Heuristic Approaches to Minimize Tour Duration for the TSP with Multiple Time Windows

Niklas Paulsen[1,2], Florian Diedrich[2], and Klaus Jansen[1]

1 **Institut für Informatik, Christian-Albrechts Universität zu Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, Germany**
{npau,kj}@informatik.uni-kiel.de

2 **FLS GmbH, Schlosskoppelweg 8, 24226 Heikendorf, Germany**

Abstract

We present heuristics to handle practical travelling salesman problems with multiple time windows per node, where the optimization goal is minimal tour duration, which is the time spent outside the depot node. We propose a dynamic programming approach which combines state labels by encoding intervals to handle the larger state space needed for this objective function. Our implementation is able to solve many practical instances in real-time and is used for heuristic search of near-optimal solutions for hard instances. In addition, we outline a hybrid genetic algorithm we implemented to cope with hard or unknown instances. Experimental evaluation proves the efficiency and suitability for practical use of our algorithms and even leads to improved upper bounds for yet unsolved instances from the literature.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases TSPTW, minimum tour duration, dynamic programming, heuristics

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.42

1 Introduction

The `Travelling Salesman Problem with Time Windows (TSPTW)` is the problem of finding a cost-minimal Hamiltonian cycle through a complete digraph on N nodes, which respects time windows given for each node. The nodes are represented by the set $V = \{0, \ldots, N-1\}$, where 0 is called the *depot*, and we define $V' := V \setminus \{0\}$ to be the other nodes. Each node $v \in V'$ has a given time window $[a_v, b_v]$ in which it has to be visited. To calculate times, $c : V \times V \to \mathbb{N}$ assigns a travel time to each edge. Arriving at a node v before a_v will lead to waiting there until the time window opens. Node dependent visit times can be encoded in the travel times; distinct start and return nodes can be combined into the node 0 by adjusting travel times from and to node 0; missing arcs can be encoded by high travel times. The time windows constrain the set of feasible solutions; in general the presence of time windows makes it NP-hard to even find a feasible tour [12]. A possible generalization from TSP to TSPTW minimizes the same objective function, namely the sum of weights of the chosen edges [4, 10]. Since we have edge weights as travel times, this objective corresponds to minimization of the total travel time. However, in workforce planning, loans make a major contribution to the planned costs and thus also waiting times are expected to have an impact on the objective function. An important choice is whether a delayed start of a given tour is counted as working time or not. If not, any tour can be started at some earliest possible time. Then, minimizing the total travel time plus any waiting time along the way corresponds to finding the tour which has the earliest return to the depot node. We call this problem `Minimum Completion Time Problem (MCTP)`. In this work, however, we allow the start to be delayed without cost, searching a tour that can be traversed with minimal tour duration from depot departure to return. We call the according problem `Minimum Tour Duration Problem (MTDP)`. This objective function is a generalization of the completion time minimization, since the latter can be expressed by fixing the starting time at the depot [3]. While for mathematical formulations, the difference between `MCTP` and `MTDP` is just a slight change in the objective function, common heuristic approaches as well as Dynamic Programming (DP) become more complicated for the latter. Modifying a given tour with known optimal departure time, a new optimal departure time needs to be searched, as Savelsbergh [13] pointed out. Despite its relevance for workforce planning, the `MTDP` has been given only little attention until recently. Tilk et al. [15] treated the `MTDP`, using a new Dynamic Programming (DP) based approach to solve many available instances to optimality or provide bounds. Although they are more focused on optimal solution rather than real-time processing in a practical use case, their solutions can serve as a good reference. We handle an even further generalized version of the problem, allowing an arbitrary number of time windows per node, as it occurs in practice for example at machine-related maintenance tasks or simply due to opening hours with lunch breaks. This complicates the search of an optimal departure time for a given tour [1].

Our focus is the development of fast algorithms for a practical workforce planning context. Our two methods are a DP based heuristic and a strongly randomized genetic algorithm. In practical workforce planning, we see multiple applications of fast heuristics for the `MTDP`:

- When planning only a single worker,
- as a frequently called local search in advance planning of big `Vehicle Routing Problems (VRPs)`,
- real-time post-optimization of planned tours after changes in online `VRP`s, and
- to obtain upper bounds that can be used by exact methods for small instances or for evaluation purposes.

In Section 2 we discuss the state-space inflation for DP inherent with allowing multiple time windows per node and propose an approach to encode multiple states into intervals. In Section 3 we outline our Genetic Algorithm, `GA`, which allows solving diverse instances like ones with very wide time windows. We report results on instance sets from the literature and on new real-world instances in Section 4 and give a final conclusion in Section 5.

1.1 Formal Definitions

We want to formalize the timings for given tours. Be $K_v > 0$ the number of time windows for $v \in V'$, $a_{v,k}$ and $b_{v,k}$ be the opening and closing time, respectively, for the k-th time window of node $v \in V'$, $0 \leq k < K_v$. The time windows of every node are presumed to be sorted, non-overlapping, and of non-negative length ($a_{v,0} \leq b_{v,0} < a_{v,1} \leq \cdots < a_{v,K_v-1} \leq b_{v,K_v-1}$ for $v \in V'$). Define Π to be the set of `TSP`-tours, represented by permutations of V, starting in the depot ($\pi(0) = 0$ for $\pi \in \Pi$). For an arrival time $t \in \mathbb{N}$ at a node $v \in V'$, the next feasible schedule time at that node is given by:

$$T^{\rightarrow}(v,t) := \min\{x \mid x \geq t \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\}$$

