10-16-2018

A Comparison of Algorithms for Finding an Efficient Theme Park Tour

Liz Bouzarth
Furman University

Richard J. Forrester
Dickinson College

Kevin Hutson
Furman University

Rahul Isaac
Furman University

James Midkiff
Dickinson College

See next page for additional authors

Recommended Citation

Elizabeth L. Bouzarth, Richard J. Forrester, Kevin R. Hutson, Rahul Isaac, James Midkiff, Danny Rivers, Leonard J. Testa, "A Comparison of Algorithms for Finding an Efficient Theme Park Tour", *Journal of Applied Mathematics,* vol. 2018, Article ID 2453185, 14 pages, 2018. https://doi.org/10.1155/2018/2453185

Hindawi
Journal of Applied Mathematics
Volume 2018, Article ID 2453185, 14 pages
https://doi.org/10.1155/2018/2453185

Research Article

A Comparison of Algorithms for Finding an Efficient Theme Park Tour

Elizabeth L. Bouzarth,[1] **Richard J. Forrester**,[2] **Kevin R. Hutson**,[1] **Rahul Isaac**,[1] **James Midkiff**,[2] **Danny Rivers**,[1] **and Leonard J. Testa**[3]

[1]*Furman University, Department of Mathematics, Greenville, SC 29613, USA*
[2]*Dickinson College, Department of Mathematics and Computer Science, Carlisle, PA 17013, USA*
[3]*TouringPlans.com, Celebration, FL 34747, USA*

Correspondence should be addressed to Richard J. Forrester; forrestr@dickinson.edu

Received 23 August 2018; Accepted 17 September 2018; Published 16 October 2018

Academic Editor: Quanke Pan

The problem of efficiently touring a theme park so as to minimize the amount of time spent in queues is an instance of the Traveling Salesman Problem with Time-Dependent Service Times (TSP-TS). In this paper, we present a mixed-integer linear programming formulation of the TSP-TS and describe a branch-and-cut algorithm based on this model. In addition, we develop a lower bound for the TSP-TS and describe two metaheuristic approaches for obtaining good quality solutions: a genetic algorithm and a tabu search algorithm. Using test instances motivated by actual theme park data, we conduct a computational study to compare the effectiveness of our algorithms.

1. Introduction

Theme parks like Walt Disney World in Orlando, Florida, attract millions of tourists each year. While these parks provide great entertainment, a common complaint is the amount of time spent waiting in line for the various attractions. Whole industries have arisen to help tourists maximize their entertainment value by offering advice on how to optimally tour parks to minimize such waiting (e.g., [1, 2]). Here, we look at the problem of finding a shortest route through an amusement park that takes into account the wait time of the attractions. This is an instance of the Traveling Salesman Problem with Time-Dependent Service Times (TSP-TS) [3].

The TSP-TS is a variation of the Traveling Salesman Problem (TSP) and was introduced by Tas et al. [3]. Given a set of locations and distances between every pair of locations, the classical TSP seeks to find the shortest possible route that visits each location and returns to the starting location. For the TSP-TS, the duration of time spent at each location is defined as a function of the arrival time to that location. The objective is to minimize the total route duration, which consists of the sum of the total travel time and the total service time. In our setting, the service times represent attraction wait times and ride times.

Time-dependency in the TSP literature is typically addressed in terms of travel times. In particular, the Time-Dependent Traveling Salesman Problem (TDTSP) seeks to find the shortest tour through the locations when the time to travel depends not only on the distance but also on the time of day the route is traversed. Because the TDTSP incorporates a more realistic touring cost structure, it has been used to model several other applications including scheduling single-machine jobs with time-dependent setup costs [4, 5], creating timetables for university exams to minimize back-to-back exams [6], production planning for car assembly lines [7], satisfying product demands at minimum travel and purchasing costs [8], and vehicle routing with varying travel times such as within regions of congestion [9–15]. Vander Wiel and Sahinidis [16, 17] note that the TDTSP is NP-hard, but little has been published in terms of heuristics, especially heuristics that incorporate a time-dependency. Multiple authors [4, 5, 12, 16–18] have proposed mixed-integer linear programs (MILP) for producing solutions to the TDTSP, but these exact solutions have generally been for

networks with tens of nodes rather than hundreds. Relaxations to these MILP formulations though have provided upper and lower bounds for TDTSP instances. Malandraki and Daskin [12] combine a MILP formulation with TSP nearest neighbor heuristics to solve random instances of up to 25 locations. Vander Wiel and Sahinidis [16] solve problems of size up to 100 locations by developing a time-dependent version of the well-known Lin-Kernighan heuristic with their MILP formulation. Cordeau et al. [19] consider a version of the TDTSP where the time range is subdivided into equal-length subintervals and the average travel speed is known. They use an algorithm by Ichoua et al. [20] to compute the travel time on each arc. Further, they develop a branch-and-cut algorithm to produce solutions for instances up to 40 vertices. Other approaches taken include dynamic programming [21], simulated annealing [22, 23], Monte Carlo methods [24], and genetic algorithms [25, 26]. With the exception of [16], most of the heuristic approaches employed are ones that were developed for the time-independent TSP and do not try to make use of the time-dependent nature of the problem.

The majority of the papers on the TDTSP focus on allowing congestion to build up on routes during certain parts of the day and consider the time spent at locations negligible [27]. However, in a theme park scenario, it is the variability in the wait time for rides that greatly increases tour time rather than variations in travel time. Tas et al. [3] introduce the Traveling Salesman Problem with Time-Dependent Service Times (TSP-TS) to model scenarios such as this. They show that in cases where service times are modeled by a linear or quadratic function with certain specifications, the service times cannot be incorporated into arc durations like those used in [19]. They propose formulations for the TSP-TS and measure the effectiveness of different subtour elimination constraints. We will show that our version of the TSP-TS also cannot be modeled as a TDTSP.

The main contributions made in this paper are as follows.

(1) We extend the TSP-TS introduced by Tas et al. [3] to include more general service time functions.

(2) By incorporating wait times into the distances between nodes, we construct an asymmetric TSP that can be used to compute a lower bound on the TSP-TS.

(3) We propose a mixed-integer linear programming model of the TSP-TS and describe a branch-and-cut algorithm based on this formulation.

(4) We create a new metric, δ, to guide tour construction and augmentation and we develop both a genetic algorithm and tabu search algorithm that can be used to find good quality solutions to the TSP-TS in an efficient manner.

(5) Motivated by actual theme park data, we introduce a collection of test problems that utilize different types of wait time distributions.

(6) Finally, we perform a detailed computational study to compare the effectiveness of our algorithms.

The remainder of this paper is organized as follows. In Section 2, we formally describe our version of the TSP-TS, propose a mixed-integer linear programming (MILP) formulation, and describe a branch-and-cut algorithm. In Section 3, we show that our version of the TSP-TS cannot be modeled as a TDTSP and describe a method for computing a lower bound on the optimal solution. Section 4 describes two different metaheuristic approaches for solving the TSP-TS using a genetic algorithm and a tabu search algorithm. We present our test data and computational results in Sections 5 and 6, respectively. Section 7 highlights our conclusion.

2. Problem Description and Formulation

In this section, we begin by introducing the notation that will be used throughout the paper. We then present a MILP model for our version of the TSP-TS and describe a branch-and-cut algorithm based on this formulation.

2.1. Notation. Let $G = (N, A)$ be a connected digraph with node set $N = \{1, 2, \ldots, n\}$ and arc set $A = \{(i, j) \mid i, j \in N,\ i \neq j\}$. Associated with each arc is a travel time from node i to node j, w_{ij}, while associated with each node i is a service time function $c_i(t)$, where t corresponds to the arrival time to node i. For our application, the nodes represent theme park attractions, w_{ij} represents the time to walk from attraction i to attraction j, and $c_i(t)$ represents the sum of the wait time and ride time when arriving at attraction i at time t. The TSP-TS aims to minimize the total tour duration, beginning and ending at node 1 (possibly representing the entrance to the park), including the total walking time and the total service time.

Tas et al. [3] considered both linear and quadratic service time functions $c_i(t)$. However, such functions are not realistic in a theme park scenario. In practice, service (wait) time data at each attraction is collected at $K + 1$ discrete times $T^0, T^1, \ldots, T^K$. In this paper, we consider two types of $c_i(t)$ functions based on data collected at these times. The first is to define $c_i(t)$ as a step function where $c_i(t)$ is set to be the wait time recorded at attraction i at time T^{m-1} for all $t \in [T^{m-1}, T^m)$. The second is to compute $c_i(t)$ using linear interpolation of the wait times recorded at T^{m-1} and T^m for all $t \in [T^{m-1}, T^m)$. While wait times computed using linear interpolation are more realistic, they are difficult to model using mixed-integer programming. However, such functions can easily be handled using metaheuristics.

