algorithms

MDPI

Article

Dynamic Shortest Paths Methods for the Time-Dependent TSP

Christoph Hansknecht [1,*], Imke Joormann [2,3] and Sebastian Stiller [1]

1 Institute for Mathematical Optimization, Technische Universität Braunschweig, Universitätsplatz 2, 38106 Braunschweig, Germany; sebastian.stiller@tu-braunschweig.de
2 Cluster of Excellence SE^{2}A—Sustainable and Energy-Efficient Aviation, Technische Universität Braunschweig, 38106 Braunschweig, Germany; i.joormann@tu-braunschweig.de
3 Institute of Automotive Management and Industrial Production, Technische Universität Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany
* Correspondence: c.hansknecht@tu-braunschweig.de

Abstract: The time-dependent traveling salesman problem (TDTSP) asks for a shortest Hamiltonian tour in a directed graph where (asymmetric) arc-costs depend on the time the arc is entered. With traffic data abundantly available, methods to optimize routes with respect to time-dependent travel times are widely desired. This holds in particular for the traveling salesman problem, which is a corner stone of logistic planning. In this paper, we devise column-generation-based IP methods to solve the TDTSP in full generality, both for arc- and path-based formulations. The algorithmic key is a time-dependent shortest path problem, which arises from the pricing problem of the column generation and is of independent interest—namely, to find paths in a time-expanded graph that are acyclic in the underlying (non-expanded) graph. As this problem is computationally too costly, we price over the set of paths that contain no cycles of length k. In addition, we devise—tailored for the TDTSP—several families of valid inequalities, primal heuristics, a propagation method, and a branching rule. Combining these with the time-dependent shortest path pricing we provide—to our knowledge—the first elaborate method to solve the TDTSP in general and with fully general time-dependence. We also provide for results on complexity and approximability of the TDTSP. In computational experiments on randomly generated instances, we are able to solve the large majority of small instances (20 nodes) to optimality, while closing about two thirds of the remaining gap of the large instances (40 nodes) after one hour of computation.

Keywords: integer programming; traveling salesman problem; shortest path problem

check for updates

Citation: Hansknecht, C.; Joormann, I.; Stiller, S. Dynamic Shortest Paths Methods for the Time-Dependent TSP. *Algorithms* **2021**, *14*, 21. https://doi.org/10.3390/a14010021

Received: 30 November 2020
Accepted: 8 January 2021
Published: 12 January 2021

1. Introduction

The traveling salesman problem (TSP) is among the best-studied combinatorial optimization problems (see [1,2] for summaries), which is in part due to a wide area of applications in logistics, manufacturing, telecommunications, and more. Considerable effort has been put into theoretical analysis of the problem, and implementations of Branch-and-Bound-based codes capable of computing optimal tours for instances consisting of several thousand nodes. Many heuristic solution techniques have been proposed as well, notably the Lin-Kernighan heuristic and its adaptations by Helsgaun [3], and numerous approaches based on genetic algorithms (see [4] for a comparative study). A prominent generalization of the TSP is the asymmetric traveling salesman problem (ATSP), which allows for different costs for the two directions in which a connection may be traversed.

There is, however, a drawback when it comes to real-world traffic problems, namely the fact that the TSP assumes that the cost required for utilizing an arc in the tour is constant, independent of the time at which the arc is traversed. This assumption does not generally hold in urban areas, where congestion, and, as a result, travel times fluctuate considerably. The problem of computing shortest paths in the presence of time-dependent travel times has been studied before, leading to significant algorithmic advances [5] rivaling those made for the shortest path problem itself.

As for the planning of time-dependent tours, several extensions of the ATSP have been examined in the literature (see below). The resulting formulations are, however, generally focused on the structure of the time-dependent cost functions rather than that of the underlying network itself. Conversely, we study the time-dependent TSP (TDTSP for short) focusing on paths through the network, leading to different Branch-Cut-and-Price formulations with time-dependent shortest path problems being solved during the column generation.

The TSP is very versatile in applications. For solving TSP, specialized IP methods have proven particularly fruitful. The importance of time-dependent cost functions is obvious at least for logistic applications. Thus, development of specialized IP methods for the TDTSP in full generality, with fully general time-dependence of the arc lengths, is of fundamental importance. To this end, we develop both arc- and path-based formulations for the TDTSP. Both formulations naturally lead to a high number of variables, naturally lending themselves to be solved by a column generation approach. It turns out that a pivotal step in this approach is the pricing problem, which yields a time-dependent shortest path problem of independent interest. This problem is to find and price paths in the time expansion of a graph G, which are acyclic when projected in the underlying graph G.

1.1. Preliminaries

The TDTSP is a generalization of the ATSP to the case of time-dependent cost functions. Specifically, we let $D = (V, A)$ be a directed graph with $n\colon = |V| \geq 3$ vertices, and $\theta^{\max} \in \mathbb{N}$ a fixed time horizon for $\Theta\colon = \{0, \dots, \theta^{\max}\}$. The travel time for an arc $a \in A$ is given by a function $c_a\colon \Theta \to \mathbb{N}$. For each sequence of arcs $(a_1, \dots, a_k)$ with $a_k = (u_k, v_k)$ and $v_k = u_{k+1}$, we can recursively define an arrival time

$$\theta^{\text{arr}}(a_1, \dots, a_k) := \begin{cases} c_{u_1,v_1}(0), & \text{if } k = 1, \text{ and} \\ \theta^{\text{arr}}(a_1, \dots, a_{k-1}) + c_{u_k,v_k}(\theta^{\text{arr}}(a_1, \dots, a_{k-1})) & \text{otherwise.} \end{cases}$$

A feasible solution of the TDTSP is a tour beginning at a source vertex $s \in V$, visiting every other vertex exactly once, which then returns to s. The TDTSP asks for a feasible solution $T = (a_1, \dots, a_n)$ minimizing the arrival time $\theta^{\text{arr}}(a_1, \dots, a_n)$ back at s, which we will also denote by $c(T)$. The specification of a source vertex is necessary for the TDTSP, as opposed to the ATSP, due to the time-dependency of the travel times c. Several special properties of travel time functions play an important role in time-dependent versions of combinatorial problems: The first-in-first-out (FIFO) property stipulates that the traveler who first enters an arc is also the first to leave it again. Formally, it must hold for each $a \in A$ that

$$\theta + c_a(\theta) \leq \theta' + c_a(\theta') \quad \forall\, \theta, \theta' \in \Theta,\, \theta \leq \theta'.$$

Shortest paths with respect to time-dependent costs can be computed efficiently using a variant of Dijkstra's algorithm if the cost functions satisfy the FIFO property [6,7].

Secondly, several well-known results (e.g., [8,9]) state that the symmetric version of the TSP can be approximated in the case of metric cost coefficients, that is, cost coefficients satisfying the triangle inequality. The definition of the triangle inequality can be easily generalized to the time-dependent case—a set of travel time functions satisfies the time-dependent triangle inequality if it holds for each $u, v, w \in V$ with $\theta, \theta + c_{uv}(\theta) \in \Theta$ that

$$\theta + c_{uw}(\theta) \leq \theta + c_{uv}(\theta) + c_{vw}(\theta + c_{uv}(\theta)). \tag{1}$$

1.2. Related Work

Several generalizations of the ATSP have been considered in the literature, such as the TSP with time windows [10,11], or the class of vehicle routing problems (VRPs) [12]. An interesting novel approach to the TSP and its variant is related to path-based formulations—the flow conservation constraints of the TSP ensure that every solution corresponds to a set of cycles in the underlying graph, making it possible to reformulate the problem in terms of path variables and solving it using a Branch-Cut-and-Price framework [13]. The same holds for the VRP [14] and other TSP-related problems [15]. By itself, this reformulation is not stronger than the original formulation due to Dantzig et al. [16]. It is, however, possible to restrict the set of path variables in order to exclude paths which are not tours. Of course, any such modification alters the pricing problem, generally having an adverse effect on its computational tractability, requiring a balance between the quality of the relaxation and the computational effort required to solve the pricing problem. Promising approaches include the generation of k-cycle-free [17], as well as so-called ng paths [18].

An early extension of the TSP to incorporate time-dependency, due to Picard and Queyranne [19], generalizes the travel time of an arc $(i, j) \in A$ to be sequence-dependent, i.e., a function $c_{ij}(k)$ $(k = 1, \ldots, n)$. This sequence-dependent variant of the TSP, which we refer to as the SDTSP, has since been studied both theoretically [15] and experimentally [19,20]. Notably, there is a correspondence between this sequence-dependent generalization of the TSP and the Minimum Latency Problem (MLP), which asks for a tour minimizing the sum of waiting times of customers. The correspondence has inspired some mixed-integer programming (MIP) formulations [21] for the MLP. The SDTSP is also closely related to identical machine scheduling, in particular $P||\sum w_j T_j$ (see [22]).

Some attempts have been made [23,24] to solve the TDTSP itself using Branch-and-Cut algorithms, with the focus of exploiting a special structure of the time-dependent cost functions. Specifically, the travel-time model introduced in [25] assumes that travel times are determined by a piecewise constant travel speed function, leading to a three-index formulation where the total number of variables depends on the complexity of the travel speed functions in terms of the number of their break points. While the approach seems successful for highly structured travel speed functions, the computational tractability apparently degrades for more irregular instances (see, for instance, the computational results in [23]). Another approach, going by the name of Dynamic Discretization Discovery, has been proposed [26,27] specifically for the TDTSP with time windows, which relies on iteratively refining a discretization of the time horizon according to optimal solutions of coarser discretizations determined during previous stages. The approach essentially substitutes one large integer program with several well-chosen smaller ones, exploiting the combination of time-dependent travel times and time windows. Conversely, we pursue an approach that does not rely on any special condition for the travel times.

A preliminary version of this paper [28] explored the (ultimately unsuccessful) usage of machine-learning techniques in order to solve the TDTSP.

1.3. Structure of This Paper

We begin by examining the complexity of the TDTSP in Section 2, establishing that the problem is hard to approximate even if an instance satisfies both the FIFO property and the time-dependent triangle inequality. On the other hand, we establish that an ATSP approximation algorithm can be used to approximate the TDTSP under some more restrictive circumstances.

