New Exact Methods for Solving Quadratic Traveling Salesman Problem

Yuxiao Chen[1], **Anubhav Singh**[1], **Ryo Kuroiwa**[2], **J. Christopher Beck**[1]

[1]University of Toronto, Canada
[2]National Institute of Informatics / The Graduate University for Advanced Studies, SOKENDAI, Tokyo, Japan
yuxiao.chen@mail.utoronto.ca, anubhav.singh@utoronto.ca, kuroiwa@nii.ac.jp, jcb@mie.utoronto.ca

Abstract

The Quadratic Traveling Salesman Problem (QTSP) is a variant of the Traveling Salesman Problem (TSP) in which the objective function depends on pairs of consecutive edges in the tour; hence, it is quadratic and generally hard to optimize. While various exact-solving approaches have been explored, many rely on specialized procedures and struggle to scale to large instances. Carefully crafted metaheuristics have demonstrated better primal bounds and scalability than the exact approaches, but, of course, cannot provide guarantees of solution quality nor prove optimality. In this work, we define encodings of QTSP in domain-independent dynamic programming (DIDP), constraint programming (CP), mixed integer quadratic programming (MIQP), and mixed integer linear programming (MILP), and compare them with the best-known exact method, a branch and cut (B&C) algorithm, and the state-of-the-art metaheuristic, a hybrid genetic algorithm (HGA). Our experimental results demonstrate that a DIDP model finds the best feasible solutions and the smallest optimality gaps on average among all exact solvers, including the B&C algorithm, for sufficiently large problems. HGA finds the best feasible solution among all approaches, with DIDP within 15% of the HGA cost on all experimental instances. Interestingly, our MILP model with the subtour elimination constraints generally finds better feasible solutions than the B&C algorithm while matching it in proving optimality, suggesting that lazily adding sub-tour elimination cuts is not particularly helpful in QTSP.

Introduction

The quadratic traveling salesperson problem (QTSP) is an extension of Traveling Salesman Problem (TSP), which seeks an optimal tour that visits all customers exactly once and returns to the starting location. However, unlike TSP, the cost of moving between two locations is not solely dependent on the pair of locations. Instead, the tour cost is quadratic, determined by pairs of consecutive edges or, equivalently, triples of consecutive customers.

Given a set of customers $N = \{0, \dots, n-1\}$, with $\sigma(i)$ being the i-th customer visited, a tour is represented as $\langle \sigma(0), \dots, \sigma(n-1) \rangle$. Assuming $\sigma(-1) = \sigma(n-1)$ and $\sigma(n) = \sigma(0)$, the cost of the tour is defined as

$$\sum_{i=0}^{n-1} c_{\sigma(i-1)\sigma(i)\sigma(i+1)}. \tag{1}$$

QTSP was originally formulated based on its applications in robotics. Aggarwal et al. (2000) introduced the angular-metric traveling salesman problem (Angle-TSP), where the tour cost is the sum of the absolute angles of rotation at each turning point, motivated by the need for smooth tours in nonholonomic robots. The authors proved that Angle-TSP is NP-hard and proposed a polynomial algorithm that approximates the optimal solution within a ratio of $O(\log n)$. Medeiros and Urrutia (2010) extended Angle-TSP to have a cost depending on a linear combination of the angle change and the travel distance to approximate another related problem from robotics, the Dubins Traveling Salesman Problem (DTSP) (Savla, Frazzoli, and Bullo 2008). The DTSP was, itself, originally approximated using the solution to the Euclidean TSP (ETSP). Medeiros and Urrutia proposed the AngleDistance-TSP to obtain a potentially more accurate approximation of DTSP, highlighting the dependence of Dubins' path length on both distances and initial and final orientations. QTSP is a slight generalization of Angle-TSP and AngleDistance-TSP with the tour cost independent of geometric constraints. A bioinformatics application of QTSP for finding realistic binding site models that better explain the transcription initiation process was introduced by Jäger and Molitor (2008).

Various specialized algorithms have been proposed to tackle the QTSP, including heuristic algorithms (Fischer et al. 2014; Pham et al. 2023; Staněk et al. 2019), branch-and-bound algorithms (Fischer et al. 2014, 2015; Jäger and Molitor 2008), and branch-and-cut algorithms (Fischer et al. 2014, 2015; Jäger and Molitor 2008; Oswin et al. 2017). However, none of these works investigate the performance of general-purpose solvers for mathematical optimization programs. While branch-and-cut algorithms are based on integer linear programming (ILP) models, they require the implementation of QTSP-specific separation algorithms that lazily add constraints to the models. In this paper, we develop, implement, and empirically evaluate a number of optimization models of QTSP in four problem paradigms.

Our contributions are *new optimization models* and an *extensive empirical study*. We propose novel encodings of

the QTSP as domain-independent dynamic programming (DIDP), a new model-based paradigm for solving dynamic programming (DP) problems (Kuroiwa and Beck 2023a, 2024), and constraint programming (CP). Additionally, we introduce mixed-integer quadratic programming (MIQP) and mixed-integer linear programming (MILP) models inspired by the existing models of TSP (Desrochers and Laporte 1991). We empirically compare our models against the best known exact approach and metaheuristic in the literature: a branch and cut (B&C) algorithm (Oswin et al. 2017), and a hybrid genetic algorithm (HGA) (Pham et al. 2023) and show that our DIDP model scales the best among all exact solvers as the size of the instance increases. Furthermore, our MILP model based on subtour elimination constraints introduced by Desrochers and Laporte (1991) performs equally well in proving optimality as the B&C approach with lazily added subtour elimination cuts and finds better feasible solutions for most of the instances. These observations indicate the importance of studying the performance of applying off-the-shelf solvers on mathematical models without specialized sub-routines.

