Dynamic Programming for the Minimum Tour Duration Problem

Christian Tilk[*,a], Stefan Irnich[a]

[a] *Chair of Logistics Management, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

The minimum tour duration problem (MTDP) is the variant of the traveling salesman problem with time windows, which consists of finding a time window-feasible Hamiltonian path minimizing the tour duration. We present a new effective dynamic programming (DP)-based approach for the MTDP. When solving the traveling salesman problem with time windows with DP, two independent resources are propagated along partial paths, one for costs and one for earliest arrival times. For dealing with tour duration minimization, we provide a new DP formulation with three resources for which effective dominance and bounding procedures are applicable. This is a non-trivial task because in the MTDP at least two resources depend on each other in a non-additive and non-linear way. In particular, we define consistent resource extension functions (REF) so that dominance is straightforward using componentwise comparison for the respective resource vectors. Moreover, one of the main advantages of the new REF definition is that the DP can be reversed consistently such that the forward DP or any of its relaxations provides bounds for the backward DP, and vice versa. Computational test confirm the effectiveness of the proposed approach.

Key words: traveling salesman problem, time windows, tour duration, dynamic programming, state-space relaxation

1. Introduction

In this paper, we consider a variant of the *traveling salesman problem with time windows* (TSPTW), in which the objective is the minimization of the tour duration. There is no consistent naming of this problem in the literature. We will use the name *minimum tour duration problem* (MTDP) in the following and start with its definition. Let $G = (V, A)$ be a digraph with node set V and arc set A. Two distinguished nodes are given, the start node $o \in V$ and the destination node $d \in V$. A *Hamiltonian o-d-path* is a simple path which start at o, ends at d, and visits all nodes exactly once. A travel time $t_{ij} > 0$ is associated with each arc $(i, j) \in A$ and a time window $[a_i, b_i]$ with each node $i \in V$. For any path $P = (i_0, i_1, \dots, i_p)$, a sequence $T = (T_0, T_1, \dots, T_p)$ of numbers is called a *schedule*. A schedule with $T_k \in [a_{i_k}, b_{i_k}]$ for all $k \in \{0, 1, \dots, p\}$ and $T_{k-1} + t_{i_{k-1},t_k} \leq T_k$ for all $k \in \{1, \dots, p\}$ is called *feasible*. A *TSPTW tour* is a Hamiltonian o-d-path P for which a feasible schedule exists. We say that T_k is the *time when service starts* at node i_k. Note that we do not explicitly consider *service times* because they can be included in the travel times t_{ij}. The MTDP is the problem of finding a TSPTW tour P with a feasible schedule $T = (T_o, \dots, T_d)$ minimizing $T_d - T_o$.

In contrast, the objective in the TSPTW is minimizing the arc-traversal costs for given arc costs c_{ij} for $(i, j) \in A$. There exists a third problem related to minimizing the completion time T_d. We call this problem *minimum completion time problem* (MCTP). It is the special case of the MTDP, in which the starting time T_o is fixed. TSPTW, MTDP, and MCTP only differ in their objectives, and these objectives are generally

*Corresponding author.
Email address: tilk@uni-mainz.de (Christian Tilk)

conflicting (Vidal *et al.*, 2011). Savelsbergh (1992) pointed out that it is important to be able to handle them.

The contribution of the paper at hand is to present a new effective dynamic-programming (DP) algorithm for the MTDP. It generalizes the approach presented by Baldacci *et al.* (2011b), who solve the TSPTW with a DP-based algorithm. Their algorithm solves all but one instance of the known benchmark sets for the TSPTW and outperforms all exact methods published in the literature so far. When solving the TSPTW, two independent resources are propagated along partial paths, one for costs and one for earliest arrival times. For dealing with tour duration minimization, there exist several possibilities to define and propagate resources. However, all these alternatives use at least three resources. We will follow ideas presented in (Irnich, 2008) in order to apply effective dominance and bounding procedures. This is a non-trivial task because in the MTDP at least two resources depend on each other in a non-additive and non-linear way. In particular, we define consistent resource extension functions (REFs, see Desaulniers *et al.*, 1998) so that dominance is straightforward using componentwise $\leq$ for the respective resource vectors. Moreover, one of the main advantages of the new REF definition is that the DP can be reversed consistently such that the forward DP or any of its relaxations provides bounds for the backward DP, and vice versa.

Using relaxations to obtain lower bounds is common practice in routing problems, e.g., using a state-space relaxation (Christofides *et al.*, 1981). We present two new relaxations for the MTDP with only one and two resource, respectively, which are attractive due to their low computational complexity. These and other relaxations can be combined with the ng-tour and ngL-tour relaxation (Baldacci *et al.*, 2011a,b).

To compute tight lower bounds, we use two methods: First, we adapt a penalty method first suggested by Christofides *et al.* (1981). Second, we generate the neighborhoods for the ng-tour and ngL-tour relaxations dynamically. This technique has been successfully applied for solving different routing problems (Roberti and Mingozzi, 2013; Bode and Irnich, 2013). To the best of our knowledge, we present the first exact algorithm for the MTDP. We provide computational results with optimal solutions for many known benchmark instances, which were originally provided for the TSPTW.

This paper is structured as follows. Section 2 briefly surveys exact solution algorithms to the TSPTW, MTDP, and MCTP. In Section 3, we present our new DP formulation for the MTDP. The computation of upper bounds with the help of a heuristic is presented in Section 4. Section 5 discusses relaxations, namely the adapted ng-tour and ngL-tour relaxations and two new relaxations with one and two resources, respectively. Moreover, a relaxation-based bounding procedure is presented. In order to further improve the lower bounds, we apply a penalty method in Section 6. Section 7 reports the computational results. Concluding remarks are given in Section 8.

2. Literature Review

The TSPTW and MCTP are often discussed problems in the literature, but only little attention was dedicated to the MTDP. Christofides *et al.* (1981) proposed a method for solving the MCTP based on state-space relaxation. They reported solutions of instances with up to 50 nodes. Baker (1983) proposed a branch-and-bound method for the MCTP producing exact solutions for instances with up to 50 nodes. Langevin *et al.* (1993) introduced a two-commodity flow formulation for the MCTP and TSPTW and developed a branch-and-bound algorithm that is able to solve instances with up to 60 nodes. Their model can easily be adapted to the MTDP.

Several DP approaches were proposed for the TSPTW: Dumas *et al.* (1995) presented a DP algorithm and advanced preprocessing procedures to reduce the number of states and state transitions. They computed solutions of instances with up to 200 nodes, but relatively tight time windows. The DP algorithm of Mingozzi *et al.* (1997) is based on another state-space relaxation. They solved the TSPTW with precedences for instances with up to 120 nodes. Li (2009) solved the TSPTW with a bi-directional resource-bounded label correcting algorithm. This algorithm is able to solve instances with up to 233 nodes. Recently, Baldacci *et al.* (2011b) presented the ng-tour and ngL-tour relaxations for the TSPTW to compute tight lower bounds. To improve the lower bounds they solve the dual of a problem that seeks a minimum-weight convex combination of non-necessarily elementary tours with a dual-ascent heuristic. Their computational results are impressive: All but one instances from several TSPTW benchmark sets are solved to proven optimality.

Balas and Simonetti (2001) presented a DP model and algorithm for a special case of the TSPTW and the MCTP, which can also be applied as a heuristic for the general case. Ascheuer *et al.* (2001) were the first to develop branch-and-cut algorithms for the TSPTW. They implemented three alternative integer programming formulations of the TSPTW and solved instances with up to 233 nodes. Dash *et al.* (2012) presented an extended formulation of the TSPTW based on partitioning the time windows into buckets. The LP-relaxation of their formulation provides strong lower bounds, which they exploited in a branch-and-cut algorithm.

To the best of our knowledge, (Savelsbergh, 1992) is the first paper dealing with the MTDP. Savelsbergh described edge-exchange improvement methods for the MTDP and the VRPTW with the objective of minimizing the route duration. The only recent papers dealing with the MTDP are an ant-colony approach by Favaretto *et al.* (2006) and a new two-commodity flow formulation by Kara *et al.* (2013). Favaretto *et al.* (2006) call the problem *temporal TSPTW* and present computational results for the MTDP on benchmark instances originally proposed for the TSPTW. Kara *et al.* (2013) solved their model with the MIP solver CPLEX to optimality for instances with up to 40 nodes.

The problem of minimizing the tour duration also occurs as a subproblem in truck driver scheduling and routing when the driving time is limited, e.g., by hours-of-service regulations as studied by Goel (2009). He presented an exact method for scheduling driving periods, breaks, rest periods and handling activities for a given tour based on a labeling algorithm. Among others, Goel and Vidal (2013) studied similar problems for regulations of different countries. They presented an algorithm that combines population-based metaheuristics with a local search that uses forward labeling procedures for checking compliance with complex hours of service regulations. Prescott-Gagnon *et al.* (2010) solve a vehicle-routing problem with time windows and European driver rules. They used a column-generation approach, which utilizes a tabu search to heuristically solve the subproblem. To check the route feasibility they model all feasibility rules as resource constraints and develop a label-setting algorithm to perform this check.

3. Dynamic-Programming Formulations

In this section, we present an exact DP formulation for the MTDP. First, we will introduce the REFs for the forward propagation and the corresponding DP recursion. Second, we will present propagation rules that limit the number of possible extensions and fathoming rules that eliminate labels which cannot lead to a feasible or optimal solution. Last, we define consistent REFs for the backward propagation and show how paths of the forward and the backward formulation can be concatenated. Consistency refers at least to the following five aspects:

1. Using the same set of resources, the MTDP can either be solved using forward or backward propagation. Both resulting DP algorithms have identical worst-case complexity.

2. Let $P = (o, \dots, i)$ be a path resulting from forward propagation, and let $P^{bw} = (i, \dots, d)$ be a path resulting from backward propagation. Moreover, let $L = L(P)$ and $L^{bw} = L^{bw}(P^{bw})$ be the corresponding labels that result from forward and backward propagation, respectively. Then, the concatenation $P \oplus P^{bw}$ is feasible w.r.t. resource consumption if and only if $L \leq L^{bw}$ holds.

3. If $P \oplus P^{bw}$ is feasible w.r.t. resource consumption, the labels allow the computation of the minimum tour duration in constant time.

4. The above label comparison enables the bidirectional solution of the MTDP by combined forward and backward propagation.

5. If an upper bound on the MTDP is known, the above label comparison also enables the use of bounding techniques: A backward label provides a valid bound for the forward label, and vice versa. Moreover, any label resulting from a relaxation may be used for bounding instead of a label produced with the exact DP.

3.1. Forward Dynamic-Programming Formulation

A *forward path* $P = (o, \dots, i)$ defines a *forward label* (k, i, S, T), where $T = (T^{time}, T^{dur}, T^{help})$ with the following semantics:

- $i \in V$ is the last visited node in P
- $S \subseteq V$ is the set of all nodes visited by the path
- k is the path length, i.e., $k = |P| = |S| - 1$
- T^{time} is the earliest feasible time, at which the last node i can be visited in the path
- T^{dur} is the minimum tour duration of the path starting at node o, visiting the set $S \subset V$ and ending at node i so that every node is visited within its time window
- T^{help} is the negative of the latest possible departure time at node o so that the time window constraints of every node in the path are satisfied and the tour duration is T^{dur}

While k, i, and S are standard attributes that can describe any partial path, the resource vector $T = (T^{time}, T^{dur}, T^{help})$ consists of the three actual resources that are specific for the MTDP. Similar to the description in (Irnich, 2008), we will now define resource windows, the initial label for the path $P = (o)$, and REFs for forward propagation.

The resource windows are defined for all nodes $i \in V$ as:

$$T_i^{time} \in \ [a_i, b_i] \tag{1a}$$

$$T_i^{dur} \ \in \ [0, UB], \text{where } UB \text{ is any bound on the MTDP (tour duration)} \tag{1b}$$

$$T_i^{help} \in \ [-b_o, \infty) \tag{1c}$$

The earliest possible time a Hamiltonian o-d-path can start is a_o and the latest is b_o. Therefore, the initial forward label for $P = (o)$ is defined as $(0, o, \{o\}, T_o)$ with $T_o = (T_o^{time}, T_o^{dur}, T_o^{help}) = (a_o, 0, -b_o)$. Forward propagation of a label (k, i, S, T_i) along an arc $(i, j) \in A$, i.e., towards node j produces the new label $(k+1, j, S \cup \{j\}, T_j)$. Herein, T_j results from the following REFs:

$$T_j^{time} = f_{ij}^{time}((k, i, S, T_i)) := \max\{T_i^{time} + t_{ij}, a_j\} \tag{2a}$$

$$T_j^{dur} = f_{ij}^{dur}((k, i, S, T_i)) := \max\{T_i^{dur} + t_{ij}, T_i^{help} + a_j\} \tag{2b}$$

$$T_j^{help} = f_{ij}^{help}((k, i, S, T_i)) := \max\{T_i^{dur} + t_{ij} - b_j, T_i^{help}\} \tag{2c}$$

Note that the third resource T^{help} has been defined to be negative so that a non-decreasing REF results. Desaulniers *et al.* (1998) were among the first who explicitly stressed the high importance of non-decreasing REFs, i.e., functions where $S \leq T$ implies $f(S) \leq f(T)$. If resources are propagated with non-decreasing REFs, standard dominance with componentwise $\leq$-comparison is valid.

The resources T_i^{dur} and T_i^{help} are interdependent. This represents, on the one hand, that when we must wait at a node j, we can shift the start time to avoid an increase of the tour duration, as long as the schedule stays feasible. On the other hand, the possible shift can be limited by the difference of the tour duration and the latest service start b_j associated with node j.

We conclude that $T_i^{time} - T_i^{dur}$ is the earliest feasible departure time at node o that avoids unnecessary waiting. Hence, $-T_i^{help} - T_i^{time} + T_i^{dur}$ is the possible amount of time that we may shift the earliest possible departure time in direction b_o. In addition $-T_i^{help} + T_i^{dur}$ is the latest feasible arrival time at node i when the path duration is T_i^{dur}.