We proceed in Section 3 by introducing several formulations for the TDTSP which do not need any specific structure of the underlying travel time functions. The formulation is based on a time expansion of the original graph, resulting in a potentially large number of variables and constraints, necessitating some form of column generation. We describe the specific pricing problem, which corresponds to computing shortest time-dependent paths, that is, shortest paths in a time-expanded network. We augment the approach by computing time-dependent k-cycle-free paths, using a dual stabilization technique in order to decrease the number of required pricing iterations. We add several valid inequalities, a propagation method, a custom branching rule, and primal heuristics in order to improve the solution process.

We demonstrate the effectiveness of our approach in Section 4 based on a computational experiment on several instances of differing sizes, providing a conclusion in Section 5.

2. Complexity

As a generalization of the TSP, the TDTSP is $\mathcal{NP}$-hard itself. Recall that while there exists no α-approximation for any $\alpha > 1$ for the general TSP, the metric TSP can be approximated [29] (p. 557). This does not hold for the metric TDTSP:

Theorem 1. *There is no α-approximation algorithm for any $\alpha > 1$ for the TDTSP unless $\mathcal{P} = \mathcal{NP}$, even if the FIFO property and the time-dependent triangle inequality are satisfied.*

Proof. Suppose there exists an α-approximation algorithm ALG for some $\alpha \geq 1$. We show that algorithm ALG could be used to solve the HAMILTONIAN CYCLE problem on an undirected graph $G = (V, E)$. To this end, let $D = (V, A)$ be the bidirected complete graph with costs

$$d_{uv} := \begin{cases} 1 & \text{if } \{u, v\} \in E \text{, and} \\ 2 & \text{otherwise.} \end{cases}$$

Note that G is Hamiltonian if, and only if D contains a tour with costs of exactly n with respect to the cost function d. Let $M := \lceil \alpha n \rceil + 1$, $s \in V$ be an arbitrary source vertex, and $\theta^{\max} := n \cdot M$ be a time horizon. For $\theta \in \Theta$ and $a \in A$, let

$$c_{uv}(\theta) := \begin{cases} M & \text{if } (v, \theta, d_{uv}) = (s, n-1, 2) \text{ or } \theta > n \text{, and} \\ d_{uv} & \text{otherwise} \end{cases} \tag{2}$$

be a set of time-dependent cost functions. These cost functions satisfy both the FIFO property and the time-dependent triangle inequality (1). We distinguish two cases with respect to the TDTSP instance $(D, c, s, \theta^{\max})$:

- G has a Hamilton cycle, and therefore, a tour $T = (a_1, \ldots, a_n)$ with costs of n exists in D: In this case, the values $d(a_i)$ for $i = 1, \ldots, n$ are all equal to one, implying that the optimal solution to the TDTSP instance has an arrival time of n. ALG yields a tour T_{apx} with an arrival time of, at most, $\alpha n < M$. Consequently, the first case of (2) is never attained, implying that each arc has a travel time of one, and ALG correctly identifies a Hamiltonian cycle in G.
- G does not possess a Hamiltonian cycle, and every tour $T = (a_1, \ldots, a_n = (u, s))$ has a cost of at least $n + 1$ with respect to d: As a result, there must be one arc $a_k \in T$ having a travel time of at least two. If $k < n$, then T arrives at u at time $\theta \geq n$, and the travel time of a_n, and consequently the arrival time of T is at least M. If, on the other hand, $k = n$, then T arrives at u at time $n - 1$ and arc a_n has a travel time of more than one, which is only possible if $d_{uv} = 2$, yielding a travel time of M for a_n. In any case, the arrival time of T is at least $M > \alpha n$.

Based on this distinction, G has a Hamilton cycle if, and only if ALG produces a tour with an arrival time of n. $\square$

Remark 1 (Dynamic Programming). *It is well-known [30] that the (asymmetric) TSP can be solved by using a dynamic programming approach. Let $C(S, v)$ be the smallest cost of an (s, v)-path consisting of the vertices $S \subseteq V$ with $s, v \in S$, and $s \neq v$. Then, $C(S, v)$ satisfies the following recursive relationship:*

$$\begin{aligned} C(\{s, v\}, v) &= c_{sv} && \forall\, v \in V, v \neq s \\ C(S, v) &= \min_{\substack{u \in S \\ u \neq s, v}} C(S \setminus \{v\}, u) + c_{uv} && \forall\, S \subseteq V, |S| \geq 3, v \in S. \end{aligned}$$

The cost of an optimal tour is then given by $\min_{v \neq s} C(V, v) + c(v, s)$, and can be determined by computing the values $C(S, v)$ in increasing order of $|S|$, yielding an $\mathcal{O}(2^n \cdot n^2)$ algorithm for the ATSP.

In order to adapt the approach to the TDTSP, we can define $C(S, v)$ as the minimum arrival time of any (s, v)-path departing from s at $\theta = 0$, consisting of the vertices $S \subseteq V$ with $s, v \in S$ and $s \neq v$. If the TDTSP instance in question satisfies the FIFO property, $C(S, v)$ satisfies a similar relationship:

$$\begin{aligned} C(\{s, v\}, v) &= c_{sv}(\theta = 0) && \forall v \in V, v \neq s \\ C(S, v) &= \min_{\substack{u \in S \\ u \neq s, v}} C(S \setminus \{v\}, u) + c_{uv}(\theta = C(S \setminus \{v\}, u)) && \forall S \subseteq V, |S| \geq 3, v \in S. \end{aligned}$$

These relations yield an algorithm for the TDTSP with the same running time as its counterpart for the ATSP. Note that the FIFO property is key here, since it ensures that only the shortest path for fixed (S, v) needs to be considered for subsequent computations.

Approximation for Special Cases

Let $\check{c} : A \to \mathbb{N}$ be the static underestimator of the time-dependent cost function c, defined as

$$\check{c}_a := \min_{\theta \in \Theta} c_a(\theta) \quad \forall a \in A.$$

If the time-dependent cost functions are of low variance, the underestimator yields TDTSP-approximations based on an underlying ATSP:

Theorem 2. *Let $\lambda \geq 1$ such that it holds for all $a \in A$, $\theta \in \{0, \dots, T\}$ that*

$$c_a(\theta) \leq \lambda \cdot \check{c}_a.$$

Then, any α-approximation of the ATSP yields an $(\alpha\lambda)$-approximation of the TDTSP.

Proof. Let T_{opt}, T_{apx} be the optimal and α-approximate tour with respect to the costs $\check{c}$ and $T_{\text{opt}}^{\mathcal{T}}$ be the optimal TDTSP tour. We have that

$$c(T_{\text{apx}}) \leq \lambda \cdot \check{c}(T_{\text{apx}}) \leq (\alpha\lambda) \cdot \check{c}(T_{\text{opt}}) \leq (\alpha\lambda) \cdot \check{c}(T_{\text{opt}}^{\mathcal{T}}) = (\alpha\lambda) \cdot c(T_{\text{opt}}^{\mathcal{T}}). \quad \square$$

Since the ATSP is inapproximable in general, further assumptions, such as a metric lower bound $\check{c}_{uv}$, are necessary to obtain any approximation results. What is more, we show in Appendix A that while it is possible to generalize the well-known one-tree relaxation [31] from the ATSP to the TDTSP, even the computation of the corresponding lower bound is an $\mathcal{NP}$-hard problem.

3. A Branch-and-Price Algorithm

Our aim in the next section is to provide a state-of-the-art mixed integer programming (MIP)-based algorithm to solve the TDTSP in full generality. We begin by formally defining time-expanded graphs and establishing both an arc-based and a path-based formulation, since it is not immediately clear which approach will work better. Both formulations consist of large numbers of variables, necessitating column generation approaches. Since feasible tours correspond to acyclic paths through the time-expanded graph, the key to solving the TDTSP efficiently is: *How can we find a shortest path through the time-expanded graph which is acyclic on the original graph?*

Optimizing over the set of acyclic paths, which we denote by $\mathcal{P}^*$, corresponds to solving the TDTSP itself. The set $\mathcal{P}$ of all paths through the time-expanded graph is much easier to handle, leading to a very fast pricing algorithm at the cost of a substantially weaker relaxation. To balance the computational effort of the pricing step and the strength of the relaxation, we generate k-cycle-free paths, that is, paths containing no cycles of size $\leq k$, where larger values of k increase the computational effort while improving the relaxation.

After discussing the subject of column generation, we strengthen both formulations based on valid inequalities. To this end, we review several classes of valid inequalities for the SDTSP and other related problems, adapting the class and their respective separation algorithms to the TDTSP. Lastly, we make several improvements relating to well-known MIP techniques, such as adding a branching rule and several primal heuristics.

