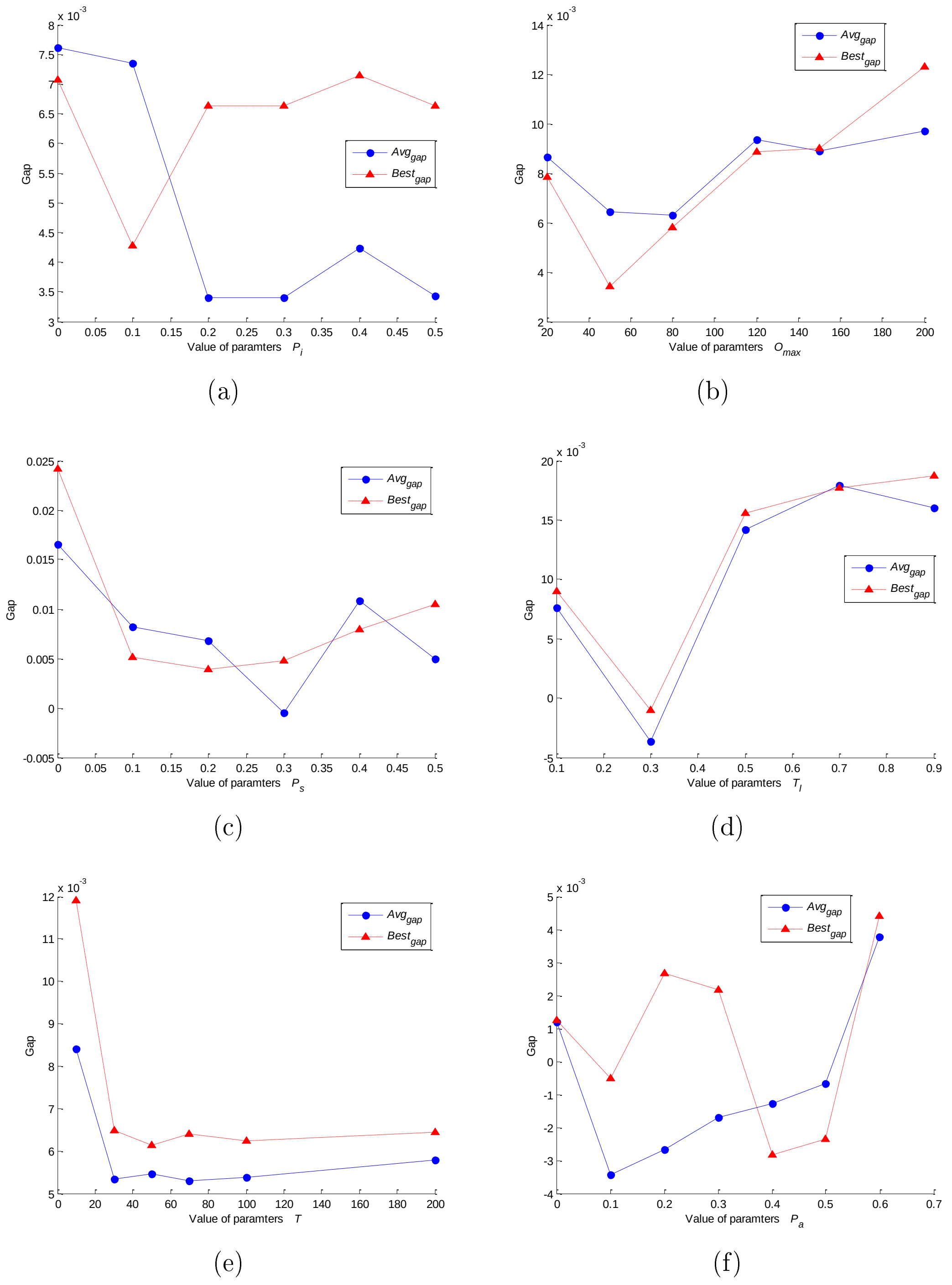

Fig. 4. Analysis of the effects of the parameters

the instances with at least 7000 vertices (cities).

Table 1
Settings of parameters

Parameters	Section	Description	Values
P_i	3.3	Greedy probability in initial solution	0.1
O_{max}	3.4	Search depth of SbTS	50
P_s	3.4.3	Select probability in intra-route optimization	0.3
T_l	3.4.3	Parameter of the tabu tenure of STS	0.3
T	3.5	used to define the destruction probability)	50
P_a	3.5	Greedy probability in local optima escaping	0.4

4.4 Computational Results

We show comparative results of our ITPLS algorithm [3] and the main ABC reference algorithm in Table 2 (Set I and Set II) and Table 3 (Set III). For each algorithm, we present the best and average objective value and the standard derivation based on 20 independent runs. We also include the best objective values reported in the literature for VNS [23], ABC [27], ACO [7] and ABC [8].

From each instance, the best (smallest) values among the compared values are indicated in boldface, while the '-' sign indicates that no result is available. From the results reported in these tables, we can make the following comments.

For the small instances of Set I, our ITPLS algorithm and ABC (of [27] and our ABC implementation) achieve the same performance in terms of the best and average objective value and the standard derivation (notice that our ABC implementation and ABC [27] report strictly the same results). Moreover, both ITPLS and ABC dominate the other competitors (VNS and ACO).

For the medium instances of Set II, we observe that only very partial results are available for the compared algorithms. We thus focus on comparing ITPLS and ABC. We observe that TPLS performs slightly better than ABC by reporting 11 (8) dominating F_{best} (F_{avg}) values against 9(7) superior F_{best} (F_{avg}) values for ABC). Besides, the Wilcoxon signed-rank test on the F_{best} and F_{avg} values of ITPLS and ABC (see Table 5) indicate that the differences between the two compared algorithms in terms of F_{best} and F_{avg} are marginal for Set II.

For the large instances of Set III, compared with ABC, ITPLS obtains 25 (21) superior F_{best} (F_{avg}) values out of the 31 instances against 6 (10) superior

[3] The certificates of the best solutions of our ITPLS algorithm are available from the page given in footnote 1.