Related Work

The first exact algorithms for QTSP were proposed by Jäger and Molitor (2008): a branch-and-bound (B&B) algorithm that visits all tours in the worst case and an integer programming (IP) based algorithm. They observed that B&B proves optimality faster than the IP on small QTSP instances but that the trend reverses with increasing size, likely owing to the exponential growth in the number of tours that B&B must visit in the worst case. Fischer et al. (2014) introduced two new exact approaches, a QTSP-to-Symmetric-TSP transformation solved by the Concorde TSP solver (Applegate et al. 2003) and a B&C approach based on an MILP model. Two key differences between the Fischer et al.'s B&C approach and the IP-based approach of Jäger and Molitor are the variables in the models and the separation problem used to add valid inequalities. Jäger and Molitor's approach only uses boolean variables representing two consecutive edges in the tour and solves the separation problem on integer solutions. In contrast, Fischer et al. uses boolean variables to represent individual edges and consecutive edge-pairs and solves a separation problem on fractional solutions.

In their subsequent work, Fisher et al. (2015) presented a DP model for QTSP and solved the quadratic Hamiltonian path problem (QHPP) arising in bioinformatics by transforming it to a QTSP. However, their DP formulation is tailored to QHPP and lacks sufficient state information to compute the QTSP tour cost accurately as it omits some terms in the QTSP cost expression. The authors observed that the B&C approach worked better at proving optimality than the B&B and DP approach.

Most recently in terms of exact approaches, Oswin et al. (2017) studied various formulations of B&C on Angle-TSP and AngleDistance-TSP and observed that, similar to the approach of Jäger and Molitor, the B&C algorithm with the standard subtour elimination constraints and separation procedures acting on integral solutions performed better at proving optimality than the B&C algorithm that added cuts on fractional solutions. This behavior is different from that of B&C on classic TSP (Pferschy and Staněk 2017) where separation on fractional solutions proves more efficacious.

In recent years, the studies of QTSP have focused on solving the problem with heuristic methods. Staněk et al. (2019) introduced metaheuristic algorithms based on the heuristics of previous works, the geometric properties of the optimal QTSP solutions, and linear programming (LP) relaxations. Their best method outperformed all of the previous heuristic algorithms that had been applied to the QTSP problems. Pham et al. (2023) proposed a hybrid genetic algorithm (HGA) inspired by the hybrid genetic search framework introduced by Vidal et al. (2012). The authors showed that the HGA is a state-of-the-art algorithm for the quality of the feasible solutions, improving the best-known primal bounds on many benchmark instances.

Optimization Models

We first develop MIQP and MILP models of QTSP as they naturally build on established TSP models. We then present novel CP and DIDP formulations of QTSP. To our knowledge QTSP has not been previously addressed by either of these latter two frameworks.

Mixed-Integer Quadratic Programming Model

Fischer et al. (2014) developed an MIQP model for the QTSP problem, but due to the exponential number of subtour elimination constraints, the model is solved by a B&C algorithm. Our MIQP model is based on the compact MILP model for TSP proposed by Desrochers and Laporte (1991) with a difference in the objective. Given n customers, we use $O(n)$ decision variables and $O(n^2)$ constraints to introduce the position of each customer in a tour. These positions are consecutive in a solution and, thus, replace the $O(2^n)$ subtour elimination constraints in the Fischer et al. model.

We use binary decision variables x_{ij} to represent visiting customer j immediately after i, and an integer decision variable u_i to denote the position of customer i in the tour. The MIQP model is as follows,

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} x_{ij} x_{jk} \tag{2a}$$

$$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1 \qquad \forall i \in N \tag{2b}$$

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 \qquad \forall i \in N \setminus \{0\}, \forall j \in N \setminus \{0, i\} \tag{2c}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, \forall j \in N \setminus \{i\} \tag{2d}$$

$$1 \leq u_i \leq n-1 \qquad \forall i \in N \setminus \{0\}. \tag{2e}$$

Constraints (2b) ensure that each customer has exactly one incoming and one outgoing edge. Constraints (2c) eliminate subtours by assigning a position to each customer in the solution tour; every customer must have a position index greater by 1 than the index of the previous customer, except for customer 0, who has a position index 0. Thus, the selected edges must form a cycle that visits all customers exactly once. The objective function (2a) includes the cost for

consecutively visiting i, j, k if and only if x_{ij} and x_{jk} are both 1.

Mixed-Integer Linear Programming Model

Our MILP model is based on our MIQP model where we linearize the objective function using additional variables. As above, we use a binary decision variable x_{ij} to represent immediately visiting customer j after customer i, but add another binary decision variable y_{ijk} to indicate the customers i, j, k are visited consecutively. The integer decision variable u_i continues to denote the position of customer i in the tour.

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} y_{ijk} \tag{3a}$$

$$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij} \quad \forall i \in N, \forall j \in N \setminus \{i\} \tag{3b}$$

$$y_{ijk} \in \{0,1\} \quad \forall i \in N, \forall j \in N \setminus \{i\}, \forall k \in N \setminus \{i,j\} \tag{3c}$$

(2b), (2c), (2d), and (2e).

Constraints (2b), (2c), (2d), and (2e) have the same interpretation as the MIQP Model. Constraints (3b) guarantee the y_{ijk} is 1 if and only if both x_{ij} and x_{jk} are 1, which means the customers i, j, k are visited consecutively in the tour.

Constraint Programming Models

We present two CP models of QTSP using different global constraints. Our first CP model uses an integer decision variable x_i to represent the i-th customer visited in a tour. An *all-different* global constraint over these variables eliminates subtours.

$$\min c_{x_{n-1}x_0x_1} + \sum_{i=0}^{n-3} c_{x_i x_{i+1} x_{i+2}} + c_{x_{n-2}x_{n-1}x_0} \tag{4a}$$

$$\mathsf{all\_different}(x_0, ..., x_{n-1}) \tag{4b}$$

$$x_0 = 0 \tag{4c}$$

$$x_i \in N \qquad i = 0, ..., n-1. \tag{4d}$$

The objective function is encoded using an *element* expression, $\mathrm{element}(\langle i, j, k \rangle \mid c) = c_{ijk}$, where i, j, and k are decision variables. Constraint (4b) ensures that each customer is visited exactly once. Constraint (4c) forces all feasible tours to start at customer 0, which reduces the symmetry without loss of generality since rooting a customer to a position does not alter the space of possible tour cycles. The above CP model is generic and convenient to implement as both element expressions and all-different constraints are commonly available in CP solvers.