Definition 1. *The set of reachable points in time, $\mathcal{T} : V \to 2^{\Theta}$ is defined as*

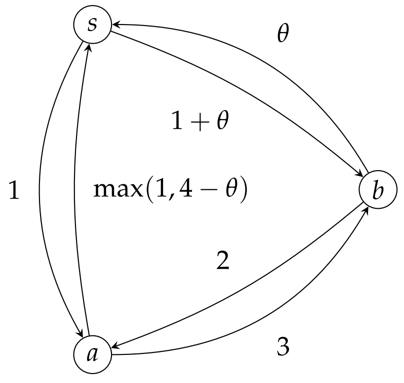
$$\mathcal{T}(v) := \{\theta \in \Theta \mid \exists(a_1, \ldots, a_k), a_1 = (s, v_1), a_k = (u_k, v), \theta^{\mathrm{arr}}(a_1, \ldots, a_k) = \theta\}.$$

The time-expanded graph $D^{\mathcal{T}} = (V^{\mathcal{T}}, A^{\mathcal{T}})$ has vertices $V^{\mathcal{T}} := \{v_\theta \mid v \in V, \theta \in \mathcal{T}(v)\}$ and arcs

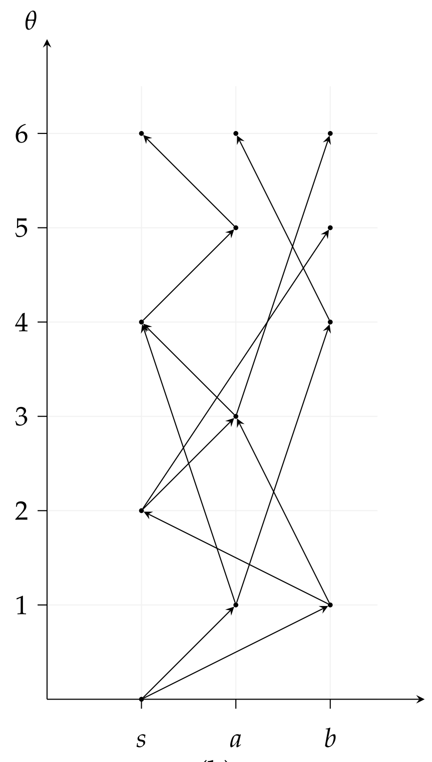
$$A^{\mathcal{T}} := \{(u_\theta, v_{\theta'}) \mid u_\theta, v_{\theta'} \in V^{\mathcal{T}}, \theta' = \theta + c_{uv}(\theta)\}.$$

We denote an arc $(u_\theta, v_{\theta'})$ by (u, v, θ) and assume from now on that $c_{uv}(\theta) > 0$ for all $(u, v) \in A, \theta \in \mathcal{T}(v)$. It follows that $D^{\mathcal{T}}$ is acyclic.

Example 1. *Figure 1 shows a directed graph with travel times for each arc and its time expansion. Any tour on D can be embedded into $D^{\mathcal{T}}$ as an (s_0, s_θ)-path.*



(**a**)



(**b**)

Figure 1. A directed graph D and its time-expansion $D^{\mathcal{T}}$. (**a**) The directed graph D; (**b**) The time-expansion $D^{\mathcal{T}}$ of D with time horizon $\theta^{\max} = 6$.

3.1. An Arc-Based Formulation

We formulate the TDTSP based on binary variables for each arc of $D^{\mathcal{T}}$ similar to the SDTSP formulation in [19]. The formulation has two kinds of constraints: A number of covering constraints ensures that each vertex in V has exactly one outgoing arc in $D^{\mathcal{T}}$, while flow conservation constraints guarantee that any feasible solution consists of a single (s_0, s_θ)-path:

$$\begin{aligned} \min_x \quad & \sum_{(u,v,\theta) \in A^{\mathcal{T}}} c_{uv}(\theta) \cdot x_{uv,\theta} \\ \text{s.t.} \quad & \sum_{\theta \in \mathcal{T}(v)} x(\delta^+(v_\theta)) = 1 && \forall\, v \in V \\ & x(\delta^+(v_\theta)) - \delta^-(v_\theta) = 0 && \forall\, v \neq s, \theta \in \mathcal{T}(v) \\ & x_a \in \{0,1\} && \forall\, a \in A^{\mathcal{T}}. \end{aligned} \tag{TDTSP-A}$$

Remark 2. *By virtue of the flow conservation constraints in (TDTSP-A), any solution of the program or its LP-relaxation can be decomposed into a set of paths leading from vertex s_0 to vertices s_θ for some values $\theta \subseteq \mathcal{T}(s)$. As a result, there exist several equivalent linear objectives, for example the arrival time objective, given by*

$$\sum_{\theta \in \mathcal{T}(s)} \sum_{(v,s,\theta') \in \delta^-(s_\theta)} \theta \cdot x_{vs,\theta}.$$

Relation to the Static ATSP

There is a strong relationship between the TDTSP and the ATSP. Let $y : A \to \mathbb{R}_{\geq 0}$ be the compound flow traversing an arc $(u,v) \in A$ with respect to a feasible solution $(x_a)_{a \in A^{\mathcal{T}}}$ of (TDTSP-A), given as

$$y_{uv} := \sum_{(u,v,\theta) \in A^{\mathcal{T}}} x_{uv,\theta}. \tag{3}$$

The covering constraints and the flow conservation yield the well-known two matching equations,

$$y(\delta^+(v)) = y(\delta^-(v)) = 1 \quad \forall\, v \in V.$$

The covering constraints and the integrality of the variables x imply that the values y are binary for any feasible solution x. However, a correct static ATSP formulation still requires subtour elimination constraints (SECs) of the form

$$y(\delta^+(S)) \geq 1 \quad \forall\, S \subset V,\ S \neq \emptyset, V.$$

Since $D^{\mathcal{T}}$ is acyclic, any solution of (TDTSP-A) is guaranteed to satisfy the additional SECs. Equivalently, flow augmentation techniques [32] for strengthening ATSP formulations are redundant for (TDTSP-A). Conversely, SECs are not necessarily satisfied by fractional solutions. Thus, (TDTSP-A) can be strengthened by separating SECs with respect to the underlying static ATSP (see Section 3.4).

Valid inequalities for the ATSP can be included in the TDTSP by first computing the compound flow y^* of a solution x^* of the LP-relaxation of (TDTSP-A), executing some separation algorithm yielding a valid inequality in the compound variables y, and finally expressing this inequality in terms of the original problem variables x.

Any feasible TDTSP solution is feasible for the underlying ATSP, and generic ATSP solutions do not necessarily produce feasible solutions of the TDTSP. Specifically, no tour $T = (a_1, \ldots, a_n)$ with $\theta^{\text{arr}}(T) > \theta^{\max}$ can be embedded into $D^{\mathcal{T}}$. The complete description of the TDTSP in terms of compound variables y can be obtained by adding forbidden path constraints of the form

$$\sum_{a \in P} y_a \leq k - 1 \quad \forall\, P = (a_1, \ldots, a_k) : \theta^{\text{arr}}(P) > \theta^{\max}.$$

As a result, facet-defining ATSP inequalities, while valid, are not necessarily facet-defining for the TDTSP.

3.2. A Path-Based Formulation

Any feasible solution of the TDTSP problem (TDTSP-A) can be decomposed into (s_0, s_θ)-paths for some values $\theta \subseteq \mathcal{T}(s)$. These paths correspond to cycles containing vertex s in D. We denote by $\mathcal{P}$ the set of all (s_0, s_θ)-paths not containing any vertex $s_{\theta'}$ with $\theta \neq \theta'$. For each path $P \in \mathcal{P}$ and each vertex $v \in V$, we let $\chi_{v,P}$ be the set of vertices in $V^{\mathcal{T}}$ that have an outgoing arc contained in P, and $\alpha_{v,P}$ its cardinality, that is,

$$\chi_{v,P} := \{v_\theta \in V^{\mathcal{T}} \mid (v, w, \theta) \in P\}, \text{ and } \alpha_{v,P} := |\chi_{v,P}|.$$

The problem can be reformulated in terms of path variables:

$$\begin{aligned} \min_x \ & \sum_{P \in \mathcal{P}} c_P \cdot x_P \\ \text{s.t.} \ & \sum_{P \in \mathcal{P}} \alpha_{v,P} \cdot x_P = 1 \quad && \forall\, v \in V \\ & x_P \in \{0, 1\} && \forall\, P \in \mathcal{P}. \end{aligned} \tag{TDTSP-P}$$

Any solution of this formulation consists of a single variable x_P set to 1 and all others set to 0, in which case P must be contained in $\mathcal{P}^*$, that is, corresponding to a tour. Any fractional solution consists of, at most, n different paths in $\mathcal{P}$ which need not be tours in D. The resulting system is small in terms of the number of constraints at the expense of the number of variables, necessitating a column generation [33] approach.

3.3. Column Generation

Let $(\lambda_v)_{v \in V}$ be the dual variables corresponding to the covering constraints in (TDTSP-P). The reduced cost of a variable x_P in the corresponding LP-relaxation is given as

$$\bar{c}_P := c_P - \sum_{v \in V} \alpha_{v,P} \cdot \lambda_v,$$

which can be rewritten as a function $A^{\mathcal{T}} \to \mathbb{R}$ via

$$\bar{c}_{uv,\theta} := c_{uv}(\theta) - \lambda_u.$$

The pricing problem therefore corresponds to a shortest path problem in $D^{\mathcal{T}}$, which can be solved in linear time since $D^{\mathcal{T}}$ is acyclic.

3.3.1. Lagrangean Pricing & Dual Stabilization

It is fairly straightforward to derive a Lagrangean relaxation of the LP-relaxation of (TDTSP-P), which is solved during the pricing problem. The constraint $\sum_{P \in \mathcal{P}} \alpha_{s,P} \cdot x_P = 1$ is equivalent to

$$\sum_{P \in \mathcal{P}} x_P = 1,$$

since every path in $\mathcal{P}$ leaves s_0 exactly once. By penalizing all covering constraints in the objective and removing all but this constraint, we obtain the Lagrangean relaxation

$$L_{\mathcal{P}}(\lambda) := \left(\sum_{v \in V} \lambda_v \right) + \begin{cases} \min\limits_x & \sum\limits_{P \in \mathcal{P}} \bar{c}_P \cdot x_P \\ \text{s.t.} & \sum\limits_{P \in \mathcal{P}} x_P \leq 1 \\ & x_P \geq 0 \quad \forall\, P \in \mathcal{P}. \end{cases}$$

If there exists a path with negative reduced costs with respect to λ, an optimal solution of $L_{\mathcal{P}}(\lambda)$ is obtained by setting $x^*_P = 1$, where $P \in \mathcal{P}$ has the minimum reduced cost. If there is no such path, we set $x^* \equiv 0$. In any case, we obtain a lower bound on the LP-relaxation of (TDTSP-P) during the pricing. It is easy to adapt this approach to (TDTSP-A), enabling us to use path-based pricing in this case as well.

We use the Lagrangean relaxation in order to perform a dual stabilization based on the weighted Dantzig–Wolfe decomposition method introduced in [22]. The weighted Dantzig–Wolfe decomposition method judges the quality of dual values according to the value of their Lagrangean relaxation. By maintaining a center of stability and only tentatively moving the center towards the current dual values, the technique can significantly decrease the time required to solve the LP-relaxations of the TDTSP formulations.