2.2. Mathematical Model. To formulate the TSP-TS as an MILP, we will assume that $c_i(t)$ is a step function over the K intervals $[T^0, T^1), [T^1, T^2), \ldots, [T^{K-1}, T^K]$. We refer to $[T^{m-1}, T^m)$ as time interval m. Our model is formulated on an expanded graph where each arc (i, j) from node i to node j is replaced by K parallel arcs from i to j, one for each time interval. In this new network we define c_i^m to represent the service time (sum of the wait time and ride time) when arriving at node i during time period m.

For notational convenience, we introduce the set $S = \{2, 3, \ldots, n\}$ to represent the set of possible successor nodes

and define $M = \{1, \ldots, K\}$ to be the set of all time intervals. The decision variables for our formulation are as follows.

$$x_{ij}^{m} = \begin{cases} 1 & \text{if node } j \text{ is visited immediately after node } i \text{ and you arrive at node } j \text{ during time interval } m \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

a_{ij} = arrival time at node j when visited immediately after node i

Using the models presented in [12, 28] as a starting point, we formulate our version of the TSP-TS as follows.

$$\text{minimize} \quad \sum_{i \in S} a_{i1} \tag{2}$$

$$\text{subject to} \quad \sum_{\substack{i \in N \\ i \neq j}} \sum_{m \in M} x_{ij}^{m} = 1 \quad \forall j \in N \tag{3}$$

$$\sum_{\substack{j \in N \\ j \neq i}} \sum_{m \in M} x_{ij}^{m} = 1 \quad \forall i \in N \tag{4}$$

$$\sum_{j \in S} a_{1j} = \sum_{j \in S} \sum_{m \in M} w_{1j} x_{1j}^{m} \tag{5}$$

$$\sum_{\substack{j \in N \\ j \neq i}} a_{ij} = \sum_{\substack{l \in N \\ l \neq i}} a_{li} + \sum_{\substack{l \in N \\ l \neq i}} \sum_{m \in M} c_{i}^{m} x_{li}^{m} + \sum_{\substack{j \in N \\ j \neq i}} \sum_{m \in M} w_{ij} x_{ij}^{m} \quad \forall i \in S \tag{6}$$

$$\sum_{\substack{l \in N \\ l \neq i}} a_{il} \leq \sum_{\substack{j \in N \\ j \neq i}} \sum_{m \in M} \left(T^{m} - 1\right) x_{ij}^{m} \quad \forall i \in S \tag{7}$$

$$\sum_{\substack{l \in N \\ l \neq i}} a_{il} \geq \sum_{\substack{j \in N \\ j \neq i}} \sum_{m \in M} T^{m-1} x_{ij}^{m} \quad \forall i \in S \tag{8}$$

$$a_{ij} \leq \sum_{m \in M} T^{K-1} x_{ij}^{m} \quad \forall i, j \in N,\ i \neq j \tag{9}$$

$$x_{ij}^{m} \text{ binary} \quad \forall i, j \in N,\ i \neq j,\ m \in M \tag{10}$$

Recall that we assume that our tour begins and ends at node 1. The objective function (2) minimizes the arrival time back to node 1 as only one variable a_{i1} can be positive. Constraints (3) and (4) ensure that each node is visited exactly once. Constraint (5) computes the arrival time to node j after leaving node 1, while constraints (6) compute the arrival time to node j visited after node $i \in S$. Note that constraints (5) and (6) act as the subtour elimination constraints. The temporal constraints (7) and (8) ensure that the correct time interval m is chosen when visiting node l after node i (note that $[T^{m-1}, T^{m}) = [T^{m-1}, T^{m} - 1]$ since we assume T^{m} is an integer for all m). Finally, constraints (9) ensure that $a_{ij} > 0$ only if one of $x_{ij}^{m} = 1$ for any m.

To strengthen the continuous relaxation of the formulation, we can include additional restrictions designed to tighten the model as suggested in [12]. Toward this end, we define the following sets:

$$A_{ij}^{m} = \left\{p \in M \mid T^{p} < \left(T^{m-1} + c_{j}^{m}\right)\right\}$$
$$B_{jl}^{p} = \left\{m \in M \mid T^{p} < \left(T^{m-1} + c_{j}^{m}\right)\right\} \tag{11}$$

We can then add in the following new restrictions, where L is a lower bound on the optimal solution.

$$x_{ij}^{m} + \sum_{\substack{l \in N \\ l \neq j}} \sum_{p \in A_{ij}^{m}} x_{jl}^{p} \leq 1 \quad \forall m \in M,\ i \in N,\ j \in S,\ i \neq j \tag{12}$$

$$\sum_{\substack{i \in N \\ i \neq j}} \sum_{m \in B_{jl}^{p}} x_{ij}^{m} + x_{jl}^{p} \leq 1 \quad \forall l \in N,\ p \in M,\ j \in S,\ j \neq l \tag{13}$$

$$\sum_{i \in S} a_{i,1} \geq L \tag{14}$$

Constraints (12) and (13) tighten the formulation as follows. Suppose $x_{ij}^{m} = 1$ so that we travel from node i to node j and arrive during time period m. Then constraints (12) ensure that if you then visit node l immediately after node j, then you must use a time interval that occurs after your arrival to j. Suppose $x_{jl}^{p} = 1$ so that you travel from node j to node l and arrive during time period p. Then constraints (13) ensure that when you traveled from node i to node j you must use a time period that finishes service at j before you leave node j to travel to node l. Finally, constraint (14) allows us to utilize a lower bound on the optimal solution, such as that computed in Section 3.

2.3. Branch-and-Cut Algorithm. In this section we describe a branch-and-cut algorithm that we developed based on the MILP described in the previous section. Our algorithm begins with a preprocessing phase that first computes a lower bound L to use in constraint (14) using the method described in Section 3. We then utilize the metaheuristics of Section 4 to obtain an incumbent solution to seed the branch-and-bound procedure. The solution obtained from the metaheuristics is also used to determine an upper bound on the number of time intervals K that are needed within the MILP formulation.

Recall that between each pair of nodes there are K parallel temporal arcs, each of which is represented by a variable x_{ij}^m. It is therefore advantageous to keep the number of temporal arcs to a minimum so as to reduce the size of the MILP formulation, which we can accomplish using the best tour found by the metaheuristics.

After the preprocessing phase, we submit the MILP (2), (3), (4), (5), (6), (7), (8), (9), (10), (12), (13), and (14) and the incumbent solution to the mixed-integer programming solver CPLEX, which was used to implement the branch-and-cut algorithm. We utilize CPLEX's callback functionality to add cuts during the enumeration. Specifically, we incorporate additional subtour elimination constraints. Recall that constraints (5) and (6) act as the subtour elimination constraints for our formulation, which are similar to the Miller-Tucker-Zemlin constraints for the TSP [30]. Note that these constraints are weaker than the typical, yet exponential in number, subtour elimination constraints for the TSP that were introduced by Dantiz, Fulkerson, and Johnson [31]. We can extend these tighter subtour elimination constraints to our formulation as follows, where S is the node set of a subtour and $|S|$ is the cardinality of S:

$$\sum_{i\in S}\sum_{j\in S}\sum_{m\in M} x_{ij}^m \le |S| - 1. \tag{15}$$

At each node of the branch-and-bound tree we identify violated subtour elimination constraints (15) of the LP relaxation using a separation technique based on a min-cut algorithm of [32] and subsequently add these constraints to the formulation. Even though these additional cuts are not necessary for the elimination of the subtours, the inequalities strengthen the linear programming relaxation of the problem. Note that the inequalities (15) added at any node of the enumeration tree are valid for all the other nodes because these inequalities are valid for the entire formulation (3), (4), (5), (6), (7), (8), (9), (10), (12), (13), and (14). Thus, at a given node, all the inequalities generated so far were incorporated into the formulation.

3. Computing a Lower Bound

We begin this section by showing that our version of the TSP-TS cannot be modeled as a TDTSP, and therefore we are unable to use the techniques that have been developed for computing lower bounds for the TDTSP. We then describe a method for computing a lower bound for the TSP-TS.

First, we need to introduce some additional notation. We define each tour of the nodes as a permutation $\pi = [1, \pi_2, \ldots, \pi_n]$, where $\pi_i \in N$, $2 \le i \le n$, and associated with tour π is a tour cost, $z(\pi) = a_{\pi_n 1}$, representing the time it takes to traverse the tour as described by permutation π and to return to $\pi_1 = 1$, the entrance. To break down this tour cost, recall we defined $a_{\pi_{i-1}\pi_i}$ to be the time that the tour arrives at the ith attraction, and let us define d_{π_i} to be the time that the tour departs the ith attraction. Note we define $d_1 = 0$. We can then recursively define the arrival and departure times for each of the attractions as follows. The arrival time at the ith attraction for $i > 1$ is

$$a_{\pi_{i-1}\pi_i} = d_{\pi_{i-1}} + w_{\pi_{i-1}\pi_i}. \tag{16}$$

The departure time of the ith attraction for $i > 1$ is

$$d_{\pi_i} = a_{\pi_{i-1}\pi_i} + c_{\pi_i}\left(a_{\pi_{i-1}\pi_i}\right). \tag{17}$$

Thus,

$$z(\pi) = a_{\pi_n 1} = d_{\pi_n} + w_{\pi_n \pi_1}. \tag{18}$$

Following the approach in [19] it seems reasonable to try to incorporate service time variability into travel times and use many of the results in the TDTSP literature to establish bounds for the TSP-TS. Much of this literature proposes to model the TDTSP by establishing a nonnegative traversal rate $v_{ij}(t)$ for arc (i, j) during times t in time period $[T^{m-1}, T^m)$. This rate is fixed during each time period but is allowed to change to $v_{ij}(t')$ when the time period changes from $[T^{m-1}, T^m)$ to $[T^m, T^{m+1})$, where $t' \in [T^m, T^{m+1})$, even if the arc traversal is not complete. This modeling technique offers the benefits of establishing lower bounds for the TDTSP. However, as we show here, the TSP-TS cannot be modeled by using this technique.

