Solving the Clustered Traveling Salesman Problem via TSP methods

Yongliang Lu, Jin-Kao Hao, and Qinghua Wu

1School of Economics and Management, Fuzhou University, 350116 Fuzhou, China
2LERIA, Université d’Angers, 2 Boulevard Lavoisier, 49045 Angers, France
3School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China

Corresponding author:
Third Author
Email address: qinghuawu1005@gmail.com

ABSTRACT

The Clustered Traveling Salesman Problem (CTSP) is a variant of the popular Traveling Salesman Problem (TSP) arising from a number of real-life applications. In this work, we explore a transformation approach that solves the CTSP by converting it to the well-studied TSP. For this purpose, we first investigate a technique to convert a CTSP instance to a TSP and then apply powerful TSP solvers (including exact and heuristic solvers) to solve the resulting TSP instance. We want to answer the following questions: How do state-of-the-art TSP solvers perform on clustered instances converted from the CTSP? Do state-of-the-art TSP solvers compete well with the best performing methods specifically designed for the CTSP? For this purpose, we present intensive computational experiments on various benchmark instances to draw conclusions.

Keywords: Traveling salesman; Heuristics; Clustered traveling salesman; Problem transformation.

INTRODUCTION

The Clustered Traveling Salesman Problem (CTSP), originally proposed by Chisman (1975), is an extension of the classic Traveling Salesman Problem (TSP) where the cities are grouped into clusters and the cities of each cluster must be visited contiguously. Formally, the problem is defined on a symmetric complete weighted graph $G = (V,E)$ with a set of vertices $V = \{1,2,...,n\}$ and a set of edges $E = \{(i,j) : i,j \in V, i \neq j\}$. The vertex set $V$ is partitioned into disjoint clusters $V_1, V_2, ..., V_m \ (V_1 \cup V_2 \cup ... \cup V_m = V)$. Let $C$ be an $n \times n$ symmetric distance matrix such that $c_{ij} \ (i,j \in 1,2,...,n,i \neq j)$ represents the travel cost between two corresponding vertices $i$ and $j$, and satisfies the triangle inequality rule. The objective of the CTSP is to find a minimum cost Hamiltonian circuit over all the vertices, where the vertices of each cluster must be visited consecutively.

The CTSP can be formally modelled as the following integer programming model described in Chisman (1975), where without loss of generality, the salesman is assumed to leave origin city 1 and return to 1.

$$\text{min } f = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij} \quad (1)$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in V \quad (2)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in V \quad (3)$$
In this model, the binary variable $x_{ij} = 1$ if city $j$ is visited immediately after city $i$; $x_{ij} = 0$ otherwise. Objective function (1) seeks to minimize the total distance traveled by the salesman. Constraints (2) and (3) ensure that each city is visited exactly once. Constraints (4) eliminate subtours, while constraints (5) guarantee that the cities of each cluster are visited contiguously. The remaining constraints are related to the decision variables.

The above subtour elimination constraints (4) are called MTZ formulation (Miller et al., 1960). Although MTZ is simple to implement, it provides a very poor linear relaxation (Campuzano et al., 2020). Many compact formulations have been proposed to replace Constraints (4). According to the literature, a multi-commodity flow formulation (Wong, 1980; Claus, 1984) was proven to provide a strong linear relaxation, without compromising its simplicity. In the multi-commodity flow formulation, let $k = 2, 3, ..., n$ be $n - 1$ commodities, and let $y^k_{ij}$ be a nonnegative decision variable which represents the flow on the arc $(i, j) \in E$ for the commodity $k$ from city 1 to city $k$. Then, another alternative mathematical model for the CTSP is constituted of the objective function (1) and the constraints (2), (3), (5), (6) along with the following subtour elimination constraints:

$$0 \leq y^k_{ij} \leq x_{ij} \quad \forall i, j, k \in V, k \neq 1$$

$$\sum_{i=2}^{n} y^k_{1i} = 1 \quad \forall k \in V \setminus \{1\}$$

$$\sum_{i=2}^{n} y^k_{ki} = 0 \quad \forall k \in V \setminus \{1\}$$

$$\sum_{i=1}^{n} y^k_{ik} = 1 \quad \forall k \in V \setminus \{1\}$$

$$\sum_{j=1}^{n} y^k_{kj} = 0 \quad \forall k \in V \setminus \{1\}$$
\[ \sum_{i=1}^{n} y_{ij} - \sum_{i=1}^{n} y_{ji} = 0 \quad \forall j, k \in V \setminus \{1\}, j \neq k \]  

Constraints (8) only allow flow in an arc \((i, j)\) if and only if it is traversed by the salesman (i.e., \(x_{ij} = 1\)). Constraints (9) ensure that city 1 is the source of one unit of each commodity \(k \in V \setminus \{1\}\) and Constraints (10) avoid that the flow of each commodity \(k \in V \setminus \{1\}\) returns to city 1. Constraints (11) and (12) guarantees that one flow unit of commodity \(k\) enters to city \(k\) and it does not leave the city \(k\). Constraints (13) ensure flow conservation at each city, apart from city 1 and for commodity \(k\) at city \(k\).

One notices that the CTSP is equivalent to the TSP when there is a single cluster or when each cluster contains exactly one vertex. Therefore, the CTSP is NP-hard, and thus computationally challenging in the general case. From a practical perspective, the CTSP is a versatile modeling tool for several operational research applications arising in a wide variety of areas, including automated warehouse routing (Chisman, 1975), emergency vehicle dispatching (Weintraub et al., 1999), production planning (Lokin, 1979), disk defragmentation (Laporte and Palekar, 2002), and commercial transactions with supermarkets, shops and grocery suppliers (Ghaziri and Osman, 2003). As a result, effective solution methods for the CTSP can help to solve these practical problems. Indeed, the computational challenge and the wide range of applications of the problem have motivated a variety of approaches that are reviewed in Section 1. However, unlike the classic TSP problem for which many powerful methods have been introduced in the past decades, studies on the CTSP are still quite limited.

Moreover, the CTSP belongs to the large class of traveling salesman problems. Among the TSP variants, the generalized traveling salesman problem (GTSP) (Srivastava et al., 1969; Cosma et al., 2021) and the family traveling salesman problem (FTSP) (Morán-Mirabal et al., 2014; Pop et al., 2018) share similarities with the CTSP. In the GTSP, the set of vertices is divided into clusters and the objective is to find a minimum-cost tour passing through one vertex from each cluster. In the FTSP, the set of vertices is also divided into clusters (called families) and the objective is to visit a predefined number of vertices in each family at a minimum cost.

In this work, we investigate the problem transformation approach proposed in Chisman (1975), which converts the CTSP to the TSP and assess the interest of popular modern TSP solvers for solving the resulting TSP instances. To our knowledge, this is the first large computational study testing modern TSP solvers on solving the CTSP. The work is motivated by the following considerations. First, intensive researches on the TSP have led to the development of many very powerful solvers. Thus, it is interesting to know whether we can take advantage of these solvers to effectively solve the CTSP. Second, the TSP instances converted from the CTSP are characterized by their cluster structures. These instances constitute interesting test cases for existing TSP solvers. This work aims thus to answer the following questions.