3.3.2. Pricing Acyclic Paths

Many paths which are generated in the pricing do not share much resemblance with tours in the underlying graph D: On the one hand, certain paths only contain few vertices and lead almost immediately back to s_θ. We will address this problem later using the propagation of lower bounds. On the other hand, paths frequently contain cycles with respect to D, that is, they contain two different copies v_θ, $v_{\theta'}$ of the same vertex $v \neq s$ with different values $\theta \neq \theta'$. Specifically, a path

$$P = (a_1 = (u_1, v_1, \theta_1), \ldots, a_k = (u_k, v_k, \theta_k))$$

in $D^{\mathcal{T}}$ forms a k-cycle if $u_1 = v_k$. A path of at least k arcs contains a k-cycle if any of its subpaths is a k-cycle. A path not containing a j-cycle for $j \leq k$ is called k-cycle-free. Naturally, a k-cycle-free path is also j-cycle-free for all $j < k$, and it is a tour if, and only if it is $(n-1)$-cycle-free.

In order to strengthen the formulations, we restrict the set of variables in the path-based formulation (TDTSP-P) to the subset $\mathcal{P}_k \subseteq \mathcal{P}$ of k-cycle-free paths, noting that

$$\mathcal{P} =: \mathcal{P}_1 \subseteq \mathcal{P}_2 \ldots \subseteq \mathcal{P}_{n-1} = \mathcal{P}^*.$$

Similarly, the sets $\mathcal{P}_k$ yield increasingly tighter Lagrangean relaxations, i.e., satisfying

$$L_{\mathcal{P}}(\lambda) \leq L_{\mathcal{P}_2}(\lambda) \leq \ldots \leq L_{\mathcal{P}_{n-1}}(\lambda).$$

The restriction to the set $\mathcal{P}_k$ requires us to adapt the pricing procedure to k-cycle-free paths. The k-cycle-free shortest problem has previously been examined [17], leading to a two-cycle-free labeling scheme with linear running time, as well as an algorithm computing k-cycle-free paths with a running time of $\mathcal{O}((k!)^2)$ in the parameter k. In order to improve the overall performance of the column generation, we use the weighted Dantzig–Wolfe decomposition described above based on the values $L_{\mathcal{P}_k}(\lambda)$.

Adapting the original formulation (TDTSP-A) is not as straightforward, since we have no control regarding the path decomposition producing the values $x_{uv,\theta}$. As a compromise, we propose to generate new columns according to the arcs of k-cycle-free paths. Since we cannot guarantee that the resulting flow can be decomposed into k-cycle-free paths, we cannot ensure that $L_{\mathcal{P}_k}(\lambda) \leq$ LP (TDTSP-A). Nonetheless, it still holds that $L_{\mathcal{P}_k}(\lambda) \leq$ (TDTSP-A), which is sufficient to ensure that, if no more k-cycle-free paths can be found, the restricted LP is a relaxation of (TDTSP-A). We are, however, unable to apply the weighted Dantzig–Wolfe decomposition to the arc-based formulation.

3.4. Valid Inequalities

We consider several classes of valid inequalities, some of which are well-known ATSP inequalities, whereas others are either adaptations of SDTSP inequalities or newly derived ones. We express the inequalities in terms of the arc variables x from (TDTSP-A), understanding that converting them to (TDTSP-P) is straightforward. The separation is performed according to a fractional solution x^* with compound values y^* given by

$$y^*_{uv} = \sum_{(u,v,\theta)\in A^{\mathcal{T}}} x^*_{uv,\theta} \quad \forall\,(u,v) \in A.$$

3.4.1. ATSP Inequalities

Apart from the subtour elimination constraints, probably the best-known family of facet-defining inequalities for the ATSP goes by the name of D_k^+-inequalities [2,34]. These inequalities are derived based on the compound variables y on a complete directed graph $D = (V, A)$. To simplify notation, for sets $S, T \subseteq V$ we let

$$\delta(u, T) := \{(u,v) \in A \mid v \in T\}.$$

The D_k^+-inequality for a sequence $(v_1, \ldots, v_k)$ of $2 \leq k < n$ distinct vertices is given by

$$\sum_{j=1}^{k-1} y_{v_j,v_{j+1}} + y_{v_k,v_1} + 2y(\delta(v_1, \{v_3, \ldots, v_k\}))$$
$$+ \sum_{j=4}^{k} y(\delta(v_j, \{v_3, \ldots, v_{j-1}\})) \leq k-1.$$

The separation of D_k^+-inequalities involves the enumeration of possible sequences in a Branch-and-Bound-like fashion. Nonetheless, the separation works well in practice, since many of the possible sequences can be pruned.

3.4.2. Incompatibilites

Incompatibilites between binary variables, collected in the so-called incompatibility graph, have proven to be highly useful in order to generate strong inequalities for mixed-integer programs in general (see [35]). For the ATSP, two arcs $(u,v) \neq (u',v')$ are incompatible if they share one or both of their endpoints, that is,

$$u = u' \quad \text{or} \quad v = v' \quad \text{or} \quad (u = v' \text{ and } u' = v).$$

The two common classes of inequalities derived from incompatibility graphs are clique and odd-cycle inequalities. Cliques in the incompatibility graph of the ATSP correspond to the sets $\delta^+(v)$ and $\delta^-(v)$. For the (TDTSP-A), clique inequalities are already implied by the constraints.

Odd-cycle inequalities for the ATSP are known as odd closed alternating trail inequalities [36], odd CATS for short. An odd CAT corresponds to a sequence $a_1, \ldots, a_{2k+1}$ of distinct arcs in A such that arcs a_i and a_{i+1} share either head or tail. The odd CAT inequality corresponding to these arcs is given by

$$\sum_{i=1}^{k} y_{a_i} \leq k.$$

Odd CAT inequalities for the ATSP can be separated heuristically by computing shortest paths in an auxiliary bipartite graph. For the TDTSP, these cuts correspond to odd cycles of cliques rather than odd cycles of simple incompatibilities.

3.4.3. Odd Path-Free Inequalities

Let $S \subseteq V \setminus \{s\}$ be a set of vertices of the underlying graph, $V^{\mathcal{T}}(S) := \{u_\theta \in V^{\mathcal{T}} \mid u \in S\}$ the corresponding vertices in $V^{\mathcal{T}}$, and $A^{\mathcal{T}}(S)$ the arcs of the subgraph of $D^{\mathcal{T}}$ induced by $V^{\mathcal{T}}(S)$. The SEC corresponding to S implies that the set $A^{\mathcal{T}}(S)$ can contain, at most, $|S| - 1$ arcs.

We obtain another class of inequalities by restricting ourselves to subsets with odd cardinality, i.e., $|S| = 2k + 1$. A subset $A' \subseteq A^{\mathcal{T}}$ is path-free with respect to S if it contains no nontrivial paths in $D^{\mathcal{T}}$, a nontrivial path consisting of more than one arc. If a set is path-free, than any arc in the intersection of A' with a tour through $D^{\mathcal{T}}$ connects two distinct vertices. As a result, the intersection can contain, at most, k arcs, yielding the following odd path-free inequality (see Figure 2):

$$\sum_{(u,v,\theta) \in A'} x_{uv,\theta} \leq k.$$

We separate these inequalities heuristically by first determining promising subsets S, and then computing the optimal set A' for each promising subset S. In order to determine a promising subset S, we note that the odd path-free inequality is strongest for small values of k. We therefore restrict ourselves to the case of $k = 1$, determining several promising sets based on the amount of induced flow $y^*(S)$. To avoid very similar inequalities, we only consider the best promising 3-set containing each vertex $v \in D \setminus \{s\}$. For each candidate set S, we compute an optimal set A' using an integer program.

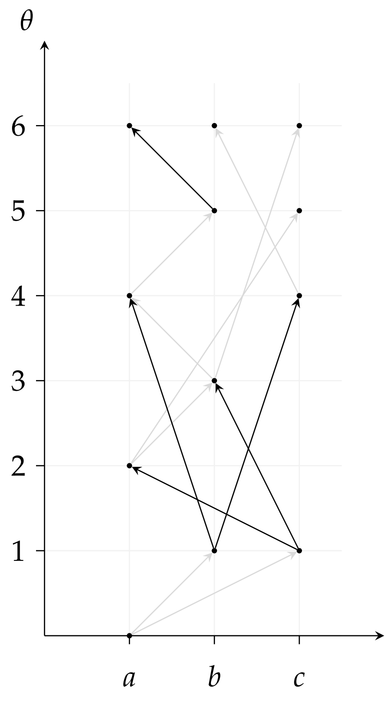


Figure 2. A path-free set of arcs on three vertices.

3.4.4. Lifted Subtour Elimination Inequalities

Subtour elimination constraints can be used to strengthen formulation (TDTSP-A). They can be separated in polynomial time by solving a series of flow problems on the underlying graph D. To further strengthen SECs for the SDTSP, it was observed in [15] that any tour has to leave S sufficiently early in order to be able to reach the vertices in $V \setminus S$ and return to s within the time horizon. We proceed to adapt the approach to the TDTSP. Let $\hat{\theta}$ be such that

$$\hat{\theta} \geq \max\{\theta \mid \text{there exists a tour } T \text{ leaving } S \text{ for the first time at } \theta\}.$$

Then, the following lifted subtour elimination constraint (LSEC) is valid for (TDTSP-A):

$$\sum_{\substack{(u,v,\theta) \in \delta^+(A^{\mathcal{T}}(S)) \\ \theta \leq \hat{\theta}}} x_{uv,\theta} \geq 1.$$

The value of $\hat{\theta}$ is maximized for a tour which first serves S, then $V \setminus S$ and returns to s immediately afterwards. Computing the optimum value of $\hat{\theta}$ is intractable in practice, as it would involve the solution of a TDTSP on S itself.

Instead of maximizing the time spent in S, we therefore minimize the time spent in $V \setminus S$. Since we do not know the optimum value of $\hat{\theta}$, we do not know the values of the cost function c. Hence, we relax the problem by replacing the time-dependent cost functions $c_{uv}(\theta)$ by its static underestimator $\check{c}_{uv}$. Minimizing the length of a static tour leaving s, traversing $V \setminus S$, and finally returning to s is easily formulated as an ATSP, yielding an optimal value of $\check{\theta}$. The latest departure time $\hat{\theta}$ can then be determined as $\theta^{\max} - \check{\theta}$.