There are additional global constraints that are suitable for encoding TSP-like problems, including *circuit* constraints, which enforce a Hamiltonian cycle on a directed graph (Hooker 2012), and specializations of disjunctive scheduling *no-overlap* constraints that enforce a separation between jobs specified by a distance-matrix (Laborie et al. 2018). The QTSP's quadratic cost function cannot be encoded using distance-matrix in the no-overlap constraints; hence, we do not use this constraint.

For our second model, we encode QTSP using a circuit constraint over integer variables $z_1, \ldots, z_n$, where z_i represents the edge (i, z_i). We add a boolean variable x_{ij} representing a directed edge from customer i to j, which allows us to describe the QTSP objective.

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} x_{ij} x_{jk} \tag{5a}$$

$$\mathsf{circuit}(z_1, \ldots, z_n) \tag{5b}$$

$$x_{ij} \leftrightarrow (z_i = j) \quad \forall i \in N, \forall j \in N \setminus \{i\} \tag{5c}$$

$$z_i \in N \setminus \{i\} \qquad \forall i \in N. \tag{5d}$$

The circuit constraint (5b) restricts the solutions to valid Hamiltonian cycles. We add additional logical constraints (5c) to channel between the boolean variables in the cost expression and the integer variables in the circuit constraint.

Dynamic Programming Models

Fischer et al. (2015) introduced a DP model; however, that DP model only solves the quadratic Hamiltonian path problem (QHPP). The QHPP differs from QTSP in its objective, as the cost of the tour $\langle \sigma(0), \ldots, \sigma(n-1) \rangle$ in QHPP is

$$\sum_{i=0}^{n-3} c_{\sigma(i)\sigma(i+1)\sigma(i+2)},$$

which excludes the costs of traveling back to the start of the tour, $c_{\sigma(n-2)\sigma(n-1)\sigma(0)}$ and $c_{\sigma(n-1)\sigma(0)\sigma(1)}$, that are part of the QTSP cost function (1). These costs are missing in the recursive function of Fischer et al.'s DP model, which makes it inaccurate for solving QTSP instances. We introduce an exact DP model for QTSP using the Bellman equation (6a)-(6b) (Bellman 1957).

$$\text{compute } V(N \setminus \{0\}, 0, 0, 0) \tag{6a}$$

$$V(U, i, j, f) = \begin{cases} \min\limits_{k \in U} V(U \setminus \{k\}, 0, k, k) & \text{if } j = 0, \\ \min\limits_{k \in U} (c_{ijk} + V(U \setminus \{k\}, j, k, f)) & \text{if } j \neq 0 \wedge U \neq \emptyset, \\ c_{j,0,f} + c_{i,j,0} & \text{if } j \neq 0 \wedge U = \emptyset. \end{cases} \tag{6b}$$

The Bellman equation has four state variables $\{U, i, j, f\}$ to represent QTSP: U is the set of unvisited customers, i is the previous customer visited, j is the current customer, and f is the first customer visited after 0. The first line of the equation (6b) defines $V(U, i, j, f)$ as the optimal cost of the tour that starts from 0 and visits all customers in U. The second and third line of the equation (6b) define $V(U, i, j, f)$ as the optimal cost of the path $\langle i, j, \sigma(1), \ldots, \sigma(|U|), 0, f \rangle$, σ being an optimal sequencing of U. The computation of objective (6a) gives us the optimal solution to the QTSP problem. Initially, there is no previous customer nor first visited customer and so we use $j = f = 0$ to represent that they are not decided. We include a detailed proof of the correctness of the Bellman equations in the appendix (Chen et al. 2025).

We implement the DP model using domain-independent dynamic programming (DIDP), a model-and-solve framework for solving DP problems (Kuroiwa and Beck 2023a, 2024). DIDP includes multiple general-purpose solvers for the DP models formulated in the Dynamic Programming Description Language (DyPDL), a solver-independent formalism for DP problems. Kuroiwa and Beck (2023b) show that the complete anytime beam search (CABS) algorithm for DIDP is the state-of-the-art model-based approach for a number of standard benchmark combinatorial optimization problems including TSP with time windows (TSPTW). We introduce two DyPDL models for Bellman equation (6a)-(6b) with different dual bound functions and use CABS to solve the models.

A DyPDL model is defined as a 7-tuple $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, \eta \rangle$, where each element is defined as follows:

- $\mathcal{V}$ is the set of *state variables*, where each state is defined uniquely by the values of the state variables.
- S^0 is the *target state*, which is the input state of the recursive function whose value is the optimal objective value.
- $\mathcal{K}$ is the set of *constant values* in the model that do not change with respect to states.
- $\mathcal{T}$ is the set of *transitions* that transform a state to another state and have associated costs.
- $\mathcal{B}$ is the set of *base case conditions*, where each base case condition is a conjunction of conditions on the state variables. A state that satisfies at least one base case condition is a termination of the recursive function.
- $\mathcal{C}$ is the set of the *state constraints* that must be satisfied by all states.
- η is a *dual bound function* that maps a state to a lower/upper bound on its optimal cost in a minimization/maximization problem.

The DyPDL model has the same state variables as the original DP model $\mathcal{V} = \{U, i, j, f\}$. Objective (6a) states that the objective is to compute the optimal cost for the *target state* $S^0 = (N \setminus \{0\}, 0, 0, 0)$. The first two lines of equation (6b) define the two sets of *transitions* in $\mathcal{T}$. The first set of transitions decides the first customer to visit after 0. These transitions have 0 cost since they form a state of only two visited customers and therefore cannot directly add to the tour cost, which depends on triples of consecutive customers. The second set of transitions decides the next customer $k \in U$ to visit at any state after the first customer has been determined. These transitions increment the cost by c_{ijk}, which is the cost of traveling to k, given the previous customer i and the current customer j. All transitions update the state variables appropriately. The third line of equation (6b) shows the *base case conditions* in this model: all customers are visited, and the current customer is not 0. The base state cost includes the cost of edge pairs ending in 0 and f. The *constant values* in this model are the costs of visiting any three customers in order, i.e., $c_{ijk} \forall i, j, k \in N$. There are no *state constraints* in this model.