For $t > b_{v,K_v}$ a minimum over $\emptyset$ leads to $T^{\rightarrow}(v,t) = \infty$. For a `TSP`-tour $\pi \in \Pi$ and departure time $t_0 \in \mathbb{N}$ the scheduled departure times $t^{\pi}_{t_0} : V \rightarrow \mathbb{N}$ can be calculated as follows: For the depot 0 it is $t^{\pi}_{t_0}(0) = t_0$ and for $v \in V'$ it is, depending on the last node visited, $v^- := \pi(\pi^{-1}(v) - 1)$:

$$t^{\pi}_{t_0}(v) := T^{\rightarrow}(v, t^{\pi}_{t_0}(v^-) + c(v^-, v))$$

Define $t^\pi_{t_0}(N) := t^\pi_{t_0}(\pi(N-1)) + c(\pi(N-1), 0)$ to be the returning time at the depot. Furthermore we define $W^{\rightarrow}(v,t) := T^{\rightarrow}(v,t) - t$ for the waiting time at node $v \in V'$, when reached at time t. The optimization goal is then to find $\pi \in \Pi$ and $t_0 \in \mathbb{N}$ minimizing $t^\pi_{t_0}(N) - t_0$.

▶ **Lemma 1.** *For $\pi \in \Pi, i < N, \tau, \delta \in \mathbb{N}$ we have $t^\pi_{\tau+\delta}(\pi(i)) \geq t^\pi_\tau(\pi(i))$. (Proof in Appendix)*

2 Adaption of Dynamic Programming for Tour Duration Minimization

A common way to solve various variants of `TSPs` is via dynamic programming (DP), based on the formulation for the classic `TSP` proposed decades ago by Bellman et al. [2]. Bellman's Principle states, generically speaking, that optimal solutions of a problem (instance) are consisting of optimal solutions to smaller sub-problems. We call sub-problems *states* and their solution a *label* of the state. The proceeding of forward-labelling is to label some initial states and use recurrence relations to propagate given labels to labels for other states.

In case of the `TSP`, sub-problems are finding a minimum cost path originating in 0, going through a given subset of nodes, $S \subset V'$, and ending in a given node, $\ell \in V' \setminus S$. We call these paths S, ℓ-paths. The calculation can be tackled in n stages, for increasing $|S|$ according to longer paths. A minimum path through a given $S \neq \emptyset$ can only be arising from an optimal path through $S \setminus \{x\}$ to $x \in S$, but with the presence of time windows, this only holds for minimizing tour completion time (`MCTP`). To solve `TSPTW` regarding minimum travel time, a two-dimensional labelling for (S, ℓ)-states is necessary, as used by Dumas et al. [4]. This is because all S, ℓ-paths are relevant that are Pareto optimal concerning cost and completion time, since during calculation it is not known which time at ℓ can lead to a feasible completion of the tour through $V' \setminus (S \cup \{\ell\})$. Recently, DP was adapted for the `MTDP` (with single time windows per node) by Tilk et al. [15]. They use labels containing 3 *resources* for the earliest possible time to complete an S, ℓ-path, the tour duration so far, and a time slack. Labelling all (non-dominated) S, ℓ-paths, Bellman's Principle holds. In the generalized case of an arbitrary number of time windows for each node, every S, ℓ-path extended to an $S \cup \{\ell\}, \ell'$-path must distinguish the times at which it is travelled: Compared to the time leading to a minimal duration of the S, ℓ-path, an earlier or later traversal with a *longer* duration may bring along a smaller waiting time at node ℓ', if a different time window of ℓ' can be taken possibly leading to a smaller tour duration. As a consequence, more labels can arise for every S, ℓ-path, corresponding to different choices of time windows for visited nodes. We show that the number of labels for each tour is growing at most linearly with the overall number of time windows. For a fixed tour $\pi \in \Pi$, the following definition is used to model a specific choice of time windows for a prefix of π.

Define a *time window path of length* $k \leq |V'| = N - 1$ to be a tuple $(s_1, \ldots, s_k)$ with $s_i < K_{\pi(i)}$ for $0 < i \leq k$ choosing time window indices for the first k nodes visited by π after the depot; we call it *schedule*, iff $k = |V'|$ and we call it *reachable*, iff a start time τ exists such that all nodes are visited within their chosen time window:

$$t^\pi_\tau(\pi(i)) \in [a_{\pi(i),s_i}, b_{\pi(i),s_i}] \quad \text{for } 0 < i \leq k. \tag{1}$$

Note that reachable time window paths are only met by delaying the departure at the depot while every other node is visited as early as possible by definition of t^π_τ.

▶ **Theorem 2.** *With a given `TSP`-Tour $\pi \in \Pi$ there are at most $1 + \sum_{v \in V'}(K_v - 1)$ reachable schedules.*

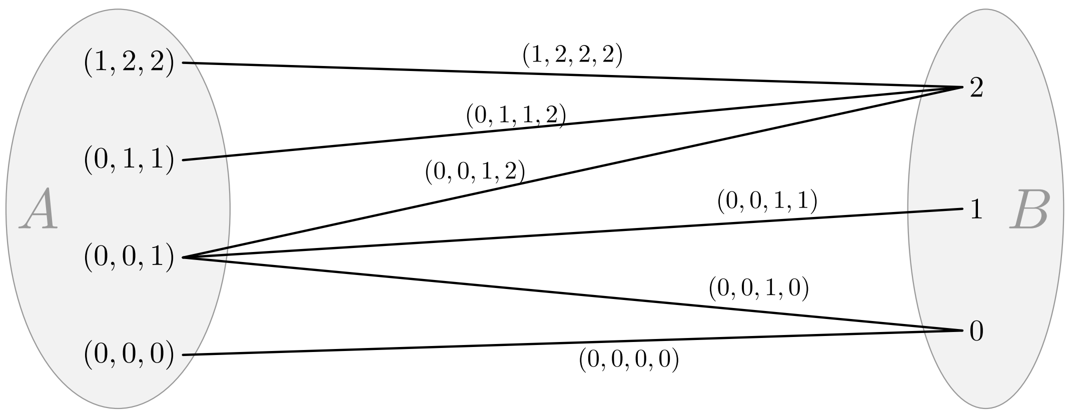

Figure 1 Illustration of an example bipartite graph H for $i = 3$.