Table 2. Computational results of the compared algorithms on Sets I and II. The best results are indicated in boldface

Instance	n	m	VNS [23]	ABC [27]	ACO [7]	ABC [8]	ABC (our implementation of [27]) F_{best}	F_{avg}	σ	ITPLS (this work) F_{best}	F_{avg}	σ
Set I												
eil21-2	21	2	**144.92**	**144.92**	**144.92**	-	**144.92**	144.92	0	**144.92**	144.92	0
eil21-3	21	3	**157.48**	**157.48**	**157.48**	-	**157.48**	157.48	0	**157.48**	157.48	0
eil31-2	31	2	**259.36**	**259.36**	261.20	-	**259.36**	259.36	0	**259.36**	259.36	0
eil31-3	31	3	**295.31**	**295.31**	**295.31**	-	**295.31**	295.31	0	**295.31**	295.31	0
eil31-4	31	4	**315.97**	**315.97**	**315.97**	-	**315.97**	315.97	0	**315.97**	315.97	0
eil41-2	41	2	**346.24**	**346.24**	349.25	-	**346.24**	346.24	0	**346.24**	346.24	0
eil41-3	41	3	**367.84**	**367.84**	437.94	-	**367.84**	367.84	0	**367.84**	367.84	0
eil41-4	41	4	**392.14**	**392.14**	392.53	-	**392.14**	392.14	0	**392.14**	392.14	0
eil51-2	51	2	**465.28**	478.08	-	-	478.08	478.08	0	478.08	478.08	0
eil51-3	51	3	**469.50**	**469.50**	470.77	-	**469.50**	469.50	0	**469.50**	469.50	0
eil51-4	51	4	**489.99**	**489.99**	-	-	**489.99**	489.99	0	**489.99**	489.99	0
eil51-5	51	5	529.38	**525.98**	526.59	-	**525.98**	525.98	0	**525.98**	525.98	0
eil76-3	76	3	**593.28**	**593.28**	597.12	-	**593.28**	593.28	0	**593.28**	593.28	0
eil76-4	76	4	**603.79**	**603.79**	606.86	-	**603.79**	603.79	0	**603.79**	603.79	0
eil76-5	76	5	**651.99**	**651.99**	704.53	-	**651.99**	651.99	0	**651.99**	651.99	0
eil76-6	76	6	675.49	**672.73**	677.67	-	**672.73**	672.73	0	**672.73**	672.73	0
eil101-4	101	4	730.47	**726.82**	896.92	-	**726.82**	726.82	0	**726.82**	726.82	0
eil101-5	101	5	781.51	**779.15**	796.77	-	**779.15**	779.15	0	**779.15**	779.15	0
eil101-6	101	6	**759.55**	**759.55**	822.29	-	**759.55**	759.55	0	**759.55**	759.55	0
eil101-7	101	7	**798.85**	**798.85**	928.01	-	**798.85**	798.85	0	**798.85**	798.85	0
Avg.	-	-	491.42	**488.84**	-	-	**488.84**	**488.84**	-	**488.84**	**488.84**	-
Set II												
gr202-12	202	12	-	-	**71924.00**	-	99871.00	100033.20	110.54	99871.00	100009.50	112.58
gr202-25	202	25	-	-	**99606.00**	-	173547.00	173596.80	54.01	173418.00	173523.80	46.77
gr202-35	202	35	-	-	**118495.00**	-	233749.00	233817.85	70.16	233749.00	233857.80	73.17
gr229-10	229	10	-	**222167.00**	-	-	**222167.00**	222354.85	164.08	**222167.00**	222347.65	103.50
gr229-15	229	15	-	**264146.00**	-	-	**264146.00**	264146.00	0.00	**264146.00**	264146.00	0.00
gr229-20	229	20	-	**319669.00**	-	-	**319669.00**	319669.00	0.00	**319669.00**	319671.90	12.97
gr229-30	229	30	-	**406664.00**	-	-	**406664.00**	407194.85	375.21	**406664.00**	406884.00	225.72
gr431-12	431	12	-	-	330554.00	-	**249031.00**	249682.25	293.07	249421.00	250036.95	613.23
gr431-25	431	25	-	-	464298.00	-	**348056.00**	348431.10	203.82	348181.00	349238.10	417.38
gr431-40	431	40	-	-	483977.00	-	**416189.00**	416758.40	249.58	416552.00	417963.75	958.14
gr666-10	666	10	-	391831.00	-	-	390188.00	392234.00	971.38	**389583.00**	396841.55	2716.00
gr666-15	666	15	-	448624.00	-	-	448604.00	449997.35	716.97	**448257.00**	449635.25	800.17
gr666-20	666	20	-	522403.00	-	-	522157.00	523583.15	937.90	**521149.00**	522650.90	1006.57
gr666-30	666	30	-	652714.00	-	-	652587.00	654001.50	633.57	**651801.00**	653318.10	927.19
Avg.	-	-	-	-	-	-	339044.64	**339678.59**	-	**338902.00**	340008.95	-

Table 3. Computational results of the compared algorithms on Set III. The best results are indicated in boldface