Example 1. We consider a path $P = (0, 1, 2, 3, 4)$ with $o = 0$ and $d = 4$ and with time windows $[a_j, b_j]$ and travel times $t_{j,j+1}$ given in the first three columns of the following table:

Node	Time window	Travel time	Resources		
j	$[a_j, b_j]$	$t_{j,j+1}$	T_j^{time}	T_j^{dur}	T_j^{help}
0	[0, 6]	1	0	0	-6
1	[2, 5]	2	2	1	-4
2	[5, 6]	3	5	3	-3
3	[11, 12]	4	11	8	-3
4	[14, 18]	–	15	12	-3

The (minimum) tour duration of the path $P_1 = (o, 1)$ coincides with the sum of travel times because waiting can be avoided by starting at any time between 1 and 4. The latest possible departure time is limited due to b_1. The path $P_2 = (o, 1, 2)$ limits the latest possible departure time in the same manner as the path P_1. The tour duration of the path $P_3 = (o, 1, 2, 3)$ consists of six units of travel time and two units of waiting time. The waiting time cannot be avoided because starting later than at time $T_o = 3$ violates the due date b_2 and the earliest time to arrive at node 3 is $a_3 = 11$. Next, the path $P_4 = (o, 1, 2, 3, 4)$ has the tour duration twelve, which comprises the sum of travel times and two units of waiting time at node 3 before a_3. The latest possible departure time of this path is three, which is limited due to b_2.

Before we start the actual DP algorithm, we modify the instance to obtain an equivalent one with fewer arcs, tighter time windows, and a set of precedences. This kind of *preprocessing* was originally suggested by Desrosiers *et al.* (1995). We iterative compute and update two types of values. $EAT(i, j)$ is the earliest feasible arrival time at node j when coming from node i, and $LDT(i, j)$ is the latest feasible departure time from node i when going to node j. Furthermore, the time window constraints impose a partial ordering of the nodes, which we use to identify node precedences: For all $j \in V$ the set $\pi(j)$ is the set of all nodes $i \in V$ that must precede j. The precedences $\pi(j)$, time windows $[a_i, b_i]$, times $EAT(i, j)$ and $LDT(i, j)$, and the (reduced) arc set A mutually affect each other (see Desrosiers *et al.* (1995) for details). Therefore, the computation should be iterated until no more modifications are made. Note that preprocessing generally reduces the possible extensions of labels, that preprocessed instances are sometimes significantly smaller, and have stronger relaxations so that they are in the end easier to solve.

Algorithm 1: Forward Dynamic Programming Labeling Algorithm

1 SET $\mathcal{L}_0 := \{(0, o, \{o\}, (a_o, 0, -b_o))\}$
2 **for** $k = 0, 1, \ldots, |V| - 1$ **do**
3   **for** $(k, i, S, T_i) \in \mathcal{L}_k$ **do**
4   **for** $(i, j) \in A : j \notin S, \pi(j) \subseteq S$ **do**
5   SET $T_j := f_{ij}(T_i)$
6   **if** *FeasibilityCheck(T_j)* **then**
7   SET $L_j := (k + 1, j, S \cup \{j\}, T_j)$
8   **if** *BoundingCheck(L_j)* **then**
9   ADD L_j to $\mathcal{L}_{k+1}$
10   CALL Dominance algorithm for $\mathcal{L}_{k+1}$
11 FIND a label $L_d^* = (|V|, d, V, T_d) \in \mathcal{L}_{|V|}$ with T_d^{dur} minimal
Result: The path P^* represented by label L_d^*

The forward DP is described in Algorithm 1. Herein, all labels are grouped according to the length k of the corresponding path, and $\mathcal{L}_k$ denotes this set. In the following, we comment on the components of Algorithm 1: The forward propagation (Step 4) always creates feasible partial paths. In particular, only arcs $(i, j) \in A$ from the preprocessed instance are allowed, the partial path must be elementary (ensured by $j \notin S$), and must respect all precedences ($\pi(j) \subseteq S$). In the feasibility check (Step 6), we first compare $T_j = (T_j^{time}, T_j^{dur}, T_j^{help})$ against the upper bounds $(b_j, UB, -a_o)$ given by (1). Moreover, there might not exist an extension to a feasible TSPTW tour due to resource consumption. We apply the following rule:

Rule 1. *(Feasibility) A label (k, i, S, T_i) cannot lead to a feasible solution if there exists a node $j \in V \setminus S$ with $T_i^{time} > LDT(i, j)$. Hence, such a label can be discarded.*

Bounding procedures try to identify those labels, for which any extension to a feasible TSPTW tour will only create a non-optimal tour. In case of the MTDP, non-optimality refers to the tour duration measured with the help of the resource T_i^{dur}. It is obvious that a label (k, i, S, T_i) cannot lead to an optimal solution if the earliest service time a_d at the destination node d minus the label's latest possible departure time at o without unnecessary waiting, i.e., $-T_i^{help}$ is greater than an upper bound. In the following we assume that an upper bound UB on MTDP is known. Step 8 of Algorithm 1 uses the following rule:

Rule 2. *(Bounding) Let UB be an upper bound for the MTDP. A label (k, i, S, T_i) cannot lead to an improved solution if $a_d + T_i^{help} \geq UB$. Hence, it can be discarded.*

A *dominance algorithm* eliminates those labels whose final extension to the destination node d produce longer tour durations compared to extensions of another label. The dominance algorithm (Step 10) applies the following rule:

Rule 3. *(Domination) Let $L = (k, i, S, T_i)$ and $L' = (k, i, S, T'_i)$ be two labels with identical path length k, set S and last node i. Let $P = P(L)$ and $P' = P(L')$ be the respective paths. If $T_i \leq T'_i$ (component-by-component), any feasible extension of P' towards d is also a feasible extension of P with non-smaller tour duration. Hence, L' can be discarded.*

The validity of this dominance rule is a direct consequence of the fact that the REFs are non-decreasing (see Desaulniers *et al.*, 1998; Irnich and Desaulniers, 2005).

3.2. Backward Dynamic-Programming Formulation

We now define consistent backward resource extensions that allow the reversal of the DP approach so that a backward DP results. Moreover, the backward DP or any of its relaxations provides bounds for the forward DP.

Irnich (2008) presented a general framework applicable to classical REFs of the form $f_{ij}(T_i) = \max\{a_{ij}, T_i + t_{ij}\}$ and REFs of the form $f_{ij}(T_i, T'_i) = (\max\{a_{ij}, T_i + t_{ij}, T'_i + u_{ij}\}, \max\{a'_{ij}, T_i + t'_{ij}, T'_i + u'_{ij}\})$ (the latter is called REFs with pairwise max-term). Herein, it is assumed that the extension along the arc $(i, j) \in A$ is feasible if $f_{ij}(T)$ and $f_{ij}(T, T')$ does not exceed b_{ij} and (b_{ij}, b'_{ij}), respectively. Then, the inverse REFs for REFs with pairwise max-term are $f_{ij}^{bw}(T_j) = \min\{b_{ij}, T_j - t_{ij}\}$ and $f_{ij}^{bw}(T_j, T'_j) = (\min\{b_{ij}, T_j - t_{ij}, T' - t'_{ij}\}, \min\{b'_{ij}, T_j - u_{ij}, T'_j - u'_{ij}\})$ (see Theorem 5 in (Irnich, 2008)).

For the MTDP, a *backward path* $(i, .., d)$ defines a *backward label* (k, i, S, T_i^{bw}). Herein, k is the length of the path, i the first visited node (i.e., the last node when propagating backward), S the set of all visited nodes, and a resource vector $T = (T_i^{time}, T_i^{dur}, T_i^{help})$. It is very important to mention that the semantics of the backward resources differs from the semantics of the forward resources: First, T_i^{time}, the latest feasible arrival time at node i. Second, T_i^{dur} is the difference between UB and the minimum tour duration of the path. Note that we assume that an upper bound for the minimal tour duration of a Hamiltonian path is known. The assumption is certainly not restrictive because $UB = b_d - a_o$ is always a valid upper bound. Moreover, the tour duration $T_i^{dur} - UB$ does not take the upper bound b_i of the time window at node i into account. Third, T_i^{help} is the difference of UB and the earliest possible arrival time at node d with tour duration T_i^{dur} that satisfies the time window constraints of every node in the path.

We define the initial state $s_d := (0, d, \{d\}, T_d)$ with $T = (T_d^{time}, T_d^{dur}, T_d^{help}) := (b_d, UB, UB - a_d)$. When we propagate a state (k, i, S, T) backward from node j to node i, i.e., in reverse direction along an arc $(i, j) \in A$, we add i to S and use the following REFs to update the resource vector T_j:

$$T_i^{time} = f_{ij}^{time,bw}(T_j) := \min\{T_j^{time} - t_{ij}, b_i\} \tag{3a}$$

$$T_i^{dur} = f_{ij}^{dur,bw}(T_j) := \min\{T_j^{dur} - t_{ij}, T_j^{help} - t_{ij} + b_j\} \tag{3b}$$

$$T_i^{help} = f_{ij}^{help,bw}(T_j) := \min\{T_j^{dur} - a_j, T_j^{help}\} \tag{3c}$$

The resource windows are defined as before by equations (1).

In general, any backward label T_i^{dur} at the start node i disregards the upper bound b_i of the time window at that node. However, the true resulting tour duration can be computed as $\max\{UB - T_i^{dur}, UB - T_i^{help} - b_i\}$.

Example 2. We consider the same example and path $P = (0,1,2,3,4)$ with time windows $[a_j, b_j]$ as in Example 1. As before the time windows and travel times $t_{j,j+1}$ are given in the first three columns of the following table. The forward resources are denoted by S and backward resources by T so that we can distinguish between the two.

Node	Time window	Travel time	Backward Resources			True duration $\max\left\{UB - T_j^{dur},\right.$	Forward Resources		
j	$[a_j, b_j]$	$t_{j,j+1}$	T_j^{time}	T_j^{dur}	T_j^{help}	$\left.UB - T_j^{help} - b_j\right\}$	S_j^{time}	S_j^{dur}	S_j^{help}
4	[14,18]	-	18	UB	∞	0	15	12	-3
3	[11,12]	4	12	$UB-4$	$UB-14$	4	11	8	-3
2	[5, 6]	3	6	$UB-7$	$UB-15$	9	5	3	-3
1	[2, 5]	2	4	$UB-11$	$UB-15$	11	2	1	-4
0	[0, 6]	1	3	$UB-12$	$UB-15$	12	0	0	-6

Note that the computed values are correct for any upper bound $UB \geq 12$ on the tour duration.

We discuss the backward labels now: First, the initial backward resource vector $(T_j^{time}, T_j^{dur}, T_j^{help}) = (18, UB, \infty)$ results from the upper bound values defined in (1). In the path $Q_3 = (3,4)$ the tour duration $UB - T_3^{dur} = 4$ and sum of travel time coincide, and $T_j^{help} = UB - 14$ means that the earliest arrival time at node $d = 4$ is 14 because the bound $b_3 = 12$ remains disregarded (see explanation of the semantics of the backward resources given above). The tour duration of the path $Q_2 = (2,3,4)$ is still 7, i.e., identical to the sum of the travel times, because also here the respective bound $b_2 = 6$ is not taken in account. The earliest possible arrival time at node d is $UB - T_3^{help} = 15$. In the path $Q_1 = (1,2,3,4)$, two units of waiting time occur. They cannot be avoided because the earliest possible arrival time at node d is 15 and the latest possible departure time at node 1 is $b_2 - t_{23} = 4$. The path $Q_0 = (0,1,2,3,4)$ has a tour duration of $UB - T_0^{dur} = 12$, which consists of the sum 10 of travel times and two units of waiting time before node 3. The earliest possible arrival time at node $d = 4$ is $UB - T_0^{help} = 15$. The table also contains the values for the forward resource vector $S_j = (S_j^{time}, S_j^{dur}, S_j^{help})$ in the last three columns. They are computed using forward partial paths as shown in Example 1. At any intermediate node j of the path $(0,1,2,3,4)$, the forward label S_j for the forward path $(0,\dots,j)$ and the backward label T_j for the backward path $(j,\dots,4)$ fulfill $S_j \leq T_j$. In this sense, the definition of forward and backward REFs are consistent.

Next, we present the fathoming rules for the backward DP. This rules are counterparts of Rules 1-3 for the forward DP. The proofs are straightforward. For the sake of completeness the entire backward dynamic labeling algorithm is given in the Appendix in Section A.1.

Rule BW 1. *(Feasibility) A backward label (k, j, S, T_j) cannot lead to a feasible solution if there exists a node $i \in V \setminus S$ with $T_j^{time} < EAT(i,j)$. Hence, such a label can be discarded.*

Rule BW 2. *(Bounding) Let UB be an upper bound for the MTDP. A backward label (k, j, S, T_j) cannot lead to an improved solution if $UB - T_j^{help} - b_o \geq UB$. Hence, it can be discarded.*

Rule BW 3. *(Domination) Let $L = (k, j, S, T_j)$ and $L' = (k, j, S, T'_j)$ be two backward labels with identical path length k, set S and first node j. Let $P = P(L)$ and $P' = P(L')$ be the respective backward paths. If $T_i \geq T'_i$ (component-by-component), any feasible extension of P' towards o is also a feasible extension of P with non-smaller tour tour duration. Hence, L' can be discarded.*

3.3. Bidirectional Dynamic Programming

As mentioned before, a bidirectional labeling approach uses forward and backward labels together and herewith allows the computation of Hamiltonian paths as concatenations of forward and backward paths. With the above definitions of REFs and labels, feasibility testing and the computation of the objective is straightforward. The corresponding statements are summarized in the following proposition.