1. How do state-of-the-art exact TSP solvers perform on clustered instances converted from the CTSP?
2. How do state-of-the-art inexact (heuristic) TSP solvers perform on clustered instances converted from the CTSP?
3. Do state-of-the-art TSP solvers compete well with the best performing methods specifically designed for the CTSP?

To our knowledge, Questions 1 and 3 have never been investigated in the literature. Regarding Question 2, two previous studies (Neto, 1999; Helsgaun, 2014) are of interest. However, they are limited because they only concern one TSP algorithm, i.e., the local search based LKH solver (Helsgaun, 2009), while ignoring other powerful TSP solvers like GA-EAX (Nagata and Kobayashi, 1997) and Concorde (Applegate et al., 2006a). Answering these questions helps to enrich the state-of-the-art of solving the CTSP and gain novel knowledge on using modern TSP methods to solve new problems. Finally, we mention that the transformation approach was also tested in Lokin (1979) and Jongens and Volgenant (1985). However, these studies are clearly outdated and don’t provide useful information as to the questions we want to investigate.

The remainder of this paper is organized as follows. Section 1 reviews existing solution methods for the CTSP. Section 2 presents the transformation of the CTSP to the TSP and three powerful TSP methods (solvers). Section 3 shows computational studies of the TSP solvers applied to the clustered instances and comparisons with existing algorithms dedicated to the CTSP. Finally, concluding remarks are provided in Section 5.
1 LITERATURE REVIEW ON EXISTING SOLUTION METHODS

There are several dedicated solution algorithms for solving the CTSP that are based on exact, approximation, and metaheuristic approaches.

Along with the introduction of the CTSP, Chisman (1975) proposed a branch-and-bound algorithm to solve the integer programming model presented in the Introduction section. Jongens and Volgenant (1985) developed an algorithm based on the 1-tree relaxation to provide lower bounds as well as a heuristic to find satisfactory upper bounds. Mestria et al. (2013) used the mathematical formulation of Chisman (1975) and IBM Parallel CPLEX solver (version 11.2) to obtain lower bounds for medium CTSP instances \( |V| \leq 1000 \).

Various \( a \)-approximation algorithms (Anily et al., 1999; Gendreau et al., 1997; Guttmann-Beck et al., 2000) have been developed for the CTSP. These approximation algorithms require either the starting and ending vertices in each cluster or a prespecified order of visiting the clusters in the tour as inputs, and solve the inter-cluster and intra-cluster problems independently. Bao and Liu (2012) presented a new 2.17-approximation algorithm where no starting and ending vertices were specified. Later, Bao et al. (2017) provided a 2.5-approximation algorithm for another version of the CTSP where the starting vertex of each cluster is given while the ending vertex is not specified. Recently, Kawasaki and Takazawa (2020) improved the approximation ratio for the CTSP by incorporating a recent approximation algorithm for the TSP by Zenklusen (2019).

Given that the CTSP is an NP-hard problem, a number of heuristic and metaheuristic algorithms have also been investigated, which aim to provide high-quality solutions in acceptable computation time, but without provable optimal guarantee of the attained solutions. For example, Laporte et al. (1997) presented a tabu search algorithm to solve a particular case of the CTSP, where the clusters are visited in a prespecified order. Potvin and Guertin (1996) developed a genetic algorithm for the CTSP that finds inter-cluster paths and then intra-cluster paths. Later, Ding et al. (2007) proposed a two-level genetic algorithm for the CTSP. In the first level, a genetic algorithm is used to find the shortest Hamiltonian cycle for each cluster. In the second level, a modified genetic algorithm is applied to merge the Hamiltonian cycles of all the clusters into a complete tour.

In addition to these early heuristic algorithms, Mestria et al. (2013) investigated GRASP (Greedy Randomized Adaptive Search Procedure) with path-relinking. Among the six proposed heuristics, one heuristic corresponds to the traditional GRASP procedure whereas the other heuristics include different path-relinking procedures. Mestria (2016) studied a hybrid heuristic, which is based on a combination of GRASP, Iterated Local Search (ILS) and Variable Neighborhood Descent (VND). Recently, Mestria (2018) presented another complex hybrid algorithm (VNRDGILS) which mixes GRASP, ILS, and Variable Neighborhood Random Descent to explore several neighborhoods. According to the computational results reported in Mestria et al. (2013); Mestria (2016, 2018), these GRASP-based algorithms are among the best performing heuristics specially designed for the CTSP currently available in the literature. In addition, Hà et al. (2022) proposed a metaheuristic method based on the ILS framework with problem-tailored operators for a version of the CTSP where the order of visiting the clusters is prespecified.

Existing studies have significantly contributed to better solving the CTSP. According to the computational results reported in the literature, due to the NP-hardness of the problem, only small CTSP instances were able to be solved to optimality with the exact algorithms. The approximation algorithms provide solutions for the CTSP within a given approximation factor. However, due to the high approximation factors involved (e.g., 5/3 (Anily et al., 1999), 3/2 Gendreau et al. (1997), 2.17 (Bao and Liu, 2012), and 2.5 (Bao et al., 2017)), these approximation algorithms are not practical for solving large instances. To deal with large CTSP instances, heuristic and metaheuristic algorithms are often preferred to find sub-optimal solutions within an acceptable computation time.

2 SOLVING THE CTSP VIA TSP METHODS

2.1 Transformation of the CTSP to the TSP

As the literature review shows, a number of dedicated solution approaches have been developed to solve the CTSP. However, one observes that these approaches have difficulty producing robustly and consistently high-quality solutions for large-scale CTSP instances with tens of thousands of vertices. Moreover, the best performing CTSP methods (e.g., VNRDGILS (Mestria, 2018), HHGILS (Mestria, 2016), and GPR1R2 (Mestria et al., 2013)) are computationally expensive (e.g., requiring 1080 seconds...
to find good solutions for instances with $1173 \leq n \leq 2000$.

On the other hand, problem transformation has been highly successful in solving several difficult optimization problems such as the Latin square completion problem via graph coloring (Jin and Hao, 2019) and the winner determination problem via weighted maximum cliques (Wu and Hao, 2015). It is known that the CTSP can be transformed to the conventional TSP (Chisman, 1975). Therefore, in principle, the CTSP can be solved by any TSP algorithm. However, to our knowledge, no computational study on using problem transformation to solve the CTSP has been presented in the literature. This work fills the gap by exploring the problem transformation approach of Chisman (1975) and testing three representative state-of-the-art TSP solvers including both exact and inexact solution approaches.

The basic idea of transforming the CTSP to the TSP is to add a large artificial cost $M$ to all inter-cluster edges in order to force the salesman to visit all the cities within each cluster before leaving it. Given a CTSP instance $G = (V, E)$ with distance matrix $C$, we define a TSP instance $G' = (V', E')$ with distance matrix $C'$ as follow.