Proof. We show via induction over $0 < i \leq |V'|$ that there are at most

$$1 + \sum_{j=1}^{i} (K_{\pi(j)} - 1) \tag{I}$$

reachable time window paths of length i. The induction base is $i = 1$, with the visit of node $\pi(1)$ in one of its $K_{\pi(1)}$ time windows.
Now assume (I) holds for an $0 < i < |V'|$. To prove that (I) also holds for $i + 1$ we need to show that only up to $K_{\pi(i+1)} - 1$ more time window paths of length $i + 1$ arise than for i. Define a bipartite graph $H = (A, B, F \subset A \times B)$ with $A \subseteq \mathbb{N}_{<K_{\pi(1)}} \times \cdots \times \mathbb{N}_{<K_{\pi(i)}}$ being the reachable time window paths of length i and $B = \mathbb{N}_{<K_{\pi(i+1)}}$ being the time window indices of the next node, $\pi(i + 1)$. An edge $f = ((c_1, \ldots, c_i), k) \in F$ shall exist, if and only if the time window path $c' := (c_1, \ldots, c_i, k)$ is reachable. Since every reachable time window path of length $i + 1$ contains a reachable time window path of length i, the edges correspond to the reachable time window paths of length $i + 1$. An example for H is shown in Figure 1.

We show that H can be drawn without crossings in the sense of Eades et al. [5]. We use the lexicographic order as $\prec_A$ on A, and the order $\prec_B$ on B which is given for the time windows by their definition. Suppose an edge (a, w) exists. Then, an edge (a', w') with $a \prec a'$ but $w' \prec w$ cannot exist, since this would mean that starting later[1] leads to reaching an earlier time window at waypoint $i + 1$, contradicting Lemma 1. This implies a crossing-free drawing of H, and thereby absence of cycles in H [5]. Being cycle-free, H is a forest and has at most $|A| + |B| - 1$ edges. With the induction hypothesis, $|A| \leq 1 + \sum_{j=1}^{i} (K_{\pi(j)} - 1)$, and with $|B| = K_{\pi(i+1)}$, (I) also holds for $i + 1$. ◀

Consider the state space $2^{V'} \times V' \times \mathbb{T}$ with $\mathbb{T} \subset \mathbb{N}$, where each state (S, ℓ, t) gets a scalar label expressing the minimum S, ℓ-path duration when ℓ is visited at time t. Even with a bounding of $\mathbb{T}$ to actually relevant times, the size of this state-space is wasteful, especially with the inherent growth with increasing temporal resolution of time encoding. Our approach is to encode for each S and ℓ the function assigning the minimal S, ℓ-path duration to each departure time t. An example of this function is shown in Figure 2 in the lower left. The

[1] With $a \prec a'$, at some point a later time window (bigger index) is taken with a', therefore starting times leading to reaching a' cannot be smaller than any starting time leading to reaching a (contraposition of Lemma 1).

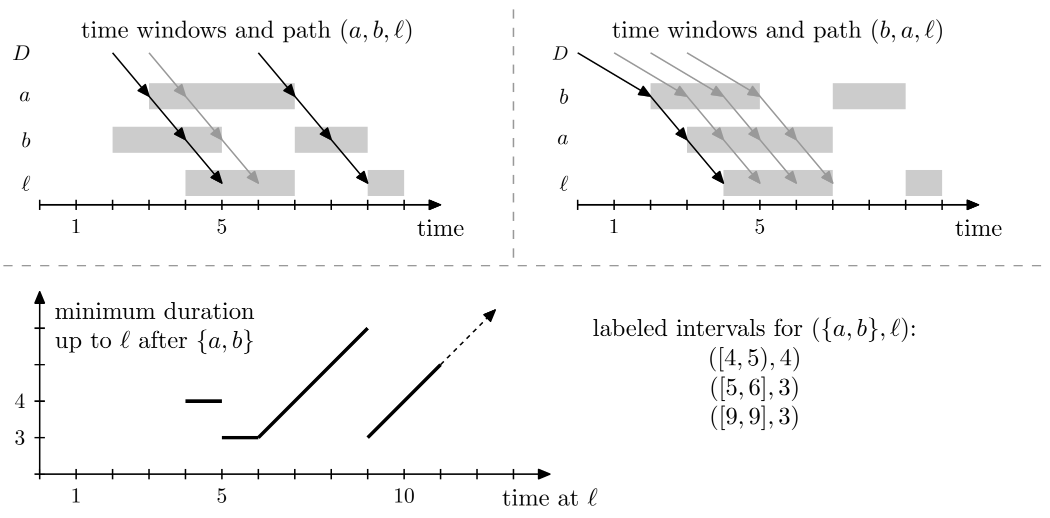
time windows and path (a, b, ℓ)
D
a
b
ℓ
1
5
time
time windows and path (b, a, ℓ)
D
b
a
ℓ
1
5
time
minimum duration up to ℓ after $\{a, b\}$
4
3
1
5
10
time at ℓ
labeled intervals for $(\{a, b\}, \ell)$:
$([4, 5), 4)$
$([5, 6], 3)$
$([9, 9], 3)$

Figure 2 Example for labelled intervals, for State $(S = \{a, b\}, \ell)$. D is the depot, travelling time between the depot and b is two hours, all others one hour.

notion is a shift of the $\mathbb{T}$ factor from the state space into the labels of the (S, ℓ)-states. If an S, ℓ-path $\mathcal{P}$ can lead to a visit of ℓ at time t with (minimal) duration T, only the following cases apply for the minimal duration T' for the "next" time $t + 1$:
(C1) $\mathcal{P}$ can be traversed later without waiting times, leading to $T' = T$, (C2a) another path leads to minimal duration when visiting ℓ at $t + 1$, (C2b) $\mathcal{P}$ with another time window combination leads to minimal duration when visiting ℓ at $t + 1$, or (C3) $\mathcal{P}$ traversed later is optimal for $t + 1$ but leads to increased waiting times along the path, with $T' = T + 1$.