Proposition 1. *Let a feasible forward path $P^{fw} = (o, \ldots, i)$ with forward label (k, i, S, T_i) and a feasible backward path $P^{bw} = (i, \ldots, d)$ with backward label (k', i, S', T'_i) be given. Then:*

(i) *The concatenation path $P = P^{fw} \oplus P^{bw}$ is a feasible Hamiltonian o-d path if and only if $k + k' = |V|$, $S \cap S' = \{i\}$, and $T \leq T'$ holds.*

(ii) *The tour duration of $P = P^{fw} \oplus P^{bw}$ is given by $z = \max\{T_i^{dur} - T_i^{'dur}, T_i^{help} - T_i^{'help}\}$.*

Note that if the tour duration is defined by the term $T_i^{help} - T_i^{'help}$ of the maximum, there occurs waiting time on the path when going forward from node i to $i + 1$. The amount of waiting time (when going from node i to $i+1$) is defined by the difference of the right and left term of the maximum. In the Example 2, the concatenation of the corresponding forward and backward labels is always feasible and the minimum tour duration is always 12. The time window bounds b_2 and a_3 and the travel time t_{23} imply that there must be waiting when going from node 2 to node 3. Hence, the tour duration at node 2 is defined by the right term of the maximum. There exist several possible strategies to conduct bidirectional labeling, e.g., discussed by Righini and Salani (2006) and Li (2009).

4. Upper Bounds

For the asymmetric traveling salesman problem (ATSP), Balas (1999) proposed and analyzed a family of large-scale neighborhoods. Although, the number of neighbor solutions is exponential (in the length of the tour), the neighborhoods can be searched efficiently by means of *very large-scale neighborhood search* (VLSNS). In VLSNS, which is a variant of local search, a best neighbor solution is found by solving another optimization problem that can be solved in (pseudo-)polynomial time. We briefly summarize the VLSNS for the so-called Balas-Simonetti neighborhood of the ATSP, before we point out its use in the TSPTW context originally discussed in (Balas and Simonetti, 2001).

Given an ATSP Hamiltonian path $x = (x_1, \ldots, x_n)$ the neighborhood $\mathcal{N}_{BS}^k(x)$, for a given parameter $k \geq 2$, consists of all tours $x' = (x_{\pi(1)}, \ldots, x_{\pi(n)})$, where π is a permutation of $\{1, \ldots, n\}$ that fulfills the following conditions: For any two indices $i, j \in \{1, \ldots, n\}$ with $i + k \leq j$, the inequality $\pi(i) \leq \pi(j)$ holds. It means that if a node x_i precedes a node x_j by at least k positions, then x_i must also precede x_j in the neighbor solution. Moreover, the nodes x_1 and x_n are typically kept fixed at positions 1 and n, respectively.

A best neighbor solution $x' \in \mathcal{N}_{BS}^k(x)$ can be determined by solving a shortest-path problem in an auxiliary graph G_k^*. Figure 1 shows an example of G_k^* for $k = 3$ and a tour $x = (x_1, \ldots, x_7)$. The auxiliary

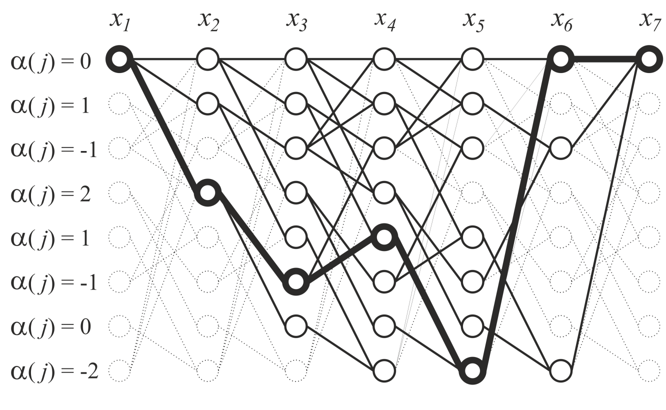

Figure 1: Auxiliary Graph G_k^* for $k = 3$ for the Balas-Simonetti Neighborhood $\mathcal{N}_{BS}^3(x)$ of $x = (x_1, \ldots, x_7)$

graph G_k^* is well-structured and consists of n identical stages for a path x of length $n - 1$. The $(k+1)2^{k-2}$ states at stage i are denoted by V_i, and $k(k+1)2^{k-2}$ arcs connect the states of consecutive stages V_i and V_{i+1}. Stage 1 contains the start state o, and stage n the sink state d. Every o-d-path in G_k^* represents a neighbor solution x', and vice versa. This property results from the fact that each state s refers to a (restricted) permutation of the nodes around position i. In particular, for a given tour $x = (x_1, \dots, x_n)$ the state s in stage i determines the permuted node $x'_i = x_{i+\alpha(s)}$ at position i in the neighbor tour x', where $\alpha(s)$ is an integer number associated with state s.

Since all induced subgraphs $G^*_k[V_i \cup V_{i+1}]$ are all identical, only one such copy needs to be constructed beforehand, and only once. Herewith, the auxiliary graph G^*_k is represented implicitly. Every neighborhood search is then a shortest-path computation on this acyclic digraph G^*_k requiring $\mathcal{O}(n \cdot k^2 2^k)$ time and space. Nevertheless, the construction of the subgraphs $G^*_k[V_i \cup V_{i+1}]$ is non-trivial, i.e., the rules that determine the arc set and the values $\alpha(s)$ for states $s \in V_i$; details can be found in the papers (Balas and Simonetti, 2001; Simonetti and Balas, 1996).

The neighboring tour associated with the path depicted in bold in Figure 1 is $x' = (x_{1+0}, x_{2+2}, x_{3-1}, x_{4+1}, x_{5-2}, x_{6+0}, x_{7+0}) = (x_1, x_4, x_2, x_5, x_3, x_6, x_7)$. Those states s that refer to positions $i + \alpha(s) < 1$ or $i + \alpha(s) > n = 7$ are states that cannot be reached by any o-d-path; unreachable states are drawn with dotted lines in Figure 1. In order to ensure that any o-d-path in G^*_k has a cost identical to the cost of the implied neighbor solution x', one has to label arc $(s, s') \in V_i \times V_{i+1}$ with cost $c_{x_{i+\alpha(s)}, x_{i+1+\alpha(s')}}$. For instance, the first bold arc in Figure 1 is labelled with cost $c_{x_{1+0}, x_{2+2}} = c_{x_1, x_4}$, the second has cost $c_{x_{2+2}, x_{3-1}} = c_{x_4, x_2}$ etc.

The auxiliary graph G^*_k can now be used to improve a given MTDP solution x by solving a shortest-path problem with resource constraints (SPPRC, Irnich and Desaulniers, 2005) on G^*_k. The initial label is $T_o = (a_o, 0, -b_0)$ associated with the initial state o of G^*_k. When propagating resources from a state s at stage i to another state s' at stage $i+1$, we use the REF $f_{i+\alpha(s), i+1+\alpha(s')}$ as defined by (2) and check the resource consumption at node $x_{i+1+\alpha(s')}$ using the resource bounds (1). Note that it is not necessary to keep track of the last node, the stage, and visited nodes (in the DP recorded by i, k and S) because the auxiliary network G^*_k ensures by construction that o-d paths are elementary whenever the input tour x was elementary. If an improving MTDP tour is found, x is replaced by the new tour and the process is iterated.

This is a local search procedure using the neighborhood $\mathcal{N}^k_{BS}(x)$. One can start the local search even with an elementary tour that is *not* resource feasible. If its neighborhood contains at least one resource feasible tour, the search can be continued.

5. Relaxations

We now describe DP relaxations for the MTDP, which can be used to compute lower bounds for the exact DP. We classify the relaxations into two groups: The first group relaxes elementary and the second relaxes resource feasibility. All presented relaxations can be used in the forward and backward DP in the same manner as in the exact DPs. Furthermore, a relaxed backward DP provides bounds (leading to another bounding procedure to be used in Step 8 of Algorithm 1) for the forward DP, and vice versa.

5.1. Relaxing Elementary

The ng-tour relaxations were first introduced by Baldacci *et al.* (2011a) for the VRP, and the ngL-tour relaxations were presented for the TSPTW by Baldacci *et al.* (2011b). Both are families of relaxations (parameterized) and allow some non-elementary tours. For the MTDP, an *ng-tour* requires the existence of a feasible schedule. We adapt the ng-tour relaxation for computing a least-duration ng-tour from o to d of length $n+1$. Clearly, if the computed tour is elementary it constitutes an optimal solution to the MTDP.

A specific ng-tour relaxation requires the definition of sets $N_i \subseteq V$ with $i \in N_i$, one for each node $i \in V$. A *forward ng-path* $P = (o, ..., i)$ is a non-necessary elementary path starting at node o and ending at node i and all nodes are visited within their time window. P defines a *forward ng-label* (k, i, S_{ng}, T_i), where k is the path length and S_{ng} is a (generally proper) subset of the visited nodes. The vector T contains the same resources as in the exact formulation, and the same resource windows and REFs for the resources are used as in the exact DP. The key point of this relaxations is the update of the set S_{ng}, which differs from the set S in the exact formulation: Forward propagation of a label (i, k, S_{ng}, T_i) along an arc $(i, j) \in A$, i.e., towards node j, produces the new label $(j, k+1, S'_{ng}, T_j)$, where $T_j = f_{ij}(T_i)$ and $S'_{ng} = (S_{ng} \cup \{j\}) \cap N_j$. The interpretation is that the new label forgets the visited nodes that are not in the set N_j so that cycles become possible. For the sake of simplicity, we skip the index $_{ng}$ and write S instead of S_{ng} in the following.

The propagation criteria and fathoming rules need to be altered also: The label (k, i, S, T) can be propagated to a node j, if the arc (i, j) exists, $j \notin S$, $T_i^{time} < LDT(i, j)$, and $k > |\pi(j)|$. The first two criteria are identical to the exact DP. The third criterion replaces the feasibility Rule 1, which is generally invalid for ng-tours. The fourth stipulates that at least $|\pi(j)|$ nodes must precede the node j. Note that we cannot require $\pi(j) \subseteq S$ (as in the exact case) because in the ng-tour relaxation all nodes $\pi(j)$ may have been visited already even if $\pi(j) \nsubseteq S$.

Next, we consider the fathoming rules for eliminating labels: An ng-label dominates another ng-label if it consumes less resources and it has more options to be propagated. For the MTDP, it means:

Rule *ng* **1.** *(ng-Domination)*
Let $L = (k, i, S, T)$ and $L' = (k, i, S', T')$ two ng-labels with identical path length k, $S \subseteq S'$ and the identical last node i. Let $P = P(L)$ and $P' = P(L')$ be the respective paths. If $T_i \leq T_i'$ (component-by-component), any feasible extension of P' towards d is also a feasible extension of P with non-smaller tour duration. Hence, L' can be discarded.

Note that Rule 2 for bounding remains applicable as in the exact DP. However, additional bounding possibilities arise when a weaker relaxation provides bounds for the relaxation under consideration. Related aspects are discussed in Section 5.4.

The *ngL-tour relaxation* is a restriction of the ng-tour relaxation. In addition to the requirements for ng-tours, it guarantees that a specific subset of nodes is visited once and only once in a given order. Such a sequence of nodes results from the preprocessing phase, e.g., by the determination of a chain of precedences $(v_0, v_1, v_2, \ldots, v_p)$ with maximum length p. Moreover, we also determine those nodes $V_i \subset V$, which can be visited between each two consecutive nodes v_ℓ and $v_{\ell+1}$ of the chain. Step 4 in Algorithm 1 then loops over all $j \in V_\ell, j \notin S$ if the partial path defining the label (k, i, S, T_i) has already visited the node v_ℓ, but not the node $v_{\ell+1}$. For a more detailed description, we refer to (Baldacci *et al.*, 2011a).

The quality of the lower bound computed by the ng-tour and ngL-tour relaxations strongly depends on the choice of the sets N_i. Clearly, larger neighborhoods produce tighter lower bounds, but they also increase the computation time of the relaxed DP. Therefore, we limit the number of neighbors of a node by a constant Δ. We use two methods to determine *promising* neighborhoods: The first method simply fills the neighborhood N_i of node i with the Δ-nearest nodes which are reachable from i and from which i is reachable. The second method is based on a dynamical augmentation of the ng or ngL neighborhoods, which was applied for the capacitated arc-routing problem by Bode and Irnich (2013) and for the delivery man problem by Roberti and Mingozzi (2013). We start with a small or empty neighborhood $(N_i)_{i \in V}$ and solve the corresponding relaxed DP. If a node i is visited more than once in the optimal ng-tour computed, we add this node to the neighborhood of all nodes, which occurs in the cycle(s) containing node i. We formalize this procedure in Algorithm 2.

Algorithm 2: Dynamic Augmentation of *ng* Neighborhoods

for $i \in V$ **do** Set $N_i := \{i\}$
repeat
  Solve the relaxed DP with neighborhoods $(N_i)_{i \in V}$; let P be the resulting ng-tour
  Detect all cycles C_j for nodes $j \in V$ visited more than once in P
  Set $\mathcal{C} := \{(j, C_j) : |N_\ell| < \Delta \text{ for all } \ell \in C_j\}$
  Select a vertex disjoint subset of $\mathcal{C}' \subseteq \mathcal{C}$ of cycles
  for $(j, C_j) \in \mathcal{C}'$ *and* $\ell \in C_j$ **do**
    Set $N_\ell := N_\ell \cup \{j\}$
until $\mathcal{C} = \varnothing$
Result: The neighborhoods $(N_i)_{i \in V}$

5.2. Relaxing Resource Feasibility

In this section, we present two new relaxations for the MTDP, the $2res$ and the $1res$ relaxation. The $2res$ *relaxation* relaxes the resource feasibility of a tour by omitting the resource T_i^{time}. A path $(o, .., i)$ defines a $2res$*-label* (k, i, S, T_i), where the vector T_i only contains the two interdependent resources T_i^{dur} and T_i^{help}. As before, k is the path length, i the last visited node, and S the set of all visited nodes. We can use the same propagation criteria as in the exact DP (Step 4 of Algorithm 1), but the fathoming rules must be altered. The feasibility Rule 1 is not applicable because we do not keep track of the resource T_i^{time}. Instead, we use the following weaker feasibility rule:

Rule $2res$ **1.** *(2res-Feasibility)*
A label (k, i, S, T_i) *cannot lead to a feasible solution, if there exists a node* $j \in V \setminus S$ *with* $a_o + T_i^{dur} > LDT(i, j)$. *Hence, such a label can be discarded.*

This rule estimates the tour duration to node j by supposing that its starting time would be the earliest possible time a_o and no time windows would imply waiting. The bounding Rule 2 and the dominance Rule 3 are applicable as in the exact case.