- Define $V = V'$ and $E = E'$.
- Define the travel distance $c_{ij}'$ in $G'$ by

$$c_{ij}' = \begin{cases} c_{ij} + M & \text{if } i \text{ and } j \text{ belong to different clusters} \\ c_{ij} & \text{otherwise} \end{cases}$$

Obviously, if the value of $M$ is sufficiently large, then the best Hamiltonian cycle in $G'$ is a feasible CTSP solution in $G$, in which the vertices of each cluster are visited contiguously.

**Property.** An optimal solution to the TSP instance corresponds to an optimal solution to the original CTSP instance.

**Proof.** Let $S'$ and $S$ be the optimal solutions of the TSP instance $G'$ and the original CTSP instance $G$, respectively. Let $m$ be the number of clusters of $G$. To minimize the total travel cost, there are only $m$ inter-cluster edges in $S'$. Therefore, $S'$ is a feasible CTSP solution for $G$ and satisfies the following relation:

$$f(S') = f(S) + m \times M$$

Obviously, $S'$ corresponds to $S$ by subtracting the constant $m \times M$.

### 2.2 Solution methods for the TSP

There are numerous solution methods for the TSP. In this work, we adopt three very powerful TSP solvers whose codes are publicly available, including one exact solver (Concorde (Applegate et al., 2006a)) and two inexact (heuristic) solvers (LHK-2 (Helsgaun, 2009) and GA-EAX (Nagata and Kobayashi, 2013)).

Notice that the TSP instance converted from a CTSP instance has a particular feature that the vertices are grouped into clusters and the distance between each pair of vertices within a same cluster is in general small, while this distance is large for two vertices from different clusters. Along with the presentation of the TSP solvers, we discuss their suitability for solving such clustered instances each time this is appropriate.

#### 2.2.1 Exact Concorde solver

Concorde is an advanced exact TSP solver for the symmetric TSP based on Branch-and-Bound and problem specific cutting plane methods (Applegate et al., 2006a). It makes use of a specifically designed QSopt linear programming solver. According to Hoos and Stutzle (2014), Concorde is the best performing exact algorithm for the TSP. As shown in Applegate et al. (2006b), Concorde can solve benchmark instances from TSPLIB with up to 1000 vertices to optimality within a reasonable computation time and it also solves large TSP instances at the cost of a long computation time.

The run time behavior of Concorde has been investigated essentially on random uniform instances. For instance, Applegate et al. (2006b) investigated the run time required by Concorde for solving random uniform instances and indicated that the run time increases as an exponential function of instance size $|V|$. Hoos and Stutzle (2014) further demonstrated that the median run time required by Concorde scales with instance size $|V|$ of the form $ab\sqrt{|V|}$ ($a \approx 0.21, b \approx 1.24$) on the widely studied class of uniform random TSP instances. To our knowledge, no study has been reported concerning the behavior of Concorde on sharply clustered instances. As a result, the current study will provide useful information on this issue.
2.2.2 Lin-Kernighan based heuristic solver

According to the TSP literature, a majority of the best performing TSP heuristic algorithms is based on the Lin-Kernighan (LK) heuristic (Lin and Kernighan, 1973) and its extensions. The LK heuristic is a variable-depth \( k \)-opt local search procedure, where the \( k \)-opt neighborhood is partially searched with a smart pruning strategy. LK explores the most promising neighbors within the \( k \)-opt neighborhood, that is, the set of feasible tours obtained by removing \( k \) edges and adding other \( k \) edges such that the resulting tour is feasible. Several improved versions of the basic LK heuristic have been introduced within the iterated local search framework (e.g., Applegate et al. (2003); Helsgaun (2000, 2009); Martin et al. (1991)).

Among these iterated LK algorithms, Helsgaun’s LKH (Helsgaun, 2000, 2009) is the uncontested state-of-the-art heuristic TSP solver. Helsgaun (2000) developed an iterated version of LK together with an efficient implementation of the LK algorithm, known as the Lin-Kernighan-Helsgaun (LKH-1) heuristic, where a 5-opt move is used as the basic move to broaden the search and an \( \alpha \)-measure method based on sensitivity analysis of minimum spanning trees is used to restrict the search to relative few of the \( \alpha \)-nearest neighbors of a vertex to speed up the search process. Later, Helsgaun (2009) further extended LKH-1 by developing a highly effective implementation of the \( k \)-opt procedure (called LKH-2), which eliminated many of the limitations and shortcomings of LKH-1. Furthermore, LKH-2 specially extended the data structures of LKH-1 to solve very large TSP instances. The main features of LKH-2 include (1) using sequential and non-sequential \( k \)-opt moves, (2) using several partitioning procedures to partition a large TSP instance into smaller subproblems, (3) using a tour merging procedure to generate a better solution from two or more local optimum solutions, and (4) applying a backbone-guided search to guide the local search to make biased local perturbations. LKH-2 is considered to be one of most effective heuristic methods for finding very high-quality solutions for various large TSP instances (Dubois-Lacoste et al., 2015).

However, the LK algorithm and any LK-based algorithms require much longer running times on clustered instances of the TSP than on uniformly distributed instances (Neto, 1999). The main reason why the LK heuristic stumbles on clustered instances is that relatively large inter-cluster edges serve as bait edges. During the LK search, when removing such a bait edge, the LK heuristic is tricked into long and often fruitless searches. More precisely, each time an edge bridging two clusters is removed, the cumulative gain rises enormously, and the procedure is encouraged to perform very deep searches. To alleviate the problem, a cluster compensation technique was proposed in Neto (1999) for the Lin-Kernighan heuristic to limit its performance degradation. Helsgaun (2009) showed that the LKH-2 algorithm performs significantly worse on sharply clustered instances than on uniform random instances. To remedy this difficulty, Helsgaun (2014) considered the unusual structure of clustered instances, and adjusted the parameter settings of LKH-2 to better solve the clustered instances. The resulting solver is named CLKH, which is used in this study.

2.2.3 Edge assembly crossover based genetic algorithm

Population-based evolutionary algorithms are another well-known approach for the TSP. A popular example is the powerful genetic algorithm introduced by Nagata and Kobayashi (2013). This algorithm (called GA-EAX, see Algorithm 1) is characterized by its powerful edge assembly crossover (EAX) operator introduced in Nagata and Kobayashi (1997); Nagata and Soler (2012) with an efficient implementation and a cost-effective selection strategy for maintaining population diversity.