Therefore the function consists (except for undefined values of t, before the arrival of the first S, ℓ-path) only of piecewise constant parts (starting with cases C2a or C2b, continued with case C1) and piecewise linear parts with a slope of 1 (case C3). Our idea is to store only the interval and assigned tour duration of constant parts to implicitly encode the function. We can then handle multiple (S, ℓ, t)-states by working with the encoded intervals.

A labelled interval $I = ([t_s, t_e], T)$ is a non-empty interval $[t_s, t_e]$ $(t_e \geq t_s)$ of $\mathbb{N}$ and an assigned tour duration. For a given (S, ℓ)-state we use

$$\mathit{Ints} : S, \ell \longmapsto \text{set of } \textit{labelled intervals} \text{ encoding labels of } (S, \ell, \cdot)$$

to label (S, ℓ)-states. An interval corresponds to a constant part of the function of minimal S, ℓ-path duration at different times. With Bellman's Principle we can demand for $S \subsetneq V', \ell \in V' \setminus S$:

$$[t_s, t_e] \cap [t'_s, t'_e] = \emptyset \quad \text{f.a. } ([t_s, t_e], T) \neq ([t'_s, t'_e], T') \in \mathit{Ints}(S, \ell) \tag{2}$$
$$t'_s > t_e \Rightarrow t'_s > t_e + (T' - T) \quad \text{f.a. } ([t_s, t_e], T), ([t'_s, t'_e], T') \in \mathit{Ints}(S, \ell). \tag{3}$$

Clearly, disjoint intervals suffice: For state (S, ℓ) only the *minimum* tour duration to reach ℓ at a time t after having visited the nodes in S is needed. To evince (3) we show:

▶ **Lemma 3.** *Suppose $\ell \in V'$ can be visited at time t after all nodes in $S \subset V'$ with a tour duration of T_1, but also such that it is left until $t + \delta$ with tour duration $T_2 \geq T_1 + \delta$, for a $\delta > 0$. Then the latter is dominated by the former (it cannot lead to a tour with a smaller duration).*

No matter how the rest of the tour is constructed through $V' \setminus (S \cup \{\ell\})$, the forward propagation of the first state is able to reach the same time windows as the forward propagation of the second. Since the waiting times can only be larger by the lead δ, this will conduct at most the same tour duration (formal proof in Appendix).

To read out the label, i.e. the minimal tour duration, for a state (S, ℓ, t), we use a function *Cost* to interpret the set of labelled intervals $\mathcal{I} = \mathit{Ints}(S, \ell)$ at the time t:

$$\mathit{Cost}(\mathcal{I}, t) := \min_{\substack{([t_s,t_e],T)\in\mathcal{I}\\ t_s\le t}} T + \max\{0, t - t_e\} \tag{4}$$

With $\mathcal{I}$ satisfying Equations (2) and (3) we can write (proof in Appendix):

$$\mathit{Cost}(\mathcal{I}, t) = T + \max\{0, t - t_e\} \text{ for } ([t_s, t_e], T) = \operatorname*{arg\,max}_{\substack{([t_s,t_e],T)\in\mathcal{I}\\ t_s\le t}} t_e \tag{5}$$

This means, to evaluate the minimum tour duration at time t, only the last labelled interval starting before t needs to be considered, which can be retrieved efficiently when the labelled intervals are stored in suitable data structures.

The labelled intervals can be initialized by

$$\mathit{Ints}(\emptyset, \ell) = \{([a_{\ell,k}, b_{\ell,k}], c(0, \ell)) : k < K_\ell\} \tag{6}$$

Forward propagation can be done for aggregated times in intervals and time windows (a pseudocode can be seen in the Appendix). When multiple labels (as labelled intervals) occur for a state (S, ℓ), the intervals can be merged, choosing for each time t the interval with the best label and respecting Equation (3), as can be seen in Figure 2.

Heuristic Adaption. To heuristically reduce the search space for larger instances, Malandraki et al. [9] used a cutoff on the number of states to keep track of after each stage in their DP heuristic for the time dependent `TSP`. By retaining only the most promising H labels after each step, the run time can be minimized drastically. For $H = 1$ it resembles a *Nearest Neighbour Heuristic*, for $H = \infty$ the Dynamic Programming for an optimal solution is not affected. We call this approach `DPH` in the following. Note that we retain H labels, containing generally more than H intervals. In our implementation the labels are simply ranked by the minimal possible duration for each (S, ℓ)-State: $\min_t \mathit{Cost}(\mathit{Ints}(S, \ell), t)$.

Adapted Cost Function. It is an easy step to generalize the `DPH` to minimize a more generic objective function, being a weighted sum of working time and travelled distance.