The $1res$ *relaxation* relaxes the $2res$ relaxation. The resource T_i^{help} is fixed to its smallest possible value $-b_o$. A path $(o, .., i)$ defines a $1res$*-label* (k, i, S, T_i^{dur}), where k is the path length, i the last visited node, S the set of all visited nodes, and T_i^{dur} is a lower bound of the path duration. The label propagation considers only the single resource T_i^{dur}, so that (2b) reduces to $T_j^{dur} := f_{ij}^{dur}((k, i, S, T_i)) := \max\{T_i^{dur} + t_{ij}, a_j - b_o\}$. We can apply the same propagation criteria and fathoming rules as for the $2res$ relaxation.

5.3. Combined Relaxations

The $1res$ or $2res$ relaxations can be combined with the ng-tour or ngL-tour relaxations. We can use the propagation and dominance rules as in the ng-tour relaxation except for the feasibility criterion that is based on the resource T_i^{time}. Instead, we forbid to propagate a *combined label* (k, i, S, T_i) (with $T_i = T_i^{dur}$ or $T_i = (T_i^{dur}, T_i^{help})$) to a node j if $a_o + T_i^{dur} > LDT(i, j)$. Figure 2 shows the hierarchy of the relaxations, in which one relaxation stands below another if the latter is a proper relaxation of the first.

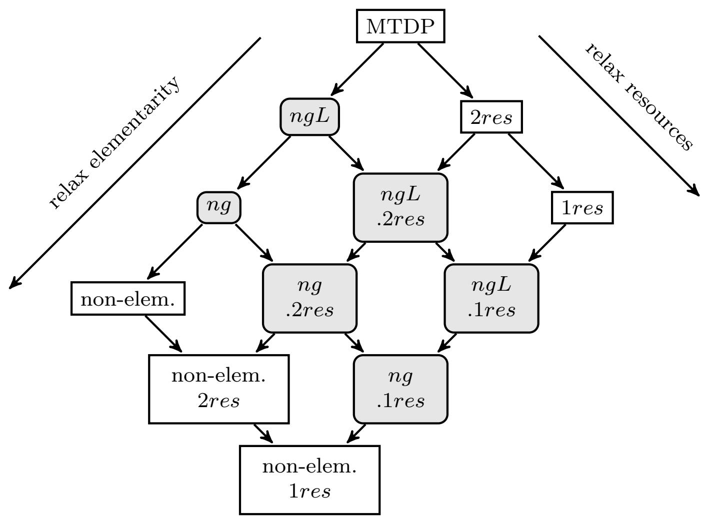

Figure 2: Hierarchy of the MTDP Relaxations. Shaded boxes are families of relaxations.

5.4. Relaxation-based Bounding

In this section, we adapt the *bounding procedure* introduced by Christofides *et al.* (1981) for the MCTP, and later applied by Baldacci *et al.* (2011b) for the TSPTW. As shown in Proposition 1, the concatenation of a forward and a backward path, given by a forward and a backward label, can easily be checked regarding feasibility. The duration of the resulting tour can be computed in constant time.

Instead of concatenating two exact labels, we can concatenate an exact and a label from a relaxation or even two relaxed labels to obtain a non-necessarily feasible TSPTW tour. Let a forward label (k, i, S, T_i) be given. The set of backward labels at stage $k' = |V| - k$, either exact or from a relaxation, is denoted by $\mathcal{L}^{bw}_{k'}$. The following lower bound on the tour duration of the concatenation is a direct consequence of Proposition 1:

$$lb^{dur}(k, i, S, T_i) := \min_{\substack{(k',i,S',T'_i)\in\mathcal{L}^{bw}_{k'}:\\ k'=|V|-k, S\cap S'=\{i\}, T_i\le T'_i}} \max\left\{T_i^{dur} - {T'}_i^{dur}, T_i^{help} - {T'}_i^{help}\right\} \tag{4}$$

Note that the part (i) of Proposition 1 provides the preconditions $k + k' = |V|, S \cap S' = \{i\}$, and $T_i \le T'_i$ for forming a feasible TSPTW tour, while part (ii) identifies the term $\max\{T_i^{dur} - {T'}_i^{dur}, T_i^{help} - {T'}_i^{help}\}$ providing the tour duration of the concatenation. The following modifications have to be made if the forward label (k, i, S, T_i) or the backward labels (k', i, S', T'_i) result from a relaxation:

- **ng, ngL:** S and/or S' have to be replaced by S_{ng} and/or S'_{ng}.
- **2res:** $T_i \le T'_i$ has to be replaced by $(T_i^{dur}, T_i^{help}) \le ({T'}_i^{dur}, {T'}_i^{help})$.
- **1res:** $T_i \le T'_i$ has to be replaced by $T_i^{dur} \le {T'}_i^{dur}$ and there is no second term in the maximum.

For combined relaxations (see Section 5.3), all of the associated modifications apply. Now we can define a bounding rule:

Rule 4. *(Bounding)*
Let UB be an upper bound for the MTDP and $lb^{dur}(k, i, S, T_i)$ be defined as in equation (4). Any label (k, i, S, T_i) with $lb^{dur}(k, i, S, T_i) \ge UB$ cannot lead to an improved solution and can be discarded.

The role of forward and backward labels can be swapped, i.e., a single backward label (k', i, S', T'_i) is given and all compatible forward labels from a relaxation provide the lower bound $lb^{dur}(k', i, S', T'_i)$. We leave the obvious formulation of the corresponding equation and bounding rule to the reader.

A final remark concerns Algorithm 2 for the dynamic neighborhood augmentation, i.e., for finding effective neighborhoods $(N_i)_{i\in V}$ for the *ng*- and *ngL*-tour relaxations. If we alternate between forward and backward DP, every time Step 2 of Algorithm 2 is executed, we can use the labels from the last iteration for bounding in the current iteration. Indeed, if the last iteration was a backward (forward) DP, its labels refer to a proper relaxation of the current forward (backward) DP. This trick drastically reduces the computation for the iterated solution of the DPs, see Section 7.

6. Improving Bounds

Also the *penalty method* was first applied by Christofides *et al.* (1981) for solving the MCTP and was later used by Baldacci *et al.* (2011b) for the TSPTW. Its purpose is to further improve lower bounds resulting from relaxations that relax elementarity. We will briefly point out specifics of the penalty method when applied to (combined) *ng*-tour or *ngL*-tour relaxation for the MTDP.

Let $V' = V \setminus \{o, d\}$ and $\mathcal{H}$ be the set of all tours generated by a given relaxation. By d_k we denote the minimum duration of the tour $k \in \mathcal{H}$ and by δ_{ik} the number of times an *ng*-tour $k \in \mathcal{H}$ visits node $i \in V'$.

The penalty method uses penalties λ_i associated with each node $i \in V'$ in order to modify the objective value of non-elementary tours. For ease of notation, we define $\lambda_o = \lambda_d := 0$ and $\Lambda := \sum_{i\in V} \lambda_i$. The objective of the MTDP is tour duration minimization so that we modify the REFs f_{ij}^{dur} and f_{ij}^{help} by subtracting the penalty λ_j. To be consistent, the resource windows defined by (1) need to be redefined as $T_o^{dur} \in [0, \infty)$, $T_i^{dur} \in (-\infty, \infty)$ for $i \in V'$, $T_d^{dur} \in (-\infty, UB-\Lambda]$ and $T_i^{help} \in (-\infty, \infty)$ for $i \in V \setminus \{o\}$. The resource bounds

$T_o^{help} \in [-b_o, \infty)$ remain unchanged. The altered tour duration of any tour $k \in \mathcal{H}$ is then $d_k - \sum_{i\in V} \delta_{ik}\lambda_i$, which is a modification by Λ for all elementary tours. For a given penalty vector $\lambda \in \mathbb{R}^{|V|}$, the value

$$lb(\lambda) := \min_{k\in\mathcal{H}}\{c_k - \sum_{i\in V} \delta_{ik}\lambda_i + \Lambda\}$$

is a valid lower bound for the MTDP. The following Lagrangian dual problem can be solved to find a tight lower bound:

$$(LD) \qquad\qquad z_{LD} := \max\{lb(\lambda) : \lambda \in \mathbb{R}^{|V|}\}$$

We use subgradient optimization and column generation to solve problem (LD): The standard subgradient algorithm, presented as Algorithm 5 in Section A.2 of the Appendix is run for $\max_{iter}$ iterations.

The column-generation algorithm for the TSPTW was suggested by Baldacci *et al.* (2011b). The linear relaxation of the master program is the following problem:

$$\min \quad \sum_{k\in\mathcal{H}} d_k y_k \tag{5a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{H}} \delta_{ik} y_k = 1 \quad \text{for all } i \in V' \tag{5b}$$

$$\sum_{k\in\mathcal{H}} y_k = 1 \tag{5c}$$

$$y_k \geq 0 \quad \text{for all } k \in \mathcal{H} \tag{5d}$$

Problem (5) provides an identical bound as (LD) and can be solved by column generation (see Desaulniers *et al.*, 2005; Lübbecke and Desrosiers, 2005) using a *restricted master program* (RMP), which is the restricted version of (5) containing only the tours found in the subgradient optimization and the tour producing the upper bound UB, see Section 4. Let λ'_i be the dual price of the constraints (5b) of the RMP and let λ'_d be the dual price to the convexity constraint (5c). Moreover, we define $\lambda'_o := 0$. The column-generation algorithm alternates between the LP re-optimization of the RMP and the column-generation pricing problem that adds additional variables (=columns) to the RMP. The pricing problem asks for a tour with negative reduced cost (=tour duration) $d_k - \sum_{i\in V} \delta_{ik}\lambda'_i$. This requires the solution of the same (relaxed) DP as in the subgradient optimization, but with λ replaced by λ', and Λ replaced by $\Lambda' := \sum_{i\in V} \lambda'_i$. The same bounding possibilities as discussed in Section 5.4 remain applicable. This includes the use of the lower bounds $lb(k, i, S, T_i)$ (see equation (4)), to which Λ' has to be added at the end. The only crucial point is that REFs and resource bounds need to be altered as described at the beginning of this section.

7. Computational Results

This section presents the computational results of the DP-based solution approaches for the MTDP. All computations were performed on a standard PC with an Intel(R) Core(TM) i7-2600 at 3.4 GHz processor with 16 GB of main memory. Algorithms were coded in C++ and compiled in release mode with MS-Visual Studio 2010. The callable library of CPLEX 12.5 was used to iteratively re-optimize the RMP in the column-generation algorithm of Section 7.4. We tested our algorithm on the instances of Potvin and Bengio (1996), Gendreau *et al.* (1998), Ohlmann and Thomas (2007), and Ascheuer (1995). The first three sets can be obtained from `http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm`. These three classes feature instances with x-y coordinates. Travel times are first computed as truncated integer Euclidean distances and then modified to satisfy the triangle inequality. The `Potvin+Bengio` benchmark set contains 30 instances with up to 46 nodes. The instances of the `Gendreau` benchmark consist of 120 instances divided in 24 subgroups of five instances each. The five instances within the same group have an identical number of nodes (between 21 and 101) and comparable time window widths (ranging from 100 to 200). Since the `Gendreau` instances have large variations in size, we divided them into two groups. `Gendreau large` includes 45 instances with more than 80 nodes, and `Gendreau small` the 75 smaller instances. The `Ohlmann+Thomas` benchmark extend the `Gendreau` benchmark regarding an increasing number of nodes (between 150 or 200) and time windows widths (from 120 to 160).

The `Ascheuer` benchmark set is available at `http://ftp.zib.de/pub/mp-testdata/tsp/atsptw/index.html` and consists of 50 asymmetric TSPTW instances with up to 233 nodes. Travel times are integer and satisfy the triangle inequality. As in Dash *et al.* (2012), we divide this class into 32 easy instances and 18 hard instances.

Before we apply our DP algorithms, we preprocess the instances according to Section 3. Note that we set $a_d = 0$ (earliest service at the destination) before preprocessing so that the tour duration can vary due to different arrival times at the destination. Furthermore, the `Ascheuer` instances are originally constrained by the origin time window $[a_o, b_o] = [0, 0]$, which we enlarge to $[0, 1000]$.

7.1. Computation of Upper Bounds

Since local search times become longer with increasing values of k for the Balas-Simonetti neighborhoods $\mathcal{N}^k_{BS}$, we iterate the local search in a *variable neighborhood descent* (VND) heuristic, where we use the values with k = 7, 9, 11, and 13. As an initial tour we use the known optimal solutions to the TSPTW or the MCTP, which can be obtained at `http://iridia.ulb.ac.be/~manuel/tsptw-instances`. Table 1 shows the aggregated results over each group of instances. The column *improved* reports the number of times the initial solution value has been improved, the column *#OPT* shows the number of times an optimal solution was computed (if an optimum is known). The maximum and average *GAP* is reported in the next two columns. The last three columns give the minimum, maximum, and average solution time.

The VND improves the objective value of the input tour for nearly all instances except for the benchmark by `Ascheuer`, in which the VND improves approximately half of the known solutions. Furthermore, for instances with a known optimal solution value (computed by us, see Section 7.4), the VND returns an optimal solution in approximately 80% of the cases. Note that the *GAP* is calculated only for those instances for which an optimal solution is known.

