An efficient timing algorithm for drivers with rest periods

Giovanni Righini[1 [0000−0001−9830−7454]] and Marco Trubian[1 [0000−0002−3523−817X]]

University of Milan, Dept. of Computer Science, Milan, Italy
giovanni.righini@unimi.it, marco.trubian@unimi.it

Abstract. We consider a timing problem arising from a vehicle routing context: it consists of optimally inserting rest periods of given duration into drivers' schedules, when the sequence of customers to visit is given, time windows are associated with customers and an upper limit is imposed on the driving time with no rest periods. We illustrate some properties that allow reformulating the problem in simpler terms, and provide the basis to design a very efficient exact optimization algorithm whose worst-case time complexity is $O(n \log n)$, where n is the number of customers to be visited.

Keywords: Timing problem · Time window constraints.

1 Introduction

Timing problems arise when a sequence of activities requires optimal start times to optimize an objective function. Unlike scheduling problems, timing problems are simpler as they do not need defining the sequence optimally. However, they are crucial due to the need to solve them for each tentative sequence, often a large number, in applications like exact optimization algorithms (e.g., branch-and-bound) and approximation algorithms (e.g., local search). In these algorithms, a vast set of partial solutions is explored, each with a different partial sequence, requiring to solve a timing problem for upper and lower bound computations. Similarly, in approximation algorithms, exploring the neighborhood of a solution demands evaluating different sequences, each requiring a timing problem solution. Another typical need is real-time re-optimization of routes with rest periods, owing to unexpected delays that make planned solutions infeasible. Hence, while timing problems may have polynomial-time solutions, the search for highly efficient exact optimization algorithms, ideally with linear or near-linear time complexity, remains significant.

For a recent overview of timing problems, refer to Vidal et al. [2], covering various settings like production scheduling, network optimization, energy optimization, and statistical inference.

Our motivation originates from a real-world vehicle routing application, where driver routes are assessed within a local search framework. Each sequence necessitates finding corresponding timings rapidly. The routing problem is complicated by two factors: customer-associated time windows and restrictions on maximum allowed driving time without rest.

In this paper, we analyze the basic version of the timing problem and we demonstrate how finding the optimal solution for a sequence of visits to n customers has the same computational complexity as sorting n elements, i.e., $O(n \log n)$. In the conclusions, we illustrate how this result easily extends to more complex problem variations.

2 Problem definition

The Timing Problem with Rest Periods (TPRP in the remainder) is defined as follows.

A sequence of customers to be visited is given. Let N be the indexed set of customers: $N = \{0, 1, 2, \ldots, n\}$, where 0 and n are the indices of the initial and final depots. Each customer $i \in N$ (including the initial and final depots) has an associated time window $[a_i, b_i]$. A solution is feasible only if the service at each customer starts within the corresponding time window. Traveling times are given for each pair of consecutive customers in the sequence. Let $d_i \geq 0$ be the time taken to travel from customer $i-1$ to customer i for each $i = 1, \ldots, n$. At each customer $i \in N$ it is required to work out some operations, whose duration $s_i \geq 0$ is known ($s_0 = s_n = 0$).

The maximum allowed driving time with no rest periods is indicated by T. The duration of a rest period is indicated by δ. All rest periods have the same duration, they cannot be interrupted and they cannot overlap with service to customers. The service time at customers is neither driving time nor rest time; while serving a customer, the driver's fatigue level remains constant. The same holds for waiting time, that occurs in case of early arrival at a customer (before the beginning of its time window).

The problem is to decide when to optimally insert the necessary rest periods. The objective is to minimize the total duration of the route.

3 Properties and reformulations

Any instance of the TPRP can be equivalently reformulated (and simplified) as follows. We distinguish four mutually exclusive types of time periods: driving time, rest time, waiting time and service time.

Elimination of service time. Service times are fixed: they can be taken into account as if the driver's clock would stop while a customer is served. An equivalent instance is obtained by redefining the time windows as shown in Algorithm 1.

Elimination of travel time. Since traveling periods have a given duration and cannot overlap with rest periods and waiting periods, they can be eliminated. An equivalent istance is obtained by recomputing all time windows again, as shown in Algorithm 2.

Algorithm 1 Elimination of service time.