Algorithm 1 GA-EAX for the CTSP

\textbf{Require:} TSP instance \( G \), population size \( p \), number of offspring solutions \( r \) generated from each parent pair

\textbf{Ensure:} best solution \( S^* \)

\begin{verbatim}
1: \( POP = \{P_1, P_2, \ldots, P_p\} \leftarrow \text{Initial\_Population}(G) \\
2: \text{while stopping condition is not met do} \\
3: \quad \text{Randomly shuffle the solutions in POP} \\
4: \quad \text{for } i = 1, 2, \ldots, p \text{ do} \\
5: \quad \quad S_1 \leftarrow P_i, S_2 \leftarrow P_i+1 \text{ / Note: } P_{p+1} = P_1 \text{ /} \\
6: \quad \quad (o_1, \ldots, o_r) \leftarrow \text{EAX}(S_1, S_2) \\
7: \quad \quad P_i \leftarrow \text{Select\_Best}(o_1, \ldots, o_r, S_i) \\
8: \quad \text{end for} \\
9: \text{end while} \\
10: S^* \leftarrow \text{Best}(POP) \\
11: \text{Return } S^*
\end{verbatim}

6/17
The key EAX operator generates, from two high-quality tours (parents), one offspring tour by first inheriting the edges from the parents to construct disjoint subtours and then connecting the subtours with new edges in a greedy fashion (similar to building a minimal spanning tree). Let $S_A$ and $S_B$ be the parents, EAX operates as follows (see Fig. 2 for an example):

1. Generate an undirected multigraph defined as $G_{AB} = (V, E_A \cup E_B)$, where $E_A$ and $E_B$ are the sets of edges of parents $S_A$ and $S_B$, respectively.

2. Extract all AB-cycles from $G_{AB}$. An AB-cycle is defined as a cycle in $G_{AB}$, such that edges of $E_A$ and edges of $E_B$ are alternately linked.

3. Construct an E-set by selecting AB-cycles according to a given selection strategy (e.g., single, k-multiple, block and block2 (Nagata and Kobayashi, 2013)), where an E-set is a set of AB-cycles.

4. Copy parent $S_A$ to an intermediate solution $o$. Then, remove the edges of $E_A$ in the E-set from $o$ and add those of $E_B$ in the E-set to $o$. This leads to an intermediate solution $o$ with one or more subtours.

5. Connect all the subtours in $o$ with new short edges to generate a complete tour (a feasible offspring solution) by using a greedy heuristic.

Note that different versions of EAX can be developed by using different selection strategies of AB-cycles for constructing E-sets. The GA-EAX algorithm employs the single and block2 strategies to generate offspring solutions from parent solutions. To maintain a healthy population diversity, GA-EAX also uses an edge entropy measure to select the solution to be used to replace a parent in the population.

Other studies (e.g., Hains et al. (2012)) also indicated the usefulness of edge-assembly-like crossovers for solving clustered instances of the TSP. As shown in the next section, the EAX-based genetic algorithm performs remarkably well on all the clustered instances transformed from the CTSP.

3 COMPUTATIONAL EXPERIMENTS
In this section, we evaluate the capacity of the TSP solvers presented in Section 2.2 to solve the CTSP via its transformation to the TSP. For this purpose, we examine their qualitative performances and run time efficiencies on various benchmark instances and make comparisons with the best dedicated CTSP algorithms in the literature.
3.1 Benchmark instances

Our computational assessments are based on three sets of 73 benchmark instances with 101 to 24,978 vertices. Sets 1 and 2 include 20 medium instances (101 ≤ |V| ≤ 1000) and 15 large instances (1173 ≤ |V| ≤ 2000), which are classical and widely used in the CTSP literature (e.g., Mestria et al. (2013); Mestria (2016, 2018)). Set 3 includes 38 large GTSP instances (1000 ≤ |V| ≤ 24,978) from Helsgaun (2014).

Sets 1 and 2 (35 instances): These instances belong to the following six types: (1) instances taken from the TSPLIB (Reinelt, 1991) where the clusters are generated by using a k-means clustering algorithm; (2) instances created from a selection of classic TSP instances (Johnson and McGeoch, 2007), where the clusters are created by grouping the vertices in geometric centers; (3) instances generated by using the Concorde interface (Applegate et al., 2006a); (4) instances generated using the layout proposed in Laporte and Palekar (2002); (5) instances similar to type 2, but generated with different parameters; (6) instances adapted from the TSPLIB (Reinelt, 1991), where the rectangular floor plan is divided into several quadrilaterals and each quadrilateral corresponds to a cluster. These instances are available at http://www2.ic.uff.br/~labic/conteudo/instance/.

Set 3 (38 instances): These large instances have 1000 to 24,978 vertices and come from GTSPLIB for the Generalised Travelling Salesman Problem (GTSP). They were generated from TSP instances by using Fischetti et al.'s clustering algorithm (Fischetti et al., 1997) and tested in Helsgaun (2014) by considering them as CTSP instances. These instances are available at http://www.ruc.dk/~keld/research/CLKH. In Helsgaun (2014), six very large instances with 31,623 to 85,900 vertices were also tested. We ignore these instances, because they are too large for the exact Concorde solver and the GA-EAX solver stops abnormally when solving these instances.

3.2 TSP solvers and experimental protocol

For our study, we employed three representative TSP solvers presented in Section 2.2, which are among the most powerful methods for the TSP in the literature.

- Exact Concorde TSP solver\(^1\): We used version Concorde-03.12.19 and ran the solver with its default parameter setting with a cutoff time of 24 CPU hours per instance.

- Inexact CLKH solver\(^2\): We used the version CLKH-1.0 which is based on the latest version 2.0.9\(^3\) of LKH-2. The default parameter setting given in Helsgaun (2014) was adopted to run CLKH. Notice that to reduce its run time, the maximum number of trials (iterations) is set to 1000 in CLKH, while this number is set to \(n\) (instance size) by default in LKH-2.

- Inexact GA-EAX TSP solver\(^4\): We used GA-EAX with its default parameter setting given in Nagata and Kobayashi (2013): \(p = 300, r = 30\) and GA-EAX terminates if the difference between the average tour length and the shortest tour length in the population is less than 0.001. Following Kerschke et al. (2018); Kotthoff et al. (2015), we reset the random seed for GA-EAX for each run (which was set to a fixed value in the official implementation).

The experiments were carried out on a computer running Linux operating system with an Intel E5-2670 processor (2.8 GHz and 4G RAM). Given the stochastic nature of CLKH and GA-EAX, we ran each algorithm 10 times for each instance while the deterministic Concorde TSP solver was run one time to solve each instance.

3.3 Computational results and comparison of popular TSP solvers

Our computational studies aim to answer the following questions: How do state-of-the-art exact TSP solvers perform on clustered instances converted from the CTSP? How do state-of-the-art inexact (heuristic) TSP solvers perform on clustered instances converted from the CTSP?