We propose two *dual bound functions*, η_1 and η_2, that lead to two different DyPDL models: DIDP-1 and DIDP-2. The dual bound function η_1 is a maximization of three lower bounds on the optimal cost of the state $V(U, i, j, f)$, i.e.,

$$\begin{aligned}
&\eta_1(U, i, j, f) = \\
&\max \begin{cases} \sum_{k \in U \cup \{f,0\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}, \\ \sum_{k \in U \cup \{j,0\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lkm}, \\ \sum_{k \in U \cup \{i,j\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{klm}, \end{cases} \\
&\leq V(U, i, j, f). \qquad (7)
\end{aligned}$$

In the first lower bound, we underestimate the cost of visiting an unvisited customer k as the minimum cost for traveling to k from any pair of customers. The summation of the estimated cost of visiting each unvisited customer is a lower bound on the total cost to visit all unvisited customers in a sequence. In addition to the set of unvisited customers, the lower bound also considers the cost to visit f and 0 since their costs are not included in the recursive function until the tour cycle is finished. Similarly, the second lower bound underestimates the cost of visiting a customer when the current customer is k, and the third line underestimates the cost of visiting a customer when k is the previous customer. These lower bounds are not required for the correctness of the DP model; we add them to solve the DP model more efficiently using the DIDP paradigm.

Our second dual bound strengthens the first dual bound function by using the set of unvisited customers U more dynamically in its calculations. The dual bound function η_2 is defined as,

$$\begin{aligned}
&\eta_2(U, i, j, f) = \\
&\max \begin{cases} \sum_{k \in U \cup \{f,0\}} \max_{e \in N \setminus U_{in}} \min_{\substack{l \in N \setminus \{k,e\}, \\ m \in N \setminus \{k,l,e\}}} c_{lmk}, \\ \sum_{k \in U \cup \{j,0\}} \max_{e \in N \setminus U_{mid}} \min_{\substack{l \in N \setminus \{k,e\}, \\ m \in N \setminus \{k,l,e\}}} c_{lkm}, \\ \sum_{k \in U \cup \{i,j\}} \max_{e \in N \setminus U_{out}} \min_{\substack{l \in N \setminus \{k,e\}, \\ m \in N \setminus \{k,l,e\}}} c_{klm}, \end{cases} \\
&\leq V(U, i, j, f), \qquad (8)
\end{aligned}$$

where $U_{in} = U \cup \{i, j, 0\}, U_{mid} = U \cup \{i, j, f, 0\}$, and $U_{out} = U \cup \{j, f, 0\}$. Different from η_1, each minimization term in the first line of η_2 underestimates the cost to visit a customer k by taking the minimum of the traveling costs to k from any pair of customers, *excluding a previously visited customer* $e \in N \setminus U_{in}$. Since any customer $e \in N \setminus U_{in}$ cannot be visited in the two steps before k, the cost terms c_{emk} and c_{lek} can be soundly excluded from the first lower bound of η_1. Hence,

$$\max_{e \in N \setminus U_{in}} \min_{\substack{l \in N \setminus \{k,e\}, \\ m \in N \setminus \{k,l,e\}}} c_{lmk},$$

is a lower bound on the cost of visiting k. Notice each of these lower bounds is stronger then the ones used in η_1. This can be established for the first lower bound from the follow-

ing inequality.

$$\begin{aligned}\forall k \in N, \forall e \in U_{in}, U_{in} \subseteq N, \max_{e' \in N \setminus U_{in}} \min_{\substack{l \in N \setminus \{k,e'\}, \\ m \in N \setminus \{k,l,e'\}}} c_{lmk} \\ \geq \min_{\substack{l \in N \setminus \{k,e\}, \\ m \in N \setminus \{k,l,e\}}} c_{lmk} \\ \geq \min_{\substack{l \in N \setminus \{k\}, \\ m \in N \setminus \{k,l\}}} c_{lmk}. \end{aligned} \tag{9}$$

The second and third lower bounds in η_2 are constructed following the same idea and they are also stronger than the corresponding lower bounds in η_1, with a similar argument. Thus, η_2 is a stronger dual bound than η_1 in all states. However, the additional maximization increases the computational complexity of η_2 to $O(n^2)$ in comparison to the $O(n)$ complexity of η_1 as U is iterated over at each state. We analyze the model with each dual bounds experimentally to determine their efficiency and evaluate whether the trade-off computational cost of η_2 is justified.

Experimental Evaluation

We compare all the approaches on the set of standard instances with customer counts of $5, 10, 15, \ldots, 200$ created by Staněk et al. (2019). To test the scalability of our approaches, we also generate larger instances with customer counts of $250, 300, 350$, and 400 that have the same metric space and cost formulation as the standard instances. For a given number of customers, the benchmark contains 10 randomly generated maps, where the customer locations are distributed uniformly at random on a 500×500 grid. The benchmark has two problem instances for each map: an Angle-TSP instance and an AngleDistance-TSP instance, which differ in their definition of the cost functions as follows.

- **Angle-TSP instances** use the turning angle as the cost. Specifically, suppose the vehicle visits location i, j, and k in order, then the cost is the turning angle α_{ijk} between the vectors $\vec{ij}$ and $\vec{jk}$, multiplied by 1000, and rounded to 12 decimal places.
- **AngleDistance-TSP instances** have a cost function that combines the turning angle with the Euclidean distances between the points in a weighted sum. Let d_{ij} represent the Euclidean distance between i and j, and $\rho \in \mathbb{R}_0^+$ be a weighting parameter, then the cost of visiting locations i, j, k in order is.

$$c_{ijk} = 100 \left(\rho \cdot \alpha_{ijk} + \frac{d_{ij} + d_{jk}}{2} \right).$$

  Notice as $\rho \to \infty$, the instance is similar to Angle-TSP instances, and as $\rho \to 0$, the instance is similar to the standard TSP instances. In this benchmark, ρ is set to 40 for all instances.

In our experiments, we compare the six novel exact approaches introduced in this work (MIQP, MILP, two CP, and two DIDP models), Oswin et al. (2017) branch and cut (B&C) algorithm, and the hybrid genetic algorithm (HGA) of Pham et al. (2023). All approaches are given a time limit of 1800 seconds and memory limit of 8GB for solving instances with less than or equal to 200 customers. For the instances with more than 200 customers, we employ the same time limit but increase the memory limit to 16GB. The experiments are executed on an Intel Xeon Gold 6148 core at 2.4GHz using GNU Parallel (Tange 2011). We implement the MILP and MIQP models in Gurobi 11.0.3 (2024), and the CP models in CP Optimizer 22.1.1.0 (Laborie et al. 2018) and Or-Tools CP-SAT 9.10.4067 (Perron and Furnon 2024). CP-SAT shows similar performance differences between the two CP models as the CP Optimizer. However, CP Optimizer significantly outperforms CP-SAT. Consequently, we only discuss CP Optimizer results and include CP-SAT's results in the appendix (Chen et al. 2025).