Consider an instance of the TSP-TS with the following conditions during time periods $[T^{m-1}, T^m)$ and $[T^m, T^{m+1})$:

(1) It is possible to arrive at node π_{i-1} at some time t_0, where $t_0 < T^m$ and arrive at node π_i at some time $t = t_0 + c_{\pi_{i-1}}(t_0) + w_{\pi_{i-1}\pi_i}$ with $T^m < t < T^{m+1}$.

(2) The slope of the service time function for some node π_{i-1} is -1 during time period $[T^m, T^{m+1})$ so that, for instance, $c_{\pi_{i-1}}(T^m) = c_{\pi_{i-1}}(T^{m+1}) + (T^{m+1} - T^m)$.

In this instance of the TSP-TS, if some tour traverses arc (π_{i-1}, π_i) starting at time T^m, then it will arrive at node π_i at some time $t' = T^m + c_{\pi_{i-1}}(T^m) + w_{\pi_{i-1}\pi_i}$. However, if some tour traverses arc (π_{i-1}, π_i) beginning at T^{m+1}, then it must arrive at node π_i at time

$$\begin{aligned} &T^{m+1} + c_{\pi_{i-1}}\left(T^{m+1}\right) + w_{\pi_{i-1}\pi_i} \\ &\quad = T^m + \left(c_{\pi_{i-1}}\left(T^{m+1}\right) + T^{m+1} - T^m\right) + w_{\pi_{i-1}\pi_i} \\ &\quad = T^m + c_{\pi_{i-1}}\left(T^m\right) + w_{\pi_{i-1}\pi_i} = t'. \end{aligned} \tag{19}$$

Note in departing node π_{i-1} at time T^m or at time T^{m+1} we arrive at node π_i at time t' in both cases.

Let us attempt to model this TSP-TS using a nonnegative traversal rate $v_{\pi_{i-1}\pi_i}(t)$ common in modeling the TDTSP. Since we arrive at node π_i at the same time, t', regardless of whether we begin traversing arc (π_{i-1}, π_i) at time T^m or time T^{m+1}, we cover the same distance, and we have

$$\begin{aligned} &v_{\pi_{i-1}\pi_i}\left(T^m\right)\left[T^{m+1} - T^m\right] + v_{\pi_{i-1}\pi_i}\left(T^{m+1}\right)\left[t' - T^{m+1}\right] \\ &\quad = v_{\pi_{i-1}\pi_i}\left(T^{m+1}\right)\left[t' - T^{m+1}\right]. \end{aligned} \tag{20}$$

This implies that $v_{\pi_{i-1}\pi_i}(T^m)[T^{m+1} - T^m] = 0$, and thus $v_{\pi_{i-1}\pi_i}(T^m) = 0$. Hence, a tour in our TSP-TS cannot cover any ground along arc (π_{i-1}, π_i) during the interval $[T^m, T^{m+1})$. This means that if some tour in the TSP-TS finishes traversing arc (π_{i-1}, π_i) at time x, then $x \leq T^m$ or $x > T^{m+1}$. This contradicts the first condition of our TSP-TS instance, which implies that a tour can complete edge (π_{i-1}, π_i) at time between T^m and T^{m+1}. Thus, this instance of a TSP-TS cannot be modeled by converting service times to edge traversal rates as is common in modeling the TDTSP. Further, this scenario is common to amusement parks, especially among attractions that have batch servicing.

Cordeau et al. [19] use variable arc traversal speeds to find a lower bound for the TDTSP. They create this lower bound by assigning, in each time interval, each arc its maximum speed found over all time intervals. This creates an asymmetric Traveling Salesman Problem (ATSP) as all arcs now have a fixed traversal time. Solutions to this ATSP provide a lower bound to the TDTSP. However, as shown in [3], solutions to the TSP in the case of variable service times are not necessarily a lower bound for the TSP-TS.

For a preliminary lower bound on the TSP-TS, one may assign to each node i a wait time equal to the minimum wait time achieved at that node throughout the day, $\mu_i = \min_t c_i(t)$. If we incorporate this time into every arc leaving node i, the problem becomes an instance of the asymmetric TSP where arc (i, j) has a weight of $\mu_i + w_{ij}$. Let $z^0(\pi)$ denote the time for a tour π to finish in this ATSP. Notice that, for any tour π, $z(\pi) \geq z^0(\pi)$, since every arc in the TSP-TS costs at least as much as the corresponding arc in the ATSP. Thus, for a tour π_A^* that is optimal to the ATSP problem, $z^0(\pi) \geq z^0(\pi_A^*)$, and as a result $z(\pi) \geq z^0(\pi_A^*)$, for all tours π. Thus $z^0(\pi_A^*)$ is a lower bound for the TSP-TS.

However, in amusement parks, wait times for different rides often increase together, and it is unlikely to create a tour with every ride achieving a minimum wait time. We can find an improved lower bound by incorporating the extra wait cost incurred by not visiting each attraction at its minimum wait time. Define $e_{\pi_i} = c_{\pi_i}(a_{\pi_{i-1}\pi_i}) - \mu_{\pi_i}$ to be the wait time above μ_{π_i} for each node π_i in a tour π. Then π has some total extra cost accumulated over all nodes, $\sum_{i=1}^{n} e_{\pi_i}$.

Consider a tour π^* that is optimal to an instance of a TSP-TS. We would like to find a tighter lower bound of $z(\pi^*)$ than $z^0(\pi_A^*)$. To that end, let $S = \{\pi_1^*, \pi_2^*, \ldots, \pi_k^*\}$ be the set of nodes corresponding the first k attractions visited by the optimal tour π^*. Let $\Pi(S)$ be the set of all permutations of the nodes in S, and let $\zeta' = [\zeta_1', \zeta_2', \ldots, \zeta_k']$ be the permutation of S that achieves $\min_{\zeta \in \Pi(S)} \sum_{i=1}^{k} e_{\zeta_i}$. Each permutation $\zeta \in \Pi(S)$ has a time that it requires to visit each of its k nodes. We call this time $\tau(\zeta)$. Let $\zeta^* \in \Pi(S)$ be the permutation of S that achieves $\min_{\zeta \in \Pi(S)} \tau(\zeta)$. That is, among all tours that begin with some permutation of the nodes in S, ζ^* finishes the tour of the nodes in S fastest, while ζ' minimizes the extra wait incurred in any tour of nodes in S. For each node $v \in N - S$ we can redefine μ_v, call it $\mu_v'(\tau(\zeta^*))$, to be the minimum value of $c_v(t)$ over all times $\tau(\zeta^*) \leq t \leq z(\pi)$, where π is any solution to the TSP-TS. Thus node $v \in N - S$ incurs an extra wait of at least $\mu_v'(\tau(\zeta^*)) - \mu_v$ because it must be visited after time $\tau(\zeta^*)$, as shown in Figure 1.

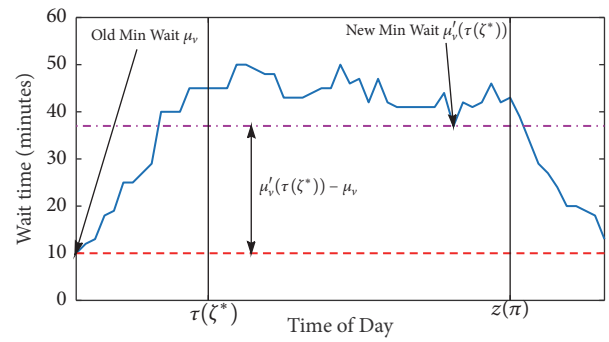

Figure 1: Extra wait time incurred by calculating a new minimum wait time $\mu_v'(\tau(\zeta^*))$ for the feasible window of time that node v will be visited, $[\tau(\zeta^*), z(\pi)]$.