Instance	n	m	VNS [23]	ABC [27]	ACO [7]	ABC [8]	ABC (our implementation of [27])			ITPLS (this work)		
							F_{best}	F_{avg}	σ	F_{best}	F_{avg}	σ
Set III												
pr1002-5	1002	5	-	-	542376.25	-	**316437.00**	317425.40	479.59	318587.00	320348.80	1058.72
pr1002-10	1002	10	-	-	588161.83	-	**382201.00**	382844.90	423.48	383112.00	384908.55	936.35
pr1002-20	1002	20	-	-	679067.97	-	**516256.00**	517481.55	452.81	517917.00	519664.85	794.31
pr1002-30	1002	30	-	-	792004.74	-	664648.00	665676.30	559.24	**664308.00**	666702.20	929.80
pr1002-40	1002	40	-	-	892296.03	-	806022.00	807838.65	786.79	**805967.00**	808503.35	1444.57
fnl2461-3	2461	3	-	-	-	111457.00	114188.00	114509.80	145.77	**110007.00**	110553.50	413.84
fnl2461-6	2461	6	-	-	-	123516.00	122312.00	122612.30	188.81	**118513.00**	119199.15	387.44
fnl2461-12	2461	12	-	-	-	151151.00	145800.00	146374.85	220.68	**145023.00**	145688.60	284.37
fnl2461-24	2461	24	-	-	-	229211.00	222465.00	223335.30	456.26	**221494.00**	221739.80	163.09
fnl2461-30	2461	30	-	-	-	274920.00	268431.00	269140.30	535.88	**267355.00**	267593.85	169.31
fnl3461-3	3461	3	-	-	-	157331.00	162335.00	162909.20	177.47	**156753.00**	157420.50	391.84
fnl3461-6	3461	6	-	-	-	169489.00	170762.00	171243.80	238.70	**165455.00**	166525.25	512.43
fnl3461-12	3461	12	-	-	-	195861.00	192874.00	193582.75	336.98	**188223.00**	188963.25	371.47
fnl3461-24	3461	24	-	-	-	270121.00	266686.00	267130.75	187.32	**265078.00**	265672.70	342.83
fnl3461-30	3461	30	-	-	-	311856.00	308742.00	308963.10	95.74	**307562.00**	308018.40	208.49
fnl3461-40	3461	40	-	-	-	387670.00	385443.00	385727.50	120.94	**385122.00**	385296.60	113.73
pla5397-20	5397	20	-	-	-	38871935.00	38335000.00	38392330.00	26980.76	**38331500.00**	38494950.00	73265.21
pla5397-30	5397	30	-	-	-	52521355.00	**51299400.00**	51340355.00	22848.87	51339600.00	51451470.00	82834.36
pla5397-40	5397	40	-	-	-	**31838496.00**	64408200.00	64476755.00	30612.77	64285900.00	64404060.00	61216.52
pla5397-50	5397	50	-	-	-	**40547303.00**	74008700.00	74019335.00	5148.66	74051200.00	74145910.00	44659.82
pla5397-60	5397	60	-	-	-	**47367475.00**	85303100.00	85324645.00	10454.99	85323100.00	85424400.00	60048.45
pla6397-20	6397	20	-	-	-	40205237.00	36672000.00	36748165.00	37220.64	**36404600.00**	36575675.00	103990.10
pla6397-30	6397	30	-	-	-	51996147.00	47689800.00	47750055.00	24229.16	**47551800.00**	47832460.00	98829.28
pla6397-40	6397	40	-	-	-	**34876904.00**	56948400.00	57022520.00	31690.54	56860500.00	56945530.00	54061.04
pla6397-50	6397	50	-	-	-	**42429271.00**	67415000.00	67485965.00	27014.35	67347700.00	67419380.00	40122.95
pla6397-60	6397	60	-	-	-	**50063869.00**	75077200.00	75118385.00	16341.50	74983600.00	75063660.00	52339.27
pla7397-20	7397	20	-	-	-	43722140.00	42262900.00	42432435.00	78855.53	**41804200.00**	42027405.00	138563.86
pla7397-30	7397	30	-	-	-	55181775.00	53648400.00	53717345.00	45595.94	**53183700.00**	53358655.00	113523.08
pla7397-40	7397	40	-	-	-	**35254833.00**	65847100.00	65919250.00	42106.65	65441600.00	65662845.00	142232.68
pla7397-50	7397	50	-	-	-	**42775482.00**	77194500.00	77265730.00	38981.29	76701700.00	76784335.00	74134.09
pla7397-60	7397	60	-	-	-	**50427942.00**	87041500.00	87103321.05	35045.55	86628200.00	86749490.00	77747.32
Avg.	-	-	-	-	-	-	29941832.32	29973335.08	-	**29847076.65**	**29915387.88**	-

F_{best} values for ABC. The statistically significant difference in terms of the best values between ITPLS and ABC is confirmed by the small p-value of 0.0012 (<0.05), while the difference in terms of average values remains marginal (p-value of 0.0599) (see Table 5). One also notices that the 5 best results reported for ACO [7] are greatly updated by ABC and ITPLS, while 17 of the 26 best results reported for the other ABC algorithm [8] are improved by ABC (1 case) and ITPLS (16 cases). These results also consolidate the above observation that our ITPLS algorithm competes very favorably with the ABC approach as implemented in [8,27].

To complement these results, we present in Appendix B (Table B.1) an additional comparison of ABC [27], our ABC implementation and ITPLS under the stopping condition of [27], i.e., a cutoff time of 1 second for instances with $21-41$ cities, 5 seconds for instances with $51-101$ cities and 60 seconds for instances with $202-666$ cities. Since ABC in [27] only reported results on Set I and some instances of Set II, this comparison is limited to these two benchmark sets. From Table B.1, we observe that ITPLS globally competes well with the two ABC implementations, which, however, converge faster than ITPLS on several small instances of Set I. Notice that our 2.3GHz processor is slower than the 3.4GHz processor used to run the ABC algorithm in [27].

To sum, our ITPLS algorithm is highly competitive compared with all existing approaches and its advantage is best demonstrated on medium and large instances. In particular, ITPLS is able to obtain new record-breaking results (new upper bounds) for 4 instances of Set II and 18 instances of Set III.

4.5 Convergence Analysis

To study the behaviors of ABC [27] and ITPLS throughout the execution, we perform an experiment to obtain the running profiles of the two algorithms on four representative instances of Set II (gr202-12, gr229-30, gr431-12, gr666-20). To eliminate the possible influence of randomness, we ran each algorithm 20 times to solve each instance with the cutoff time of 600 seconds per run, and record the best objective values during the process. Fig. 5 illustrates the running profiles which show how the average best objective values found evolve with the running time. We notice that the two algorithms are able to improve the solution quality quickly in the beginning (during the first 100 to 150 seconds), but ABC converges more quickly. However, ITPLS has generally a better performance on the long term. Indeed, ABC began to slow down or even stagnate on the best solution after 150 seconds, while ITPLS continued its search to find still better solutions. This experiment indicates that ABC converges faster than ITPLS, but ITPLS can benefit more run time to find better solutions.