The results of the three TSP solvers (Concorde, CLKH, GA-EAX) on the 20 medium and 15 large CTSP benchmark instances are summarized in Table 1 (Set 1) and Table 2 (Set 2). Columns 1 to 3 show the basic information of each instance: the instance name (Instance), the number of vertices (|V|) and

\(^1\)http://www.math.uwaterloo.ca/tsp/concorde/index.html
\(^2\)http://www.ruc.dk/~keld/research/CLKH
\(^3\)http://akira.ruc.dk/~keld/research/LKH/
\(^4\)https://github.com/sugia/GA-for-TSP
the number of clusters (m). Column 4 gives the optimal objective value reported by the exact Concorde TSP solver, followed by the required run time in seconds. For both the CLKH and GA-EAX solvers, we show the best (Gap_best) and average (Gap_avg) results over 10 independent runs in the form of the percentage gap to the optimal solution, as well as the average run time in seconds. If the best solution over 10 independent runs equals the optimal solution obtained with the exact Concorde TSP solver, the corresponding cell in column Ga_best Shows ‘=’ along with the number of runs that succeeded in finding the optimal solution. Finally, row ‘Avg.’ provides the average run time in seconds for each approach, and the average gap between the average objective values obtained with CLKH/GA-EAX and the optimal values obtained with the Concorde TSP solver.

Table 1. Computational results of the TSP solvers Concorde, CLKH and GA-EAX on medium CTSP instances (Set 1).

| Instance | | m | Min | Max | Opt. | Opt. | Opt. | Opt. |
|----------|-----------------|---|-----|-----|------|------|------|------|
| 10-lin158  | 10  | 6  | 32984 | 2.2  | 19.5  | 0.0000  | 15.8  | 1.7 |
| 10-pcb442  | 442  | 10  | 37342 | 20.7  | 46.9  | 0.0000  | 19.6  | 6.3 |
| CLKH  | 1000  | 10  | 1325027 | 21.9  | 128.6  | 0.0001  | 112.6  | 16.3 |
| 1500-50-505  | 1500  | 50  | 12928125 | 22.3  | 70.6  | 0.0000  | 112.6  | 5.3 |
| 2000-10-x2  | 2000  | 10  | 33509 | 170.9  | 160.1  | 0.0000  | 29.9  | 37.3 |
| 2000-10-a  | 2000  | 10  | 105360 | 7214.3  | 0.0038  | 0.0155  | 401.9  | 0.0826  | 0.1167  | 45.3 |
| Avg  | 1133.8  | 0.0793  | 257.3  | 0.0131  | 33.6 |

Table 2. Computational results of the TSP solvers Concorde, CLKH and GA-EAX on large CTSP instances (Set 2).

| Instance | | m | Min | Max | Opt. | Opt. | Opt. | Opt. |
|----------|-----------------|---|-----|-----|------|------|------|------|
| 10-lin158  | 10  | 6  | 32984 | 2.2  | 19.5  | 0.0000  | 15.8  | 1.7 |
| 10-pcb442  | 442  | 10  | 37342 | 20.7  | 46.9  | 0.0000  | 19.6  | 6.3 |
| CLKH  | 1000  | 10  | 1325027 | 21.9  | 128.6  | 0.0001  | 112.6  | 16.3 |
| 1500-50-505  | 1500  | 50  | 12928125 | 22.3  | 70.6  | 0.0000  | 112.6  | 5.3 |
| 2000-10-x2  | 2000  | 10  | 33509 | 170.9  | 160.1  | 0.0000  | 29.9  | 37.3 |
| 2000-10-a  | 2000  | 10  | 105360 | 7214.3  | 0.0038  | 0.0155  | 401.9  | 0.0826  | 0.1167  | 45.3 |
| Avg  | 1133.8  | 0.0793  | 257.3  | 0.0131  | 33.6 |

From Tables 1-2, we can make the following observations:

First, the exact Concorde TSP solver performs very well on these 35 instances and is able to solve all of them exactly. Specifically, the 20 medium instances can be solved easily in a short run time (an average of about 14 seconds). The 15 large instances are more difficult and the run time needed to solve these instances increases considerably (an average of 1133.8 seconds, reaching 7214.3 seconds for the most difficult instance).

Second, the CLKH solver performs globally very well on these 35 instances. For the 20 medium instances, CLKH attains all the optimal solutions but one with an average run time of 47.8 seconds. For the 15 large instances, CLKH reaches the optimal solutions for 13 instances with an average run time of 257.3 seconds.

Third, the GA-EAX solver performs remarkably well by attaining the optimal values for all 35 instances but one. For the 20 medium instances, GA-EAX consistently hits the optimal solutions for each of its 10 run (except for one instance for which it has a hit of 9 out of 10). The average run time is only 5.7 seconds for the medium instances and 33.6 seconds for the large instances. Compared to Concorde and CLKH, GA-EAX is thus extremely time efficient. Moreover, contrary to the Concorde and CLKH solvers, the computation time required by GA-EAX remains very stable across the instances of the same
For a large value $\tau$ the algorithm $s$ can achieve many times the performance of the best algorithm $r$ that the Concorde solver failed to exactly solve three instances in 24 hours, the corresponding cell (in parentheses) in column ‘Optimum’ indicates the best tour length (best upper bound) found by CLHK and GA-EAX. In this case, the percentage gaps ($Gap_{\text{best}}$ and $Gap_{\text{avg}}$) are calculated by using the best bound, and column $Gap_{\text{best}}$ shows ‘=’ the number of runs for an algorithm to find the best bound.

From Table 3, we can make the following observations. First, Concorde manages to optimally solve 21 large GTSP instances with up to 3162 vertices with a run time ranging from 17.4 seconds to 45008.4 seconds while its solution time is not completely consistent with the size of the problem instances. For the 21 instances that can be solved exactly by Concorde, CLKH attains 15 best upper bounds, while GA-EAX reaches all best upper bounds in less computing time. Second, for most of the instances with $|V| < 10,000$, compared with CLHK, GA-EAX has a better performance both in terms of solution quality and computation time. For the instances with 10,000 $\leq |V| \leq 24,978$, the solution quality of GA-EAX is better than that of CLHK in most cases, while requiring more computation time.

To sum up, the exact Concorde solver is very efficient for the instances with up to 1000 vertices (order of seconds) and can even find optimal solutions for instances with up to some 3000 vertices at a price of more run time (order of minutes to hours). For larger instances, both inexact solvers (CLHK and GA-EAX) are reliable alternatives to find optimal or sub-optimal solutions with some advantages for GA-EAX. These heuristic solvers also perform very well on smaller instances.

To deepen our computational study, we call upon to the performance profile, an analytic tool for evaluating the performance of multiple compared optimization algorithms (Dolan and Moré, 2002). The performance profile uses a cumulative distribution function for a performance metric, such as run time, objective function values, number of iterations, and so on. Precisely, let $S$ be a set of algorithms and $P$ be a set of problem instances. For a given performance metric $f_{p}$ (that is the performance of algorithm $s \in S$ solving instance $p \in P$), the performance ratio is defined by $r_{p} = \frac{f_{p}}{\min\{f_{p} | p \in S\}}$. Then, for each algorithm $s \in S$, the performance function is given by $\rho_{s}(\tau) = \frac{|\{p | f_{p} r_{p} \leq \tau\}|}{|P|}$. Thus, the value of $\rho_{s}(\tau)$ corresponds to the fraction of problem instances that algorithm $s$ can achieve many times the performance of the best algorithm (meaning the probability that the algorithm $s$ will win over the rest of the compared algorithms). For a large value $\tau$, the value of $\rho_{s}(\tau)$ indicates a high robustness of algorithm $s$.