To build a new lower bound on $z(\pi^*)$, we start with the observation that for the particular S used by π^*, the extra wait incurred by the first k nodes of π^* is bounded below by the minimum possible extra wait over all permutations of the nodes in S:

$$\sum_{i=1}^{k} e_{\pi_i^*} \geq \sum_{i=1}^{k} e_{\zeta_i}. \tag{21}$$

Additionally, the extra wait time for all nodes visited in π^* after the kth node must be at least as large as the difference of the updated minimum wait after $\tau(\zeta^*)$ and the original minimum wait time:

$$\sum_{i=k+1}^{n} e_{\pi_i^*} \geq \sum_{v \in N-S} \left[\mu_v'(\tau(\zeta^*)) - \mu_v\right] \tag{22}$$

Starting with the idea of adding extra wait times incurred to the existing lower bound $z^0(\pi_A^*)$ and incorporating inequalities (21) and (22), we observe the following:

$$\begin{aligned} z(\pi^*) &\geq z^0(\pi_A^*) + \sum_{i=1}^{n} e_{\pi_i^*} \\ &= z^0(\pi_A^*) + \sum_{i=1}^{k} e_{\pi_i^*} + \sum_{i=k+1}^{n} e_{\pi_i^*} \\ &\geq z^0(\pi_A^*) + \sum_{i=1}^{k} e_{\zeta_i'} + \sum_{v \in N-S} \left[\mu_v'(\tau(\zeta^*)) - \mu_v\right]. \end{aligned} \tag{23}$$

Define the lower bound of the extra wait time of the nodes in S as

$$E(S) = \sum_{i=1}^{k} e_{\zeta_i'} + \sum_{v \in N-S} \left[\mu_v'(\tau(\zeta^*)) - \mu_v\right]. \tag{24}$$

Since we do not know which k nodes of N the optimal tour π^* starts with, we exhaustively consider each of the $\binom{n}{k}$ possible sets S that π^* may start with and examine all $k!$ possible

orderings of each S to find ζ', ζ^*, and ψ, the set of k nodes that minimizes $E(S)$. This would then form a lower bound z_ψ that improves upon the previous lower bound, $z^0(\pi_A^*)$. Continuing from inequality (23), we have

$$z(\pi^*) \geq z^0(\pi_A^*) + E(S) \geq z^0(\pi_A^*) + E(\psi) = z_\psi. \quad (25)$$

As π^* must start with some k nodes, it must incur extra wait based on some S that was considered, and so must have extra wait at least as large as the minimum, $E(\psi)$. This extra wait was completely ignored in the ATSP. In using this approach to form the lower bound, we typically use $5 \leq k \leq 7$.

4. Metaheuristics

In this section we describe two different metaheuristics that can be used to solve our version of the TSP-TS. In particular, we develop both a genetic algorithm and a tabu search algorithm.

Several authors have proposed metaheuristic approaches to solving instances of the TDTSP. Testa et al. [26] implement eight different genetic operators on 50 randomly generated instances of the TDTSP and found certain combinations of genetic operators were effective at producing high-quality solutions. They show that the crossover operators edge recombination and cycle crossover along with high levels of mutation produce particularly good solutions on these instances and better solutions than can be generated via dynamic programming in a fraction of the time (see Section 4.1.4 for crossover descriptions). Li et al. [25] use a chained Lin-Kernighan algorithm with a double-bridge kick mutator within a genetic algorithm framework to effectively produce solutions to problems with realistic traffic assumptions (including time periods and areas with traffic jam travel restrictions).

Before we discuss our metaheuristic approaches, we define a new metric for comparing possible adjustments to a tour that takes into account differences from the average wait times of attractions. Certain tour-construction heuristic approaches will attempt to build a tour making choices using local information at each step. Two nodes i and j with equal service times $c_i(t) = c_j(t)$ can seem equally attractive to visit in time t. However, a twenty minute wait for an attraction that has an average wait time of an hour at time t is more of a bargain than an attraction with an average wait of fifteen minutes. We let $\overline{c_i}$ be the average wait for attraction i. Then for each time t, a node i has a deviation from its average

$$\gamma_i(t) = c_i(t) - \overline{c_i}. \quad (26)$$

Nodes that have little variation from average can be visited during any time period without much bonus or penalty. However, nodes that show large deviation from average can potentially provide huge tour time savings and also carry enormous time penalties. For example, Figure 2 shows an attraction whose wait time is broken down into 15 minute time periods. The average wait time, approximately 40 minutes, is shown with the dashed line. We note that $\gamma_i(t_1) = 8 - 40 = -32$ while $\gamma_i(t_{35}) = 48 - 40 = 8$. If this

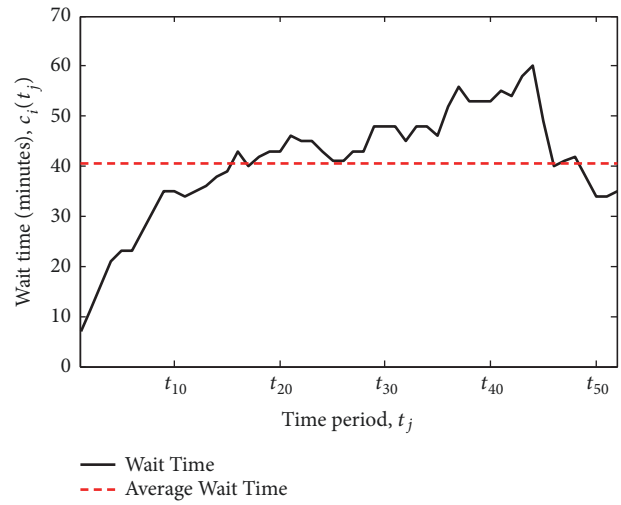

Figure 2: Wait time data for an attraction shown with 15 minute time periods. Dashed line demonstrates the average wait time, $\overline{c_i}$.

node can be scheduled earlier or later in the tour, there are time-saving benefits (when $\gamma_i < 0$), and there are penalties for scheduling it during the middle of the day (when $\gamma_i > 0$).

We also take into account an average travel time between nodes. Let $\omega_{ij} = w_{ij} - \overline{w_i}$, where $\overline{w_i}$ is the average of all $n - 1$ travel times associated with edges starting at node i over all t. Using Equations (16), (17), and (26) we define a metric that incorporates these averages of both wait times and travel times:

$$\delta_{ij}(d_i) = \gamma_{\pi_j}(d_i + w_{ij}) + \omega_{ij} = \gamma_{\pi_j}(a_{ij}) + \omega_{ij}. \quad (27)$$

To the best of our knowledge, this metric has not appeared in the literature.

4.1. Genetic Algorithms for the TSP-TS. Genetic algorithms can be customized to fit a problem and are normally defined by choices in the following components [33]: population representation and initialization, fitness evaluation, reproduction selection, and choice of genetic and replacement operators.

We define each candidate solution in the population to be represented by a permutation π of the n locations, which we refer to as a tour, and having fitness $z(\pi)$, the tour duration given in (18). We keep 40 candidate tours at every generation. At each iteration of a genetic algorithm, we choose an operator via dynamic operator selection as described in [26]. With this selection process, operators that have demonstrated success have a higher likelihood to be chosen to be used, where success is measured by how many direct children, grandchildren, etc. were inserted into the top half of the population as a result of this operator. Either one or two tours from the population are chosen at random, depending on the number of tours the operator takes as input. The operator produces an offspring tour. As in [26], we employ a $(P+1)$ reproduction approach. That is, if x is the candidate solution with the highest (worst) fitness value, and

y is the new offspring tour, we compare the tour costs of x and y and keep the tour with the lower (better) fitness value.

4.1.1. Population Initialization. In generating the initial population, we randomly generate a portion of the 40 tours. We combine these random tours with tours generated by more sophisticated algorithms. We tested many of the most commonly used construction heuristics, such as nearest neighbor and a dynamic programming algorithm proposed by Malandraki and Dial [21]. We also tested a variant of nearest neighbor introduced here, called **δ-Nearest Neighbor (δNN)**. This heuristic is the same as nearest neighbor except at each iteration of the tour construction, if we are located at node i at time period t, we choose a node j that minimizes the value of $\delta_{ij}(t)$ in (27).

4.1.2. Local Search Operators. Many genetic algorithms employ a local search heuristic to improve population tours. It has been shown in [34] to be an effective way to improve solution quality. A common local search heuristic for the TSP is the k-opt exchange heuristic. These heuristics remove k arcs of a current solution and replace them with k different arcs to reproduce a tour. This procedure is repeated until no improvements can be found. The 2-opt heuristic has been shown to be particularly effective for improving solutions for the TSP. Figure 3 shows a 2-opt improvement for a tour of eight attractions.

The 2-opt in Figure 3 reverses arcs (W, U), (U, S), and (S, P) meaning that these attractions are visited at different times of day than previously, which can significantly change the tour length. In a time-independent case, this would not cause the same effect.

The Lin-Kernighan algorithm [35] is a variable k-opt procedure that dynamically determines how many exchanges will be considered and has been shown to be very effective at producing high-quality solutions for both the symmetric TSP and the TDTSP. We considered three versions of the Lin-Kernighan (LK) heuristic differentiated by whether it took the first improvement, exhaustively searched to find the best 2-opt improvement, or used the δ metric. We found that the version that uses the first improving 2-opt, we refer to this as LKTD-1.