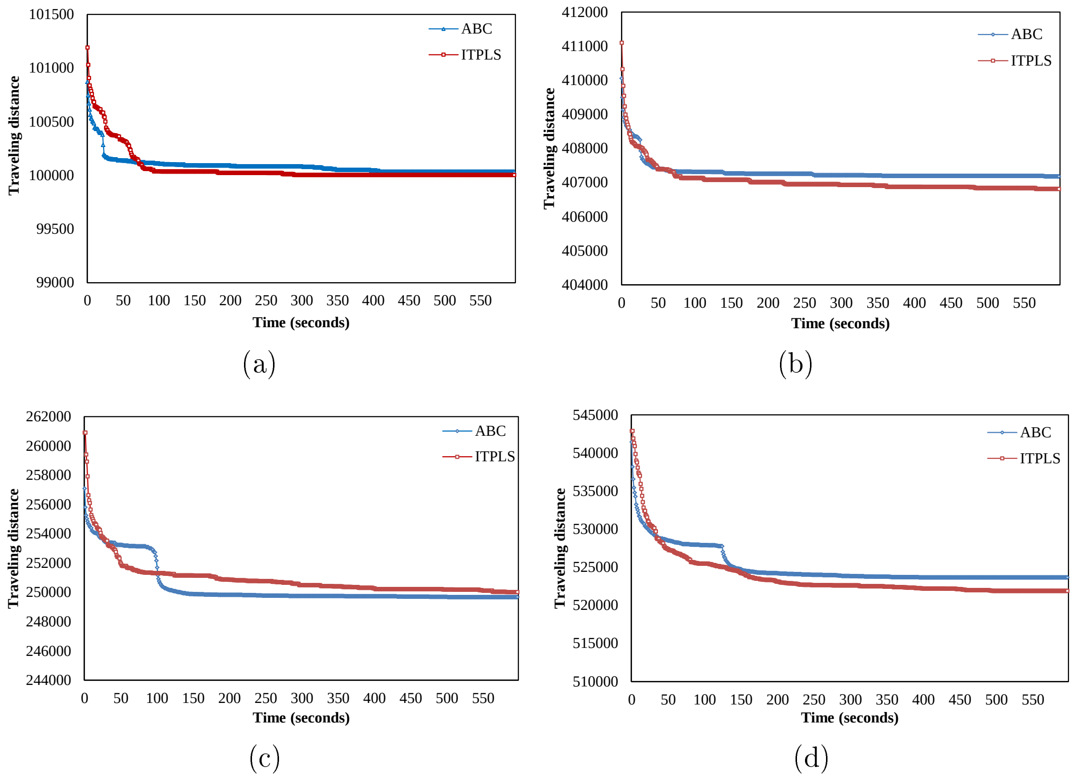

Fig. 5. Convergence charts (running profiles) of ITPLS and ABC (our implementation of [27]) for solving four representative instances of Set II (gr202-12, gr229-30, gr431-12, gr666-20). The results were obtained from 20 independent executions of each compared algorithm

4.6 Additional Computational Results of ITPLS

In this section, we are interested in the following question. Can we use our ITPLS algorithm as a post-optimizer to further improve high-quality solutions provided by another method? Such a study is relevant and allows us to test the ability of an algorithm to boost another powerful method [22].

For this purpose, we choose solutions achieved by the ABC algorithm of [27], which proves to be among the best performing algorithms. For this experiment, we disabled, in ITPLS, its greedy randomized initialization procedure of Section 3.3 and ran, under the same conditions as before, the algorithm with the best solution from ABC as its starting solution (denoted by ABC+ITPLS). The cutoff time of ABC+ITPLS is thus twice that of ITPLS. Since the instances of Set I are rather easy, we conducted this experiment only on Sets II and III. The results are reported in Tables 4 where $Gap_{ITPLS} = (F_{ITPLS+ABC} - F_{ITPLS})/F_{ITPLS} \times 100$ and $Gap_{ABC} = (F_{ITPLS+ABC} - F_{ABC})/F_{ABC} \times 100$, while the p-values from the Wilcoxon signed-rank test for different pairwise comparisons are shown in Table 5.

Table 4
Computational results of ABC+ITPLS on Sets II and III.

	ABC+ITPLS				
Instance	F_{best}	F_{avg}	σ_F	Gap_{ABC}	Gap_{ITPLS}
Set II					
gr202-12	99871	99925.45	100.37	0.00	0.00
gr202-25	173439	173558.35	52.25	-0.06	0.01
gr202-35	233749	233811.8	64.38	0.00	0.00
gr229-10	222167	222330.15	150.73	0.00	0.00
gr229-15	264146	264146	0	0.00	0.00
gr229-20	319669	319669	0	0.00	0.00
gr229-30	406664	406851.05	236.23	0.00	0.00
gr431-12	249031	249598.15	277.54	0.00	-0.16
gr431-25	348056	348419.15	205.6	0.00	-0.04
gr431-40	416189	416749.45	245.35	0.00	-0.09
gr666-10	388344	390898.4	1597.63	-0.47	-0.32
gr666-15	448240	449287.7	526.7	-0.08	0.00
gr666-20	520245	522339.15	922.95	-0.37	-0.17
gr666-30	651767	652998.7	784.45	-0.13	-0.01
Avg.	338684.07	339327.32			
Set III					
pr1002-5	316436	317180.85	534.12	0.00	-0.68
pr1002-10	381977	382711	417.75	-0.06	-0.30
pr1002-20	516238	517357.7	537.22	0.00	-0.32
pr1002-30	663247	665439.75	733.72	-0.21	-0.16
pr1002-40	805650	806890.45	971.64	-0.05	-0.04
fnl2461-3	109381	109956.25	268.56	-4.21	-0.57
fnl2461-6	118480	118880.45	350.81	-3.13	-0.03
fnl2461-12	143763	144287.4	306.39	-1.40	-0.87
fnl2461-24	221212	221421.95	159.32	-0.56	-0.13
fnl2461-30	267296	267498.5	156.51	-0.42	-0.02
fnl3461-3	156129	156850	335.96	-3.82	-0.40
fnl3461-6	165021	165474.75	329.31	-3.36	-0.26
fnl3461-12	187969	188572.1	326.99	-2.54	-0.13
fnl3461-24	264423	264917.85	224.25	-0.85	-0.25
fnl3461-30	307406	307589.65	137.97	-0.43	-0.05
fnl3461-40	384715	384817.3	49.5	-0.19	-0.11
pla5397-20	38144800	38210130	39590.88	-0.50	-0.49
pla5397-30	51180300	51216895	16850.28	-0.23	-0.31
pla5397-40	64199900	64267840	40279.67	-0.32	-0.13
pla5397-50	73996500	74001165	3625.53	-0.02	-0.07
pla5397-60	85269500	85282115	9904.29	-0.04	-0.06
pla6397-20	36161900	36247120	62012.04	-1.39	-0.67
pla6397-30	47419400	47479965	26265.17	-0.57	-0.28
pla6397-40	56677400	56746670	36123.05	-0.48	-0.32
pla6397-50	67222700	67271750	27741.9	-0.29	-0.19
pla6397-60	74850900	74906385	24934.32	-0.30	-0.18
pla7397-20	41464300	41658850	137498.13	-1.89	-0.81
pla7397-30	52813400	53010650	86252.3	-1.56	-0.70
pla7397-40	65075300	65234890	67013.38	-1.17	-0.56
pla7397-50	76554400	76635565	55852.77	-0.83	-0.19
pla7397-60	86382900	86484395	58943.59	-0.76	-0.28
Avg.	29755579	29795943			