We implement our DIDP models in DIDPPy 0.8.0 (Kuroiwa, Chen, and Beck 2024) and solve them using complete anytime beam search (CABS). CABS is an anytime algorithm based on beam search, introduced by Zhang (1998) and implemented in DIDP by Kuroiwa and Beck (2023b). The CABS algorithm implements an iterative beam search with increasing beam width, starting with a beam width of 1 and doubling it after each iteration. With a sufficiently large beam width, no nodes are discarded due to the limit and thus the algorithm terminates with proven optimality as the entire search space is traversed. Note that primal solutions found from earlier iterations are used to soundly prune nodes with equal or greater dual bound. We chose CABS because, as noted above, Kuroiwa and Beck (2023b) showed it was the best performing of the anytime state-based search methods implemented in the DIDP framework across a number of benchmark problems.

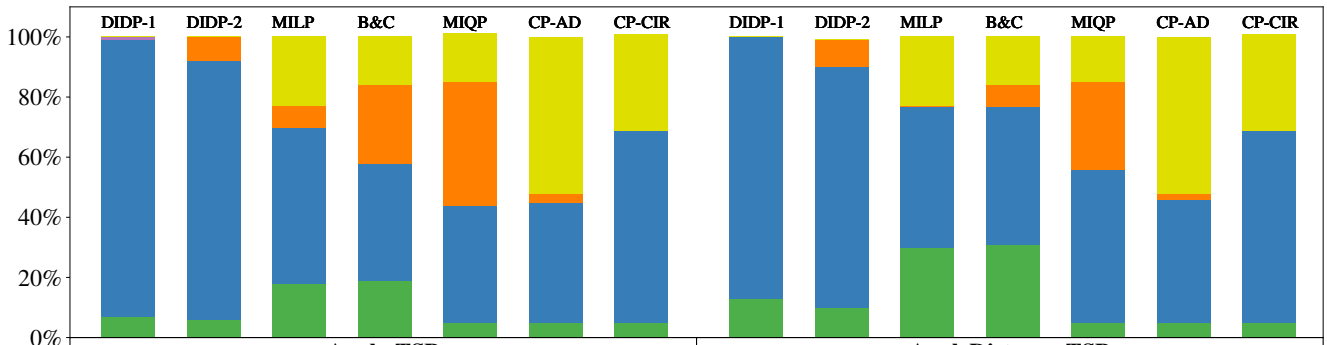
Following the best approach of Oswin et al. (2017), we implemented the B&C algorithm with cuts added only on integral solutions. We also use Gurobi 11.0.3 for the B&C implementation, employing the callback feature to lazily add subtour elimination cuts. The callback function for cut generation is implemented in Python. The Python algorithm does not cause any noticeable overhead, since we observe the time spent on the callback function is always less than 10 seconds.

We use the original implementation of Pham et al. (2023) for HGA, with the same hyper-parameter settings as the authors used in their experiments.

Results

Figure 1 shows the results of all exact solvers on the benchmark instances. It shows the number of instances (out of 440) in which a solver could prove the optimal solution (*opt*), find a feasible solution (*fs*) without proving optimality, or find no solution (*ns*). For the latter two categories, the results are split depending on the reason for termination: time-out (*to*) or memory-out (*mo*). Recall that DIDP-1 and DIDP-2 represent the DIDP model with the dual bound function η_1 (7) and η_2 (8). CP-AD represents the CP model with all-different constraint and CP-CIR represents the one with circuit constraint.

The plot highlights two significant findings: DIDP's strong performance in finding feasible solutions to QTSP in-



	Angle-TSP							AngleDistance-TSP						
	DIDP-1	**DIDP-2**	**MILP**	**B&C**	**MIQP**	**CP-AD**	**CP-CIR**	**DIDP-1**	**DIDP-2**	**MILP**	**B&C**	**MIQP**	**CP-AD**	**CP-CIR**
ns-mo	0	0	100	70	70	230	140	0	0	100	70	64	230	140
ns-to	0	34	30	116	180	13	0	0	40	1	32	128	10	0
fs-mo	4	0	0	0	0	0	0	0	0	0	0	0	0	0
fs-to	406	378	229	172	170	177	280	384	354	205	202	226	180	280
opt	30	28	81	82	20	20	20	56	46	134	136	22	20	20

Figure 1: The performance profiles of the exact approaches DIDP-1, DIDP-2, MILP, B&C, MIQP, CP-AD, and CP-CIR. The table captures the number and the bar graph shows the percentage of the 440 instances with each solution status: *opt* when the instance is solved to optimality, *fs*-{*to, mo*} when a feasible solution is found, but the algorithm terminates because of time-out (*to*) or memory-out (*mo*) before proving optimality, and *ns*-{*to, mo*} when no solutions are found within time or memory limit.

	DIDP-1		DIDP-2	
Class	# exp.	exp. time	# exp.	exp. time
Ang-5	40.5	1.32E-5	39.8	3.05E-5
Ang-10	3.63E4	2.73E-6	1.95E4	7.19E-5
AngD-5	37.4	1.58E-5	35.2	3.15E-5
AngD-10	2908.3	4.91E-6	1353.8	1.00E-4
AngD-15	5.33E4	4.00E-6	1.70E4	3.34E-4
AngD-20	1.15E6	5.66E-6	1.63E5	9.11E-4
AngD-25	1.99E7	7.89E-6	5.32E5	2.45E-3

Table 1: The average number of states expanded (# exp.) and the average node expansion time (exp. time) in seconds per node, over 10 instances in each class (Problem type-size), where Ang and AngD stand for Angle and AngleDistance, respectively. Only the sizes where both DIDP models solve all instances to optimality are included.