Table 3. Computational results of the TSP solvers Concorde, CLKH and GA-EAX on large GTSP instances (Set 3).
To make a fair and meaningful comparison with this tool, we focus on the two inexact solvers CLKH and GA-EAX and run each solver 10 times on each of the 73 instances. We use the software 'perprof-py' (Siqueira et al., 2016) to draw the performance profiles (see Figure 3) where the quality of the solution is measured by the average objective value and average run time. These performance profiles tend to show an advantage of GA-EAX over CLKH for solving these clustered instances with up to 24,978 vertices.

3.4 TSP solvers v.s. state-of-the-art CTSP heuristics

Table 4. List of the reference algorithms for the CTSP

| Algorithm name | Reference | Search strategy                        |
|----------------|-----------|----------------------------------------|
| VNRDGILS      | Mestria (2018) | A hybrid heuristic based on GRASP, ILS and VNRD |
| HHGILS        | Mestria (2016) | A hybrid heuristic based on GRASP, ILS and VND |
| GPR1R2        | Mestria et al. (2013) | A GRASP with Path Relinking PR1 and PR2 |
| GPR1          | Mestria et al. (2013) | A GRASP with Path Relinking PR1          |
| GPR2          | Mestria et al. (2013) | A GRASP with Path Relinking PR2          |
| GPR3          | Mestria et al. (2013) | A GRASP with Path Relinking PR3          |
| GPR4          | Mestria et al. (2013) | A GRASP with Path Relinking PR4          |
| GRASP         | Mestria et al. (2013) | A traditional GRASP heuristic          |
| TLGA          | Ding et al. (2007) | A two-level genetic algorithm            |

In Section 3.3, we observed that the exact Concord TSP solver and the inexact CLKH and GA-EAX TSP solvers are powerful tools for solving clustered TSP instances converted from the CTSP. We now answer the following question: Do these general TSP solvers compete well with state-of-the-art CTSP heuristics specially designed for the problem?

For this purpose, we adopt GA-EAX as our representative TSP solver and compare it with three best performing CTSP heuristics in the literature: VNRDGILS (Mestria, 2018), HHGILS (Mestria, 2016), and GPR1R2 (Mestria et al., 2013). Indeed, according to the experimental studies reported in Mestria et al. (2013); Mestria (2016, 2018), these three heuristics perform the best among the recent CTSP heuristics available in the literature (see Table 4). This study is based on the 35 medium and large instances of Sets 1 and 2 (no results for the three CTSP heuristics are available on the large GTSP instances of Set 3).

Table 5 provides the comparative results of the GA-EAX TSP solver along with the results reported by the three CTSP algorithms on the medium and large instances. For each instance and algorithm, columns \( f_{\text{best}} \), \( f_{\text{avg}} \) and \( t(s) \) show respectively the best objective value over 10 independent runs, the average objective value and the average run time in seconds. Furthermore, the row ‘Avg.’ shows the average performances for each compared algorithm, including the average percentage gap of the best/average result to the optimal result obtained with the Concorde TSP solver and the average run time in seconds. To determine whether there exists a statistically significant difference in performance between the GA-EAX TSP solver and each CTSP algorithm in terms of best and average results, the \( p \)-values from the Wilcoxon
Table 5. Comparative results between the GA-EAX TSP solver and three CTSP algorithms on medium and large CTSP instances.

| Instance | (F) | m | GA-EAX | VNRDGILS | HHGILS | GPR1R2 |
|----------|-----|---|--------|----------|--------|--------|
| 10-nl-4 | 10  | 310 | 13543 | 135483  | 135510 | 135578 |
| 10-nl-5 | 10  | 529  | 529584| 530604  | 530871 | 530817 |
| 10-nl-6 | 10  | 1050 | 105044| 105061  | 105094 | 105033 |
| 10-nl-7 | 10  | 1400 | 140040| 140061  | 140094 | 140033 |
| 10-nl-8 | 10  | 1750 | 175040| 175061  | 175094 | 175033 |
| 10-nl-9 | 10  | 2100 | 210040| 210061  | 210094 | 210033 |
| 10-nl-10| 10  | 2500 | 250040| 250061  | 250094 | 250033 |
| 10-nl-11| 10  | 3000 | 300040| 300061  | 300094 | 300033 |
| 10-nl-12| 10  | 3500 | 350040| 350061  | 350094 | 350033 |
| 10-nl-13| 10  | 4000 | 400040| 400061  | 400094 | 400033 |
| 10-nl-14| 10  | 4500 | 450040| 450061  | 450094 | 450033 |
| 10-nl-15| 10  | 5000 | 500040| 500061  | 500094 | 500033 |
| 10-nl-16| 10  | 5500 | 550040| 550061  | 550094 | 550033 |
| 10-nl-17| 10  | 6000 | 600040| 600061  | 600094 | 600033 |
| 10-nl-18| 10  | 6500 | 650040| 650061  | 650094 | 650033 |
| 10-nl-19| 10  | 7000 | 700040| 700061  | 700094 | 700033 |
| 10-nl-20| 10  | 7500 | 750040| 750061  | 750094 | 750033 |
| 10-nl-21| 10  | 8000 | 800040| 800061  | 800094 | 800033 |
| 10-nl-22| 10  | 8500 | 850040| 850061  | 850094 | 850033 |
| 10-nl-23| 10  | 9000 | 900040| 900061  | 900094 | 900033 |
| 10-nl-24| 10  | 9500 | 950040| 950061  | 950094 | 950033 |
| 10-nl-25| 10  | 10000| 1000040| 1000061  | 1000094 | 1000033 |
signed-rank tests are given in the last row of Table 5. Entries with "-" mean that the corresponding results are not available in the literature. The best objective values obtained by the compared algorithms are indicated in bold if they attain the optimal solution. Notice that the results of the CTSP algorithms (VNRDGILS, HHGILS and GPR1R2) correspond to 10 executions per instance on a computer with 2.83 GHz Intel Core 2 CPU and 8 GB RAM and the time limit per run was set to 720 seconds for medium instances and 1080 seconds for large instances.