The results show that ITPLS can greatly raise the quality of the solutions provided by ABC in terms of the best and the average objective values and performs better than ITPLS with its greedy randomized initialization. The p-values from the Wilcoxon signed-rank test indicate that the improvements are statistically significant. This experiment demonstrates that ITPLS can be beneficially combined with other algorithms to find high-quality solutions that cannot be discovered by running the underlying algorithms separately.

Table 5
Statistical results (p-values) from the Wilcoxon signed-rank test with a confidence level of 95% of different pairwise comparisons for the three benchmark sets

Algorithm pair	Set I		Set II		Set III	
	F_{best}	F_{avg}	F_{best}	F_{avg}	F_{best}	F_{avg}
ITPLS vs ABC	1	1	0.3125	0.9460	0.0012	0.0599
ABC+ITPLS vs ABC	-	-	0.0234	0.0017	1.17E-06	1.17E-06
ABC+ITPLS vs ITPLS	-	-	0.0391	1.22E-04	7.89E-06	3.62E-04

To push this study even further, we performed a complementary experiment to investigate the influence of the initial solution on the performance of ITPLS. For this purpose, we replaced the greedy randomized initialization procedure of ITPLS by ABC (denoted by ABC+ITPLS). For this experiment, we adopted the same experimental protocol of Section 4.2. Since ABC converges faster than ITPLS, we only assigned a fraction of the total run time to ABC and used the remaining time to run ITPLS. We experimented two cases where ABC was given the first 10% and 20% of the total run time, respectively. We use ITPLS+ABC(1) and ITPLS+ABC(2) to represent these two cases. We conducted this experiment only on Sets II and III since Set I is too easy for this study. Computational results are illustrated in Table C.1, where $Gap = (F_{ITPLS+ABC} - F_{ITPLS})/F_{ITPLS} \times 100$, while the p-values from the Wilcoxon signed-rank test for different pairwise comparisons are shown in Table C.2.

The results show that with the same run time, the combined use of ABC and ITPLS can reach better results than ITPLS alone in terms of the best and average values, especially for instances of Set III. In other words, high-quality initial solutions can help ITPLS to find still better solutions. The p-values from the Wilcoxon signed-rank test indicate that the improvements are statistically significant. One notices that the results of ABC+ITPLS(2) are better than ABC+ITPLS(1). Thus, ITPLS could be beneficially combined with other algorithms to find high-quality solutions that cannot be accessed by running the underlying algorithms separately.

5 Conclusions

We introduced the iterated two-phase local search algorithm for the challenging colored traveling salesman problem which has a number of real applications. The proposed algorithm relies on a combination of a local optima exploration phase and a local optima escaping phase. The local optima exploration phase is responsible for finding solutions of increasing quality by alternating inter-route optimization between routes and intra-route optimization of individual route, while the local optima escaping phase uses a solution destruction-reconstruction procedure to create new starting solutions for the local optima exploration phase.

Computational results of the proposed algorithm on three sets of 65 benchmark instances from the literature demonstrated its effectiveness and competitiveness compared to the existing methods. Especially, the algorithm was able to update the previous best-known results (improved upper bounds) for 22 instances (4 instances in Set II and 18 instances in Set III). These new upper bounds can be used by researchers for future research on CTSP (e.g., as reference values for new algorithm assessment, as initial bounds for exact algorithms). Moreover, given that CTSP can model several real problems, the code of our algorithm (that will be publicly available) can help practitioners to solve these practical applications.

For future research, there are several possibilities. First, given that existing studies on CTSP mainly focused on bio-inspired population frameworks such as genetic algorithms, ant colony optimization, and artificial bee colony, this work opens the way for designing effective algorithms based on other search frameworks such as local search and hybrid methods. Second, since CTSP is tightly related to other routing problems, it would be interesting to verify whether proven methods developed for these related problems could be effective for solving CTSP. Third, the basic idea of the proposed approach, in particular, mixing inter-route optimization and intra-route optimization is of general nature. It is worth investigating similar ideas to solve other routing problems such as the multiple traveling salesmen problem [2,11] and the colored balanced traveling salesman problem [6]. Finally, to the best of our knowledge, no exact algorithm exists for CTSP in the literature. There is thus much room for research in this direction.

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions which helped us to improve the paper. We would like to thank Prof. Jun Li, Prof. Alok Singh and Dr. Venkatesh Pandiri for providing their test problems. 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] T. Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures, Omega 34 (3) (2019) 209–219.

[3] J. Brandão, A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem, European Journal of Operational Research 284 (2) (2020) 559–571.

[4] I. M. Chao, B. Golden, E. Wasil, A computational study of a new heuristic for the site-dependent vehicle routing problem, INFOR: Information Systems and Operational Research 37 (3) (1999) 319–336.

[5] G. A. Croes, A method for solving traveling salesman problems, Operations Research 6 (6) (1958) 791–812.

[6] X. Dong, Y. Cai, A novel genetic algorithm for large scale colored balanced traveling salesman problem, Future Generation Computer Systems 95 (2019) 727–742.

[7] 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.

[8] 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.

[9] Z. H. Fu, J. K. Hao, A three-phase search approach for the quadratic minimum spanning tree problem, Engineering Applications of Artificial Intelligence 46 (2015) 113–130.

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

[11] B. Gavish, A note on "the formulation of the m-salesman traveling salesman problem", Management Science 22 (6) (1976) 704–705.

[12] F. Glover, M. Laguna, Tabu Search, Springer Science+Business Media New York, 1997.