4.1.3. Mutators. Mutators take one tour as input and produce a new tour by augmenting the selected tour in some fashion. We considered several mutators and our tests show the following to be effective for this problem.

(i) **Uniform Order-based Mutation (UOM)**: This operator, described in [36], works on a single parent tour by performing a simple swap of two randomly selected nodes in that tour.

(ii) **δ-Mutate (δM)**: This mutation operator, like UOM, performs a single swap of nodes in a parent tour by finding the node after the arc that has the greatest penalty in terms of the δ metric from Equation (27) and swaps it with a random other node.

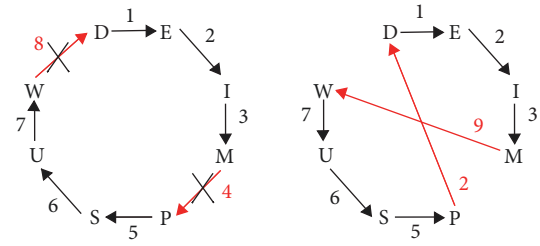

Figure 3: An example of the 2-opt heuristic on eight attractions removes edges (W, D) and (M, P), with weights 8 and 4, respectively, and replaces them with edges (M, W) and (P, D), with weights 9 and 2, respectively. This 2-opt improves the tour fitness from 36 to 35.

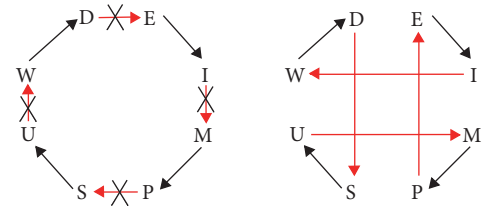

Figure 4: An example of the double-bridge kick operator on eight attractions that removes edges (D, E), (I, M), (P, S), and (U, W) and replaces them with edges (D, S), (P, E), (I, W), and (U, M).

(iii) **Double-Bridge Kick Operator (BK)**: As described in Li et al. [25], this operator, which cannot be achieved with 2-opts alone and which is useful in escaping local minima, randomly removes four arcs from the tour and relinks the nodes in a different manner, as shown in Figure 4.

4.1.4. Crossover Operators. Crossover operators take two parent tours from the population and try to strategically combine them to produce an offspring. The crossover operators we found most effective were the following.

(i) **Cycle Crossover (CC)**: This operator, described in [37], produces an offspring from two parents by ensuring any node appearing in position i in the offspring tour must appear in position i in at least one of the parent tours.

(ii) **Edge Recombination (ER)**: First proposed for the TSP in [38], ER produces a single offspring tour from two parent tours by building a list of edges present in both parents and transferring these edges to the offspring. The offspring is then completed by adding in the remaining nodes offering preference to nodes with smaller numbers of outgoing edges. As noted in [26], the motivation behind ER is that in the TSP connections between locations contribute more to the tour length than the position of the locations in the tour.

(iii) **Node Recombination (NR) via Path Relinking**: This operator follows the strategy of path relinking in [39]. A path is constructed between two parent tours by searching the neighborhood of the tours. Given

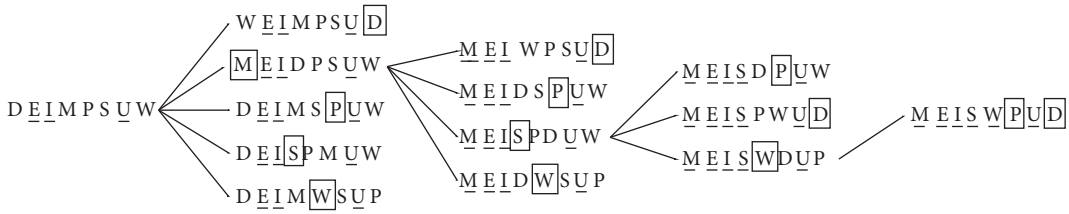

Figure 5: Path relinking shown on a tour of eight attractions. Underlined nodes signify agreement between the two original parent tours. Boxed nodes signify nodes that are in the correct spot due to the swap performed during that step.

two parent tours, x and y, we search solutions that share nodes in the same position in the tour as both parents. To describe the process, Figure 5 shows two parent tours, $x = [D, E, I, M, P, S, U, W]$ and $y = [M, E, I, S, W, P, U, D]$, which share common nodes in positions 2, 3, and 7. In positions where x and y do not have common elements, these elements are changed in succession as follows. If x differs from y in position i, x_i is moved to its position in y, say position j, and the element in x_j is moved to position i while all other nodes are held constant. We thus create a tour x' with more nodes in common with y than x had in common with y. For example, in Figure 5, x differs from y in position 1. The element in position 1 in tour x, node D, is moved to its position in y, position 8, and the element in position 8 in x, node W, is moved to position 1 in x to create a new tour. This change is performed sequentially for each $1 \leq i \leq n$ where the tours have noncommon elements and each of these resulting tours is evaluated for its fitness. The tour with the best fitness after these switches is chosen as the new parent tour x, tour $[M, E, I, D, P, S, U, W]$ in Figure 5, and the process is repeated until any switch results in tour y. The best tour found in the intermediate steps is chosen as the offspring of x and y.

(iv) **Sort Crossover (SX)**: This operator is essentially δNN where the selection set is restricted to the two parent tours. The SX operator takes two tours x and y and constructs the offspring tour z so that at each iteration i, z_i is chosen to be the node with the minimum δ metric (equation (27)) out of the next node in x or in y that has not already appeared in the constructed tour z.

4.1.5. Preliminary Computational Study. To test the effectiveness of different local search operators, mutators, and crossover operators when applied to instances of the TSP-TS, we completed an extensive computational study on randomly generated instances of the TSP-TS (see Section 5 for details of our test problems). For the sake of brevity, we omit the specific details of our tests here. The most effective combination of initialization, local search operators, mutators, and crossover mutators is described as Algorithm A in Table 1. To provide a basis of comparison we also describe the genetic algorithm proposed in [26], identified as Algorithm B in Table 1. Note that all of our tests regarding genetic algorithms in Section 6 will focus on these two algorithms.

Table 1: Genetic algorithms used for computational study. Algorithm A is our best combination of operators and Algorithm B is the best found in the literature [26].

Algorithm	A	B
Initialization	Random, δNN	Random
Local Search	LKTD-1	LKTD-1
Mutator	δM	UOM
Crossover(s)	NR, SX	CX

4.2. Tabu Search Algorithm. In this section we describe the tabu search algorithm used to solve TSP-TS. Tabu search, originally introduced by Glover [40] and later formalized in [41–43], is a metaheuristic algorithm that is known to be quite effective for hard combinatorial optimization problems. Tabu search begins with an initial solution and uses a local search procedure to move from the current solution to the best one in its neighborhood, even if the move leads to a worsening of the objective function value. To prevent becoming stuck at a local optima plateau, attributes of solutions that have recently been visited are declared "tabu" for a certain number of iterations.

4.2.1. Search Neighborhoods. In our tabu search, which is similar to [44], we use the nearest neighbor heuristic to generate an initial solution. Our search neighborhoods are defined by performing all possible *swap* and *shift* moves that are not tabu. Given a tour $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$, where π_i represents the attraction in position i, the swap move chooses two attractions π_i and π_j and then exchanges the attractions so that each attraction is located in the position previously occupied by the other one. The shift move chooses two attractions i and j with $i < j$ and a number m with $m \leq i$, which defines the number of attractions to move. It then relocates attractions $i - m$ through i to j, shifting any attractions between i and j to the left. In our algorithm, we consider all possible shift moves for every pair of positions i and j and for every value $m \leq i$.

The two neighborhoods defined by the swap and shift moves are determined separately and the best allowable moves from each neighborhood are compared to choose the one to be performed. During the search process, we use the standard aspiration criteria which allows a tabu move if it

leads to a tour with a better solution than the best solution found thus far. Our tabu search stops after a specified amount of time has passed, which we will discuss further in Section 6.

4.2.2. Tabu Status and Tenures. For our tabu search, we used a fixed tabu tenure of $7.5\ln(n)$ as suggested in [44]. When a swap occurs, exchanging attractions i and j, swapping any attractions located at the positions i and j is declared tabu. That is, swapping the index positions becomes tabu rather than swapping the attractions themselves. When a shift occurs, shifting i and j with $i < j$, moving attractions i and j to their previous positions is made tabu. This is similar to how [44] handled making swaps tabu, but we found it useful when performing shifts. A shift move of i and j is considered tabu if moving attraction i to the position of j is tabu. Note that the swap and shift tabu lists are independent of each other.