			GAP [%]		Time [s]		
Instances	improved	#OPT	max	ø	min	max	ø
`Potvin+Bengio`	27/30	17/28	7.4	1.0	0.1	7.7	1.1
`Ascheuer easy`	13/32	27/32	0.1	0.1	0.1	2.4	0.5
`Ascheuer hard`	11/18	17/18	1.1	0.1	0.9	8.3	3.2
`Gendreau small`	73/75	58/72	12.3	0.5	0.2	6.1	1.8
`Gendreau large`	44/45	30/32	3.5	0.1	1.3	16.5	6.6
`Ohlmann+Thomas`	25/25	1/1	0.0	0.0	10.8	105.5	35.9
Overall	193/225	151/183					

Table 1: Aggregated Results for Upper Bounds computed with the Balas-Simonetti Neighborhood

7.2. Comparison of the Relaxations

Now we compare the different relaxations introduced in Section 5 regarding their computation times and lower bounds. In pre-tests we found that the ngL-tour relaxations almost always outperform the corresponding ng-tour relaxations. Therefore, we limit our study to the $2res$, $1res$, ngL, $ngL.2res$, and $ngL.1res$ relaxations. Recall that the last three relaxations are parameterized with neighborhoods $(N_i)_{i \in V}$ and a sequence of nodes in precedence. For the comparison, we use a priori generated neighborhoods N_i with the $\Delta = 10$ closest neighbors to node i. The next section analyzes the impact of varying sizes and generation methods of the neighborhoods. The node sequence in precedence is arbitrarily chosen as a longest path in the precedence graph.

We set a hard time limit of 600 seconds for the computation of the lower bound as well as for the computation of the exact DP. The relaxation uses a forward DP, while the exact DP is computed backwards so that the relaxation provides valid bounds according to (4). Table 2 shows the aggregated results for all relaxations and the different benchmark sets. Column *#LB* displays the number of times a lower bound is computed, i.e., the relaxation is solved within the time limit. The next columns show the average and maximum GAP between the computed lower bound and the best known solution, followed by the average and maximum time. Column $\#SOL$ gives the number of times the exact DP was solved, and the following two columns report the average and maximum time for this step. The last two columns show the overall solution time for each instance on average and the maximum. In order to provide a fair comparison, the maximum and average gaps and times are taken only over those instances for which bounds were computed by all relaxations. Similarly, the times for solving the exact DP and the total time include only those instances solved by all five variants. We briefly summarize the results: The $2res$ and $1res$ relaxations need

		#LB	GAP [%]		Time LB [s]		#SOL	Time exact [s]		Time all [s]	
Instances	Relaxation		ø	max	ø	max		ø	max	ø	max
	$2res$	18	0.9	5.9	77.2	556.0	18	0.1	1.3	77.2	556.0
`Potvin`	$1res$	19	3.6	9.7	23.6	289.7	19	0.1	1.4	23.7	289.7
`+Bengio`	ngL	21	1.7	9.4	13.5	186.2	21	0.5	4.5	14.0	188.7
($n = 30$)	$ngL.2res$	30	2.4	13.4	0.2	1.3	21	1.2	15.2	1.4	16.4
	$ngL.1res$	30	6.0	19.1	0.1	0.3	21	2.0	27.4	2.1	27.7
	$2res$	26	0.0	0.4	1.2	22.8	26	0.0	0.0	1.2	22.8
`Ascheuer`	$1res$	27	0.9	6.9	1.0	18.3	27	0.3	6.1	1.1	18.3
`easy`	ngL	32	3.1	30.3	0.2	2.4	28	1.9	27.7	2.1	27.8
($n = 32$)	$ngL.2res$	32	3.1	30.3	0.1	0.7	28	2.0	28.0	2.1	28.0
	$ngL.1res$	32	4.1	30.3	0.1	0.9	27	7.9	111.6	8.5	111.7
	$2res$	10	0.0	0.1	43.5	248.9	10	0.0	0.0	43.5	248.9
`Ascheuer`	$1res$	10	1.1	6.7	34.4	233.6	10	4.2	13.9	38.6	245.0
`hard`	ngL	18	1.0	5.9	7.5	41.0	14	3.1	10.4	10.7	41.4
($n = 18$)	$ngL.2res$	18	1.0	6.0	1.3	4.0	13	3.4	12.0	4.8	15.7
	$ngL.1res$	18	1.7	12.5	1.5	4.0	13	3.9	13.1	5.4	15.0
	$2res$	37	0.2	3.4	26.7	335.7	37	0.0	0.2	26.7	335.7
`Gendreau`	$1res$	42	2.5	8.6	12.5	239.7	42	0.1	2.4	12.6	242.1
`small`	ngL	67	7.9	21.8	13.0	95.4	52	2.8	95.1	15.7	138.3
($n = 75$)	$ngL.2res$	75	8.8	24.9	0.6	2.7	54	2.5	86.7	3.1	89.5
	$ngL.1res$	75	12.8	27.9	0.3	0.9	52	4.2	147.9	4.5	148.9
	$2res$	2	0.0	0.0	317.2	483.1	2	0.0	0.0	317.2	483.1
`Gendreau`	$1res$	2	0.0	0.0	243.6	359.7	2	21.8	35.4	265.4	395.1
`large`	ngL	19	0.0	0.0	6.6	12.5	8	66.7	88.3	73.3	89.0
($n = 45$)	$ngL.2res$	45	0.0	0.0	1.2	1.8	9	70.1	95.4	71.3	95.9
	$ngL.1res$	45	0.0	0.0	1.4	1.5	7	161.3	223.4	162.7	224.8
`Ohlmann`	$ngL.2res$	25	4.3	14.7	94.1	369.0	1	0.1	0.1	52.5	52.5
`+Thomas`	$ngL.1res$	25	4.3	14.7	12.5	23.7	1	0.1	0.1	13.7	13.7
($n = 25$)											
	$2res$	93	0.3	5.9	37.4	556.0	93	0.0	1.3	37.8	556.0
Overall	$1res$	100	2.1	9.7	18.7	359.7	100	1.1	35.4	20.0	395.1
($n = 225$)	ngL	157	4.4	30.3	8.8	186.2	123	3.5	95.1	12.4	188.7
	$ngL.2res$	225	4.9	30.3	0.5	4.0	125	3.7	95.4	4.2	95.9
	$ngL.1res$	225	7.6	30.3	0.4	4.0	120	8.2	223.4	8.7	224.8

Table 2: Aggregated Results for Different Relaxations

much computation time and, therefore, they solve the fewest relaxations. If they are able to compute lower bounds, however, the exact DP is always solved. In comparison, these two relaxations are too slow and therefore not beneficial.

The $ngL.2res$ and $ngL.1res$ relaxations are able to solve the relaxations on all problem instances and need comparably small time to terminate. When using the ngL-tour relaxation, its time is on average approximately 20 times higher compared to the $ngL.2res$ and $ngL.1res$ relaxations. Moreover, none of the three relaxations $2res$, $1res$, and ngL is able to compute a lower bound for the `Ohlmann+Thomas` instances. Consequently, rows for these relaxations are omitted in Table 2.

Comparing ngL with $ngL.2res$, gaps are slightly in favor of the ngL-tour relaxation. However, the exact DP in combination with the $ngL.2res$ relaxation is able to solve the largest number of instances to optimality, 126 out of 225, which is three more than with the ngL-tour relaxation. Also the overall computation times (relaxation plus exact DP) are in favor of the $ngL.2res$ relaxation. Therefore, the $ngL.2res$ relaxation seems to be the best compromise.

7.3. Comparison of ngL Neighborhood Sizes

Next, we analyze the impact of different neighborhoods $(N_i)_{i \in V}$ for the $ngL.2res$ relaxations. Recall from Section 5.1 that the neighborhoods can either be generated a priori (static version) or be generated using Algorithm 2 (dynamic version). We compare the different maximal sizes of the neighborhoods given by $\Delta = 8, 10, 12$, and 14. Pre-tests have shown that smaller or larger sizes are not beneficial (in a stand-alone algorithm not using a penalty method) because either the lower bounds are to weak or the computation times are too large. As in the previous analysis, we set a hard time limit of 600 seconds for the computation of the relaxation as well as for the exact DP.

					GAP [%]				Time LB [s]				Time Exact [s]			
ngL.2res	#LB		#SOL		ø		max		ø		max		ø		max	
with	S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D
`Potvin+Bengio` instances (n=30)																
$\Delta = 8$	30	30	22	22	6.1	5.2	21.9	20.5	0.7	1.9	9.4	10.5	5.7	5.3	96.8	94.5
$\Delta = 10$	30	30	22	22	5.3	4.3	20.0	17.2	1.8	5.9	17.7	42.9	4.7	4.4	81.3	80.9
$\Delta = 12$	30	30	22	22	4.9	3.7	19.6	16.0	4.7	20.9	39.2	127.1	3.9	3.6	66.6	65.3
$\Delta = 14$	30	28	22	22	4.7	3.3	19.6	15.5	26.6	69.4	482.7	496.1	3.1	2.7	50.8	46.8
easy `Ascheuer` instances (n=32)																
$\Delta = 8$	32	32	27	27	3.1	1.6	34.2	15.0	0.1	1.1	0.1	10.6	2.6	0.4	35.4	3.9
$\Delta = 10$	32	32	27	27	2.8	2.5	30.3	25.9	0.1	1.1	0.4	1.7	1.9	0.2	28.2	2.0
$\Delta = 12$	32	32	27	28	2.6	1.9	29.5	21.8	0.2	1.1	1.6	5.0	0.5	0.2	6.3	1.8
$\Delta = 14$	32	32	27	28	1.6	1.1	15.0	9.3	1.1	2.9	10.6	18.1	0.4	0.1	3.9	1.8
hard `Ascheuer` instances (n=18)																
$\Delta = 8$	18	18	13	14	1.4	1.2	7.0	4.9	0.3	2.5	0.7	9.3	2.8	1.6	12.8	8.4
$\Delta = 10$	18	18	13	14	1.3	1.1	6.0	4.9	0.8	7.2	2.0	25.2	2.8	1.5	12.8	8.3
$\Delta = 12$	18	18	13	14	1.2	1.0	5.6	4.9	2.7	21.4	8.3	92.4	2.8	1.5	12.7	8.2
$\Delta = 14$	18	18	13	14	1.1	1.0	4.9	4.9	9.7	91.5	33.0	351.8	2.7	1.4	12.7	8.1
`Gendreau small` instances (n=75)																
$\Delta = 8$	75	75	51	54	10.5	9.2	29.4	27.7	0.7	3.0	4.5	16.6	21.3	13.9	149.8	113.3
$\Delta = 10$	75	75	53	54	9.7	8.1	28.3	27.1	1.6	9.5	8.3	78.0	14.3	12.6	112.4	106.7
$\Delta = 12$	75	75	54	55	8.8	7.2	28.3	27.1	4.0	34.0	24.2	273.5	13.1	10.8	102.2	90.2
$\Delta = 14$	75	74	55	55	8.2	6.5	27.1	25.9	11.9	96.6	88.9	598.0	12.7	10.4	101.9	89.4
`Gendreau large` instances (n=45)																
$\Delta = 8$	45	45	8	9	4.6	4.6	21.8	21.8	2.1	11.6	8.8	42.8	14.4	2.8	58.6	22.7
$\Delta = 10$	45	45	9	9	4.6	4.6	21.8	21.8	5.0	35.2	16.7	136.8	13.0	2.8	54.8	22.6
$\Delta = 12$	45	43	10	10	4.6	4.6	21.8	21.8	14.0	100.5	52.7	327.0	12.5	2.7	54.5	21.8
$\Delta = 14$	45	37	10	10	4.6	4.6	21.8	21.8	38.9	241.8	148.4	596.3	12.3	2.7	54.3	21.3
`Ohlmann+Thomas` instances (n=25)																
$\Delta = 8$	25	23	1	1	0.3	0.3	0.5	0.5	17.4	19.4	21.1	34.6	0.1	0.1	0.1	0.1
$\Delta = 10$	25	16	1	1	0.3	0.3	0.5	0.5	40.7	35.7	52.5	67.1	0.1	0.1	0.1	0.1
$\Delta = 12$	24	8	1	1	0.3	0.3	0.5	0.5	106.5	237.4	125.4	470.1	0.1	0.1	0.1	0.1
$\Delta = 14$	16	2	1	1	0.3	0.3	0.5	0.5	336.1	237.4	345.0	470.1	0.1	0.1	0.1	0.1
`Overall` (n=225)																
$\Delta = 8$	200	200	122	127	4.8	4.1	0.6	27.7	0.6	2.7	9.4	42.8	5.9	3.5	149.8	113.3
$\Delta = 10$	200	200	125	128	4.3	3.5	1.4	27.1	1.4	8.1	17.7	136.8	4.5	3.1	112.4	106.7
$\Delta = 12$	200	198	127	130	4.0	3.0	3.8	27.1	3.8	25.3	52.7	326.9	3.8	2.7	102.2	90.2
$\Delta = 14$	200	189	128	130	3.6	2.7	14.7	25.9	14.7	78.2	482.8	598.0	3.4	2.3	101.9	89.4

Table 3: Aggregated Results comparing different ngL Neighborhoods

Table 3 shows the aggregated results for the different parametrization. Herein, S denotes the static generation of the neighborhood and D the dynamic augmentation. The first column $\#LB$ denotes the number of times the respective relaxation was solved, followed by the number $\#SOL$ of times the exact DP was solved. The next columns report the average and maximal GAP, the average and maximum time to compute lower bounds and the average and maximum time to solve the exact DP. As before, to ensure a fair comparison, the average and maximum gaps refer to those instances that were solved by all methods. Similarly, the reported times for the exact DP refer to the instances solved to optimality by all variants.

Clearly, a larger Δ requires more computation time for the relaxation leading to generally smaller gaps and, in turn, smaller times for the exact DP. The static neighborhood generation needs less time than the dynamic version because in the latter case more relaxed DPs have to be solved. On the other hand, the dynamic neighborhood augmentation provides smaller gaps, more solutions and better solution times for the exact DP. For a few instances, the static approach solves the exact DP faster than the dynamic one. In these cases, the time windows impose a minimum for the lower bound of the MTDP. Since the dynamic neighborhood augmentation does not necessarily fill all neighborhoods $(N_i)_{i \in V}$ to the maximum size Δ opposed to the static augmentation, the latter may provide better lower bounds.

7.4. Results of the Column-Generation Algorithm and Exact Dynamic Program

In this section, we report results of the column-generation algorithm (see Section 6) and the successive application of the exact DP. Recall that the column-generation method provides optimal penalties $(\lambda_i^*)_{i \in V}$. Both the lower bound LB and the solution help solving the exact DP faster: On the one hand, the solution of each pricing problem provides a lower bound LB for the MTDP, and this bound may suffice to close the gap to the upper bound computed with the VND (see Section 4). On the other hand, the solution itself, i.e., the labels produced by the $ngL.2res$ relaxation can be exploited for bounding using the bounds (4). We describe the overall approach in Algorithm 3.