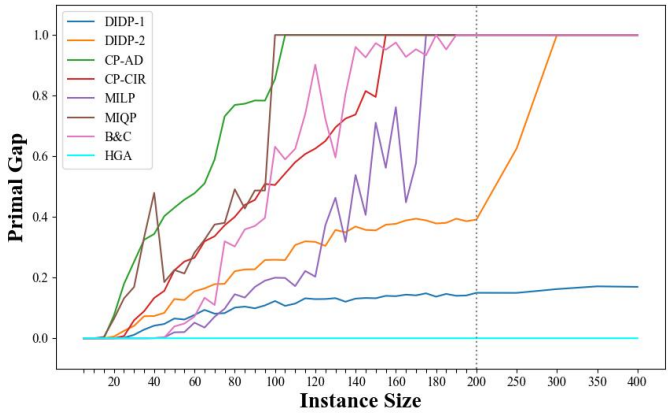
stances and the MILP and B&C approaches' performance in proving optimality. We observe that the DIDP-1 model finds feasible solutions for all problems and the DIDP-2 model finds feasible solutions for all except some of the large instances. This points to the strength of the CABS algorithm, which behaves similarly to greedy depth-first search (DFS) at the beginning of the search, allowing the solver to find feasible solutions quickly. The DIDP-2 model fails to find any feasible solution for large instances. In fact, for all instances over 250 customers, DIDP-2 is unable to even complete a single iteration of CABS (i.e., with width equal to 1) within the time limit due to the additional computational complexity of the dual bound function η_2. The impact of the more expensive but stronger dual bound can be seen in Table 1, where the DIDP-2 model always spends more time on each state expansion. We also observe that the stronger dual bound η_2 allows the DIDP-2 model to prove optimality with fewer state expansions. However, the reduced search space does not pay off due to the increase in expansion time and we observe an increase in the total runtime.

CABS also shows an advantage in memory usage, as it only reaches the memory limit on a small number of Angle-TSP instances while solving DIDP-1. When CABS exhausts memory in DIDP-1, it is because it progresses quickly and reaches high beam widths that consume a large amount of memory. The weaker dual bound prevents it from pruning sub-optimal nodes. DIDP-2 does not experience the same memory-out issue because it has a slower state expansion and stronger pruning but runs out of time before running out of memory. This result suggests that improvements in proving optimality require stronger and/or more efficient dual bounds than η_1 and η_2.

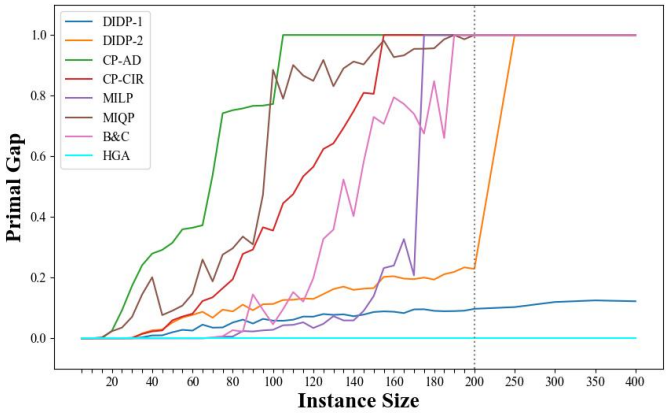
The LP-based approaches, MILP and B&C, show a significant advantage in proving optimality. MILP and B&C solve instances up to 75 customers optimally, while DIDP only does so on instances up to size 30. This result itself is interesting as it shows that the MILP model with the subtour elimination constraints introduced by Desrochers and Laporte (1991) matches the performance of the state-of-the-art B&C algorithm in proving optimality. At the same time, the MILP model finds feasible solutions for more instances. While MILP's close performance to B&C is unexpected, it is not entirely surprising that Desrochers and Laporte's constraints have not been previously explored for QTSP, given B&C is generally considered a superior model for classic TSP (Laporte 1992; Fischer et al. 2014). The B&C approach holds a slight advantage in terms of memory consumption, but this does not translate into more solutions being found.

Both our CP models reach the memory limit without finding a feasible solution for a large number of instances. In most of these instances, the CP solver terminated before the

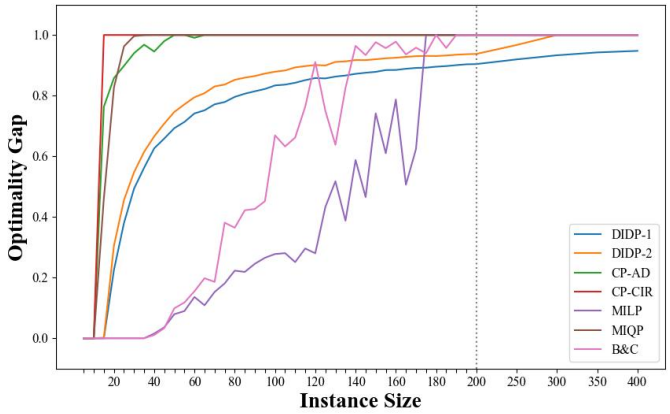
(a) Average *primal gap* for Angle-TSP instances.



(b) Average *primal gap* for AngleDistance-TSP instances.



(c) Average *optimality gap* for Angle-TSP instances.



(d) Average *optimality gap* for AngleDistance-TSP instances.

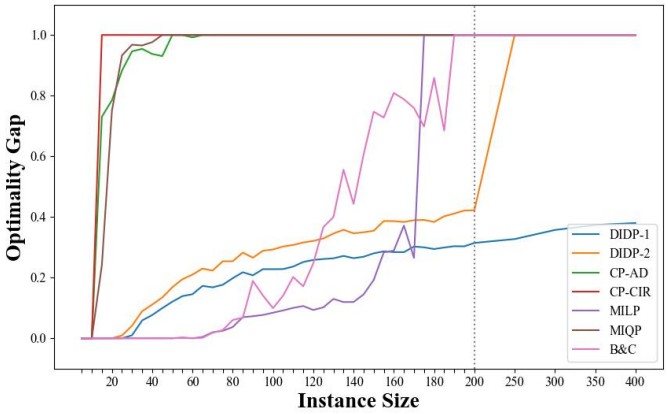


Figure 2: The plots of average *primal gap* and *optimality gap* found by each solver.

search could even start, indicating significant preprocessing effort in both time and memory. In CP-AD, this behavior is somewhat unexpected as the only constraint is all-different, and it is not hard to satisfy. It appears that the representation of the cost via the element expressions results in significant preprocessing that exceeds the memory limits.