4.2.3. Diversification Strategies. To help drive the search into new regions of the solution space we use both *light* and *strong* diversification techniques. A light diversification is applied when the current tour's fitness value has not changed by over 0.5% in the past $\lfloor n/4 \rfloor$ iterations or when the current tour's fitness value has not changed at all in the past five iterations. When the light diversification is applied, the exhaustive swap and shift neighborhoods are determined using an alternate fitness function that adds a small penalty to the true objective value. Specifically, for every tour π', the frequency penalty $p(\pi')$ is equal to the sum of the iterations that each node k has been at index j in the current tour, divided by the total number of iterations. The new fitness value for the tour is then equal to $\widehat{F}(\pi') = F(\pi') + 0.1p(\pi')F(\pi')/\sqrt{n}$. This new fitness function urges the exhaustive searches to find tours outside of the local optima plateau. This type of function was first suggested by [45] and later used by [44].

A strong diversification is utilized when the current tour's fitness value has not changed by over 0.5% in the past $\lfloor n/8 \rfloor$ iterations or when the best fitness value has not changed for over $9n$ iterations. When strong diversification is applied, a subset of $\lfloor n/5 \rfloor$ nodes from the current tour are randomly permuted to yield a new current tour. To select the subset of nodes to be scrambled, we keep track of the number of iterations each node has been at the same index and select the $\lfloor n/5 \rfloor$ nodes that have been at their current index the longest. This diversification can have a significant impact in terms of sending the algorithm into an unexplored area of the search space.

5. Test Instances

Within a theme park, the wait time data varies from attraction to attraction and from hour to hour. Using data supplied by touringplans.com [29], Figure 6 shows estimated wait times for 29 attractions at Walt Disney World's Magic Kingdom for the date October 20, 2014. As evidenced from the figure, some rides have high wait times throughout the day while others have wait times that peak early, such as thrill rides, peak midday, such as kid attractions, or exhibit a sawtooth pattern, such as bulk-entry shows.

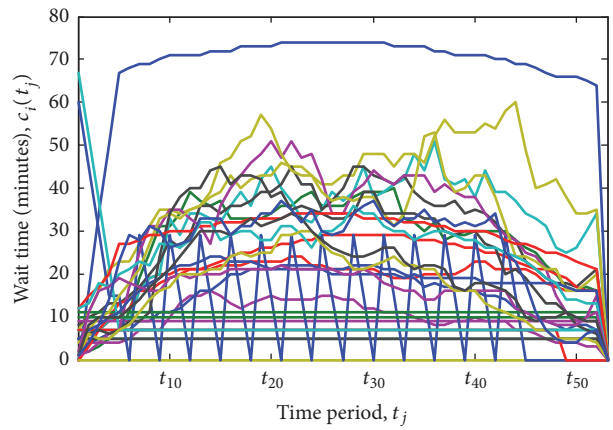

FIGURE 6: Wait times for 29 attractions at the Magic Kingdom [29].

To develop test problems that mimic the types of wait time distributions seen at Walt Disney World, we generate a wait time $c_i(t)$ at each attraction i that resembles either a uniform, sawtooth, or peaking distribution. We also generate a travel time $w_{ij}(t)$ between attractions i and j. Because we focus on the scenario that the wait times, $c_i(t)$, dwarf the walking times, $w_{ij}(t)$, we generated the walking time functions independent of t. To generate w_{ij}, each node was randomly assigned a position on a 1000 by 1000 grid, and the Euclidean distance was calculated between each pair of nodes. This result was divided by 100, to give a result generally between 0 and 15, but largely clustered around the 5-9 range. This aligned well with the travel times seen in real-world scenarios.

We used the actual theme park data to motivate the collection of distributions from which we generate wait times for networks of various sizes. For each distribution, we have a collection of parameters that we sample from the indicated range to give each node its own wait time distribution. There are five types of distributions we consider to generate a distribution on the interval $x \in [0, x_e]$. In each case, we compute a scaled distribution $\widetilde{f}(x)$ incorporating the average of the distribution over the interval $[0, x_e]$, $\overline{f}$, and a randomly chosen scaling factor, c:

$$\widetilde{f}(x) = \frac{cf(x)}{\overline{f}}. \tag{28}$$

(1) **Uniform distribution**: Let $f(x) = 1$, where the parameter c is randomly chosen from the interval $[5, 15]$.

(2) **Sawtooth distribution**: Let $f(x) = s - (x \bmod s)$, where s is randomly selected from the set $\{x_e/26, x_e/13\}$ and $c = s/2$. For many of our test cases, $x_e = 780$ minutes (this represents a common theme park operating schedule, e.g., 9am–10pm), so s is randomly selected from the set $\{30, 60\}$ and c is either 15 or 30.

(3) **Bimodal distribution**: Let

$$f(x) = \frac{\exp\left(-(x-\mu_1)^2/2\sigma_1^2\right)}{\sqrt{2\pi}\sigma_1} + \frac{\exp\left(-(x-\mu_2)^2/2\sigma_2^2\right)}{\sqrt{2\pi}\sigma_2}, \tag{29}$$

where the following parameters are chosen randomly from the given intervals:

$$\begin{aligned} \mu_1 &\in [0.2, 0.4]\, x_e \\ \mu_2 &\in [0.6, 0.8]\, x_e \\ s_1 &\in [2, 5] \\ s_2 &\in [2, 5] \\ \sigma_1 &\in \frac{\mu_2 - \mu_1}{s_1} \\ \sigma_2 &\in \frac{\mu_2 - \mu_1}{s_2} \\ c &\in [15, 70] \end{aligned} \tag{30}$$

(4) **Beta Distribution**: Let

$$f(x) = \frac{x}{x_e}\left(1 - \frac{x}{x_e}\right)^{\beta-1}, \tag{31}$$

where β is chosen randomly from the set $\{2, 3, 4, 5\}$ and c is chosen randomly from the interval $[15, 70]$.

(5) **Erlang Distribution**: Let

$$f(x) = \frac{1}{(k-1)!}\left(\frac{k}{4}\right)^k \left(\frac{7x}{x_e}\right)^{k-1} \exp\left(-\frac{7kx}{4x_e}\right), \tag{32}$$

where k is randomly chosen from the set $\{2, 3, 4\}$ and c is randomly chosen from the interval $[15, 70]$.

The Beta and Erlang distributions can produce asymmetric distributions, so we also introduce a randomly chosen reflection option where we use $f(x_e - x)$ instead of $f(x)$. Figure 7 shows examples of each of these distributions.

As mentioned before, the distributions in Figure 7 attempt to mimic wait times that might typically occur in a theme park. The Beta distribution with $\beta = 5$ and $c = 40$ might describe an attraction that has a larger wait time at the beginning of the day but diminishes throughout the day. Whereas the bimodal distribution represents an attraction that gets crowded during the early morning hours and during the afternoon but has lower wait times during the lunch hours. The sawtooth pattern might represent an attraction that takes in visitors at fixed points in time, like a show or movie attraction that runs on the hour and half-hour.

Using this collection of distributions, we developed ten different types of networks. These networks are described in Table 2, which provides the percentage of attractions that are assigned to each type of distribution for each network type. We randomly generated both a size 30 and 50 node problem for each network type, yielding a test bed of 20 randomly generated instances. In addition to these synthetic problems, we also conducted tests on three amusement parks in Walt Disney World in Orlando, Florida: the Magic Kingdom, Disney's Hollywood Studios, and Disney's Animal Kingdom. The data for these parks corresponds to the wait times for October 20, 2014, and was provided by touringplans.com [29].

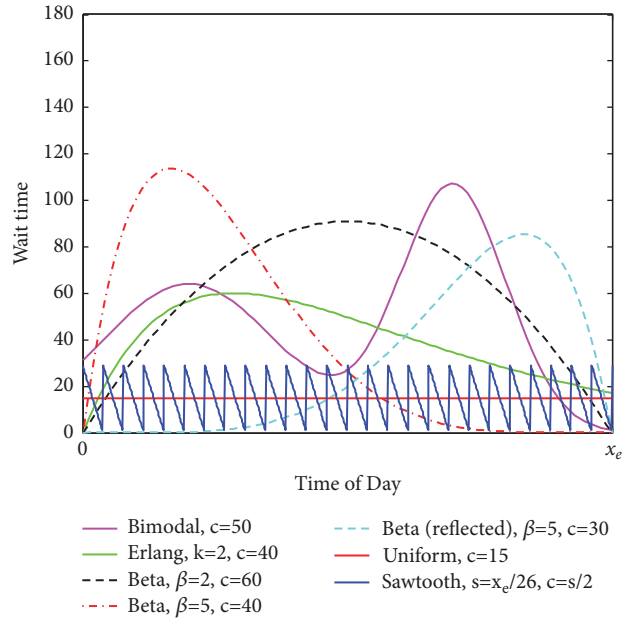

FIGURE 7: Examples of distributions used to generate wait times for test problems.