Algorithm 3: Overall Algorithm

1 CALL preprocessing // → $(A, \pi(i), \sigma(j), EAT_{ij}, LDT_{ij}, [a_i, b_i])$
2 CALL VND with Balas-Simonetti neighborhoods // → (UB, P^*)
3 COMPUTE static $ngL.2res$ neighborhoods $(N_i)_{i \in V}$ of size $\Delta_1 = 3$ // → (N_i)
4 CALL subgradient method with $\max_{iter} = 10$ // → $(LB, \lambda_i^*, ngL$ tours)
5 CALL column-generation algorithm // → (LB, λ_i^*)
6 CALL dynamic augmentation of neighborhoods with penalties $(\lambda_i^*)_{i \in V}$ and $\Delta_2 = 14$ // → (LB, N_i)
7 **while** $LB < UB$ **do**
8   SET $B = \min\{\lceil 1.01 \cdot LB \rceil, UB\}$
9   CALL exact DP with upper bound B; bounding with labels of Step 6 // → $(UB, P^*$ or failed)
10   **if** *DP failed* **then** SET $LB = B + 1$

Result: optimal MTDP tour P^* with minimum tour duration UB

The results of the last two subsections suggest the $ngL.2res$ relaxation to be used in the subgradient and the column-generation algorithms as well as the dynamic augmentation of the neighborhoods (Steps 4, 5, and 6). Pre-tests have shown that with small-sized neighborhoods N_i a significantly higher number of column-generation iterations is possible. Herewith, lower bounds generally improve more due to good penalties compared to larger-sized neighborhoods. We have obtained the best results with a neighborhood size of $\Delta_1 = 3$ (Step 3). To improve the bounds of this relaxed DP, we augment the neighborhood dynamically (Algorithm 2 called in Step 6) after good penalties are found up to a neighborhood size of $\Delta_2 = 14$.

Furthermore, preliminary tests revealed that the running time of the exact DP is strongly impacted by the quality of the upper bound UB: A relatively weak upper bound causes that bounding procedures almost always to fail so that the DP algorithm will not terminate (or terminate with an 'out of memory' message) due to an enormous set of labels that has to be generated and stored. Therefore, it is computationally advantageous to try tentative upper bounds B whenever the gap $UB - LB$ is relatively large. The loop (Steps 8–10), iteratively tries to increase the tentative upper bounds B, where the gap between B and LB never exceeds 1% (plus 1). When the exact DP is called with a tight upper bound, more labels can be bounded and the computation time is reduced. If a tentative bound is too small, the DP will fail in the sense that no $ngL.2res$ tour is computed. In this case, however, one knows that the tentative upper bound is in fact a lower bound (Step 10). We set a hard time limit of one hour for the computation of the lower bounds by the column-generation algorithm as well as for the dynamic neighborhood augmentation and the computation of the exact DP.

For the ease of presentation, Algorithm 3 does not detail all possible termination points. Indeed, if any of the relaxed DPs terminates with a solution that is an elementary tour with a feasible schedule, this tour constitutes an optimal solution to the MTDP. Therefore, we perform this kind of check in Steps 4, 5, and 6.

Moreover, if the rounded up value of any lower bound equals UB (computed by the subgradient or column-generation algorithm or Algorithm 2), the tour that has produced UB is optimal. Hence, Algorithm 3 is stopped whenever this happens for a valid upper bound UB (not for tentative upper bounds B).

Tables 4–6 show the results of Algorithm 3 aggregated over each instance group. Results for the `Ohlmann+Thomas` instances are left out, since Algorithm 3 did not solve additional instances compared to Section 7.3. In each table, n denotes the number of remaining instances to be solved. In turn, $\#SOL$ denotes the number of instances solved to optimality. The average and maximum gap as well as the average and maximum solution time are reported for each of the Steps 5, 6, and 9 of Algorithm 3.

Table 4 shows the results of Algorithm 3 up to Step 5. The column-generation penalty method is able to solve 108 of the 200 instances. Results for the remaining 92 instances are then presented in Table 5, where additionally the dynamic neighborhood augmentation procedure (Algorithm 2) is applied making use of the best penalties $(\lambda_i^*)_{i \in V}$ computed before. Another 25 instances are now solved so that for the remaining 67 instances the exact DP algorithm is invoked (Steps 8–10). The results of these final computations are summarized in Table 6. Another 46 instances are solved by the exact DP. In summary, the overall algorithm is able to solve 179 out of 200 instances.

			GAP [%]		Time [s]	
Instances	n	#SOL	ø	max	ø	max
`Potvin+Bengio`	30	8	2.9	15.1	120.4	1943.3
`Ascheuer easy`	32	20	0.1	0.4	3.6	70.6
`Ascheuer hard`	18	16	0.1	0.4	137.4	625.5
`Gendreau small`	75	41	0.8	7.1	283.4	3308.1
`Gendreau large`	45	23	0.5	4.6	1921.0	3600.0
Overall	200	108	0.9	15.1	582.5	3600.0

Table 4: Aggregated Results of the Overall Algorithm for all Instances

We now highlight some of the results visible in the tables: For the column-generation algorithm, the average GAP over all instances is smaller than one percent and the average solution time is smaller than ten minutes. The neighborhood augmentation procedure further reduces the average GAP of the remaining instances by 0.4 percent. Notably, computation times of this step are generally smaller than those for the column-generation method. Recall that alternating between forward and backward DP using bounds from the preceding iteration is possible for the neighborhood augmentation, but not for the column-generation algorithm due to the iterative modification of penalties in this case.

Moreover, the computation times of the neighborhood augmentation algorithm strongly depend on the size of the instance: For the large instances from the `Gendreau large` group (comprising 81 and 101 nodes) much more computation time is needed than for the other groups.

		Column Generation				Dyn. ng neighb. augment.				
		GAP [%]		Time [s]			GAP [%]		Time [s]	
Instances	n	ø	max	ø	max	#SOL	ø	max	ø	max
`Potvin+Bengio`	22	3.9	15.1	136.8	1943.3	4	3.5	15.1	17.9	100.7
`Ascheuer easy`	12	0.1	0.4	1.2	3.5	2	0.1	0.4	0.2	0.7
`Ascheuer hard`	2	0.2	0.4	313.8	598.6	1	0.1	0.4	68.9	135.8
`Gendreau small`	34	1.6	7.1	310.3	3308.1	16	0.9	7.1	60.5	573.9
`Gendreau large`	22	0.9	4.6	2553.3	3600.0	2	0.8	4.6	872.8	2934.9
Overall	92	1.8	15.1	764.9	3600.0	25	1.4	15.1	236.9	2934.9

Table 5: Aggregated Results of the Overall Algorithm for Instances not solved by Step 5

Finally, detailed results for every instance showing bounds, gaps, and computation times of the different solution procedures included in Algorithm 3 are listed in Section B of the Appendix. An interesting detail in these results is that for only three instances the solution time of the exact DP exceeds four minutes. We can conclude that either an instance can be solved fast by the exact DP, or the solution time may easily exceed one hour.

		Column Generation				Dyn. ng neighb. augment.				Exact DP		
		GAP [%]		Time [s]		GAP [%]		Time [s]			Time [s]	
Instances	n	ø	max	ø	max	ø	max	ø	max	#SOL	ø	max
`Potvin+Bengio`	18	4.7	15.1	166.0	1943.3	4.3	14.0	21.8	100.7	15	110.1	1283.6
`Ascheuer easy`	10	0.1	0.3	1.2	3.5	0.1	0.4	0.2	0.7	10	0.1	0.1
`Ascheuer hard`	1	0.4	0.4	29.0	29.0	0.1	0.1	1.9	1.9	1	0.3	0.3
`Gendreau small`	18	2.0	7.1	512.9	3308.1	1.7	5.5	111.1	573.9	15	221.8	3108.7
`Gendreau large`	20	1.0	4.6	2448.4	3600.0	0.8	4.3	904.1	2934.9	5	92.0	458.2
Overall	67	2.1	15.1	913.9	3600.0	1.9	14.0	305.6	2934.9	46	118.3	3108.7

Table 6: Aggregated Results of the Overall Algorithm for Instances not solved by Steps 5 and 6

During the time we designed the overall algorithm, we tested various other setups and parameters. Furthermore, we also did some longer computational tests. As a result, we were able to compute five more optimal solutions to the MTDP instances. The tables in Section B of the Appendix indicate these optimal solutions with the symbol †.

8. Conclusions

In this paper, we address the *minimum tour duration problem* (MTDP), which is a variant of the TSPTW with the objective of minimizing the time between the departure at the start and the arrival at the destination. The MTDP is a fundamental problem when routing vehicles whose movements are constrained by time windows.

We have presented algorithmic components for a DP-based approach tailored to the MTDP such as relaxed and exact forward and backward DPs, a VND, a dynamic ng neighborhood augmentation procedure, and a penalty method based on subgradient and column-generation algorithms. There exists a plethora of possible algorithmic designs to combine and parameterize these components. Our findings are the following:

First, for bounding purposes, a combined $ngL.2res$ relaxation provides an excellent tradeoff between strength of the resulting bounds and the computational effort. While ngL-based relaxations gradually allow some non-elementary tours, the $2res$ relaxation disregards the time window constraints, but keeps the computation of the tour duration exact. No pure relaxation exclusively relaxing elementarity constraints or resource constraints was found competitive with $ngL.2res$.

Second, a penalty algorithm which penalizes those routes that are non-elementary (and therefore also not Hamiltonian) is often more effective than enlarging the ng neighborhoods which specify the actual ngL-tour relaxation. For this reason, we decided for our approach that the column-generation method precedes the neighborhood augmentation procedure.

Third, with optimized penalties and carefully dynamically augmented ng neighborhoods, excellent lower bounds on the MTDP can be achieved. Our computational analyses show that these lower bounds often suffice to either close the gap or to make the exact DP solution relatively easy.

Fourth, upper bounds produced with the Balas-Simonetti neighborhood-based VND are generally tight (below 1% on average). However, since the exact DP is very sensible regarding the quality of the presented upper bound, it seems computationally advantageous to provide very tight tentative upper bounds for the exact DP. In case that the bound was wrongly chosen, i.e., too small, a repeated solution of exact DPs with increased tight upper bounds is most of the time faster than solving a single exact DP with a low-quality upper bound.

Summarizing the presented computational results, the proposed DP-based approach can solve 90% of the instances with up to 125 nodes from the standard benchmark sets to proven optimality. Of course, time window constraints certainly have a significant impact on the practical hardness of an instance. Therefore, we cannot make this observation a general statement even if some benchmark instances have relative wide time windows. One must keep in mind that the TSPTW is already a (practically) very hard combinatorial problem. For example, the `Ascheuer` benchmark set contained several open instances for more than a decade before it was completely solved by the work of Baldacci *et al.* (2011b). Due to the more involved REFs and resource constraints, the MTDP is certainly an even harder combinatorial problem.

We see one main contribution of the paper at hand in the consistent way that resources and REFs are defined for both forward and backward DP including relaxations. This consistency (see Section 3) is the theoretic background for the later algorithm design. As a result, the overall algorithm solves 183 out of 200 instances from the three benchmark sets `Potvin+Bengio`, `Ascheuer`, and `Gendreau` to optimality.

References

Ascheuer, N. (1995). *Hamiltonian Path Problems in the On-Line Optimization of Flexible Manufacturing Systems*. Ph.D. thesis, Technische Universität Berlin.

Ascheuer, N., Fischetti, M., and Grötschel, M. (2001). Solving the asymmetric traveling salesmann problem with time windows by branch-and-cut. *Mathematical Programming, Series A*, **90**, 475–506.

Baker, E. (1983). An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, **31**, 938–945.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011a). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011b). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*.

Bode, C. and Irnich, S. (2013). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*. (Forthcoming.).

Christofides, N., Mingozzi, A., and Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, **11**, 145–164.

Dash, S., Günlük, O., Lodi, A., and Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(1), 132–147.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F. (1995). Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. Elsevier, Amsterdam.

Dumas, Y., Desrosiers, J., Gelinas, E., and Solomon, M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, **43**(2), 367–371.

Favaretto, D., Moretti, E., and Pellegrini, P. (2006). An ant colony system approach for variants of the traveling salesman problem with time windows. *Journal of information and optimization sciences*, **27**(1), 35.

Gendreau, M., Hertz, A., Laporte, G., and Stan, M. (1998). A generalized insertion heuristics for the traveling salesman problem with time windows. *Operations Research*, **43**(3), 330–335.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, **43**(1), 17–26.

Goel, A. and Vidal, T. (2013). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science*, **forthcoming**.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Kara, I., Koc, O. N., Altıparmak, F., and Dengiz, B. (2013). New integer linear programming formulation for the traveling salesman problem with time windows: minimizing tour duration with waiting times. *Optimization*, **62**(10), 1309–1319.

Langevin, A., Desrochers, M., Desrosiers, J., Gélinas, S., and Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, **23**, 631–640.

Li, J.-Q. (2009). A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Technical report, Working paper, University of California, Berkeley, Berkeley.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Mingozzi, A., Bianco, L., and Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, **45**(3), 365–377.

Ohlmann, J. and Thomas, B. (2007). A Compressed-Annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **19**(1), 80–90.

Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows Part II: Genetic search. *INFORMS Journal on Computing*, **8**(2), 165–172.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., and Rousseau, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, **44**(4), 455–473.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Roberti, R. and Mingozzi, A. (2013). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*. (Forthcoming.).

Savelsbergh, M. (1992). The vehicle routing problem with time windows: Minimizing route duration. *Operations Research Society of America*, **4**(2), 146–154.

Simonetti, N. and Balas, E. (1996). Implementation of a linear time algorithm for certain generalized traveling salesman problems. In W. Cunningham, S. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 316–329. Springer Berlin Heidelberg.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2011). A unifying view on timing problems and algorithms. Technical Report 2011-43, CIRRELT, Montréal, Canada.