[13] J. Grefenstette, R. Gopal, B. Rosmaita, D. Van Gucht, Genetic algorithms for the traveling salesman problem, in: Proceedings of the first International Conference on Genetic Algorithms and their Applications, vol. 160, Lawrence Erlbaum, 1985.

[14] 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.

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

[16] X. Lai, J. K. Hao, Iterated variable neighborhood search for the capacitated clustering problem, Engineering Applications of Artificial Intelligence 56 (2016) 102–120.

[17] X. Lai, D. Yue, J. K. Hao, F. W. Glover, Solution-based tabu search for the maximum min-sum dispersion problem, Information Sciences 441 (2018) 79–94.

[18] J. Li, X. Meng, X. Dai, Collision-free scheduling of multi-bridge machining systems: a colored traveling salesman problem-based approach, IEEE CAA J. Autom. Sinica 5 (1) (2018) 139–147.

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

[20] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problems, Operations Research 21 (2) (1973) 498–516.

[21] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, in: F. W. Glover, G. A. Kochenberger (eds.), Handbook of Metaheuristics, vol. 57 of International Series in Operations Research & Management Science, Kluwer / Springer, 2003, pp. 320–353.

[22] Z. Lu, J. K. Hao, Y. Zhou, Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem, Computers & Operations Research 111 (2019) 43–57.

[23] 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.

[24] T. Moyaux, E. Marcon, Cost of selfishness in the allocation of cities in the multiple travelling salesmen problem, Engineering Applications of Artificial Intelligence 89 (2020) 103429.

[25] V. Nguyen, C. Prins, C. Prodhon, A multi-start iterated local search with tabu list and path relinking for the two-echelon location-routing problem, Engineering Applications of Artificial Intelligence 25 (1) (2012) 56–71.

[26] V. Pandiri, A. Singh, Two metaheuristic approaches for the multiple traveling salesperson problem, Applied Soft Computing 26 (2015) 74–89.

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

[28] Y. B. Park, A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines, International Journal of Production Economics 73 (2) (2001) 175–188.

[29] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (8) (2007) 2403–2435.

[30] M. M. Silva, A. Subramanian, L. S. Ochi, An iterated local search heuristic for the split delivery vehicle routing problem, Computers and Operations Research 53 (2015) 234–249.

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

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

[33] L. Tang, J. Liu, A. Rong, Z. Yang, A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex, European Journal of Operational Research 124 (2) (2000) 267–282.

[34] Y. Wang, Q. Wu, F. W. Glover, Effective metaheuristic algorithms for the minimum differential dispersion problem, European Journal of Operational Research 258 (3) (2017) 829–843.

[35] D. L. Woodruff, E. Zemel, Hashing vectors for tabu search, Annals of Operational Research 41 (2) (1993) 123–137.

[36] X. Xu, J. Li, M. Zhou, Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems, IEEE Transactions on Intelligent Transportation Systems (2020) 1–11.

[37] Y. Zhou, J. K. Hao, An iterated local search algorithm for the minimum differential dispersion problem, Knowledge Based Systems 125 (2017) 26–38.

A Streamlined computation technique

This Appendix presents the streamlined computation technique for fast updates of the move gain matrix δ used by the solution-based tabu search procedure. In this procedure, all neighbor solutions are represented by $s \oplus Insert(\cdot)$ where s is the current solution. The cost variation between the two solutions (i.e., the move gain of $Insert(k_1, i_1, k_2, i_2)$) is given by $\delta(k_1, i_1, k_2, i_2) = F(s \oplus Insert(\cdot)) - F(s)$. The move gain $\delta(k_1, i_1, k_2, i_2)$ can be calculated efficiently by Eq. (A.1),

$$\delta(k_1, i_1, k_2, i_2) = c_{i_1 i_2} + c_{i_2 i_1^n} - c_{i_1 i_1^n} + c_{i_2^p i_2^n} - c_{i_2^p i_2} - c_{i_2 i_2^n} \qquad \text{(A.1)}$$

where i_2^p and i_2^n are the previous and next city of i_2. The matrix δ saves the move gains of all neighbor solutions $s \oplus Insert(\cdot)$. Computing $\delta(k_1, i_1, k_2, i_2)$ with Eq. (A.1) needs $O(1)$ time, instead of $O(n)$ by Eq. (8).

However, the solution-based tabu search procedure needs to select the best eligible neighbor solution to update the current solution, and this requires $O(|S| \times n)$ time by computing δ for all neighbor solutions based on Eq. (A.1). To accelerate this computation and update of move gains, we adapted the streamlined technique [10], which was initially developed for the graph coloring problem.

During the first step of the solution-based tabu search procedure, we fill the gain matrix $\delta(\cdot)$ for all neighbor solutions of $s \oplus Insert(\cdot)$. If $Insert(k_1, i_1, k_2, i_2)$ is performed during the search, we just need to update parts of the gain matrix. As shown in Eqs. (A.2-A.5), the gain matrix $\delta(\cdot)$ for $Insert(k_3, i_3, k, i)$ can be updated after performing $Insert(k_1, i_1, k_2, i_2)$ as follows.

$$\begin{aligned}\delta(k_3, i_3, k, i) = \Big(&c_{i_3 i} + c_{i i_3^n} - c_{i_3 i_3^n} + c_{i^p i^n} - c_{i^p i} - c_{i i^n}\\ k_3 \in \{k_1, k_2\}, k \in M \backslash \{k_1, k_2\}, i \in S, i_3 &\in \{i_1, i_2, i_2^p\}\Big)\end{aligned} \tag{A.2}$$

$$\begin{aligned}\delta(k_3, i_3, k, i) = \Big(&c_{i_3 i} + c_{i i_3^n} - c_{i_3 i_3^n} + c_{i^p i^n} - c_{i^p i} - c_{i i^n}\\ k_3 \in \{k_1\}, k \in \{k_2\}, i_3 \in \{i, i_2\}, i &\in S \backslash \{i_2^p, i_2^n\}\Big)\end{aligned} \tag{A.3}$$

$$\begin{aligned}\delta(k_3, i_3, k, i) = \Big(&c_{i_3 i} + c_{i i_3^n} - c_{i_3 i_3^n} + c_{i^p i^n} - c_{i^p i} - c_{i i^n}\\ k_3 \in \{k_2\}, k \in \{k_1\}, i_3 \in \{i_2^p\}, i &\in S \backslash \{i_1, i_2, i_1^n\}\Big)\end{aligned} \tag{A.4}$$

$$\begin{aligned}\delta(k_3, i_3, k, i) = \Big(&c_{i_3 i} + c_{i i_3^n} - c_{i_3 i_3^n} + c_{i^p i^n} - c_{i^p i} - c_{i i^n}\\ k_3 \in M \backslash \{k\}, k \in \{k_1 k_2\}, i_3 \in V, i &\in \{i_1, i_2, i_1^n i_2^p, i_2^n\}\Big)\end{aligned} \tag{A.5}$$