This estimation of the optimal $\hat{\theta}$ can be rather rough if the cost functions c differ significantly from their underestimators $\check{c}$. As a result, the computational effort required to solve the ATSP does not seem merited. We choose to instead bound the static tour cost from below by computing an arborescence of minimum weight.

3.4.5. Cycle Inequalities

While it is possible to generate variables for the (TDTSP-A) according to k-cycle-free paths, solutions of the LP relaxation generally do not possess a k-cycle-free path decomposition. We adapt the cycle inequalities introduced in [15] for the SDTSP to the TDTSP, cutting off solutions based on the elimination of k-cycles. Consider a path

$$P = (a = (u, u_1, \theta), a_1 = (u_1, v_1, \theta_1), \ldots, a_k = (u_k, v_k, \theta_k))$$

in $D^{\mathcal{T}}$ satisfying the following properties:

1. The arcs $a_1, \ldots, a_k$ form a k-cycle C not containing s.
2. The arc a enters the cycle C, that is, $u \notin \{u_1, \ldots, u_k\}$.

Let T be a tour entering the cycle C via arc a. Since T is k-cycle-free, it can contain, at most, $j \leq k-1$ of the arcs in C, leaving the cycle via an arc (u_j, w, θ_j) with $w \neq u_{j+1}$. It also holds that w cannot coincide with u_ℓ for $\ell < j$ or u (otherwise T would contain a cycle of length $\leq j$). Based on these observations, the cycle inequality associated with P, defined as

$$x_{u_1 v_1, \theta_1} \leq \sum_{j=1}^{k-1} \sum_{\substack{(u_j, w, \theta_j) \in A^{\mathcal{T}} \\ w \notin \{u, u_1, \ldots, u_{j+1}\}}} x_{u_j w, \theta_{j+1}},$$

is valid for (TDTSP-A). We separate these inequalities using a simple heuristic: Based on x^* we compute a decomposition into paths $P_1, \ldots, P_k \subseteq \mathcal{P}$. For each path P_i, we determine whether it contains a k-cycle, adding the corresponding inequality if necessary.

Note that cycle inequalities were reexamined in [37], leading to the stronger class of Time-Dependent Cycle Inequalities (TDCIs) for the SDTSP. The techniques required to derive these inequalities are, however, specific to the SDTSP and do not generalize to the TDTSP.

3.4.6. Unitary AFCs

Unitary admissible flow constraints (AFCs) were introduced in [15] for the SDTSP. In terms of the TDTSP, they can be derived as follows:

Summing up the flow conservation constraints of (TDTSP-A) over a set $S \subseteq V^{\mathcal{T}}$ not containing any vertices s_θ yields the equation $x(\delta^-(S)) = x(\delta^+(S))$. Based on an incoming arc $(u, v, \theta) \in \delta^-(S)$, it can be relaxed to $x_{uv,\theta} \leq x(\delta^+(S))$. This constraint by itself is obviously redundant. However, any tour T that enters S via (u, v, θ) has to leave S using some arc $(u', v', \theta') \in \delta^+(S)$ such that $v' \neq u, v$, yielding the unitary AFC inequality

$$x_{uv,\theta} \leq \sum_{\substack{(u', v', \theta') \in \delta^+(S) \\ v' \neq u, v}} x_{u'v',\theta'}.$$

To separate these inequalities, we first observe that stronger unitary AFC inequalities correspond to smaller cuts, that is, a set S' with $(u, v, \theta) \in \delta^-(S')$ and $\delta^+(S') \subseteq \delta^+(S)$ produces a stronger inequality than S. As a result, any vertex in S except for $v_{\theta+c_{uv}(\theta)}$ can be removed, provided that it does not contain any arc entering from another vertex in S. We can therefore assume that S only contains vertices reachable from $v_{\theta+c_{uv}(\theta)}$. Similarly, we can also assume that S contains no copies of u and v, other than $v_{\theta+c_{uv}(\theta)}$ itself. Hence, the separation of unitary AFC inequalities can be conducted by solving a min-cut problem for each arc (u, v, θ) with $x^*_{uv,\theta} > 0$, with capacities based on the values of x^*.

3.5. Additional Techniques

The addition of cutting planes already significantly strengthens the TDTSP formulations. There are, however, several other techniques which can be used to speed up the computation of the optimal tour in a Branch-and-Bound framework:

3.5.1. Propagation

During the solution process, we have a dual bound $\underline{\theta}$ based on the value of the LP-relaxation of the current Branch-and-Bound node and a primal bound $\overline{\theta}$ based on the best-known integral solution (we let $\underline{\theta} := -\infty$ and $\overline{\theta} := +\infty$ if the bounds are not available). The LP-relaxation frequently contains (s_0, s_θ)-paths containing only a few arcs each, leading back to s at $\theta < \underline{\theta}$. Similarly, there are long paths which cannot be part of an optimal solution. Formally, variable $x_{vw,\theta}$ can be fixed to zero if

$$\theta > \overline{\theta} \quad \text{or} \quad (\theta + c_{vw}(\theta) < \underline{\theta} \text{ and } w = s).$$

We propagate any improvement in the bounds $\underline{\theta}$ and $\overline{\theta}$ by fixing the appropriate variables. We also include the propagation into the pricing problem.

3.5.2. Compound Branching

Branching on variables $x_{uv,\theta}$ likely yields a highly unbalanced Branch-and-Bound tree, since fixing a variable $x_{uv,\theta}$ to one is a strong restriction, while fixing it to zero is a very weak one. We instead add the binary compound flow variables y and coupling constraints (3) to the formulations and assign branching priorities in order to force the solver to branch on compound flow variables whenever possible, leading to a more balanced tree. Since any binary solution y^* corresponds to a tour, it is never necessary to branch on the variables x. Since the number of compound variables is generally much lower than the number of arcs in $A^{\mathcal{T}}$, the addition of the compound variables is unlikely to significantly affect computational performance.

3.5.3. Primal Heuristics

We use an incremental construction heuristic to obtain a path P starting at s_0. At each turn, we append an arc leading to vertices whose counterparts in D are still unexplored by P, finishing when the path forms a tour. The selection of the arcs is based on the current fractional solution (x^*, y^*). Arcs that are fixed to zero are always disregarded. If there are multiple possible arcs to be added, we score each arc (u, v, θ), and then choose one with probability proportional to the score. We use three different scoring functions:

– The inverse of the travel time $c_{uv}(\theta)$ (breaking ties arbitrarily);
– The variable value of $x^*_{uv,\theta}$ using travel times to break ties or
– The compound value y^*_{uv} using the same tie-breaking rule.

Since this construction is computationally inexpensive, we can increase the chance of finding an improved tour by using all scoring functions and performing several iterations with different random seeds.

4. Computational Experiments

We proceed to report on the empirical findings on a set of artificially generated TDTSP instances. All experiments were carried out based on an implementation in the C++ programming language that compiled with full optimization. We used version 7.0.1 of the SCIP [38] optimization suite and version 9.0.0 of GUROBI [39] as an underlying LP solver. We ran the experiments on an AMD Epyc 7742 processor clocked at up to 3.40 GHz, and imposed a time limit of 3600 s for all computations. Each individual computation was carried out by a single-threaded process.

4.1. Instances

We generate several problem instances, each given by a complete directed graph and cost functions associated with its arcs. We embed the vertices of the graph into $\{0, \dots, 100\}^2$ and introduce (symmetric) costs c_a using rounded-down Euclidean distances between the points of the embedding. Based on the costs c_a, we generate a piecewise linear function $f_a(\theta)\colon \mathbb{N} \to \mathbb{Z}$ with $M \in N$ breaking points $\theta_1 < \theta_2 < \dots < \theta_M$ within Θ:

1. We let $f_a(0)\colon = 0$ and fix the slope at zero to $+1$.
2. The slope alternates between $+1$ and -1 with break points at θ_i for $i = 1, \dots, M$.

Based on f_a, the time-dependent cost function $c_a\colon \{0, \dots, \theta^{\max}\} \to \mathbb{N}$ is given as

$$c_a(\theta) := c_a + \max(\min(f_a(\theta), \lambda \cdot c_a), 0),$$

where the parameter $\lambda > 1$ controls which multiple of $c_a(\theta)/c_a$ can be attained (see Figure 3 for an example). Throughout the experiments we set $\lambda\colon = 3$ and distribute a $M\colon = 360$ break point over a set of 3600 points in time.

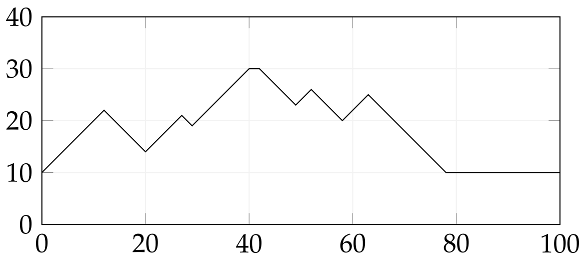


Figure 3. A sample plot of the travel time function for costs of 10, a time horizon of $\theta^{\max} = 100$ and $M = 10$ break points. The cost is constrained by a factor of $\lambda = 3$.

By construction, the function f satisfies the FIFO property independently of the choice of the time steps θ. We then use (time-dependent) shortest-path distances with respect to $c(\cdot)$ instead of c directly, in order to ensure that the (time-dependent) triangle inequality is satisfied as well.

We generate 50 small instances consisting of 20 vertices each, as well as 20 large instances consisting of 40 vertices based on different random seeds. For each instance, we compute an optimal tour T_0 with respect to the time-independent costs. The tour T_0 serves both as an initial feasible solution and as a means to determine a suitable time horizon $\theta^{\max}$ to construct a time-expanded graph $D^{\mathcal{T}}$.

There is a significant increase in the number of arcs during the time-expansion—while the original number of 20 vertices of the small instances leads to 380 arcs in the underlying graph, the number of arcs in the time-expanded graphs $D^{\mathcal{T}}$ ranges from about 80,000 to over 130,000. For the large instances, the number of arcs in $D^{\mathcal{T}}$ lies between 800,000 and 1,000,000.