A. Algorithms

A.1. Backward Dynamic Programming Algorithm

Algorithm 4 is the backward DP labeling algorithm. Herein, the set $\sigma(i)$ is the set of all nodes $j \in V$ that must succeed i.

Algorithm 4: Backward Dynamic Programming Labeling Algorithm

SET $\mathcal{L}_0^{bw} := \{(0, d, \{d\}, (b_d, UB, \infty))\}$
for $k = 0, 1, \ldots, |V| - 1$ **do**
  for $(k, j, S, T_j) \in \mathcal{L}_k^{bw}$ **do**
    for $(i, j) \in A : i \notin S, \sigma(i) \subseteq S$ **do**
      SET $T_i := f_{ij}^{bw}(T_j)$
      if *FeasibilityCheck(T_i)* **then**
        SET $L_i := (k + 1, i, S \cup \{i\}, T_i)$
        if *BoundingCheck(L_i)* **then**
          ADD L_i to $\mathcal{L}_{k+1}^{bw}$
  CALL Dominance algorithm for $\mathcal{L}_{k+1}^{bw}$
FIND a label $L_o^* = (|V|, o, V, T_o) \in \mathcal{L}_{|V|}^{bw}$ with T_o^{dur} maximal
Result: The path P^* represented by label L_o^*

A.2. Subgradient Optimization Algorithm

Algorithm 5 is the (standard) subgradient algorithm for the resolution of the Lagrangian-dual problem (LD) (see Section 6).

Algorithm 5: Standard Subgradient Optimization Algorithm

SET $t := 0$, $LB := 0$, $\lambda_l^* = \lambda_l^0 := 0 \ \forall l \in V$
for $t < \max_{iter}$ **do**
  CALL DP Algorithm 1 with selected relaxation, modified REFs and penalties $(\lambda_l^t)_{l \in V}$
  FIND a label $L_d^* = (|V|, d, V, T_d) \in \mathcal{L}_{|V|}$ with T_d^{dur} minimal
  if $T_d^{dur} + \sum_{j \in V} \lambda_j^t > LB$ **then**
    SET $LB := T_d^{dur} + \sum_{j \in V} \lambda_j^t$, $\lambda^* := \lambda$
  COMPUTE ng-tour k and coefficients $(\delta_{lk})_{l \in V}$ corresponding to label L_d^*
  for $j \in V$ **do**
    SET $\lambda_j^{t+1} := \left(\lambda_j^t - \left(LB - 1.2\left(LB + \sum_{l \in V} \delta_{lk} \lambda_l^t\right)\right)\right) \cdot (2 - 2\delta_{jk}) \,/ \left(\sum_{l \in V} (2 - 2\delta_{lk})^2\right)$
Result: Lower Bound LB and best computed penalties $(\lambda_l^*)_{l \in V}$

B. Detailed Computational Results

This section reports, for all instances individually, the results produced with Algorithm 3. For each instance group a separate table is presented resulting in nine different tables (Tables 7–15). Herein, $|V|$ denotes the number of nodes of the instance and *UB BS* the upper bound computed with the VND (see Section 4). These are followed by columns *LB*, *GAP* and computation *Time* for the column-generation algorithm (Step 5). The same information is shown for the dynamic augmentation of the neighborhoods (Step 6). The last two columns denote the computation time of the exact DP (Step 9) and the optimal solution value, if it is known. Otherwise, an interval $[LB, UB]$ for the optimum is given. (Note that the lower bound *LB* shown for the open instances may differ from the lower bound resulting from the neighborhood augmentation, since the exact DP improves lower bounds every time Step 10 is reached.) The symbol '†' indicates that an optimal solution was computed by another setup during pre-tests. An entry '−' in a column means that invoking this part of the algorithm was not necessary, while 'TL' denotes that the time limit of 3600 seconds in this part of the algorithm was reached.

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
rc_201.1.txt	20	50352	50352.0	0.0	0.1	–	–	–	–	50352
rc_201.2.txt	26	75633	75633.0	0.0	0.0	–	–	–	–	75633
rc_201.3.txt	32	81605	81605.0	0.0	0.3	–	–	–	–	81605
rc_201.4.txt	26	81207	80999.0	0.3	0.1	81207.0	0.0	0.1	–	81207
rc_202.1.txt	33	78820	75720.2	1.9	15.8	76324.0	1.2	1.3	0.2	77215
rc_202.2.txt	14	31504	31504.0	0.0	0.2	–	–	–	–	31504
rc_202.3.txt	29	89203	87057.5	0.0	1.1	87069.0	0.0	0.3	–	87069
rc_202.4.txt	28	79446	77899.0	1.9	9.2	77899.0	1.9	0.1	0.2	79446
rc_203.1.txt	19	45340	45066.3	0.6	0.3	45340.0	0.0	0.1	–	45340
rc_203.2.txt	33	80942	80637.5	0.2	96.8	80637.5	0.2	1.8	0.2	80798
rc_203.3.txt	37	88733	84702.1	3.1	273.0	84883.7	2.9	100.7	6.7	87403
rc_203.4.txt	15	31841	31841.0	0.0	0.0	–	–	–	–	31841
rc_204.1.txt	46	89889	87687.1	0.4	1943.3	87687.1	0.4	64.3	126.8	88069
rc_204.2.txt	33	67462	65038.4	3.1	196.3	65060.3	3.1	8.6	TL	67120†
rc_204.3.txt	24	45495	43919.0	3.5	59.3	43919.0	3.5	1.1	0.2	45495
rc_205.1.txt	14	37549	37549.0	0.0	0.0	–	–	–	–	37549
rc_205.2.txt	27	79495	74189.0	5.9	0.1	74189.0	5.9	0.2	0.1	78869
rc_205.3.txt	35	82764	82738.1	0.0	20.2	82764.0	0.0	0.3	–	82764
rc_205.4.txt	28	78937	77051.0	2.4	0.2	77051.0	2.4	0.1	0.1	78937
rc_206.1.txt	4	11784	11784.0	0.0	0.0	–	–	–	–	11784
rc_206.2.txt	37	84349	79950.7	5.2	10.4	80949.0	4.0	1.2	0.4	84349
rc_206.3.txt	25	57723	57035.2	1.2	1.8	57565.0	0.3	0.2	0.1	57723
rc_206.4.txt	38	84770	80242.3	4.3	8.5	81334.5	3.0	12.2	0.2	83830
rc_207.1.txt	34	77056	72627.0	0.9	36.2	72627.0	0.9	0.2	0.1	73260
rc_207.2.txt	31	70347	62736.4	10.5	6.3	63298.0	9.7	8.7	1283.6	70116
rc_207.3.txt	33	73286	63113.3	7.5	22.3	63372.4	7.1	64.9	225.2	68230
rc_207.4.txt	6	11961	11961.0	0.0	0.0	–	–	–	–	11961
rc_208.1.txt	38	79904	68981.0	13.7	247.4	68981.0	13.7	0.3	TL	[73432,79904]
rc_208.2.txt	29	55478	51350.2	3.8	7.9	51656.0	3.2	41.9	8.1	53369
rc_208.3.txt	36	67902	57625.1	15.1	54.1	58407.4	14.0	84.2	TL	[61302,67902]

Table 7: Detailed Results of the Column-Generation Method for the `Potvin+Bengio` Instances

Instance	$\lvert V \rvert$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
rbg010a.tw	11	2975	2975.0	0.0	0.0	–	–	–	–	2975
rbg016a.tw	17	2465	2457.1	0.3	0.0	2465.0	0.0	0.1	–	2465
rbg016b.tw	17	1304	1299.0	0.4	0.2	1299.0	0.4	0.0	0.1	1304
rbg017.2.tw	16	1351	1351.0	0.0	0.1	–	–	–	–	1351
rbg017.tw	16	1756	1756.0	0.0	0.0	–	–	–	–	1756
rbg017a.tw	18	4296	4296.0	0.0	0.1	–	–	–	–	4296
rbg019a.tw	20	2448	2448.0	0.0	0.0	–	–	–	–	2448
rbg019b.tw	20	2975	2975.0	0.0	0.1	–	–	–	–	2975
rbg019c.tw	20	4536	4526.0	0.2	0.4	4526.0	0.2	0.1	0.1	4536
rbg019d.tw	20	2917	2917.0	0.0	0.0	–	–	–	–	2917
rbg020a.tw	21	4689	4689.0	0.0	0.0	–	–	–	–	4689
rbg021.2.tw	20	4528	4526.0	0.0	0.4	4526.0	0.0	0.2	0.1	4528
rbg021.3.tw	20	4528	4519.6	0.2	0.5	4520.0	0.2	0.2	0.1	4528
rbg021.4.tw	20	4525	4516.0	0.2	0.8	4516.0	0.2	0.0	0.1	4525
rbg021.5.tw	20	4516	4510.0	0.1	0.8	4510.0	0.1	0.1	0.1	4516
rbg021.6.tw	20	4489	4483.5	0.0	2.2	–	–	–	–	4484
rbg021.7.tw	20	4481	4479.0	0.0	2.8	4479.0	0.0	0.3	0.1	4479
rbg021.8.tw	20	4481	4478.0	0.0	2.2	4478.0	0.0	0.6	–	4478
rbg021.9.tw	20	4481	4478.0	0.0	2.2	4478.0	0.0	0.7	0.1	4478
rbg021.tw	20	4536	4526.0	0.2	0.4	4526.0	0.2	0.1	0.1	4536
rbg027a.tw	28	5093	5088.7	0.1	3.5	5090.0	0.1	0.5	0.1	5093
rbg031a.tw	32	2953	2953.0	0.0	1.0	–	–	–	–	2953
rbg033a.tw	34	3157	3157.0	0.0	1.3	–	–	–	–	3157
rbg034a.tw	35	2714	2714.0	0.0	0.5	–	–	–	–	2714
rbg035a.2.tw	36	2715	2715.0	0.0	5.0	–	–	–	–	2715
rbg035a.tw	36	2874	2874.0	0.0	2.6	–	–	–	–	2874
rbg038a.tw	39	5115	5115.0	0.0	1.4	–	–	–	–	5115
rbg040a.tw	41	5079	5079.0	0.0	0.1	–	–	–	–	5079
rbg050a.tw	51	11450	11450.0	0.0	10.5	–	–	–	–	11450
rbg055a.tw	56	6367	6367.0	0.0	4.5	–	–	–	–	6367
rbg067a.tw	68	9736	9736.0	0.0	0.3	–	–	–	–	9736
rbg125a.tw	126	13652	13652.0	0.0	70.6	–	–	–	–	13652

Table 8: Detailed Results of the Column-Generation Method for the `Ascheuer easy` Instances

Instance	$\lvert V \rvert$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
rbg041a.tw	42	3245	3245.0	0.0	4.7	–	–	–	–	3245
rbg042a.tw	43	2962	2949.8	0.4	29.0	2959.0	0.1	1.9	0.3	2962
rbg048a.tw	49	9793	9793.0	0.0	2.6	–	–	–	–	9793
rbg049a.tw	50	12657	12657.0	0.0	1.0	–	–	–	–	12657
rbg050b.tw	51	11357	11357.0	0.0	53.7	–	–	–	–	11357
rbg050c.tw	51	10431	10431.0	0.0	76.9	–	–	–	–	10431
rbg086a.tw	87	16299	16299.0	0.0	0.2	–	–	–	–	16299
rbg092a.tw	93	11924	11924.0	0.0	1.0	–	–	–	–	11924
rbg132.2.tw	131	17524	17524.0	0.0	9.0	–	–	–	–	17524
rbg132.tw	131	17929	17929.0	0.0	0.8	–	–	–	–	17929
rbg152.3.tw	151	16455	16455.0	0.0	74.8	–	–	–	–	16455
rbg152.tw	151	17019	17019.0	0.0	72.5	–	–	–	–	17019
rbg172a.tw	173	17221	17213.5	0.0	598.6	17220.1	0.0	135.8	–	17221
rbg193.2.tw	192	20401	20401.0	0.0	44.2	–	–	–	–	20401
rbg193.tw	192	20869	20868.8	0.0	625.5	–	–	–	–	20869
rbg201a.tw	202	20818	20818.0	0.0	306.7	–	–	–	–	20818
rbg233.2.tw	232	25143	25143.0	0.0	39.5	–	–	–	–	25143
rbg233.tw	232	25691	25691.0	0.0	531.9	–	–	–	–	25691

Table 9: Detailed Results of the Column-Generation Method for the `Ascheuer hard` Instances

Instance	$\|V\|$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n20w120.001.txt	21	296	295.3	0.2	0.1	–	–	–	–	296
n20w120.002.txt	21	220	217.3	1.2	0.4	220.0	0.0	0.1	–	220
n20w120.003.txt	21	303	303.0	0.0	0.0	–	–	–	–	303
n20w120.004.txt	21	312	308.0	1.3	0.5	308.0	1.3	0.1	0.1	312
n20w120.005.txt	21	277	274.3	1.0	0.8	276.6	0.1	0.1	–	277
n20w140.001.txt	21	188	187.3	0.4	0.2	–	–	–	–	188
n20w140.002.txt	21	280	277.0	1.1	0.1	–	–	–	–	280
n20w140.003.txt	21	252	251.3	0.3	0.8	–	–	–	–	252
n20w140.004.txt	21	280	275.5	1.6	0.4	280.0	0.0	0.1	–	280
n20w140.005.txt	21	231	230.3	0.3	0.5	–	–	–	–	231
n20w160.001.txt	21	284	283.0	0.4	0.7	284.0	0.0	0.1	–	284
n20w160.002.txt	21	205	204.7	0.1	0.1	–	–	–	–	205
n20w160.003.txt	21	277	275.7	0.5	0.2	277.0	0.0	0.0	–	277
n20w160.004.txt	21	222	222.0	0.0	0.3	–	–	–	–	222
n20w160.005.txt	21	284	283.2	0.3	0.8	–	–	–	–	284
n20w180.001.txt	21	303	285.0	5.9	0.9	302.5	0.2	0.2	–	303
n20w180.002.txt	21	319	302.6	1.4	0.7	307.0	0.0	0.2	–	307
n20w180.003.txt	21	273	270.0	1.1	0.3	270.0	1.1	0.1	0.1	273
n20w180.004.txt	21	234	232.0	0.9	0.9	234.0	0.0	0.1	–	234
n20w180.005.txt	21	207	199.6	0.7	1.3	201.0	0.0	0.1	–	201
n20w200.001.txt	21	233	230.5	1.1	0.7	232.0	0.4	0.1	0.1	233
n20w200.002.txt	21	211	209.0	0.9	1.6	209.0	0.9	0.0	0.1	211
n20w200.003.txt	21	271	254.8	2.8	0.7	262.0	0.0	0.2	–	262
n20w200.004.txt	21	320	279.7	7.1	0.7	285.0	5.3	0.8	0.1	301
n20w200.005.txt	21	229	226.0	0.4	0.5	–	–	–	–	227