The time complexity for these operations is $O(3 \times S)$, $O(2 \times |s_k|)$, $O(|s_{k_1}|)$, $O(5 \times n)$ respectively. Therefore, the time complexity of updating the move gain matrix becomes $O(n)$, which is significantly smaller than $O(|S| \times n)$. This technique thus accelerates greatly the update of the move gain matrix by avoiding many unnecessary computations.

B Comparative results under the cutoff times of [27]

This Appendix (Table B.1) shows detailed results of ABC [27], our implementation of [27] and ITPLS on 34 instances of Sets I and II under the cutoff times (see column 't(s)') used in [27]: 1 second for instances with $21-41$

cities, 5 seconds for instances with $51-101$ cities and 60 seconds for instances with $202-666$ cities. ABC in [27] was ran on a 3.4GHz processor, while our implementation of ABC and ITPLS were ran on a slower 2.3GHz processor.

Table B.1
Comparative results of ABC [27] and ABC (our implementation of [27]) and ITPLS under the stopping conditions of ABC [27]. Unavailable results are indicated by the symbol '-' while the best results of the compared methods are highlighted in bold.

		ABC [27]		ABC (our implementation of [27])			ITPLS (this work)			
Instance	t(s)	f_{best}	f_{avg}	f_{best}	f_{avg}	σ	f_{best}	f_{avg}	σ	*Gap*(%)
eil21-2	1	**144.92**	144.92	**144.92**	144.92	0.00	**144.92**	149.50	3.13	0.00
eil21-3	1	**157.48**	157.48	**157.48**	157.48	0.00	**157.48**	160.62	2.71	0.00
eil31-2	1	**259.36**	259.36	**259.36**	261.01	0.75	262.32	266.86	3.53	1.14
eil31-3	1	**295.31**	295.31	**295.31**	295.31	0.00	295.36	300.20	2.88	0.02
eil31-4	1	**315.97**	315.97	**315.97**	315.97	0.01	316.02	319.49	3.06	0.02
eil41-2	1	**346.24**	346.24	**346.24**	347.68	1.17	347.86	354.94	5.63	0.47
eil41-3	1	**367.84**	367.84	368.81	369.32	0.72	**367.84**	374.90	6.04	0.00
eil41-4	1	**392.14**	392.14	**392.14**	393.20	0.62	392.49	398.75	3.73	0.09
eil51-2	5	**478.08**	478.08	**478.08**	478.08	0.00	**478.08**	478.08	0.00	0.00
eil51-3	5	**469.50**	469.50	**469.50**	469.50	0.00	**469.50**	469.50	0.00	0.00
eil51-4	5	**489.99**	489.99	**489.99**	489.99	0.00	**489.99**	489.99	0.00	0.00
eil51-5	5	**525.98**	525.98	**525.98**	525.98	0.00	**525.98**	525.98	0.00	0.00
eil76-3	5	**593.28**	593.28	**593.28**	593.28	0.00	**593.28**	594.07	1.41	0.00
eil76-4	5	**603.79**	603.79	**603.79**	603.79	0.00	**603.79**	603.82	0.13	0.00
eil76-5	5	**651.99**	651.99	**651.99**	651.99	0.00	**651.99**	652.91	0.95	0.00
eil76-6	5	**672.73**	672.73	**672.73**	672.73	0.00	**672.73**	673.15	1.29	0.00
eil101-4	5	**726.82**	726.82	**726.82**	726.82	0.00	**726.82**	727.08	0.63	0.00
eil101-5	5	**779.15**	779.15	**779.15**	779.15	0.00	**779.15**	779.15	0.25	0.00
eil101-6	5	**759.55**	759.55	**759.55**	759.55	0.00	**759.55**	759.55	0.00	0.00
eil101-7	5	**798.85**	798.85	**798.85**	798.85	0.00	**798.85**	798.85	0.00	0.00
gr202-12	60	-	-	100032.00	100173.10	60.41	**99871.00**	100073.55	158.65	-0.16
gr202-25	60	-	-	173547.00	173796.90	45.77	**173427.00**	173566.25	85.35	-0.07
gr202-35	60	-	-	**233749.00**	234093.75	38.01	**233749.00**	234017.35	109.00	0.00
gr229-10	60	**222167.00**	222408.40	**222167.00**	222722.20	170.41	222279.00	222573.50	170.83	0.05
gr229-15	60	**264146.00**	264225.20	**264146.00**	265178.85	361.58	**264146.00**	264280.25	324.14	0.00
gr229-20	60	**319669.00**	319669.90	**319669.00**	320298.00	288.67	**319669.00**	319990.30	493.09	0.00
gr229-30	60	**406664.00**	406768.50	**406664.00**	408154.75	162.49	**406664.00**	407181.20	382.65	0.00
gr431-12	60	-	-	250535.00	251674.25	577.30	**249562.00**	251836.00	2117.36	-0.39
gr431-25	60	-	-	349644.00	350708.30	519.31	**349400.00**	351285.15	1374.60	-0.07
gr431-40	60	-	-	417773.00	419297.90	682.31	**417161.00**	420031.25	2059.85	-0.15
gr666-10	60	**391831.00**	393949.00	396752.00	401124.95	1680.20	399376.00	406624.00	3818.98	1.93
gr666-15	60	**448981.00**	449978.60	452475.00	454511.50	828.90	449369.00	458238.85	5616.10	0.09
gr666-20	60	**522403.00**	523358.60	527001.00	528630.21	940.14	523747.00	527617.42	3361.88	0.26
gr666-30	60	653224.00	653857.20	656486.00	658433.30	1047.39	**652926.00**	657084.80	3088.77	-0.05
Avg.	-	-	-	140602.06	**141136.25**	217.83	**140328.82**	141302.27	682.25	0.09

C Additional comparative results

This Appendix (Tables C.1 and C.2) shows comparative results of ABC+ITPLS(1) and ABC+ITPLS(2) on Sets II and III. The cutoff time per run for both algorithms was set to be same as ITPLS. ABC was given the first 10% and 20% of the total run time, while the remaining time was allocated to ITPLS.