$\Delta \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  $a_i \leftarrow a_i - \Delta$
  $b_i \leftarrow b_i - \Delta$
  $\Delta \leftarrow \Delta + s_i$

Algorithm 2 Elimination of travel time.

$\Delta \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  $\Delta \leftarrow \Delta + d_i$
  $a_i \leftarrow a_i - \Delta$
  $b_i \leftarrow b_i - \Delta$

Symmetry. After reformulating the generic problem instance, the service at each customer is instantaneous and hence a variable t_i can represent both the arrival time and the departure time at/from each customer $i \in N$.

The constraint on rest periods imposes to insert a minimum number λ_i of rest periods before each customer $i \in N$:

$$\lambda_i = \left\lceil \frac{\sum_{j=1}^{i} d_j}{T} \right\rceil - 1.$$

For the same reason, symmetrically, the number of rest periods after each customer $i \in N$ is lower bounded by a limit μ_i:

$$\mu_i = \left\lceil \frac{\sum_{j=i+1}^{n} d_j}{T} \right\rceil - 1.$$

To minimize the total duration of the travel, the total number of rest periods must be minimum, that is equal to

$$\pi = \left\lceil \frac{\sum_{j=1}^{n} d_j}{T} \right\rceil - 1.$$

For each customer $i \in N$, either $\pi = \lambda_i + \mu_i$ or $\pi = \lambda_i + \mu_i + 1$.

Proof. Indicating for brevity $u = \dfrac{\sum_{j=1}^{i} d_j}{T}$, $v = \dfrac{\sum_{j=i+1}^{n} d_j}{T}$ and observing that $\dfrac{\sum_{j=1}^{n} d_j}{T} = \dfrac{\sum_{j=1}^{i} d_j}{T} + \dfrac{\sum_{j=i+1}^{n} d_j}{T}$, it holds $\lambda = \lceil u \rceil - 1$, $\mu = \lceil v \rceil - 1$ and $\pi = \lceil (u+v) \rceil - 1$. Hence, $u-1 \leq \lambda < u$, $v-1 \leq \mu < v$ and therefore, summing up the inequalities, $(u-1)+(v-1) \leq \lambda+\mu < (u+v)$. Analogously, $(u+v)-1 \leq \pi < (u+v)$. Hence, by difference from the inequalities above, $-1 < \pi - (\lambda + \mu) < 2$. Since λ, μ and π are integers, strict inequalities imply $0 \leq \pi - (\lambda + \mu) \leq 1$ (Q.E.D.).

Example. In a problem instance A: $n = 2$, $d_1 = d_2 = 12$ and $T = 10$. Then $\pi = 2$ and for customer 1, $\lambda_1 = 1$ and $\mu_1 = 1$. In a problem instance B: $n = 2$, $d_1 = d_2 = 18$ and $T = 10$. Then $\pi = 3$ and for customer 1, $\lambda_1 = 1$ and $\mu_1 = 1$. The second rest period in instance B can be placed before or after customer 1.

We partition customers for which $\pi = \lambda_i + \mu_i$ from those for which $\pi = \lambda_i + \mu_i + 1$. We define U to be the set of the former ones and V the set of the latter ones. U and V are a partition of N. Now, we further partition U in subsets U_k for $k = 0, \ldots, \pi$, each one containing customers $i \in U$ for which $\lambda_i = k$. For sure, the depot 0 belongs to U_0 and depot n belongs to U_π. Analogously, for each value of $k = 1, \ldots, \pi$ let V_k be the subset of customers $i \in V$ such that $\lambda_i = k - 1$ and $\mu_i = \pi - k$. All subsets V_k are disjoint and they form a partition of V.

Elimination of rest periods. For all customers $i \in U$ the number of rest periods before and after the visit is forced to the pre-computed values λ_i and μ_i, respectively, and it can be eliminated, i.e. treated as a constant as for travel and service time.

For customers $i \in V_k$, for each value of $k = 1, \ldots, \pi$ only two cases are possible, that is with $\lambda_i = k - 1$ rest periods before the visit and $\mu_i + 1 = \pi - k + 1$ rest periods after it and with $\lambda_i + 1 = k$ rest periods before the visit and $\mu_i = \pi - k$ rest periods after it. This correponds to generate pairs of time windows, indicated by $[a'_i, b'_i]$ and $[a''_i, b''_i]$ for each $i \in V_k$ for each $k = 1, \ldots, \pi$, as shown in Algorithm 3.