Remark 3. *By means of Theorem 2, the choice of objective enabled us to compute a lower bound on the optimal objective values of our TDTSP instances. These bounds are oftentimes better than* *the objective values of the LP-relaxations of the root nodes. However, in order to evaluate our formulations, we do not use them during our computations.*

4.2. Computational Results

4.2.1. Comparisons between Formulations

We begin by comparing the performance of the different formulations, namely, the arc-based formulation (TDTSP-A), both with and without path-based pricing, and its path-based counterpart (TDTSP-P). The increased size of the instances (see above) has immediate consequences regarding the computational performance (see Table 1 for the solution statistics for the large instances): Formulation (TDTSP-A) spends an average of about 13 min solving the root relaxations of the problem instances. Enabling column generation decreases this time to about 55 s, whereas the path-based formulation (TDTSP-P) averages at 17 s. While the difference in running times is less pronounced for the small instances, it is clear that column generation is necessary in order to solve larger TDTSP instances.

Despite the fact that the path-based formulation solves the initial relaxation the fastest, it explores only seven nodes on average during its execution, compared to 29 nodes explored by the arc-based formulation. Consequently, the average remaining gaps are 0.57 and 0.55, respectively. The gap here is defined as $(p-d)/d$, with p being the objective function value of the best-known feasible solution, and d the best-known lower bound.

4.2.2. Performance of Different Pricers

We proceed to evaluate the pricing methods introduced in Section 3.3. As a first step, we study the difference between pricing individual arcs and entire paths with respect to the arc-based formulation (TDTSP-A). While both approaches yield the same gaps at the root node, path-based pricing is substantially faster than pricing arcs, while yielding relaxations with substantially fewer variables.

We then examine the effect of pricing k-cycle-free paths, based on the algorithms introduced in [17], the experimental results being depicted in Table 2. Clearly, increasing the value of k increases the computational effort. On the other hand, larger values of k yield tighter relaxations. To obtain a realistic picture of the running times, we solved the LP-relaxation of (TDTSP-A) and measured the required running time, thereby accounting for deviations in the size of the programs arising from the use of different pricing techniques. After solving the relaxations, we compute the relative gaps between the different dual bounds and the primal bound obtained from the original path-based pricing method, thereby ignoring any distortions due to improved primal bounds.

We find that, independently of the formulation, pricing two-cycle-free paths incurs a negligible computational overhead of less than a minute, while closing almost 30% of the relative gap. Increasing k beyond two immediately results in a significant increase to the computational effort, requiring almost 40 min for large instances for the arc-based formulation. What is more, the effect of pricing k-cycle-free paths on the relative gap becomes less pronounced for larger values of k.

Regarding the formulations, the pricing procedure becomes significantly more difficult for the path-based formulation (TDTSP-P). While the effect is mitigated when we add dual stabilization, the difference remains significant. A likely explanation is that, when solving the arc-based formulation, the LP solver can compose the arcs of previously added paths in order to obtain new paths, which may not have been explicitly added before. As a result, the solver is quickly able to compute near-optimal dual values, thereby guiding the column generation. What is more, the difference between the formulations with respect to the relative gaps is very slight and does not merit the computational effort required.

Based on these observations we proposed to use the arc-based formulation while pricing two-cycle-free paths in order to achieve a reasonable trade-off between computational overhead and decrease in relative gap.

Table 1. Statistics with respect to the large instances for miscellaneous formulations, consisting of **p**rimal, **d**ual values, remaining **g**ap, the number of examined **n**odes, and the **t**ime required to solve the root node.

Instance	(TDTSP-A)					(TDTSP-A) **with Pricing**					(TDTSP-P) **with Pricing**				
	p	**d**	**g**	**n**	t_{root}	**p**	**d**	**g**	**n**	t_{root}	**p**	**d**	**g**	**n**	t_{root}
1	759	439.85	0.73	1	3 384.62	759	447.77	0.70	33	32.90	759	443.94	0.71	7	13.96
2	952	566.09	0.68	1	3 426.43	952	579.02	0.64	25	54.04	952	565.64	0.68	3	20.50
3	733	471.61	0.55	1	–	733	471.23	0.56	28	36.62	733	470.51	0.56	5	11.41
4	962	492.88	0.95	1	3 438.55	962	499.10	0.93	24	75.91	962	490.63	0.96	4	19.06
5	764	508.43	0.50	1	–	764	501.86	0.52	35	40.97	764	494.42	0.55	4	16.06
6	818	526.35	0.55	1	3 495.04	818	527.73	0.55	21	32.48	818	525.38	0.56	4	13.49
7	745	518.59	0.44	1	3 468.89	745	522.99	0.42	29	49.53	745	522.35	0.43	8	11.66
8	772	492.36	0.57	1	–	772	483.32	0.60	23	41.59	772	476.44	0.62	4	20.80
9	786	576.16	0.36	1	–	786	577.25	0.36	32	39.56	786	577.23	0.36	9	12.81
10	897	535.59	0.67	1	3 392.87	897	543.79	0.65	24	42.06	897	532.30	0.69	5	15.16
11	794	497.36	0.60	1	3 450.85	794	496.43	0.60	26	64.05	794	495.94	0.60	8	17.17
12	854	509.18	0.68	1	3 411.36	854	518.79	0.65	21	49.47	854	513.38	0.66	4	13.15
13	870	553.34	0.57	1	–	870	553.83	0.57	22	36.36	870	551.60	0.58	4	16.52
14	771	520.03	0.48	1	3 460.53	771	523.41	0.47	32	44.75	771	517.18	0.49	11	10.82
15	721	543.23	0.33	1	–	721	548.23	0.32	35	29.64	721	544.67	0.32	15	9.84
16	767	535.25	0.43	1	3 505.49	767	545.93	0.40	25	44.69	767	545.51	0.41	7	12.48
17	751	459.39	0.63	1	3 256.30	751	459.52	0.63	32	37.10	751	457.83	0.64	5	12.65
18	799	528.77	0.51	1	3 405.66	799	545.27	0.47	26	67.33	799	531.38	0.50	8	17.77
19	800	503.82	0.59	1	3 460.39	800	512.01	0.56	32	66.72	800	505.73	0.58	5	20.77
20	822	566.01	0.45	1	3 310.60	822	584.36	0.41	34	77.51	822	564.98	0.45	4	16.15

Table 2. Running time and relative gap for different pricers.

		Running Time [s]		**Relative Gap**	
	k	**Small**	**Large**	**Small**	**Large**
Original		1.14	18.57	0.43	0.57
arc-based, k-cycle-free	2	1.28	23.39	0.32	0.45
	3	69.44	1960.60	0.28	0.40
	4	1706.55	–	0.26	–
path-based, k-cycle-free	2	7.38	38.19	0.31	0.43
	3	1203.36	–	0.26	–
path-based, stabilized, k-cycle-free	2	4.22	54.44	0.31	0.43
	3	585.13	–	0.26	–

4.2.3. Impact of Valid Inequalities

We evaluate the separators introduced in Section 3.4 using the same approach as above, comparing the running times and relative gaps for all separators (see Table 3). In terms of the relative gap, the subtour inequalities, as well as their lifted counterparts (see Section 3.4.4) perform best, closing about 25% of the gap of the original formulation. The computational overhead is relatively lower compared to pricing three- or four-cycle-free paths. We evaluated miscellaneous combinations of different separators, finding the combination of lifted subtour and D_k^+ inequalities to be most efficient.

Table 3. Running time and relative gap for different separators.

		Running Time [s]		**Relative Gap**	
		Small	**Large**	**Small**	**Large**
Original		1.14	18.57	0.43	0.57
Separator	Cycle	5.66	107.12	0.38	0.54
	D_k^+	6.19	286.34	0.33	0.45
	LSEC	9.41	378.85	0.29	0.43
	Odd CAT	4.05	152.71	0.39	0.52
	Odd path-free	5.06	87.30	0.41	0.56
	SEC	9.38	373.89	0.30	0.43
	unitary AFC	23.43	263.34	0.38	0.55
	LSEC and D_k^+	10.16	484.12	0.29	0.42

4.2.4. Performance of Primal Heuristics

In order to judge the effectiveness of the LP-based construction heuristics introduced in Section 3.5.3, we solve the initial relaxations of instances, then apply the heuristics in order to improve the primal bound. Once more, we record both the execution time and the gap relative to the original dual bound. The results, shown in Table 4, demonstrate the effectiveness of these fairly simple heuristics—while the computational effort is minor, with increases in running time well below 10 s, up to 30% of the gap is closed solely based on improved primal bounds.

Table 4. Running time and relative gap for different heuristics.

		Running Time [s]		Relative Gap	
		Small	**Large**	**Small**	**Large**
Original		1.14	18.57	0.43	0.57
Heuristic	Compound value	2.53	47.93	0.30	0.53
	Inverted travel time	2.34	45.22	0.37	0.53
	Variable value	1.99	46.38	0.37	0.50

4.2.5. Final Algorithm

Based on the previous experiments, we augment the arc-based formulation (TDTSP-A) with two-cycle-free path-based pricing, the combination of lifted subtour and D_k^+ inequalities, and all of the primal heuristics. In addition, we use the propagation method introduced in Section 3.5.1 to fix binary variables whenever possible.

Based on the combined improvements, we are able to solve 46 of the 50 small instances to optimality. The remaining gap of the four unsolved instances averages 4.5%.

While we are unable to solve any of the large instances to optimality, we reduce the original remaining gap from 55% to 19% (see Tables 1 and 5). In order to compare our algorithm with a state-of-the-art solver, we used GUROBI as a black-box solver on the same instances. As a black-box solver, GUROBI uses neither a column-generation approach, nor the specialized heuristics and valid inequalities we introduced above, consequently averaging at a significantly larger gap of 44% on the large instances.

Table 5. Running time and relative gap for the full formulation.