6. Computational Results

We begin by examining our results for the three metaheuristics: genetic algorithms A and B described in Section 4.1, and the tabu search algorithm described in Section 4.2. The algorithms were implemented in Java and executed on a Dell Precision T5610 Workstation equipped with dual 2.6 GHz processors and 32 GB of RAM running 64-bit Windows 7. To allow for comparisons between the three algorithms, we utilized a fixed time limit of five minutes for each algorithm, as opposed to using a fixed number of generations for the genetic algorithms or a fixed number of iterations for the tabu search.

In Table 3 we report the minimum objective values (tour fitness $z(\pi)$ from equation (18)) found using the three different algorithms, along with the lower bounds computed using the method described in Section 3, for the ten networks of size 30. Within the table, the first column identifies the network type (as described in Table 2), while the second column gives the lower bound computed. The next three columns give the minimum objective values found by genetic algorithm A,

Table 2: Percentage of testing networks comprised by each type of distribution.

Network	Constant	Bimodal	Erlang (L)	Erlang (R)	Beta (L)	Beta (R)	Sawtooth
1	20	20	10	10	10	10	20
2	100	0	0	0	0	0	0
3	0	100	0	0	0	0	0
4	0	0	50	0	50	0	0
5	0	0	25	25	25	25	0
6	50	50	0	0	0	0	0
7	50	0	12.5	12.5	12.5	12.5	0
8	40	0	10	10	10	10	20
9	40	40	0	0	0	0	20
10	40	20	5	5	5	5	20

Table 3: Results for size 30 networks.

		Linear Interpolation			Step		
Network	LB	GA-A	GA-B	Tabu Search	GA-A	GA-B	Tabu Search
1	145	276	275	**263**	**204**	207	**204**
2	388	**388**	**388**	**388**	**388**	**388**	**388**
3	67	**93**	**93**	94	**92**	**92**	94
4	160	**521**	**521**	**521**	**403**	**403**	404
5	112	512	**510**	512	**424**	**424**	444
6	227	**257**	**257**	**257**	**257**	**257**	**257**
7	338	**609**	**609**	**609**	**504**	**504**	536
8	220	298	298	**288**	**242**	245	245
9	199	**238**	244	**238**	217	217	**213**
10	179	240	251	**239**	**214**	**214**	**214**

genetic algorithm B, and tabu search, respectively, when the service times $c_i(t)$ are computed using linear interpolation. The last three columns give the same information for when the service times $c_i(t)$ are computed using a step function. The boldfaced entries indicate the minimum value achieved over all cases.

A number of observations can be made from Table 3. First, note that when the service times were computed using linear interpolation, the tabu search algorithm was the most effective, achieving the minimum tour length found among the three algorithms for eight of the ten networks. However, when the service times were computed using a step function, genetic algorithm A was superior, achieving the minimum tour length found for nine out of the ten networks, while the tabu search algorithm performed poorly. Second, note that for Network 2, which consists entirely of wait times that are constant throughout the day, all three algorithms were able to achieve a tour length equal to the lower bound of 388, which must therefore be the optimal solution. This is not surprising in that when all the wait times are constant, the TSP-TS essentially reduces to a standard TSP, which is not difficult to solve for 30 nodes. Third, the quality of the lower bounds obtained varied significantly depending on the network. For example, the lower bounds for Networks 1, 4, 5, and 7 were quite poor in that the gap between the bound and the best tour found is quite large. We believe this is due to the abundance of attractions with Beta and Erlang distributions and the nature in which we formulate the lower bound. Recall, the lower bound tests permutations of $5 \leq k \leq 7$ attractions to find a permutation with low tour costs and combine that with minimum wait times within the remaining time window from the attractions left to tour. In most theme parks, this works well in achieving a good lower bound because most attractions achieve a minimum wait time during the first few time periods and all attractions see an increase in wait times as time period progress. In networks with left and right Erlang and Beta distributions, there are attractions achieving their minimum wait times during the first few time periods and others achieving their minimum in later time periods. Hence, this tends to create lower bound tours utilizing attractions with minimum wait times achieved in the first few time periods as part of the set ζ but still have many attractions achieving minimum wait times later, thus not increasing the lower bound substantially.

In Table 4 we report the same information as in Table 3, but for the size 50 networks. Genetic algorithm A and the tabu search algorithm performed similarly for both methods of computing the service times, achieving the minimum objective value found for about half of the network instances. Genetic algorithm B appears to be the weakest of the three algorithms, only achieving the minimum objective value found in three of the network instances for both methods of computing the service times. Note that for Network 2, which consists entirely of wait times that are constant

Table 4: Results for size 50 networks.

		Linear Interpolation			Step		
Network	LB	GA-A	GA-B	Tabu Search	GA-A	GA-B	Tabu Search
1	214	482	481	**448**	379	388	**348**
2	593	**593**	**593**	594	**593**	**593**	594
3	102	**210**	216	217	**207**	211	211
4	153	**678**	682	681	648	647	**640**
5	114	807	**806**	**806**	773	773	**765**
6	356	**382**	383	387	**381**	**381**	391
7	374	**552**	**552**	557	**497**	**497**	**497**
8	300	496	493	**467**	**397**	409	399
9	279	391	419	**381**	359	356	**352**
10	297	501	512	**494**	444	469	**434**

Table 5: Results for Disney Parks.

		Linear Interpolation			Step		
Park	LB	GA-A	GA-B	Tabu Search	GA-A	GA-B	Tabu Search
Magic Kingdom	543	**651**	652	652	**638**	**638**	642
Animal Kingdom	369	**407**	411	**407**	**400**	404	**400**
Hollywood Studios	329	**752**	**752**	**752**	**752**	**752**	**752**

throughout the day, both genetic algorithms were able to obtain the optimal solution. Networks 4 and 5 continue to be troublesome because of the difficulty these networks present for achieving an adequate lower bound.

In Table 5 we provide the results of the metaheuristics applied to the three Disney theme parks, which uses the same layout as Tables 3 and 4. Genetic algorithm A is the most effective algorithm in that it was able to achieve the minimum tour value found for all three theme parks. The big difference between the lower bound gap at Hollywood Studios has to due with the abundance of shows with a sawtooth distribution at this theme park. It is difficult to find tours that would be able to achieve the minimum wait time of 0 for each of these shows (equivalent to arriving at each of these attractions exactly as the show is starting). Most of these shows do not even open until later time periods, so they would not be included in the ζ set in calculating the lower bound and thus would have a minimum wait time in later time periods of 0.

We now discuss our results of the MILP and the branch-and-cut method of Section 2. Our algorithm was implemented in Java using ILOG Concert and CPLEX 12.6, and was executed on the same machine as the metaheuristics. Unfortunately, our results were fairly disappointing. First, we found that adding the cuts based on the stronger subtour elimination constraints (15) did not improve the effectiveness of the model. That is, we found that simply submitting the formulation (2), (3), (4), (5), (6), (7), (8), (9), (10), (12), (13), and (14) to CPLEX was more efficient than adding the additional stronger cuts during the enumeration. This is curious because many other authors have found success utilizing this methodology (for example, see [12, 28]). We believe that the issue is related to the large number of parallel temporal arcs K. While other authors utilize similar temporal arcs, the number of arcs that these authors consider is much smaller. For example, in [12, 28] they consider formulations with $K = 3$ time periods, while in comparison we utilized approximately $K = 20$ temporal arcs for our size 30 networks (recall that we use the metaheuristics to help determine the number of arcs needed). Second, we found that our MILP was unable to solve any of our test networks to optimality, and therefore does not appear to be an effective solution method for the TSP-TS.

While our MILP, and the branch-and-cut algorithm based on this model, were not successful at optimizing our test cases of the TSP-TS, we found that it was useful in quantifying the quality of the solutions generated by our metaheuristics for when the service times are computed using a step function. For the 10 randomly generated networks of size 30, we formulated (2), (3), (4), (5), (6), (7), (8), (9), (10), (12), (13), and (14) and submitted the models to CPLEX using the best solution found by the three metaheuristics as an incumbent solution. We selected the CPLEX parameter to emphasize optimality over feasibility and set a time limit of 54,000 CPU seconds (15 hours). Our results are presented in Table 6, where the first column identifies the network type, the second column provides the minimum solution fitness found among all the considered metaheuristics, and the third column gives MIP gap between the best integer objective and the objective of the best node remaining upon reaching the time limit.

We first note that CPLEX was never able to improve upon the incumbent solutions generated by the metaheuristics. As we can see from Table 6, five out of the ten solutions have MIP gaps of less than 10%. However, it is important to note that since CPLEX was terminated before achieving optimality, it is quite likely that the actual gaps are smaller than those listed in Table 6. This table suggests that our metaheuristics are able to generate good quality, if not optimal, solutions. Not surprisingly, Networks 4, 5, and 7 have the largest optimality

Table 6: MIP gaps of minimum objective values found.

Network	Min. Fitness	Optimality Gap
1	204	13.16%
2	388	0%
3	92	12.91%
4	403	60.30%
5	424	63.67%
6	257	7.95%
7	504	32.94%
8	242	9.09%
9	213	4.07%
10	214	7.67%

gaps. Again, we believe this is due to the presence of one-sided Beta and Erlang distributions in large quantities.