Table 6. Statistical results for the GA-EAX TSP solver and three state-of-the-art CTSP algorithms on Set 1 (medium instances) and Set 2 (large instances). Dominating values are indicated in bold.

|               | GA-EAX | VNRDGILS | HHGILS | GPR1R2 |
|---------------|--------|----------|--------|--------|
| **Set 1**     |        |          |        |        |
| Optimal solutions | 20/20  | 0/20     | 1/20   | 0/20   |
| Average $\text{Gap}_{\text{best}}/\text{Gap}_{\text{avg}}$ (%) | 0.00/0.00 | 0.18/0.30 | 0.21/0.40 | 0.39/0.73 |
| Average time (s) | 5.7    | 720.0    | 720.0  | 720.0  |
| **Set 2**     |        |          |        |        |
| Optimal solutions | 14/15  | 0/15     | 0/15   | 0/15   |
| Average $\text{Gap}_{\text{best}}/\text{Gap}_{\text{avg}}$ (%) | 0.00/0.01 | 8.61/11.39 | 8.92/11.04 | 12.25/15.51 |
| Average time (s) | 33.6   | 1080.0   | 1080.0 | 1080.0 |

Table 6 summarizes the statistical results for each compared algorithm on the two sets of medium and large instances. The first row indicates the number of optimal solutions found by each approach. The average percentage gap of the best/average result from the optimal result is provided in row ‘Average $\text{Gap}_{\text{best}}/\text{Gap}_{\text{avg}}$’. Finally, row ‘Average time (s)’ provides the average run time in seconds for each algorithm.

From Tables 5 and 6, we observe that the GA-EAX solver significantly outperforms the three CTSP algorithms on the medium and large instances in terms of both the best and the average results. For the large instance set, the improvement gaps between the results of GA-EAX and those of the CTSP methods are very high, ranging from 10.39% to 15.49%. Furthermore, in terms of the average run time, GA-EAX is about 30 to 130 times faster than the CTSP algorithms. The above results thus indicate that the GA-EAX TSP solver has a strong dominance over current best performing CTSP approaches in the literature. In addition, the small $p$-values (<0.05) from the Wilcoxon signed-rank tests further confirm the statistically significant difference of the compared results.

Figure 4. Boxplots of the normalized average objective values for the medium instance set and large instance set.

To have a finer analysis of the compared algorithms, Figure 4 provides boxplot graphs to compare the distribution and range of the average results for each compared algorithm, except GPR1R2 for the medium
instances since its results on several medium instances are not available. In this figure, the average objective value $f_{avg}$ of a given algorithm is normalized according to the relation $y = 100 \times \frac{(f_{avg} - f_{opt})}{f_{opt}}$, where $f_{opt}$ is the optimal value. The plots in Figure 4 show clear differences in the distributions of the average results between GA-EAX and each compared CTSP heuristic, which further confirms the efficiency of the GA-EAX TSP solver with respect to these dedicated CTSP heuristics.

Finally, considering the results of the Concorde solver and the CLKH solver reported in Section 3.3, we conclude that these TSP solvers also dominate the current best CTSP algorithms in the literature.

4 DISCUSSION

We now provide additional explanations regarding the behaviors of the three TSP solvers. First, given the NP-hard nature of the CTSP and the exponential time complexity of the exact Concorde solver, it is expected that the exact Concorde solver reaches its limit when the instance to be solved reaches some size (about 3000 vertices for the studies instances). Indeed, when the search space becomes extremely large, the exact Branch-and-Bound search even equipped with the best problem specific cutting plane methods cannot effectively enumerate all candidate solutions. In fact, such a behavior has already been observed in previous studies on Concorde applied to classical TSP instances (Applegate et al., 2006b; Hoos and Stützle, 2014). Second, regarding the two heuristic solvers CLKH and GA-EAX, the CLKH solver exhibits a worse performance compared to GA-EAX. As discussed in Section 2.2.2, the underlying LK heuristic stumbles on clustered instances because relatively large intercluster edges serve as bait edges. With the presence of these bait edges, the LK heuristic may be tricked into long and often fruitless search trajectories. Third, the GA-EAX solver performs its search mainly with its edge assembly crossover, which inherits the edges of the parents to construct disjoint subtours and then connect the subtours. This crossover proves to be meaningful and helps the algorithm avoid local optimal traps. Once again, the excellent behavior of GA-EAX on the CTSP instances is consistent with its performance on conventional TSP instances as shown in Nagata and Kobayashi (2013).

5 CONCLUSION

This work presents the first extensive computational study on the transformation approach of solving the Clustered Traveling Salesman Problem with general TSP solvers. Based on the results from the exact Concorde solver and the heuristic CLKH and GA-EAX solvers on 20 medium ($101 \leq |V| \leq 1000$) and 15 large ($1173 \leq |V| \leq 2000$) CTSP benchmark instances and 38 large GTSP benchmark instances (with up to 24,978 vertices) available in the literature, we can draw the following conclusions.

- The exact Concorde solver can optimally solve all medium and large CTSP instances. It also solves exactly large GTSP instances with up to 3162 vertices in a reasonable time, but fails to solve larger GTSP instances in 24 hours. Its solution time is not completely consistent with the size of the problem instances.

- The heuristic CLKH and GA-EAX solvers perform very well both in terms of solution quality and computational efficiency. Both solvers have a good scalability, making them particularly suitable for solving very large instances with at least several thousands of vertices. For the tested instances with up to some 24,978 vertices, GA-EAX exhibits a better performance than CLKH.

- The general TSP solvers significantly dominate, both in terms of solution quality and computational efficiency, the current best performing CTSP heuristics specially designed for the problem. In particular, the TSP heuristics are several orders of faster than the state-of-the-art CTSP heuristics to find much better results.

This study indicates that the existing CTSP benchmark instances in the literature are not challenging for modern TSP solvers even if they remain difficult for the existing CTSP algorithms.

Finally, given the findings of this study, it would be interesting to investigate the problem transformation approach for solving other TSP variants that can be converted to the TSP or to a TSP variant for which effective algorithms are available.
ACKNOWLEDGMENTS

We are grateful to the Associated Editor and anonymous reviewers for their valuable suggestions and comments, which helped us to improve the paper. This work was partially supported by the National Natural Science Foundation Program of China [Grant No. 72122006].

REFERENCES

Anily, S., Bramel, J., and Hertz, A. (1999). A 53-approximation algorithm for the clustered traveling salesman tour and path problems. *Operations Research Letters*, 24(1-2):29–35.

Applegate, D., Bixby, R., and Chvatal, V. (2006a). Concorde tsp solver [http://www.math.uwaterloo.ca/tsp/concorde/index.html].

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006b). The traveling salesman problem: a computational study. *Princeton University Press*.

Applegate, D., Cook, W., and Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92.

Bao, X. and Liu, Z. (2012). An improved approximation algorithm for the clustered traveling salesman problem. *Computers & Operations Research*, 39(12):3866–3870.

Bao, X., Liu, Z., Yu, W., and Li, G. (2017). A note on approximation algorithms of the clustered traveling salesman problem. *Information Processing Letters*, 127(54–57).

Campuzano, G., Obreque, C., and Aguayo, M. M. (2020). Accelerating the miller-tucker-zemlin model for the asymmetric traveling salesman problem. *Expert Systems with Applications*, 148(1):113229.

Chisman, J. A. (1975). The clustered traveling salesman problem. *Computers & Operations Research*, 2(2):115–119.

Claus, A. (1984). A new formulation for the travelling salesman problem. *SIAM Journal on Algebraic Discrete Methods*, 5(1):21–25.