Instance	Full Formulation					GUROBI			
	p	**d**	**g**	**n**	t_{root}	**p**	**d**	**g**	**n**
1	627	527.08	0.19	18	775.79	676	444	0.52	1
2	832	644.08	0.29	10	763.66	952	566	0.68	1
3	660	551.06	0.20	26	296.38	733	476	0.54	3
4	740	609.09	0.21	8	1950.61	818	494	0.66	1
5	714	609.00	0.17	22	348.29	764	502	0.52	3
6	723	599.49	0.21	17	710.42	765	529	0.45	1
7	667	571.00	0.17	28	273.41	612	522	0.17	1
8	682	573.00	0.19	17	593.06	772	482	0.60	3
9	700	612.00	0.14	27	199.94	786	583	0.35	1
10	753	637.00	0.18	18	800.22	897	533	0.68	1
11	659	579.02	0.14	16	302.16	793	499	0.59	1
12	749	606.71	0.23	11	936.39	854	509	0.68	1
13	779	634.00	0.23	17	693.32	805	561	0.44	1
14	744	600.29	0.24	21	415.43	746	520	0.44	1
15	701	607.08	0.15	31	201.47	721	546	0.32	1
16	721	608.00	0.19	11	922.34	767	545	0.41	1
17	688	548.00	0.26	15	551.36	751	464	0.62	2
18	645	593.49	0.09	14	496.26	799	532	0.50	1
19	742	571.00	0.30	12	809.39	800	511	0.57	1
20	773	622.59	0.24	21	464.79	803	566	0.42	1

5. Conclusions and Future Work

In this paper, we have discussed several theoretical and empirical properties of the TDTSP. Since the TDTSP is a generalization of the ATSP, many of the complexity-specific theoretical results, such as $\mathcal{NP}$-hardness and inapproximability, carry over to the TDTSP.

Unfortunately, several positive results regarding the ATSP are not retained in the TDTSP. Specifically, the TDTSP remains inapproximable even if a generalized triangle inequality is satisfied. Furthermore, even simple relaxations, such as time-dependent trees, cannot be used to determine combinatorial lower bounds on the TDTSP.

From a practitioner's point of view, the increase in problem size poses significant problems when trying to solve even moderate-sized instances of the TDTSP. The authors of [15] conclude that there are challenging instances of the SDTSP with less than 100 vertices. While the results date back some years, the increase in computational complexity is apparent even in the case of the SDTSP.

To be able to tackle the TDTSP, a sophisticated pricing routine is necessary. It is apparent that generating columns according to time-dependent paths is superior to generating columns for individual arcs. More recent advances regarding specialized shortest path variants facilitate the solution process by achieving improved dual bounds. Further improvements in this area will likely benefit the TDTSP as well.

The connection between TDTSP and ATSP yields a variety of feasible classes of inequalities which help to significantly improve dual bounds. Unfortunately, the generalizations of SDTSP-type inequalities do not perform equally well in comparison. Objective value propagation and primal heuristics decrease the gap even further. The primal heuristics profit from the connection to the ATSP, significantly outperforming the heuristics built into the solver itself.

We have focused on developing an algorithm capable of solving instances with fairly arbitrary travel times. It may be worth investigating the performance on instances with more regular travel time functions by conducting a computational study. Similarly, the existing formulations can be extended to incorporate time windows, suggesting further comparative experiments.

Author Contributions: Conceptualization, C.H. and I.J. and S.S.; software, C.H.; investigation, C.H.; writing—original draft preparation, C.H. and I.J.; writing—review and editing, C.H. and I.J. and S.S.; funding acquisition, I.J. and S.S. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge support by the German Research Foundation and the Open Access Publication Funds of Technische Universität Braunschweig. The second author was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2163/1—Sustainable and Energy Efficient Aviation—Project-ID 390881007.

Data Availability Statement: The implementation of the algorithm as well as the generator for all random instances is available at https://github.com/chrhansk/time-dependent-tsp.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Time-Dependent Spanning Trees

Relaxations play an important role in integer programming in general and the TSP in particular. They provide lower bounds which can be used to obtain quality guarantees for solutions. The prevalent relaxation of combinatorial problems formulated as integer programs is given by their fractional relaxations. In several cases however, it is possible to derive purely combinatorial relaxations. In case of the symmetric traveling salesman problem, a popular combinatorial relaxation is given by one-trees [31]. A one-tree with respect to a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ consists of a spanning tree together with an additional edge joining vertex $s{:} = 1$ to another vertex $v \in V$. Since every tour is a one-tree, the one-tree of minimum cost provides a lower bound on the cost of a tour. The computation of such a one-tree in the static case involves the computation of a minimum spanning tree (MST), which can be carried out efficiently. The approach generalizes to the ATSP, in which case the one-tree corresponds to an arborescence with root s together with a single arc entering s from another vertex v.

An important question to ask is whether these approaches carry over to the time-dependent case. We begin by generalizing the definition of spanning trees to the time-dependent case. To this end, we assume throughout this subsection that D is bidirected, corresponding to an undirected graph G_D, and that the time-dependent cost function c is symmetric, i.e., $c_{uv}(\theta) = c_{vu}(\theta)$ holds for all $(u, v) \in A, \theta \in \Theta$, making the cost function c correspond to the undirected edges of G_D. We consider a spanning tree $T = (V, F)$ of G_D. For each vertex $v \in V$, there exists a unique (s, v)-path P_v in T, enabling us to define an arrival time $\theta_T^{\mathrm{arr}}(u)$ induced by T via P_u for each $u \in V$. Based on these arrival times we define the following:

Definition A1. *A time-dependent minimum spanning tree (TDMST) is a spanning tree T of G_D minimizing $c(T)$, which is defined as*

$$c(T) := \sum_{\{u,v\} \in F} c_{uv}(\theta_T^{\mathrm{arr}}(u)).$$

The corresponding optimization problem asks for the value of a TDMST for an instance (G_D, c, s). As was the case before, any tour becomes a spanning tree once we remove its last arc, the arrival time of the resulting path at its target coinciding with the cost of the tree. Consequently, the optimal value of the corresponding TDTSP instance is bounded from below by the cost of any minimum spanning tree. Unfortunately, approximating the TDMST is an $\mathcal{NP}$-hard problem itself:

Theorem A1. *There is no α-approximation algorithm for any $\alpha > 1$ for the TDMST problem unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Consider an instance of the 3SAT problem (see [29] (p. 391)), given in terms of a set $X := \{x_1, \ldots, x_n\}$ of Boolean variables and a set $\mathcal{Z} := \{Z_1, \ldots, Z_m\}$ of clauses, each clause consisting of exactly three literals in $X \dot\cup \overline{X}$. We construct a suitable instance of the TDMST problem using a number of components. First we define a component A_i for each variable $x_i \in X$. The component is shown in Figure A1a, the edges being annotated with their (constant) travel times.

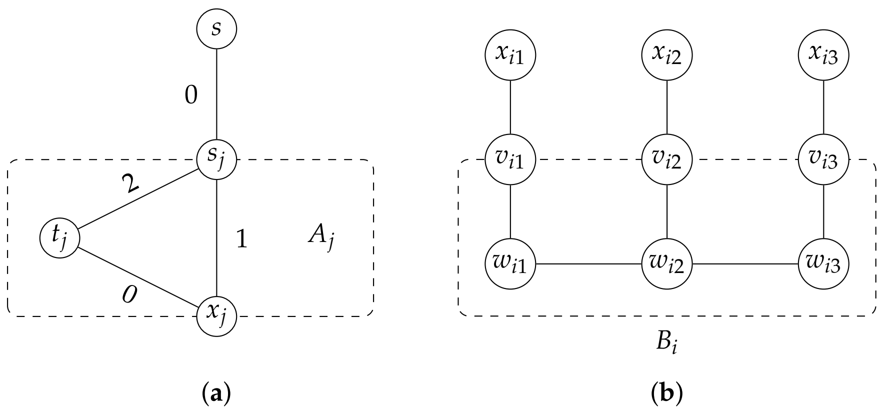


Figure A1. Gadgets used in the proof of Theorem A1. (**a**) A component which queries whether a variable is set; (**b**) A component which determines whether a clause is satisfied.

We define a component B_i for each clause $Z_i \in \mathcal{Z}$. To this end, let x_{i1}, x_{i2}, x_{i3} denote the variables whose literals appear in Z_i and $M \geq 2n + 6m + 1$. The edges in the component have the following travel times:

1. The edges $\{w_{i1}, w_{i2}\}$ and $\{w_{i2}, w_{i3}\}$ have a constant travel time of 1.
2. The edge $\{v_{i1}, w_{i1}\}$ has a travel time of

$$c_{v_{i1}w_{i1}}(\theta) := \begin{cases} 1 & \text{if } \theta \leq 2, \text{ and} \\ M & \text{otherwise.} \end{cases}$$

The same holds true for the edges $\{v_{i2}, w_{i2}\}$ and $\{v_{i3}, w_{i3}\}$.

3. The travel time of the edge connecting x_{ik} and v_{ik} depends on whether x_k or $\overline{x}_k$ appears in the clause Z_j. In the former case the travel time is constantly 1, whereas in the latter it is given by
$$c_{x_{ik}v_{ik}}(\theta) := \begin{cases} 0 & \text{if } \theta \geq 2, \text{ and} \\ 2 & \text{otherwise.} \end{cases}$$

The instance including the components, depicted in Figure A2, has a TDMST of cost less than M if and only if the 3SAT instance is satisfiable:

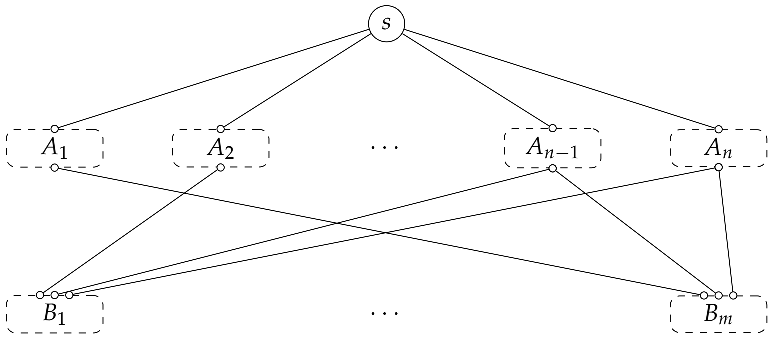


Figure A2. The TDMST construction used to prove Theorem A1.