Algorithm 3 Elimination of rest periods.

for $i = 1, \ldots, n$ **do**
  if $i \in U$ **then**
    $a_i \leftarrow a_i - \lambda_i \delta$
    $b_i \leftarrow b_i - \lambda_i \delta$
  else
    $a'_i \leftarrow a_i - \lambda_i \delta$
    $b'_i \leftarrow b_i - \lambda_i \delta$
    $a''_i \leftarrow a_i - (\lambda_i + 1)\delta$
    $b''_i \leftarrow b_i - (\lambda_i + 1)\delta$

Tightening the time windows. Some parts of the time windows can be eliminated, because they cannot be used in any feasible solution. A too small value of a_i may correspond to a point in time that is too early, so that the vehicle cannot reach customer i in that moment. Symmetrically, a too large value of b_i may correspond to a point in time that is too late, because leaving customer i at time b_i would not allow the vehicle to reach the next customer in time. Time windows can be tightened by comparing consecutive time windows, as shown in Algorithm 4. All these preprocessing steps are illustrated by the example described in Table 1. It refers to an instance with 6 customers, where $T = 10$ and $\delta = 4$. The total distance to travel is 42; hence $\pi = 4$.

i	d_i	$\sum_{j=1}^{i} d_j$	$\sum_{j=i+1}^{n} d_j$	λ_i	μ_i	Set	$[a, b]$	$[a'', b'']$	$[a', b']$	$[a, b]$	$[a'', b'']$	$[a', b']$	$[a, b]$
0		0	42	0	4	U_O	$[0, 50]$			$[0, 50]$			$[0, 18]$
1	8	8	34	0	3	V_1	$[12, 25]$	$[4, 17]$	$[8, 21]$		$[4, 17]$	$[8, 18]$	
2	10	18	24	1	2	V_2	$[14, 37]$	$[2, 25]$	$[6, 29]$		$[4, 18]$	$[8, 18]$	
3	3	21	21	2	2	U_2	$[16, 30]$			$[8, 22]$			$[8, 18]$
4	10	31	11	3	1	U_3	$[13, 39]$			$[1, 27]$			$[8, 18]$
5	3	34	8	3	0	V_4	$[26, 38]$	$[6, 18]$	$[10, 22]$		$[8, 18]$	$[10, 22]$	
6	2	36	6	3	0	V_4	$[19, 44]$	$[-1, 24]$	$[3, 28]$		$[8, 24]$	$[10, 28]$	
7	6	42	0	4	0	U_4	$[28, 50]$			$[12, 34]$		$[12, 34]$	

Table 1. An example with 6 customers besides the depot. Column 8 reports the time windows after the elimination of service times and travel times. Columns $9 - 11$ report the time windows after the elimination of rest periods. Columns $12 - 14$ report the tightened time windows.

Algorithm 4 Time windows tightening.

$a \leftarrow a_0$
for $i \in U_0$ **do**
  $a_i \leftarrow \max\{a_i, a\}$
  $a \leftarrow \max\{a, a_i\}$
$a'' \leftarrow a$
$a' \leftarrow a$
for $k = 1, \ldots, \pi$ **do**
  for $i \in V_k$ **do**
    $a'_i \leftarrow \max\{a'_i, a'\}$
    $a' \leftarrow \max\{a', a'_i\}$
    $a''_i \leftarrow \max\{a''_i, a''\}$
    $a'' \leftarrow \max\{a'', a''_i\}$
  $a \leftarrow a''$
  for $i \in U_k$ **do**
    $a_i \leftarrow \max\{a_i, a\}$
    $a \leftarrow \max\{a, a_i\}$
  $a'' \leftarrow a$
  $a' \leftarrow a$
$b \leftarrow b_n$
for $k = \pi, \ldots, 1$ **do**
  for $i \in U_k$ **do**
    $b_i \leftarrow \min\{b_i, b\}$
    $b \leftarrow \min\{b, b_i\}$
  $b' \leftarrow b$
  $b'' \leftarrow b$
  for $i \in V_k$ **do**
    $b'_i \leftarrow \min\{b'_i, b'\}$
    $b' \leftarrow \min\{b', b'_i\}$
    $b''_i \leftarrow \min\{b''_i, b''\}$
    $b'' \leftarrow \min\{b'', b''_i\}$
  $b \leftarrow b'$
for $i \in U_0$ **do**
  $b_i \leftarrow \min\{b_i, b\}$
  $b \leftarrow \min\{b, b_i\}$