Cosma, O., Pop, P. C., and Cosma, L. (2021). An effective hybrid genetic algorithm for solving the generalized traveling salesman problem. *Lecture Notes in Computer Science*, 12886:113–123.

Ding, C., Cheng, Y., and He, M. (2007). Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale tsp. *Tsinghua Science and Technology*, 12(4):459–465.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

Dubois-Lacoste, J., Hoos, H. H., and Stützle, T. (2015). On the empirical scaling behaviour of state-of-the-art local search algorithms for the euclidean tsp. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (pp. 377-384).

Fischetti, M., Salazar González, J. J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394.

Gendreau, M., Laporte, G., and Hertz, A. (1997). An approximation algorithm for the traveling salesman problem with backhauls. *Operations Research*, 45(4):639–641.

Ghaziri, H. and Osman, I. H. (2003). A neural network algorithm for the traveling salesman problem with backhauls. *Computers & Industrial Engineering*, 44(2):267–281.

Guttmann-Beck, N., Hassin, R., Khuller, S., and Raghavachari, B. (2000). Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437.

Hä, M. H., Nguyen Phuong, H., Tran Ngoc Nhat, H., Langevin, A., and Trépanier, M. (2022). Solving the clustered traveling salesman problem with-relaxed priority rule. *International Transactions in Operational Research*, 29(2):837–853.

Hains, D., Whitley, D., and Howe, A. (2012). Improving lin-kernighan-helsgaun with crossover on clustered instances of the tsp. In *International Conference on Parallel Problem Solving from Nature* (pp. 388-397). *Springer, Berlin, Heidelberg*.

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

Helsgaun, K. (2009). General k-opt submoves for the lin-kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Helsgaun, K. (2014). Solving the clustered traveling salesman problem using the lin-kernighan-helsgaun algorithm. *Computer Science Research Report*, (142)(142):1–45.
508 Hoos, H. H. and Stützle, T. (2014). On the empirical scaling of run-time for finding optimal solutions to
509 the travelling salesman problem. European Journal of Operational Research, 238(1):87–94.
510 Jin, Y. and Hao, J.-K. (2019). Solving the latin square completion problem by memetic graph coloring.
511 IEEE Transactions on Evolutionary Computation, 23(6):1015–1028.
512 Johnson, D. S. and McGeoch, L. A. (2007). Experimental analysis of heuristics for the tsp. In The
513 Traveling Salesman Problem and its Variations (pp. 369-443). Springer, Boston, HEA.
514 Jongens, K. and Volgenant, T. (1985). The symmetric clustered traveling salesman problem. European
515 Journal of Operational Research, 19(1):68–75.
516 Kawasaki, M. and Takazawa, K. (2020). Improving approximation ratios for the clustered traveling
517 salesman problem. Journal of the Operations Research Society of Japan, 63(2):60–70.
518 Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. (2018). Leveraging tsp solver
519 complementarity through machine learning. Evolutionary computation, 26(4):597–620.
520 Kotthoff, L., Kerschke, P., Hoos, H., and Trautmann, H. (2015). Improving the state of the art in inexact
521 tsp solving using per-instance algorithm selection. In International Conference on Learning and
522 Intelligent Optimization (pp. 202-217). Springer, Cham.
523 Laporte, G. and Palekar, U. (2002). Some applications of the clustered travelling salesman problem.
524 Journal of the Operational Research Society, 53(9):972–976.
525 Laporte, G., Potvin, J. Y., and Quilleret, F. (1997). A tabu search heuristic using genetic diversification
526 for the clustered traveling salesman problem. Journal of Heuristics, 2(3):187–200.
527 Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem.
528 Operations Research, 21(2):498–516.
529 Lokin, F. C. J. (1979). Procedures for travelling salesman problems with additional constraints. European
530 Journal of Operational Research, 3(2):135–141.
531 Martin, O., Otto, S. W., and Felten, E. W. (1991). Large-step markov chains for the traveling salesman
532 problem. Oregon Graduate Institute of Science and Technology, Department of Computer Science and
533 Engineering.
534 Mestria, M. (2016). A hybrid heuristic algorithm for the clustered traveling salesman problem. Pesquisa
535 Operacional, 36(1):113–132.
536 Mestria, M. (2018). New hybrid heuristic algorithm for the clustered traveling salesman problem.
537 Computers & Industrial Engineering, 116(1):1–12.
538 Mestria, M., Ochi, L. S., and de Lima Martins, S. (2013). Grasp with path relinking for the symmetric
539 euclidean clustered traveling salesman problem. Computers & Operations Research, 40(12):3218–3229.
540 Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling
541 salesmen problems. Journal of ACM, 7(4):326–329.
542 Morán-Mirabal, L. F., Velarde, J. L. G., and Resende, M. G. C. (2014). Randomized heuristics for the
543 family traveling salesperson problem. International Transactions in Operational Research, 21(1):41–
544 57.
545 Nagata, Y. and Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for
546 the travelling salesman problem. 7th International Conference on Genetic Algorithms (pp. 450-457).
547 Morgan Kaufmann, San Francisco.
548 Nagata, Y. and Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for
549 the traveling salesman problem. INFORMS Journal on Computing, 25(2):346-363.
550 Nagata, Y. and Soler, D. (2012). A new genetic algorithm for the asymmetric traveling salesman problem.
551 Expert Systems with Applications, 39(10):8947–8953.
552 Neto, D. (1999). Efficient cluster compensation for lin-kernighan heuristics. PhD thesis, University of
553 Toronto.
554 Pop, P., Matei, O., and Pintea, C. (2018). A two-level diploid genetic based algorithm for solving the
555 family traveling salesman problem. In Proceedings of the Genetic and Evolutionary Computation
556 Conference (pp. 340–346).
557 Potvin, J. Y. and Guertin, F. (1996). The clustered traveling salesman problem: A genetic approach. In
558 Meta-Heuristics (pp. 619-631). Springer, Boston.
559 Reinelt, G. (1991). Tsp-lib-a traveling salesman problem library. ORSA Journal on Computing, 3(4):376–
559 384.
560 Siqueira, A. S., da Silva, R. C., and Santos, L. R. (2016). Perprof-py: A python package for performance
561 profile of mathematical optimization software. Journal of Open Research Software, 4(1).
Srivastava, S. S., Kumar, S., Garg, R. C., and Sen, P. (1969). Generalized traveling salesman problem through n sets of nodes. *CORS Journal*, 7:97–101.

Weintraub, A., Aboud, J., Fernandez, C., Laporte, G., and Ramirez, E. (1999). An emergency vehicle dispatching system for an electric utility in chile. *Journal of the Operational Research Society*, 50(7):690–696.

Wong, R. (1980). Integer programming formulations of the travelling salesman problem. *Proceedings of the IEEE International Conference of Circuits and Computers* (pp. 149-152).

Wu, Q. and Hao, J.-K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1):355–365.

Zenklusen, R. (2019). A 1.5-approximation for path tsp. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1539-1549).