Turning to a comparison of solution quality, Figures 2a and 2b show the *primal gap* found by each solver in the Angle-TSP and AngleDistance-TSP instances, averaged over ten instances for each problem size. The primal gap is defined as

$$\frac{|\text{Primal Bound} - \text{Best Known Solution}|}{\text{Primal Bound}},$$

where the Primal Bound is the cost of the best solution found by each solver and the Best Known Solution is either a known optimal solution provided in the benchmark or the best solution across all six solvers in our experiments. The benchmark provides optimal solutions up to instance size 75 for Angle-TSP and up to instance size 100 for AngleDistance-TSP. The primal gap is a measurement for the solution quality: a smaller primal gap means a better feasible solution.

From Figure 2a and 2b, we see that the HGA metaheuristic always finds best feasible solutions. Recall that HGA is customized for QTSP (Pham et al. 2023), enabling quick improvements of the primal bounds. Furthermore, HGA does not manage any lower bound information nor provide any guarantee on solution quality, allowing better scalability in the memory usage compared to the exact approaches.

Considering the exact approaches only, we observe the DIDP-1 model finds the best feasible solutions for the large instances of both problem type: for 80 or more customers for Angle-TSP and 145 or more for AngleDistance-TSP instances. However, the more expensive dual bound reduces the performance of DIDP-2: the solver is unable to explore as many states as DIDP-1 within the time limit, resulting in a larger primal gap. At the extreme, as DIDP-2 did not complete even a single iteration of CABS for larger instances, no feasible solutions are found.

The MILP model finds the best feasible solutions for the smaller instances, but fails to find any feasible solutions for the instances with size greater than 170. From Figure 1, we observe that most of instances for which MILP cannot find a solution exhibit a memory out. As with the CP models, the culprit appears to be the pre-solving stage: most of the instances that reach the memory limit terminate before search. While the B&C algorithm finds a feasible solution on a smaller number of instances than MILP, it has a slightly better memory usage than the MILP approach, allowing it to find a few more feasible solutions for instances with size 180 to 190 which we see in Figure 2a and 2b.

Figures 2c and 2d show the average optimality gap in the

Angle-TSP and AngleDistance-TSP instances, respectively. The optimality gap is calculated by

$$\frac{|\text{Primal Bound} - \text{Dual Bound}|}{\text{Primal Bound}},$$

where the Dual Bound is the best lower bound on the optimal solution proved by the corresponding solver. Since HGA does not compute any lower bound it does not provide an optimality gap and we exclude its results.

From Figure 2c and 2d, we observe that the DIDP models provide the tightest range for the optimal solution costs of the instances with more than 175 customers. Unlike the other approaches, the DIDP models achieve a much smaller optimality gap in AngleDistance-TSP instances than the Angle-TSP instances. The difference is caused by the larger cost range in Angle-TSP instances compared to the AngleDistance-TSP instances. Recall the Angle-TSP cost depends only on the angle between visit-triples, which can take on any value in $[0, 2\pi]$. In contrast, the AngleDistance-TSP cost includes both angle and distance, and the distance distributes among the hundreds of uniformly distributed nodes. Thus, in an Angle-TSP instance, the ratio between the mean and the minimum of the c_{ijk} terms over all $i, j, k \in N$ is larger than the one for the corresponding AngleDistance-TSP (see the appendix). Given that the dual bound functions in our DIDP models rely on the smallest cost for visiting a customer in any tour, a higher ratio between the mean and the smallest cost results in a weaker dual bound.

For the small instances, the MILP model has the best performance among all exact solvers. The performance degradation of the MILP model as the problem size increases is caused by its poor scalability on memory usage as discussed above. We also observe that the MILP and B&C models obtain some optimality gaps that are similar to their primal gaps, indicating that the LP-based approaches find tight dual bounds as also shown by the performance in proving optimality (Figure 1). In contrast, both DIDP models have an optimality gap that is much larger than the primal gap. In fact, the MILP and B&C models can achieve a better optimality gap with a worse primal solution in many instances compared to the DIDP models.

The CP models find and prove the fewest optimal solutions (Figure 1), and they are always outperformed by MILP and DIDP-1 in both the primal gap and the optimality gap measurements (Figure 2). By comparing the CP models to each other, we observe the CP-CIR finds a better feasible solution (Figure 2a and 2b), but delivers a worse optimality gap (Figure 2c and 2d) for all instances. In the log files we observe that the search speed (branches per second) of the solver while solving CP-CIR is much higher than while solving CP-AD. Thus, the CP-CIR model provides a better feasible solution by exploring more search space but the CP-AD model spends more effort on computing the dual bound for each branch.

Conclusion

Of the exact models we present for QTSP, DIDP-1 stands out as the most scalable, finding feasible solutions for all benchmark instances, even where other approaches run out of memory. This result is somewhat surprising given the general belief in the high memory consumption of dynamic programming approaches. However, this contradiction can be resolved by distinguishing between DP *models* and DP *algorithms*. While DIDP solves DP models, its does not necessarily solve such models with memory-intensive DP methods. Indeed by separating the model from the solver, DIDP provides an opportunity for substantial further research in solution techniques for DP models.

HGA, the state-of-the-art metaheuristic, is also highly scalable and finds the best solutions with up to 15% lower costs. DIDP-1 provides guarantees on solution quality, which HGA cannot, and surpasses all other methods in terms of optimality gap on large instances. However, DIDP-1 solves fewer instances to optimality and has a larger optimality gap on small and medium-sized instances than the MILP approach. Furthermore, although DIDP-2 employs a stronger dual bound and decreases the number of expansions required to prove optimality, it introduces a substantial computation time, indicating a need for more efficient lower-bounding methods.

While the state-of-the-art B&C algorithm scales slightly better in terms of memory consumption, a MILP model with position-based subtour elimination constraints (Desrochers and Laporte 1991) finds better feasible solutions while matching B&C performance in proving optimality. This result suggests that lazily generating sub-tour elimination cuts may not be required for QTSP.