4 Minimization of the total waiting time

After these pre-processing steps, only waiting times are left: travels, rest periods and customer services have got null duration. Therefore, the solution is to be found by defining a sequence of points in time t_i, representing the visit to each customer $i \in N$ in the modified instance, such that

1. $t_i \geq t_{i-1}\ \forall i = 1, \ldots, n$;
2. $a_i \leq t_i \leq b_i\ \forall i \in U$.

Hence, should $b_i < a_i$ occur for some $i \in U$, the instance would be infeasible.

The time window constraint for customers in V can be formulated as follows: for each $k = 1, \ldots, \pi$, the set V_k must be partitioned into two subsets V'_k and V''_k of consecutive customers, where all customers in V'_k precede those in V''_k so that

1. $a'_i \leq t_i \leq b'_i\ \forall i \in V'_k$;
2. $a''_i \leq t_i \leq b''_i\ \forall i \in V''_k$.

This choice corresponds to insert the rest period with index k after the customers in V'_k and before those in V''_k. As a special case, one of the two subsets can be left empty.

Special case. If $a_n \leq b_0$, the modified instance may allow for an optimal solution of null duration, where $t_i = t\ \forall i \in N$ with $t \in [a_n, b_0]$, i.e. with no waiting time. This is guaranteed to happen if all time windows have an initial width not smaller than δ.

Proof. Owing to the non-decreasing order of the endpoints of the time windows ($a_{i+1} \geq a_i$ and $b_{i+1} \geq b_i$ for all $i \in N$), the inequality $t \leq b_0$ implies $t \leq b_i\ \forall i \in U$ and $t \leq b'_i\ \forall i \in V$; symmetrically, $t \geq a_n$ implies $t \geq a_i\ \forall i \in U$ and $t \geq a''_i\ \forall i \in V$.

If all time windows have an initial width greater than or equal to δ, this guarantees that after the elimination of rest periods and before time windows tightening $b_i - a_i \geq \delta \ \forall i \in U$ and $b''_i \geq a'_i \ \forall i \in V$. Therefore, any value of t between a''_i and b'_i for $i \in V$ is feasible with respect to at least one of the two time windows of i. In other terms, no "holes" are generated between the two time windows of any customer $i \in V$. Furthermore, when going from i to $i+1$ it is never necessary to pass from a time window $[a''_i, b''_i]$ to a time window $[a'_{i+1}, b'_{i+1}]$, because it cannot happen that $a'_i > b''_{i+1}$.

Hence, solutions with no waiting time are feasible (Q.E.D.).

For instance, this happens in the example illustrated in Table 1, where $a_n = 12$ and $b_0 = 18$.

Finally, if $b_0 < a_n$, then there exists an optimal solution with waiting time equal to $b_0 - a_n$, whose feasibility is proven in the same way. It is always possible to set $t_0 = b_0$ and $t_n = a_n$ and to find non-decreasing values of t_i for all intermediate customers, complying with all time windows, since no "holes" exist between them.

General case. In a more general case, where time windows may be initially smaller than δ, the above proof is no longer valid. In this case, empty intervals may occur between $[a''_i, b''_i]$ and $[a'_i, b'_i]$ for some $i \in V$. It is still true that $a_{i+1} \geq a_i$ and $b_{i+1} \geq b_i$ for all i, and possible "holes" may occur only between b''_i and a'_i for some $i \in V$ (not before a''_i nor after b'_i).