Preprocessing and Trimming the Search Space. The search space can be trimmed by preprocessing the instance, see [3]. Also, when calculating the labelled intervals for a state (S, ℓ), only times need to be considered, which allow to reach all unvisited nodes $v \in V' \setminus (S \cup \{\ell\})$ before the end of their last time windows, b_{v,K_v-1}. Assuming the triangle-inequality (which holds often, especially with visit times present), an easy bound for relevant departure times at ℓ is:

$$B_{S,\ell} := \min\{b_{v,K_v-1} - c(\ell, v) : v \in V' \setminus (S \cup \{\ell\})\}. \tag{7}$$

Algorithm 1: Dynamic Programming for tour duration minimization.

```
1  $\mathcal{H} \leftarrow$ empty hashtable for labels assigned to $S, \ell$-states;
2  label $(\emptyset, \ell)$ with $\{([a_{\ell,k}, b_{\ell,k}], c(0,\ell)) : k < K_\ell\}$ for $\ell \in V'$;
3  for stage from 1 to $N - 2$ do
4      for state $(S, \ell)$ with label $\mathcal{I}$ and $|S| = stage - 1$ do
5          for $\ell' \in V' \setminus (S \cup \{\ell\})$ do
6              calculate new interval set $\mathcal{I}'$ by propagating $\mathcal{I}$ towards $\ell'$;
7              Trim interval ranges to be $\leq B_{S \cup \{\ell\}, \ell'}$;
8              if $\mathcal{H}$ contains label $\mathcal{I}''$ for $(S \cup \{\ell\}, \ell')$ then
9                  $\mathcal{H}(S \cup \{\ell\}, \ell') \leftarrow$ Merged intervals of $\mathcal{I}'$ and $\mathcal{I}''$;
10             else
11                 $\mathcal{H}(S \cup \{\ell\}, \ell') \leftarrow \mathcal{I}'$;
12     retain only best $H$ labels with $|S| =$ stage in $\mathcal{H}$;
13 return $\min_{\ell \in V'} \min_{([t_s,t_e],T) \in \mathcal{H}(V' \setminus \{\ell\}, \ell)} T + c(\ell, 0)$;
```

2.1 Pseudocode

Algorithm 1 illustrates the principal DPH flow. States are expressed by a combined binary representation of S and ℓ. Order constraints between nodes are also saved in a binarily represented set of nodes that have to be visited before a given node. It can be checked with little computation whether all required nodes have been visited when extending toward a node ℓ' (not shown, line 5). The hashtable lookup in line 8 can be done very efficiently. The merge step in line 9 only takes time linear in the number of intervals to be merged. By iterating all times t_s, t_e for $([t_s, t_e], \cdot) \in \mathcal{I} \cup \mathcal{I}'$ in ascending order, one simply has to chose the minimal intervals between the times and trim them to fit Equation (3). Note that with merging labels each state gets at most one label. Backtracking information is included for every labelled interval.

3 A Genetic Algorithm

To find high-quality solutions for instances with arbitrary or unknown properties in real-time, we developed a genetic algorithm that builds and refines a set of solutions, called the population. It builds on the general concepts of genetic algorithms, like the one of Sengoku et al. [14]. In iterations called generations, *mutation* is trying to bring some randomly chosen solutions to near local optima, *selection* focuses the search by removing the least promising solutions from the population, and *multiplication* makes up for deletions by combining existing solutions into crossovers, in hope of finding new local optima. An initial population is generated with randomized `Insertion` heuristics inserting nodes iteratively at a position which is chosen with higher probabilities towards positions that lead to lower overall cost. For an additional start solution, the DPH is run with $H = 200$. Mutation of the population makes use of local search strategies on one third of the solutions picked randomly. The main local search is a repeated search in randomly chosen fixed-size subsets of 3-Opt [8] neighbourhoods. We experienced a randomized 3-Opt to be more effective than searching in the full neighbourhood of weaker local searches like 2-Opt. The fixed number of checked neighbours leads to execution times growing only about linearly in the number of nodes, for relevant instance sizes. The basic crossover operation is `CommonEdgesCrossover`, which chooses randomly three parent solutions from the population and constructs a new tour by choosing randomly the edges to traverse. Edges occurring in more parent solutions are chosen with a higher probability and infeasible tours are prohibited if possible. Selection removes the worst fifth of the population, but is also allowed to eliminate solutions based on their affinity to the other solutions in the population to encourage diversity. The said affinity is valued by computing the longest common subsequence shared with some randomly picked solutions from the rest of the population. The population size and the number of generations can be set with a single aggregated parameter, γ, which controls the overall number of performed mutations, with $\gamma = 0$ leading to 750 mutations. For instances with more nodes, those mutations are deployed over more generations but with a smaller population, which we found to be more efficient. We suppose that this is due to bigger neighbourhoods with possibly more potential for bigger instances. The execution of the genetic algorithm can generally be stopped any time leading to the return of the best solution so far, which allows for interruption by users or timing. Infeasible solutions are tolerated but highly penalized: If for a tour $\pi \in \Pi$ and start time τ, a node $v \in V'$ is reached after its last time window, we correct adjust the timing to be ($v^- := \pi(\pi^{-1}(v) - 1)$):

$$t^\pi_\tau(v) = t^\pi_\tau(v^-) + c(v^-, v) + P_\infty,$$

where P_∞ is a *soft infinity* penalty, higher than any feasible tour duration. As a consequence, the tour duration increases by the number of nodes not yet visited ($N - \pi^{-1}(v)$) times P_∞. This allows to improve infeasible tours while always favouring tours that (feasibly) visit more nodes.

4 Experimental Results

The following experimental analysis was conducted based on the rationale of [7]. The test system is a Dell OptiPlex 980 equipped with 16 GB RAM and an Intel Core i7-880 CPU (8MB Cache, 3.06 GHz clock rate) running Windows 7. The algorithms were implemented in Microsoft C# 4.0 and compiled with Microsoft Visual Studio 2010. To evaluate the computation times and solution quality and provide comparable results, we use available instances from the literature and additional real-world instances to rate suitability for use.