Table 10: Detailed Results of the Column-Generation Method for the `Gendreau 20` Instances

Instance	$\|V\|$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n40w120.001.txt	41	446	431.0	3.4	63.1	431.0	3.4	0.2	0.2	446
n40w120.002.txt	41	514	509.4	0.9	42.3	513.1	0.2	6.1	–	514
n40w120.003.txt	41	420	412.3	1.1	51.8	416.0	0.2	9.8	0.3	417
n40w120.004.txt	41	347	343.8	0.9	16.9	346.8	0.0	1.6	–	347
n40w120.005.txt	41	418	417.3	0.2	9.1	–	–	–	–	418
n40w140.001.txt	41	402	401.0	0.2	18.5	–	–	–	–	402
n40w140.002.txt	41	401	401.0	0.0	2.6	–	–	–	–	401
n40w140.003.txt	41	429	428.0	0.2	42.8	–	–	–	–	429
n40w140.004.txt	41	400	399.1	0.2	76.8	–	–	–	–	400
n40w140.005.txt	41	390	389.1	0.2	41.6	–	–	–	–	390
n40w160.001.txt	41	418	418.0	0.0	18.0	–	–	–	–	418
n40w160.002.txt	41	388	383.7	1.1	63.5	388.0	0.0	2.9	–	388
n40w160.003.txt	41	368	367.2	0.2	15.8	–	–	–	–	368
n40w160.004.txt	41	361	356.3	0.2	252.5	357.0	0.0	14.3	–	357
n40w160.005.txt	41	316	315.1	0.3	213.5	–	–	–	–	316
n40w180.001.txt	41	399	388.8	1.3	125.8	389.8	1.1	138.2	11.6	394
n40w180.002.txt	41	379	378.1	0.2	38.4	–	–	–	–	379
n40w180.003.txt	41	346	345.5	0.1	45.6	–	–	–	–	346
n40w180.004.txt	41	378	369.0	0.3	71.0	–	–	–	–	370
n40w180.005.txt	41	363	362.0	0.3	301.3	–	–	–	–	363
n40w200.001.txt	41	345	339.0	0.0	537.4	339.0	0.0	11.9	–	339
n40w200.002.txt	41	343	337.2	1.4	222.3	337.2	1.4	49.7	0.2	342
n40w200.003.txt	41	391	342.2	1.7	77.4	344.8	0.9	135.9	5.0	348
n40w200.004.txt	41	377	368.3	0.5	33.8	369.3	0.2	25.4	120.9	370
n40w200.005.txt	41	350	347.0	0.9	77.1	347.0	0.9	269.7	TL	[347,350]

Table 11: Detailed Results of the Column-Generation Method for the `Gendreau 40` Instances

Instance	$\lvert V\rvert$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n60w120.001.txt	61	484	483.0	0.2	268.9	483.0	0.2	36.9	55.0	484
n60w120.002.txt	61	553	552.1	0.2	265.2	–	–	–	–	553
n60w120.003.txt	61	488	488.0	0.0	2.1	–	–	–	–	488
n60w120.004.txt	61	556	556.0	0.0	186.8	–	–	–	–	556
n60w120.005.txt	61	549	548.2	0.1	526.4	–	–	–	–	549
n60w140.001.txt	61	560	560.0	0.0	175.5	–	–	–	–	560
n60w140.002.txt	61	597	594.1	0.5	230.1	595.3	0.3	26.4	8.7	597
n60w140.003.txt	61	567	567.0	0.0	266.1	–	–	–	–	567
n60w140.004.txt	61	567	566.0	0.2	510.3	–	–	–	–	567
n60w140.005.txt	61	501	497.0	0.0	310.0	497.0	0.0	42.7	16.1	497
n60w160.001.txt	61	614	614.0	0.0	671.5	–	–	–	–	614
n60w160.002.txt	61	614	614.0	0.0	13.7	–	–	–	–	614
n60w160.003.txt	61	507	507.0	0.0	851.4	–	–	–	–	507
n60w160.004.txt	61	505	504.0	0.2	633.2	–	–	–	–	505
n60w160.005.txt	61	561	560.3	0.1	698.9	–	–	–	–	561
n60w180.001.txt	61	488	487.7	0.1	172.5	–	–	–	–	488
n60w180.002.txt	61	503	501.5	0.3	398.5	502.0	0.2	19.1	–	503
n60w180.003.txt	61	526	525.1	0.2	282.1	–	–	–	–	526
n60w180.004.txt	61	577	577.0	0.0	207.1	–	–	–	–	577
n60w180.005.txt	61	486	466.0	4.1	1818.0	466.0	4.1	425.1	TL	[466,486]
n60w200.001.txt	61	465	446.5	3.8	2642.4	451.1	2.8	264.8	3108.7	464
n60w200.002.txt	61	504	503.1	0.2	545.4	–	–	–	–	504
n60w200.003.txt	61	525	494.5	5.8	3308.1	496.2	5.5	573.9	TL	[497,525]
n60w200.004.txt	61	512	511.1	0.2	1740.1	–	–	–	–	512
n60w200.005.txt	61	545	545.0	0.0	1771.4	–	–	–	–	545

Table 12: Detailed Results of the Column-Generation Method for the `Gendreau 60` Instances

Instance	$\lvert V\rvert$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n80w100.001.txt	81	664	660.2	0.4	919.5	661.5	0.2	183.7	458.2	663
n80w100.002.txt	81	707	683.0	0.0	415.6	683.0	0.0	113.5	0.9	683
n80w100.003.txt	81	717	717.0	0.0	11.0	–	–	–	–	717
n80w100.004.txt	81	753	753.0	0.0	3.5	–	–	–	–	753
n80w100.005.txt	81	664	661.5	0.4	181.6	661.5	0.4	167.7	0.3	664
n80w120.001.txt	81	620	620.0	0.0	910.5	–	–	–	–	620
n80w120.002.txt	81	695	695.0	0.0	16.1	–	–	–	–	695
n80w120.003.txt	81	622	621.0	0.2	2999.2	–	–	–	–	622
n80w120.004.txt	81	591	590.1	0.2	2331.0	–	–	–	–	591
n80w120.005.txt	81	690	689.0	0.1	2332.5	–	–	–	–	690
n80w140.001.txt	81	635	635.0	0.0	838.8	–	–	–	–	635
n80w140.002.txt	81	591	588.0	0.5	588.5	588.0	0.5	525.3	TL	[588,591]
n80w140.003.txt	81	617	612.7	0.7	TL	614.4	0.4	857.2	TL	[615,617]
n80w140.004.txt	81	561	549.0	2.1	TL	549.0	2.1	81.8	TL	[549,561]
n80w140.005.txt	81	687	685.0	0.3	638.0	685.0	0.3	80.6	0.3	687
n80w160.001.txt	81	564	561.7	0.4	TL	563.1	0.2	1090.0	–	564
n80w160.002.txt	81	609	601.5	1.2	TL	602.0	1.1	2400.0	TL	[603,609]
n80w160.003.txt	81	638	628.2	1.5	TL	630.0	1.2	1380.0	TL	[633,638]
n80w160.004.txt	81	596	596.0	0.0	2033.0	–	–	–	–	596
n80w160.005.txt	81	584	573.7	1.8	1689.5	573.7	1.8	295.5	TL	[583,584]
n80w180.001.txt	81	617	610.2	0.5	TL	610.7	0.4	631.9	TL	613†
n80w180.002.txt	81	570	555.7	2.5	TL	559.5	1.8	2529.1	TL	[564,570]
n80w180.003.txt	81	623	623.0	0.0	3553.7	–	–	–	–	623
n80w180.004.txt	81	592	592.0	0.0	2641.0	–	–	–	–	592
n80w180.005.txt	81	571	568.3	0.5	TL	569.4	0.3	1115.5	TL	[570,571]
n80w200.001.txt	81	584	557.3	4.6	TL	558.8	4.3	2934.9	TL	[559,584]
n80w200.002.txt	81	550	545.4	0.8	TL	548.5	0.3	1541.3	TL	[549,550]
n80w200.003.txt	81	617	617.0	0.0	74.6	–	–	–	–	617
n80w200.004.txt	81	611	606.1	0.8	TL	608.3	0.4	1625.8	TL	611†
n80w200.005.txt	81	564	561.0	0.5	1707.6	561.0	0.5	522.6	0.3	564

Table 13: Detailed Results of the Column-Generation Method for the `Gendreau 80` Instances

Instance	$\|V\|$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n100w120.001.txt	101	825	825.0	0.0	783.9	–	–	–	–	825
n100w120.002.txt	101	846	843.0	0.4	853.2	843.0	0.4	386.6	TL	[843,846]
n100w120.003.txt	101	852	852.0	0.0	41.9	–	–	–	–	852
n100w120.004.txt	101	868	868.0	0.0	923.0	–	–	–	–	868
n100w120.005.txt	101	806	806.0	0.0	545.1	–	–	–	–	806
n100w140.001.txt	101	956	956.0	0.0	42.9	–	–	–	–	956
n100w140.002.txt	101	949	948.0	0.1	2306.6	948.0	0.1	415.0	TL	[948,949]
n100w140.003.txt	101	783	783.0	0.0	3114.7	–	–	–	–	783
n100w140.004.txt	101	791	791.0	0.0	39.6	–	–	–	–	791
n100w140.005.txt	101	733	731.0	0.3	TL	731.0	0.3	293.3	TL	733†
n100w160.001.txt	101	802	802.0	0.0	2928.1	–	–	–	–	802
n100w160.002.txt	101	731	729.8	0.2	TL	730.1	0.1	29.7	–	731
n100w160.003.txt	101	882	882.0	0.0	52.7	–	–	–	–	882
n100w160.004.txt	101	751	751.0	0.0	2711.1	–	–	–	–	751
n100w160.005.txt	101	855	855.0	0.0	1344.2	–	–	–	–	855

Table 14: Detailed Results of the Column-Generation Method for the `Gendreau 100` Instances

Instance	$\|V\|$	UB BS	Column Generation LB	GAP [%]	Time [s]	Dyn. ng neighb. augment. LB	GAP [%]	Time [s]	Exact DP Time [s]	OPT
n150w120.001.txt	151	9200	9098.0	1.1	TL	9098.0	1.1	860.5	TL	[9098,9200]
n150w120.002.txt	151	8692	8670.0	0.3	TL	8670.0	0.3	1381.3	TL	[8681,8692]
n150w120.003.txt	151	8611	8477.0	1.6	TL	8477.0	1.6	TL	TL	[8598,8611]
n150w120.004.txt	151	8759	8648.0	1.3	TL	8648.0	1.3	1379.9	TL	[8751,8759]
n150w120.005.txt	151	8555	8509.0	0.5	TL	8509.0	0.5	991.4	TL	[8509,8555]
n150w140.001.txt	151	9560	9560.0	0.0	729.1	9560.0	0.0	3.3	–	9.560
n150w140.002.txt	151	9811	9596.0	2.2	TL	9596.0	2.2	TL	TL	[9722,9811]
n150w140.003.txt	151	7915	7860.0	0.7	TL	7860.0	0.7	TL	TL	[7860,7915]
n150w140.004.txt	151	8494	8210.0	3.3	TL	8210.0	3.3	TL	TL	[8210,8494]
n150w140.005.txt	151	8712	8448.0	3.0	TL	8448.0	3.0	2834.2	TL	[8568,8712]
n150w160.001.txt	151	9062	8953.0	1.2	TL	8953.0	1.2	TL	TL	[8953,9062]
n150w160.002.txt	151	8905	8023.0	9.9	TL	8023.0	9.9	TL	TL	[8023,8905]
n150w160.003.txt	151	8886	8827.0	0.7	TL	8827.0	0.7	1283.5	TL	[8827,8886]
n150w160.004.txt	151	8632	8404.0	2.6	TL	8404.0	2.6	TL	TL	[8539,8632]
n150w160.005.txt	151	8669	8530.0	1.6	TL	8530.0	1.6	TL	TL	[8530,8669]
n200w120.001.txt	201	10454	10360.0	0.9	TL	10360.0	0.9	TL	TL	[10410,10454]
n200w120.002.txt	201	10225	10150.0	0.7	TL	10150.0	0.7	1224.9	TL	10225†
n200w120.003.txt	201	10810	10734.0	0.7	TL	10734.0	0.7	1720.9	TL	[10777,10810]
n200w120.004.txt	201	10177	10146.0	0.3	TL	10146.0	0.3	917.5	TL	[10146,10177]
n200w120.005.txt	201	10385	10195.0	1.8	TL	10195.0	1.8	1344.1	TL	[10197,10385]
n200w140.001.txt	201	10893	10813.0	0.7	TL	10813.0	0.7	TL	TL	[10813,10893]
n200w140.002.txt	201	10398	10078.0	3.1	TL	10078.0	3.1	2986.3	TL	[10284,10398]
n200w140.003.txt	201	10404	10290.0	1.1	TL	10290.0	1.1	TL	TL	[10290,10404]
n200w140.004.txt	201	10518	10416.0	1.0	TL	10416.0	1.0	1051.3	TL	[10416,10518]
n200w140.005.txt	201	10743	10652.0	0.8	TL	10652.0	0.8	TL	TL	[10692,10743]

Table 15: Detailed Results of the Column-Generation Method for the `Ohlmann+Thomas` Instances