7. Concluding Remarks

On crowded days at theme parks, the wait time of an attraction depends heavily on the time of day that the attraction is visited. Given that frustration over waiting in lines can ruin vacations, there is a need for effective solutions to the TSP-TS that incorporates the time-dependency of the problem. In this paper, we have extended the TSP-TS literature to include these real-world service time functions and showed how a lower bound on an optimal solution can be calculated. We investigated a variety of approaches to generate optimal and near-optimal solutions including a new metric used in construction approaches that gives preference for using attractions with high potential time savings, a branch-and-cut algorithm for our proposed MILP, and metaheuristic approaches. Our genetic algorithm and tabu search produced promising and efficient results, but our computational study highlighted the need for more sophisticated methods to yield tighter lower bounds to aid the branch-and-cut algorithm.

These results are encouraging, but there are further constraints that could be included in the model to make it more realistic. As it stands, our tours give patrons no time to eat or take breaks during the day, two highly desirable characteristics of theme park tours. In particular, meals could be incorporated as additional attractions where the wait times could be interpreted as the time it takes to complete a meal at a given establishment. Breaks could be implemented in a similar fashion. Each of these could have user-defined preferences for the times of day that these occur.

Further, theme parks have innovated around the wait time issue. Most parks now offer a limited number of queue priorities, which are passes that allow for shorter wait times where patrons are given a time window to arrive to receive priority entrance to the attraction. Sometimes these passes are unlimited, but patrons are charged extra for their use. In other instances, the number is limited, but the opportunity is included with the price of admission. With the introduction of these priority passes, an interesting question arises for future study. Mainly, how can patrons optimally choose a limited number of queue priorities to further reduce the time it takes to tour a theme park?

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was funded by the Furman Advantage Summer Research Fellowship, the Furman University Summer Mathematics Undergraduate Research Fellowship, and the Dickinson College Curley Endowment for Student-Faculty Research.

References

[1] B. Guides, *Birnbaum's 2019 Walt Disney World: The Official Guide*, Birnbaum Guides, 2018.

[2] B. Sehlinger and L. Testa, *Unofficial Guide to Walt Disney World 2019*, Unofficial Guides, 2019.

[3] D. Tas, M. Gendreau, O. Jabali, and G. Laporte, "The traveling salesman problem with time-dependent service times," *European Journal of Operational Research*, vol. 248, no. 2, pp. 372–383, 2016.

[4] K. Fox, B. Gavish, and S. Graves, *The Time Dependent Traveling Salesman Problem and Extensions*, vol. 7904, Graduate School of Management, University of Rochester, 1979.

[5] J.-C. Picard and M. Queyranne, "The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling," *Operations Research*, vol. 26, no. 1, pp. 86–110, 1978.

[6] N. Balakrishnan, A. Lucena, and R. T. Wong, "Scheduling examinations to reduce second-order conflicts," *Computers & Operations Research*, vol. 19, no. 5, pp. 353–361, 1992.

[7] F. Jaehn and H. A. Sedding, "Scheduling with time-dependent discrepancy times," *Journal of Scheduling*, vol. 19, no. 6, pp. 737–757, 2016.

[8] E. Angelelli, M. Gendreau, R. Mansini, and M. Vindigni, "The traveling purchaser problem with time-dependent quantities," *Computers & Operations Research*, vol. 82, pp. 15–26, 2017.

[9] B. Fleischmann, M. Gietz, and S. Gnutzmann, "Time-varying travel times in vehicle routing," *Transportation Science*, vol. 38, no. 2, pp. 160–173, 2004.

[10] M. Gendreau, G. Ghiani, and E. Guerriero, "Time-dependent routing problems: A review," *Computers & Operations Research*, vol. 64, pp. 189–197, 2015.

[11] A. Haghani and S. Jung, "A dynamic vehicle routing problem with time-dependent travel times," *Computers & Operations Research*, vol. 32, no. 11, pp. 2959–2986, 2005.

[12] C. Malandraki and M. S. Daskin, "Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms," *Transportation Science*, vol. 26, no. 3, pp. 185–200, 1992.

[13] S. Mancini, "A combined multistart random constructive heuristic and set partitioning based formulation for the vehicle

routing problem with time dependent travel times," *Computers & Operations Research*, vol. 88, pp. 290–296, 2017.

[14] J.-Y. Potvin, Y. Xu, and I. Benyahia, "Vehicle routing and scheduling with dynamic travel times," *Computers & Operations Research*, vol. 33, no. 4, pp. 1129–1137, 2006.

[15] R. Spliet, S. Dabia, and T. Van Woensel, "The time window assignment vehicle routing problem with time-dependent travel times," *Transportation Science*, vol. 52, no. 2, pp. 261–276, 2018.

[16] R. J. V. Wiel and N. V. Sahinidis, "Heuristic bounds and test problem generation for the time-dependent traveling salesman problem," *Transportation Science*, vol. 29, no. 2, pp. 167–183, 1995.

[17] R. J. V. Wiel and N. V. Sahinidis, "An exact solution approach for the time-dependent traveling-salesman problem," *Naval Research Logistics (NRL)*, vol. 43, no. 6, pp. 797–820, 1996.

[18] A. Montero, I. Mndez-Daz, and J. J. Miranda-Bront, "An integer programming approach for the time-dependent traveling salesman problem with time windows," *Computers & Operations Research*, vol. 88, pp. 280–289, 2017.

[19] J.-F. Cordeau, G. Ghiani, and E. Guerriero, "Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem," *Transportation Science*, vol. 48, no. 1, pp. 46–58, 2014.

[20] S. Ichoua, M. Gendreau, and J. Potvin, "Vehicle dispatching with time-dependent travel times," *European Journal of Operational Research*, vol. 144, no. 2, pp. 379–396, 2003.

[21] C. Malandraki and R. B. Dial, "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem," *European Journal of Operational Research*, vol. 90, no. 1, pp. 45–55, 1996.

[22] J. Schneider, "The time-dependent traveling salesman problem," *Physica A: Statistical Mechanics and Its Applications*, vol. 314, no. 1-4, pp. 151–155, 2002.

[23] Y. Xiao and A. Konak, "A simulating annealing algorithm to solve the green vehicle routing & scheduling problem with hierarchical objectives and weighted tardiness," *Applied Soft Computing*, vol. 34, pp. 372–388, 2015.

[24] J. Bentner, G. O. G. Bauer, I. Morgenstern, and J. Schneider, "Optimization of the time-dependent traveling salesman problem with monte carlo methods," *Physical Review E*, 2001.

[25] B. Golden, S. Raghavan, and E. Wasil, *Solving the Time Dependent Traveling Salesman Problem*, vol. 29, Springer US, 2005.

[26] L. Testa, A. Esterline, G. Dozier, and A. Homaifar, "A comparison of operators for solving time-dependent traveling salesman problems using genetic algorithms," *GECCO*, pp. 995–1102, 2000.

[27] A. Lucena, "Time-dependent traveling salesman problem—the deliveryman case," *Networks*, vol. 20, no. 6, pp. 753–763, 1990.

[28] G. Stecco, J.-F. Cordeau, and E. Moretti, "A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times," *Computers & Operations Research*, vol. 35, no. 8, pp. 2635–2655, 2008.

[29] https://touringplans.com/.

[30] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, pp. 326–329, 1960.

[31] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[32] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, 1997.

[33] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999.

[34] H. Tamaki, H. Kita, N. Shimizu, K. Maekawa, and Y. Nishikawa, "Comparison study of genetic codings for the Traveling salesman problem," in *Proceedings of the 1st IEEE Conference on Evolutionary Computation. Part 1 (of 2)*, pp. 1–6, June 1994.

[35] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, pp. 498–516, 1973.

[36] L. Davis, Ed., *Handbook of, Genetic Algorithms*, Von Hostrand Reinhold, 1991.

[37] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," *Choice Reviews Online*, vol. 27, no. 02, pp. 27-0936–27-0936, 1989.

[38] D. Whitely, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in *Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaugmann*, pp. 133–140, 1989.

[39] F. Glover, "A template for scatter search and path relinking," in *Artificial Evolution*, vol. 1363 of *Lecture Notes in Computer Science*, pp. 1–51, Springer, Berlin, Germany, 1998.

[40] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[41] "Tabu search - part 1," *ORSA Journal on Computing*, vol. 4, pp. 190–206, 1989.

[42] "Tabu search - part 2," *ORSA Journal on Computing*, vol. 2, pp. 4–32, 1989.

[43] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization, Vol. 3*, pp. 621–757, Kluwer Acad. Publ., Boston, MA, 1998.

[44] G. Stecco, J.-F. Cordeau, and E. Moretti, "A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine," *Journal of Scheduling*, vol. 12, no. 1, pp. 3–16, 2009.

[45] É. Taillard, "Parallel iterative search methods for vehicle routing problems," *Networks*, vol. 23, no. 8, pp. 661–673, 1993.