We use the instances from `Gendreau` [6] and `Potvin+Bengio` [11][2] and processed them according to Tilk et al. [15]. The former are 120 instances with different parameters for node count (21–101) and width of time windows; the latter are 30 instances with 4 to 46 nodes. All instances have only single time windows per node. The other instances treated by Tilk et al. [15] were omitted since one set originates from a stacker crane context and the other one has instances with more than 126 nodes, for which the current `DPH` implementation is not capable (and which may arise for mobile workforce day trips only in very special circumstances).

Being interested in practicability of the algorithms for real-world scenarios, we adduce another test set consisting of 332 stops assigned to 17 tours with 16 to 24 nodes. The data originates from a logistics company bringing goods from and to collecting points within an urban area. Time windows in most cases resemble a full workday, a half workday, or a full one with a midday closure, with varying times. Around 30% of the nodes have a midday closure leading to two time windows. A fixed-time lunch break for the drivers has already

[2] Both sets downloaded from `http://iridia.ulb.ac.be/~manuel/tsptw-instances`.

Table 1 Aggregated results for different instance groups. (*)-marked averages are only taken over the found solutions. Stats for `GA` are reported as averages of 5 runs.

Instances	Program		#Feas.	#UB	#Imp	UBGAP [%] ∅	UBGAP [%] max	Time [s] ∅	Time [s] max
Gendreau small	`DPH`	H=1500	75	52	2	1.1	16.0	0.4	1.4
(75 instances)	`DPH`	H=5k	75	62	3	0.5	8.3	1.6	5.3
	`DPH`	H=15k	75	64	3	0.2	6.4	4.7	17.2
	`GA`	γ= -10	75	44	3	0.2	2.7	0.6	1.0
	`GA`	$\gamma = 0$	75	53	3	0.0	1.1	2.1	3.4
Gendreau big	`DPH`	H=1500	45	19	1	2.6	10.8	1.9	3.5
(45 instances)	`DPH`	H=5k	45	28	2	1.7	10.6	7.5	13.3
	`DPH`	H=15k	45	29	3	1.2	7.7	23.5	40.0
	`GA`	γ = -10	45	17	1	0.8	5.8	1.9	2.8
	`GA`	$\gamma = 0$	45	25	1	0.2	3.0	6.5	9.4
Potvin+Bengio	`DPH`	H=1500	29	19		1.6*	15.5	0.2	1.3
(30 instances)	`DPH`	H=5k	29	21		1.3*	13.7	0.8	5.3
	`DPH`	H=15k	28	21	1	0.8*	13.4	2.5	13.9
	`GA`	γ = -10	30	15	1	0.2	2.3	0.3	0.6
	`GA`	$\gamma = 0$	30	18	2	0.0	1.0	1.3	1.8
Real data	`DPH`	H=1500	17	5		3.4	11.2	0.2	0.3
(17 instances)	`DPH`	H=5k	17	9		1.6	10.9	0.7	1.5
	`DPH`	H=15k	17	10		1.0	9.6	2.6	5.5
	`GA`	γ = -10	17	15		0.1	0.8	0.6	1.1
	`GA`	$\gamma = 0$	17	17		0.0	0.0	2.6	4.5

been incorporated into the time windows. Compared to the other instances, time windows are rather broad (7:53 hours open spread over the day on average). Other properties differ from the simulated instances, like that 200 of 332 nodes have their first time window starting exactly at 8 o'clock, instead of time windows more randomly scattered around the day or simulated around a reference tour. The sum of travel costs assigned to edges and working time loans plus overtime fees are to be minimized.

Table 1 shows results for the Genetic Algorithm and the `DPH`, with different parameters H aggregating the instances according to Tilk et al. [15] *#Feas.* is the number of instances, for which a solution was found; *#UB* and *#Imp* the count of upper bounds (including optima) reported by Tilk et al. [15] which were hit exactly or improved, respectively. For the new instance set, optima were instead previously calculated using the `DPH` with $H = \infty$. Since, for real-time post-processing of tours, we are interested in good solutions rather than guaranteed optimality, the solution qualities for yet unsolved instances are reported in relation to the upper bounds from Tilk et al. [15], which we consider a good reference due to the overall quality of their algorithm (and effort expended for calculation). *UBGAP* for a given upper bound u and a solution value v is defined as $\frac{v-u}{u}$. Averages of *UBGAP* are over all respective instances, including those, where the upper bound was hit (zero gap) or improved (negative gap). Although the `DPH` leads to satisfying solution quality, it is outperformed by the `GA` on these instance sets, especially the real-world one with very wide time windows. However, this also depends on the composition of the test sets: Table 2 shows results on the `Gendreau`

Table 2 Aggregated results for `Gendreau` instances with narrow time windows. `GA*` is `GA` without the `DPH` start solution. Stats for `GA(*)` are reported as averages of 5 runs.

Instances	Program		#UB	#Imp	UBGAP [%] ∅	UBGAP [%] max	Time [s] ∅	Time [s] max
Gendreau	`DPH`	H=1500	38	1	0.40	5.71	0.9	2.5
w120 + w140	`DPH`	H=5000	46	2	0.01	0.77	3.3	9.4
(50 instances)	`GA`	γ= -10	28		0.36	2.85	1.2	2.9
	`GA`	$\gamma = 0$	34	2	0.13	1.73	4.3	10.0
	`GA*`	γ = -10	22		0.54	2.94	1.0	2.7
	`GA*`	$\gamma = 0$	31	1	0.16	1.69	4.2	9.8

instances with rather tight time windows, for which the `DPH` solves the instances very close to optimality within seconds. The `GA` is noticeably weaker, and when ran without its additional start solution from `DPH` (marked with `*`), even more.
Regarding execution times, not only for instances presented here, we found that the `GA` has a running time roughly linear in the number of nodes (tested up to 200) that grows by about 15% when incrementing the parameter γ by one. The running times of `DPH` are varying stronger and also expectedly depend on the time window width.
We conclude that a combination of both algorithms is promising, for example by running `DPH` on visibly easier (few nodes and/or tight time windows) instances.