Table C.1. Computational results of the ABC+ITPLS with sets II and III.

	ABC+ITPLS(1)				ABC+ITPLS(2)			
Instance	F_{best}	F_{avg}	σ_F	Gap	F_{best}	F_{avg}	σ_F	Gap
set II								
gr202-12	99871	99994	114.94	0.00	99871	99963.5	109.59	0.00
gr202-25	173353	173520.8	81.26	-0.04	173415	173534.9	70.12	0.00
gr202-35	233749	233819.5	69.35	0.00	233749	233826.95	73.96	0.00
gr229-10	222167	222395.05	152.4	0.00	222167	222363.85	163.98	0.00
gr229-15	264146	264191.2	119.34	0.00	264146	264184.2	117.58	0.00
gr229-20	319669	319669	0	0.00	319669	319669	0	0.00
gr229-30	406664	406843.2	227.25	0.00	406664	406814.5	220.55	0.00
gr431-12	249644	250250.1	394.45	0.09	249557	250173.35	373	0.05
gr431-25	348612	349123.4	306.08	0.12	348187	348787.35	309.67	0.00
gr431-40	417035	418169.35	590.81	0.12	416564	418041.2	814.17	0.00
gr666-10	389620	392741.2	1435.54	0.01	389916	393175.45	1607.31	0.09
gr666-15	448364	449410.75	851.89	0.02	448352	449539.05	772.16	0.02
gr666-20	520642	522785.7	1145.85	-0.10	520664	522938.8	1371.09	-0.09
gr666-30	652007	653554.35	917.37	0.03	650898	653317.35	1153.52	-0.14
Avg.	338967.36	339747.69			338844.21	339737.82		
set III								
pr1002-5	317972	318848.55	683.8	-0.19	317423	318621.65	517.41	-0.37
pr1002-10	382181	383784.45	824.34	-0.24	382058	383731.65	700.66	-0.28
pr1002-20	516471	517822.2	837.05	-0.28	516011	517612.15	1062.74	-0.37
pr1002-30	663447	665384.6	809	-0.13	664126	665291.75	655.36	-0.03
pr1002-40	805631	807752.95	1227.3	-0.04	805446	807387.9	1096.99	-0.06
fnl2461-3	109797	110290.2	292.13	-0.19	109676	110331.15	303.03	-0.30
fnl2461-6	118363	119062.45	304.98	-0.13	118607	118984.5	280.24	0.08
fnl2461-12	144086	144438.45	257.65	-0.65	143860	144420.4	296.6	-0.80
fnl2461-24	221185	221450.25	144.96	-0.14	221089	221405.8	153.95	-0.18
fnl2461-30	267405	267618.7	139.26	0.02	267303	267596.5	202.15	-0.02
fnl3461-3	156263	157082.8	445.69	-0.31	155882	156820.2	469.34	-0.56
fnl3461-6	165303	165995.7	388.21	-0.09	165105	165803.3	432.31	-0.21
fnl3461-12	188128	188975.25	543.15	-0.05	188417	188975.05	369.78	0.10
fnl3461-24	264925	265254.2	218.69	-0.06	264630	265121.6	237.43	-0.17
fnl3461-30	307415	307726.05	165.08	-0.05	307541	307714.6	149.63	-0.01
fnl3461-40	384766	384866.15	55.27	-0.09	384729	384894.75	68.95	-0.10
pla5397-20	38147300	38286715	49197.3	-0.48	38195400	38254655	40671.67	-0.36
pla5397-30	51211900	51279410	40246.27	-0.25	51216500	51256400	30042.01	-0.24
pla5397-40	64276400	64358615	49999.71	-0.01	64213900	64332570	49797.59	-0.11
pla5397-50	74002500	74011550	7204.57	-0.07	73998200	74005500	5714.62	-0.07
pla5397-60	85271900	85297540	15381.99	-0.06	85270900	85290610	15434.95	-0.06
pla6397-20	36214800	36375825	87395.01	-0.52	36238000	36346385	74332.25	-0.46
pla6397-30	47474400	47546215	39199.32	-0.16	47454300	47518230	38155.97	-0.21
pla6397-40	56728200	56816670	47028.35	-0.23	56712700	56791770	35066.93	-0.26
pla6397-50	67282900	67329035	30575	-0.10	67225600	67303575	30562.19	-0.18
pla6397-60	74908400	74964100	35971.36	-0.10	74902800	74961445	29143.61	-0.11
pla7397-20	41483600	41727000	99191.52	-0.77	41604900	41722805	83035.59	-0.48
pla7397-30	52961500	53103975	61320.19	-0.42	52898400	53116050	74979.07	-0.54
pla7397-40	65211100	65351920	64780.82	-0.35	65155900	65347285	75985.82	-0.44
pla7397-50	76613500	76746165	87943.95	-0.12	76581100	76710595	69038.08	-0.16
pla7397-60	86566700	86633247.37	35396.8	-0.07	86365100	86529380	97144.14	-0.30
Avg.	29786079	29834011			29775665	29822967		

Table C.2
Statistical results (p-values) from the Wilcoxon signed-rank test with a confidence level of 95% of different pairwise comparisons for the three sets.

Algorithm pair	set I		set II		set III	
	F_{best}	F_{avg}	F_{best}	F_{avg}	F_{best}	F_{avg}
ITPLS vs ABC	1	1	0.3125	0.9460	0.0012	0.0599
ITPLS vs ABC+ITPLS(1)	-	-	0.3125	0.8552	1.30E-06	1.58E-06
ITPLS vs ABC+ITPLS(2)	-	-	0.8438	0.6698	2.56E-06	1.58E-06
ABC vs ABC+ITPLS(1)	-	-	0.25	0.7354	3.75E-06	4.97E-06
ABC vs ABC+ITPLS(2)	-	-	0.1289	0.7354	3.10E-06	4.53E-06