Our CP models fall short of the B&C, MILP, and DIDP approaches. The CP solver exceeds the memory limit during preprocessing on many instances, even though both models compactly represent QTSP using global constraints. The CP solvers that we tested clearly experience scalability limitations in managing the quadratic cost expression of QTSP.

Overall, our findings underscore the significance of compact mathematical models of QTSP, which commercial and open-source exact solvers can directly solve. However, further improvements to these approaches are still necessary to boost their performance. DIDP, in particular, would benefit from more efficient dual bound calculations. Stronger dual bounds could be specified in the DIDP model, but their formulation is restricted to the expressions available in DyPDL. Alternatively, and possibly ultimately, adapting automatic abstraction-based heuristics studied in AI planning (Culberson and Schaeffer 1998; Helmert et al. 2014; Seipp and Helmert 2018) and state abstractions in operations research (Holte and Fan 2015; Baldacci, Mingozzi, and Roberti 2011) will be an essential step in improving DIDP. The correspondence between factored transition systems seen in AI Planning and those in DIDP is not, however, immediately apparent.

Acknowledgements

Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. Scinet is funded by ISED Canada; the Digital Research Alliance of Canada; Ontario Research Fund:RE; and the University of Toronto. This research was supported by the Natural Sciences and Engineering Council of Canada.

References

Aggarwal, A.; Coppersmith, D.; Khanna, S.; Motwani, R.; and Schieber, B. 2000. The Angular-Metric Traveling Salesman Problem. *SIAM Journal on Computing*, 29(3): 697–711.

Applegate, D.; Bixby, R. E.; Chvătal, V.; and Cook, W. J. 2003. Concorde. https://www.math.uwaterloo.ca/tsp/concorde/index.html. Accessed: 2024-10-29.

Baldacci, R.; Mingozzi, A.; and Roberti, R. 2011. New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem. *Operations Research*, 59(5): 1269–1283.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Chen, Y.; Singh, A.; Kuroiwa, R.; and Beck, J. C. 2025. Appendix to New Exact Methods for Solving Quadratic Traveling Salesman Problem. https://doi.org/10.5281/zenodo.15306819. Accessed: 2025-04-29.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence*, 14(3): 318–334.

Desrochers, M.; and Laporte, G. 1991. Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints. *Operations Research Letters*, 10(1): 27–36.

Fischer, A.; Fischer, F.; Jäger, G.; Keilwagen, J.; Molitor, P.; and Grosse, I. 2014. Exact Algorithms and Heuristics for the Quadratic Traveling Salesman Problem With an Application in Bioinformatics. *Discrete Applied Mathematics*, 166: 97–114.

Fischer, A.; Fischer, F.; Jäger, G.; Keilwagen, J.; Molitor, P.; and Grosse, I. 2015. Computational Recognition of RNA Splice Sites by Exact Algorithms for the Quadratic Traveling Salesman Problem. *Computation*, 3(2): 285–298.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com. Accessed: 2024-10-29.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Holte, R. C.; and Fan, G. 2015. State Space Abstraction in Artificial Intelligence and Operations Research. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 55–60.

Hooker, J. N. 2012. *Integrated Methods for Optimization*. Springer.

Jäger, G.; and Molitor, P. 2008. Algorithms and Experimental Study for the Traveling Salesman Problem of Second Order. In *International Conference on Combinatorial Optimization and Applications*, 211–224. Berlin, Heidelberg: Springer.

Kuroiwa, R.; and Beck, J. C. 2023a. Domain-Independent Dynamic Programming: Generic State Space Search for Combinatorial Optimization. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS)*, 236–244. Palo Alto, California USA: AAAI Press.

Kuroiwa, R.; and Beck, J. C. 2023b. Solving Domain-Independent Dynamic Programming Problems with anytime heuristic search. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS)*, 245–253.

Kuroiwa, R.; and Beck, J. C. 2024. Domain-Independent Dynamic Programming. arXiv:2401.13883 [cs.AI]. arXiv:2401.13883.

Kuroiwa, R.; Chen, Y.; and Beck, J. C. 2024. DIDP. https://didp.ai. Accessed: 2024-10-29.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilím, P. 2018. IBM ILOG CP Optimizer for Scheduling: 20+ Years of Scheduling With Constraints at IBM/ILOG. *Constraints*, 23: 210–250.

Laporte, G. 1992. The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research*, 59(2): 231–247.

Medeiros, A. C.; and Urrutia, S. 2010. Discrete Optimization Methods to Determine Trajectories for Dubins' Vehicles. *Electronic Notes in Discrete Mathematics*, 36: 17–24.

Oswin, A.; Fischer, A.; Fischer, F.; Meier, J. F.; Pferschy, U.; Pilz, A.; and Staněk, R. 2017. Minimization and Maximization Versions of the Quadratic Travelling Salesman Problem. *Optimization*, 66(4): 521–546.

Perron, L.; and Furnon, V. 2024. OR-Tools. https://developers.google.com/optimization. Accessed: 2024-10-29.

Pferschy, U.; and Staněk, R. 2017. Generating Subtour Elimination Constraints for the TSP From Pure Integer Solutions. *Central European journal of operations research*, 25: 231–260.

Pham, Q. A.; Lau, H. C.; Hà, M. H.; and Vu, L. 2023. An Efficient Hybrid Genetic Algorithm for the Quadratic Traveling Salesman Problem. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS)*, 343–351.

Savla, K.; Frazzoli, E.; and Bullo, F. 2008. Traveling Salesperson Problems for the Dubins Vehicle. *IEEE Transactions on Automatic Control*, 53(6): 1378–1391.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

Staněk, R.; Greistorfer, P.; Ladner, K.; and Pferschy, U. 2019. Geometric and LP-based Heuristics for Angular Travelling Salesman Problems in the Plane. *Computers & Operations Research*, 108: 97–111.

Tange, O. 2011. GNU Parallel - The Command-Line Power Tool. *The USENIX Magazine*, 36: 42–47.

Vidal, T.; Crainic, T. G.; Gendreau, M.; Lahrichi, N.; and Rei, W. 2012. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. *Operations Research*, 60(3): 611–624.

Zhang, W. 1998. Complete Anytime Beam Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 425–430.