Improving Upper Bounds of Unsolved Instances

We ran the `DPH` on the instances with no more than 101 nodes that have not been solved to optimality yet. These are 17 instances with 36 to 101 nodes from the `Gendreau` and `Potvin+Bengio` instance sets. Running the `DPH` with increasing parameter $H \in \{10^3, 10^5, 10^6\}$ (stopping, if the lower bound was reached) we tried to improve the upper bounds reported by Tilk et al. [15]. Detailed results are shown in Table 3. *LB* and *UB* are the bounds previously reported. *Time* is the sum of the execution times in case multiple parameters were run. For the two `Potvin+Bengio` instances we also ran the `GA`, since the `DPH` seemed less effective. For five instances, the upper bound was met exactly and for ten it was improved (values underlined in Table 3), with an average improvement of 2.44%. In five cases our upper bound equals the known lower bound, so that those instances are now solved to optimality. The other five reduced the gap of the best upper bound to the best lower bound by more than half, on average.

5 Conclusion

We presented two algorithmic concepts to treat `TSP`s with (multiple) time windows for which the tour duration is to be minimized, like it is common in many areas of mobile workforce planning. The DP approach is based on aggregating state labels into efficient data structures by encoding intervals of times. It can be used to seek (and prove) optimal solutions, but also as a heuristic for harder instances. Our genetic algorithm applies local searches in a strongly randomized manner leading to good solution qualities, even with very broad time windows. It has a simple parameter to balance running time and (expected) solution quality and can be interrupted, e.g. for online problems, if the input needs to be modified. Both

Table 3 Results on open instances with up to 101 nodes, all times in seconds.

			Result for $H =$			for	Time	Improved
Instance	n	[LB,UB]	1K	100K	1M	`GA`	[s]	UB [%]
n40w200.5	41	[347,350]	347				1	0.9
n60w180.5	61	[466,486]	501	466			150	4.1
n60w200.3	61	[497,525]	497				1	5.3
n80w140.2	81	[588,591]	592	589	589		765	0.3
n80w140.3	81	[615,617]	632	617	617		912	
n80w140.4	81	[549,561]	583	550	550		962	2.0
n80w160.2	81	[603,609]	637	629	629		1659	
n80w160.3	81	[633,638]	676	651	633		2061	0.8
n80w160.5	81	[583,584]	627	584	584		1771	
n80w180.2	81	[564,570]	615	591	570		2059	
n80w180.5	81	[570,571]	573	571	571		2210	
n80w200.1	81	[559,584]	620	566	564		2809	3.4
n80w200.2	81	[549,550]	603	582	560		2901	
n100w120.2	101	[843,846]	843				2	0.4
n100w140.2	101	[948,949]	954	949	949		1722	
rc_208.1	38	[73432,79904]	-	-	83683	79348	524	0.7
rc_208.3	36	[61302,67902]	84723	64499	64123	63436	1604	6.6

implementations are applicable for practical real-time optimization and post-processing. Computational results showed that very satisfying solutions can be found with minimal computing times. We even improved the best solution reported so far for 10 out of the 17 unsolved instances from the `Gendreau` and `Potvin+Bengio` benchmarks.

Acknowledgements. The authors would like to thank Thomas Brechtel for many fruitful discussions. Furthermore, the authors would like to thank the anonymous referees for many helpful suggestions which led to the improvement of the presentation.

References

1 Slim Belhaiza, Pierre Hansen, and Gilbert Laporte. A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. *Computers & Operations Research*, 52:269–281, 2014.

2 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

3 Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

4 Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

5 Peter Eades, Brendan D McKay, and Nicholas C Wormald. On an edge crossing problem. In *Proc. 9th Australian Computer Science Conference*, volume 327, page 334, 1986.

6 Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.

7 David S Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59:215–250, 2002.

8 Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal, The*, 44(10):2245–2269, 1965.

9 Chryssi Malandraki and Robert B Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45–55, 1996.

10 Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. On the flexibility of constraint programming models: From single to multiple time windows for the traveling salesman problem. *European Journal of Operational Research*, 117(2):253–263, 1999.

11 Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part II: genetic search. *INFORMS journal on Computing*, 8(2):165–172, 1996.

12 Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

13 Martin WP Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

14 Hiroaki Sengoku and Ikuo Yoshihara. A fast TSP solver using GA on Java. In *Third International Symposium on Artificial Life, and Robotics (AROB III'98)*, pages 283–288, 1998.

15 Christian Tilk and Stefan Irnich. Dynamic programming for the minimum tour duration problem. Technical Report LM-2014-04, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany, 2014.

A Appendix: Proofs

Proof of Lemma 1. We show the claim by induction over i; fix $\tau, \delta \in \mathbb{N}$.
The equality $t^{\pi}_{\tau+\delta}(\pi(0)) = \tau + \delta \geq \tau = t^{\pi}_{\tau}(\pi(0))$ yields the induction base.
For the induction step, let $0 < i < N,\ v := \pi(i),\ v^- = \pi(i-1)$ and assume

$$t^{\pi}_{\tau+\delta}(v^-) \geq t^{\pi}_{\tau}(v^-). \tag{H}$$

Then it follows, by definitions of $T^{\rightarrow}$ and $t^{\pi}_{t_0}$ and (H):

$$\begin{aligned}
t^{\pi}_{\tau+\delta}(v) &= T^{\rightarrow}(v, t^{\pi}_{\tau+\delta}(v^-) + c(v^-, v)) \\
&= \min\{x \mid x \geq t^{\pi}_{\tau+\delta}(v^-) + c(v^-, v) \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\} && (T^{\rightarrow}) \\
&\geq \min\{x \mid x \geq t^{\pi}_{\tau}\quad(v^-) + c(v^-, v) \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\} && (\text{H}) \\
&= T^{\rightarrow}(v, t^{\pi}_{\tau}(v^-) + c(v^-, v)) && (T^{\rightarrow}) \\
&= t^{\pi}_{\tau}(v) && \blacktriangleleft
\end{aligned}$$

Proof of Lemma 3. Construct states $s1 := (S, \ell, t)$ and $s2 := (S, \ell, t + \delta)$ according to the supposition.