Consider a satisfying truth assignment of the 3SAT instance. For each variable x_j set to `true` we choose the path (s, s_j, x_j, t_j) in component A_j. If the variable is set to `false` we choose the path (s, s_j, t_j, x_j) to be part of the spanning tree. Thus, the arrival time of the tree at x_j is 1 if x_j is set to `true` and 2 otherwise. For each clause Z_i we add the edges between the vertices corresponding to its variables and their respective v_i counterparts. Since the clause is satisfied, the arrival time at one of v_{i1}, v_{i2}, or v_{i3} is at most 2, enabling us to add the edges $\{v_{ik}, w_{ik}\}$, $\{w_{i1}, w_{i2}\}$, and $\{w_{i2}, w_{i3}\}$ at a cost of 1 each. The total cost of the resulting tree is at most $2n + 6m < M$.

Conversely, consider a tree T with costs less than M. We first consider the (s, s_j)-paths in T for all $j = 1, \ldots, n$. If this path does not contain $\{s, s_j\}$ itself, then it must start by t traversing a variable gadget $A_{j'}$ for $j' \neq j$ continued by a clause gadget B_i, entering via v_{ik} and leaving via v_{il}. As was established above, the earliest arrival time at v_{ik} is 2, implying that the time of arrival at w_{il} is at least 4, resulting in a travel time of at least M for the traversal of edge $\{w_{il}, v_{il}\}$, thereby contradicting the assumption of $c(T) < M$. For the same reason, T must contain either the path (s, s_j, x_j) or (s, s_j, t_j, x_j). Based on the optimality of T, the tree must contain (s, s_j, x_j, t_j) in the former case. We can therefore identify the choice of paths with a variable assignment as we did above.

To see that the assignment is feasible, we note that for each clause Z_i one of the edges $\{v_{ik}, w_{ik}\}$, say $\{v_{i1}, w_{i1}\}$, is in T and directed away from v_{i1}. Since $c(T) < M$, the travel time of this edge must be less than M, implying that the arrival time of T at v_{i1} is at most 2. If x_{i1} appears in Z_j, then the arrival time at x_{i1} is 1, and the variable is set to `true` satisfying Z_i. If $\overline{x}_{i1}$ appears in Z_j, then it follows that the arrival time at x_{i1} is 2, the variable is set to `false` and clause Z_i is satisfied as well.

Now assume the existence of an α-approximation for the TDMST problem. We construct the instance introduced above for $M := \lceil \alpha \cdot (2n + 6m) \rceil$ and apply the approximation. If the resulting tree has costs less than M, the 3SAT instance is satisfiable. Otherwise, the optimal TDMST has costs at least $M/\alpha \geq 2n + 6m$, and the instance is not satisfiable. $\square$

References

1. Applegate, D.L.; Bixby, R.E.; Chvatal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton University Press: Princeton, NJ, USA, 2011.
2. Gutin, G.; Punnen, A.P. *The Traveling Salesman Problem and Its Variations*; Springer: Berlin, Germany, 2006; Volume 12.
3. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [CrossRef]
4. McMenemy, P.; Veerapen, N.; Adair, J.; Ochoa, G. Rigorous Performance Analysis of State-of-the-Art TSP Heuristic Solvers. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*; Springer: Berlin, Germany, 2019; pp. 99–114. [CrossRef]
5. Batz, G.V.; Delling, D.; Sanders, P.; Vetter, C. Time-Dependent Contraction Hierarchies. In Proceedings of the 2009 Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX), New York, NY, USA, 3 January 2009; pp. 97–105. [CrossRef]
6. Kaufman, D.E.; Smith, R.L. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *J. Intell. Transp. Syst.* **1993**, *1*, 1–11. [CrossRef]
7. Delling, D.; Wagner, D. Time-Dependent Route Planning. *Robust Online Large-Scale Optim.* **2009**, *5868*, 207–230. [CrossRef]
8. Rosenkrantz, D.J.; Stearns, R.E.; Philip, M.; Lewis, I. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput.* **1977**, *6*, 563–581. [CrossRef]
9. Christofides, N. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*; Technical Report; Carnegie-Mellon Univ. Pittsburgh Pa Management Sciences Research Group: Pittsburgh, PA, USA, 1976.
10. Ascheuer, N.; Fischetti, M.; Grötschel, M. A Polyhedral Study of the Asymmetric Traveling Salesman Problem with Time Windows. *Networks* **2000**, *36*, 69–79. [CrossRef]
11. Ascheuer, N.; Fischetti, M.; Grötschel, M. Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut. *Math. Program.* **2001**, *90*, 475–506. [CrossRef]
12. Toth, P.; Vigo, D. *The Vehicle Routing Problem*; SIAM: Philadelphia, PA, USA, 2002.
13. Fukasawa, R.; Barboza, A.S.; Toriello, A. On the Strength of Approximate Linear Programming Relaxations for the Traveling Salesman Problem. Available online: www2.isye.gatech.edu/~atoriello3/bcpalp.pdf (accessed on 2 October 2020).
14. Fukasawa, R.; Longo, H.; Lysgaard, J.; De Aragão, M.P.; Reis, M.; Uchoa, E.; Werneck, R.F. Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Math. Program.* **2006**, *106*, 491–511. [CrossRef]
15. Abeledo, H.; Fukasawa, R.; Pessoa, A.; Uchoa, E. The time dependent traveling salesman problem: Polyhedra and algorithm. *Math. Program. Comput.* **2013**, *5*, 27–55. [CrossRef]
16. Dantzig, G.; Fulkerson, R.; Johnson, S. Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.* **1954**, *2*, 393–410. [CrossRef]
17. Irnich, S.; Villeneuve, D. The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for $k \geq 3$. *INFORMS J. Comput.* **2006**, *18*, 391–406. [CrossRef]
18. Roberti, R.; Mingozzi, A. Dynamic ng-Path Relaxation for the Delivery Man Problem. *Transp. Sci.* **2014**, *48*, 413–424. [CrossRef]
19. Picard, J.C.; Queyranne, M. The Time-Dependent Traveling Salesman Problem and its Application to the Tardiness Problem in One-Machine Scheduling. *Oper. Res.* **1978**, *26*, 86–110. [CrossRef]
20. Bigras, L.P.; Gamache, M.; Savard, G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discret. Optim.* **2008**, *5*, 685–699. [CrossRef]
21. Ángel-Bello, F.; Cardona-Valdés, Y.; Álvarez, A. Mixed integer formulations for the multiple minimum latency problem. *Oper. Res.* **2019**, *19*, 369–398. [CrossRef]
22. Pessoa, A.; Uchoa, E.; de Aragão, M.P.; Rodrigues, R. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Math. Program. Comput.* **2010**, *2*, 259–290. [CrossRef]
23. Cordeau, J.F.; Ghiani, G.; Guerriero, E. Analysis and Branch-and-Cut Algorithm for the Time-Dependent Travelling Salesman Problem. *Transp. Sci.* **2014**, *48*, 46–58. [CrossRef]
24. Arigliano, A.; Calogiuri, T.; Ghiani, G.; Guerriero, E. A branch-and-bound algorithm for the time-dependent travelling salesman problem. *Networks* **2018**, *72*, 382–392. [CrossRef]
25. Ichoua, S.; Gendreau, M.; Potvin, J.Y. Vehicle dispatching with time-dependent travel times. *Eur. J. Oper. Res.* **2003**, *144*, 379–396. [CrossRef]
26. Vu, D.M.; Hewitt, M.; Boland, N.; Savelsbergh, M. Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows. *Transp. Sci.* **2020**, *54*, 703–720. [CrossRef]
27. Boland, N.L.; Savelsbergh, M.W. Perspectives on integer programming for time-dependent models. *TOP* **2019**, *27*, 147–173. [CrossRef]
28. Hansknecht, C.; Joormann, I.; Stiller, S. Cuts, Primal Heuristics, and Learning to Branch for the Time-Dependent Traveling Salesman Problem. *arXiv* **2018**, arXiv:1805.01415.
29. Korte, B.; Vygen, J. *Combinatorial Optimization: Theory and Algorithms*; Springer: Berlin, Germany, 2012. [CrossRef]
30. Held, M.; Karp, R.M. A Dynamic Programming Approach to Sequencing Problems. *J. Soc. Ind. Appl. Math.* **1962**, *10*, 196–210. [CrossRef]
31. Held, M.; Karp, R.M. The traveling-salesman problem and minimum spanning trees. *Oper. Res.* **1970**, *18*, 1138–1162. [CrossRef]

32. Gouveia, L.; Voß, S. A classification of formulations for the (time-dependent) traveling salesman problem. *Eur. J. Oper. Res.* **1995**, *83*, 69–82. [CrossRef]
33. Lübbecke, M.E.; Desrosiers, J. Selected Topics in Column Generation. *Oper. Res.* **2005**, *53*, 1007–1023. [CrossRef]
34. Grötschel, M.; Padberg, M.W. Lineare Charakterisierungen von Travelling Salesman Problemen. *Z. Oper. Res.* **1977**, *21*, 33–64. [CrossRef]
35. Atamtürk, A.; Nemhauser, G.L.; Savelsbergh, M.W. Conflict graphs in solving integer programming problems. *Eur. J. Oper. Res.* **2000**, *121*, 40–55. [CrossRef]
36. Balas, E. The Asymmetric Assignment Problem and Some New Facets of the Traveling Salesman Polytope on a Directed Graph. *SIAM J. Discret. Math.* **1989**, *2*, 425–451. [CrossRef]
37. Miranda-Bront, J.J.; Méndez-Díaz, I.; Zabala, P. Facets and valid inequalities for the time-dependent travelling salesman problem. *Eur. J. Oper. Res.* **2014**, *236*, 891–902. [CrossRef]
38. Achterberg, T. SCIP: Solving constraint integer programs. *Math. Program. Comput.* **2009**, *1*, 1–41. [CrossRef]
39. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*; Gurobi Optimization, LLC: Beaverton, OR, USA. 2020.