Case 1). It is clear that for $S \cup \{\ell\} = V'$, state $s1$ leads, by extension towards the depot, to a TSP-Tour with tour duration $T_1 + c(\ell, 0)$ which dominates the extension of $s2$ to the depot, concluding a TSP-Tour with duration $T_2 + c(\ell, 0) \geq T_1 + c(\ell, 0)$.

Case 2). For $S \cup \{\ell\} \subsetneq V'$, fix an arbitrary $\ell' \in V' \setminus (S \cup \{\ell\})$ and regard the forward propagation of $s1$ and $s2$ towards ℓ', leading to labelling of states $s1'$ and $s2'$, respectively.
State $s1' = (S \cup \{\ell\}, \ell', T^{\rightarrow}(\ell', t + c(\ell, \ell')))$ is labelled with $T_1' = T_1 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + c(\ell, \ell'))$.
$s2' = (S \cup \{\ell\}, \ell', T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')))$ is labelled with $T_2' = T_2 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell'))$.
Set $\delta' := T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - T^{\rightarrow}(\ell', t + c(\ell, \ell'))$. Then:

$$\begin{aligned}
T_2' - T_1' &= T_2 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) \\
&\quad - (T_1 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + c(\ell, \ell'))) \\
&= T_2 - T_1 + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - W^{\rightarrow}(\ell', t + c(\ell, \ell')) \\
&\geq \delta + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - W^{\rightarrow}(\ell', t + c(\ell, \ell')) \\
&= \delta + T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - (t + \delta + c(\ell, \ell')) \\
&\quad - T^{\rightarrow}(\ell', t + c(\ell, \ell')) + (t + c(\ell, \ell')) \\
&= T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - T^{\rightarrow}(\ell', t + c(\ell, \ell')) = \delta'
\end{aligned}$$

The initial situation is reiterated. Since S is of increasing cardinality this iteration converges to Case 1). ◀

Proof of Equation 5. We prove that Equation 5 follows from Equation 4, if (2),(3) hold. It is to be shown that of the labelled intervals from $\mathcal{I}$ with $t_s \leq t$, the one with maximal t_e (uniquely defined with (2) holding) also maximizes $T + \max\{0, t - t_e\}$. This is clear, if there is only one labelled interval in $\mathcal{I}$ with $t_s \leq t$. Otherwise, fix two distinct labelled intervals $([t_s, t_e], T), ([t_s', t_e'], T') \in \mathcal{I}$ with $t_s, t_s' \leq t$. With (2), one of them is earlier, say $t_e < t_e'$ and $t_e < t_s'$. With (3) we have $t_s' > t_e + (T' - T)$. This leads to:

$$\begin{aligned}
T' + \max\{0, t - t_e'\} &\leq T' + \max\{0, t - t_s'\} && (t_e' \geq t_s') \\
&= T' + t - t_s' && (t_s' \leq t) \\
&< T + t - t_e && (t_s' > t_e + (T' - T)) \\
&= T + \max\{0, t - t_e\} && (t_e < t_s' \leq t)
\end{aligned}$$

◀

B Appendix: Additional Pseudocode

Propagation of labelled intervals. The forward propagation of labels is shown in Algorithm 2. Adjusting the intervals to conform to Equation 3 is omitted here. We write $\mathcal{I}[i]$ for the i-th labelled interval of a sorted set $\mathcal{I}$ of labelled intervals, and write a labelled interval i as $([i.t_s, i.t_e], i.T)$.

Algorithm 2: Propagation of labelled intervals.

```
Data: Labelled intervals $\mathcal{I}$ for $(S, \ell)$ satisfying equations (2) and (3),
Travel time $c = c(\ell, \ell')$ from node $\ell$ to next node $\ell' \in V' \setminus (S \cup \{\ell\})$.
Result: $\mathcal{I}'$: Propagated intervals $\mathcal{I}$ towards node $\ell'$.
$i \leftarrow 0$;
for $k$ from 0 to $K_{\ell'} - 1$ do
    while $i < |\mathcal{I}| - 1$ and $\mathcal{I}[i+1].t_s + c \leq a_{\ell',k}$ do
        i++;
    if $i \geq |\mathcal{I}|$ then
        break;
    if $\mathcal{I}[i].t_e + c < a_{\ell',k}$ then
        Add $([a_{\ell',k}, a_{\ell',k}], \mathcal{I}[i].T + a_{\ell',k} - \mathcal{I}[i].t_e)$ to $\mathcal{I}'$;
        i++;
    while $i < |\mathcal{I}|$ and $\mathcal{I}[i].t_s + c \leq b_{\ell',k}$ do
        $t'_s \leftarrow \max(\mathcal{I}[i].t_s + c,\ a_{\ell',k})$;
        $t'_e \leftarrow \min(\mathcal{I}[i].t_e + c,\ b_{\ell',k})$;
        Add $([t'_s, t'_e], \mathcal{I}[i].T + c)$ to $\mathcal{I}'$;
        i++;
```