To deal with this more general case, we exploit the constraint that all time windows $[a', b']$ must be traversed before all time windows $[a'', b'']$ for the customers in each set V_k. Hence, each set of customers V_k originates $|V_k| + 1$ possible sequences of time windows. Each of them corresponds to the choice of inserting the rest period in position $\ell = 0, \ldots, |V_k|$ in the ordered sequence of the customers in V_k. For each choice of $\ell = 0, \ldots, |V_k|$ the intersection of the time windows of the customers in V_k is $[a^k_\ell, b^k_\ell]$ with endpoints in

$$a^k_\ell = \max\{\max_{i \in V_k : i > \ell}\{a''_i\}, \max_{i \in V_k : i \leq \ell}\{a'_i\}\}$$

$$b^k_\ell = \min\{\min_{i \in V_k : i > \ell}\{b''_i\}, \min_{i \in V_k : i \leq \ell}\{b'_i\}\}.$$

If $a^k_\ell > b^k_\ell$ for some ℓ, then the corresponding time window does not exist. In this way, each subset V_k can be replaced by a single dummy customer with multiple time windows. If two or more of its time windows overlap, they can be merged in a single time window.

For instance, considering the two customers numbered 5 and 6 in the example above, both belonging to set V_4, there are three alternatives: if $\ell = 0$, then the rest period is inserted before customer 5 and $[a^k_\ell, b^k_\ell] = [8, 18]$; if $\ell = 1$, then the rest period is inserted between customers 5 and 6 and $[a^k_\ell, b^k_\ell] = [10, 22]$; if $\ell = 2$, then the rest period is inserted after customer 6 and $[a^k_\ell, b^k_\ell] = [10, 22]$. Since the three time windows overlap, the subset V_4 can be replaced by a dummy customer with a single time window $[8, 22]$.

On the contrary, in the example illustrated in Figure 1 and Table 2 multiple time windows are generated. The example refers to an instance with 3 customers, where

$T = 100$ and $\delta = 20$. The total distance to travel is 110; hence $\pi = 1$. All three customers belong to the same subset V_1, that can be traversed in four ways indexed by $\ell = 1, \ldots, 4$ corresponding to the four possible insertion positions of the rest period. The corresponding time windows are: $a_0^1 = [5, 12]$, $a_1^1 = [15, 19]$, a_2^1 empty, $a_3^1 = [25, 32]$. Hence the subset V_1 can be replaced by a dummy customer with three disjoint time windows.

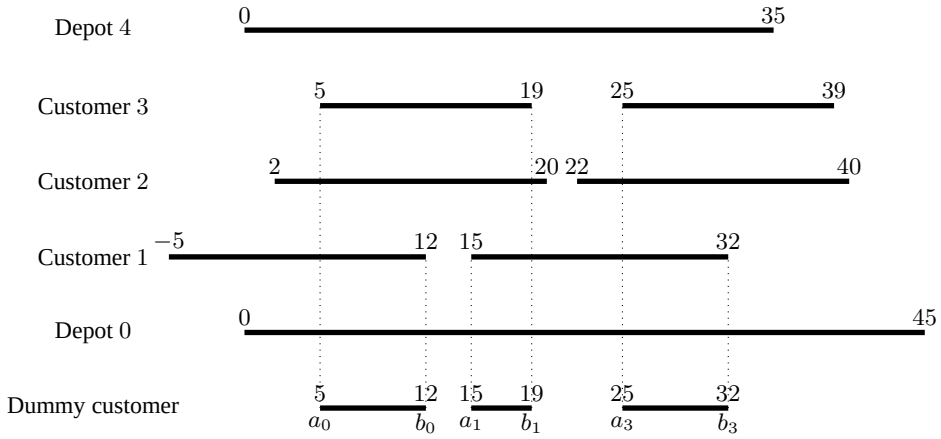

Fig. 1. Time windows $[a', b']$ and $[a'', b']$ of three customers replaced by multiple time windows of a single dummy customer.

i	d_i	$\sum_{j=1}^{i} d_j$	$\sum_{j=i+1}^{n} d_j$	λ_i	μ_i	Set	$[a, b]$	$[a'', b'']$	$[a', b']$	$[a, b]$	$[a'', b'']$	$[a', b']$	$[a, b]$
0		0	110	0	1	U_O	$[0, 45]$			$[0, 45]$			$[0, 32]$
1	60	60	90	0	0	V_1	$[15, 32]$	$[-5, 12]$	$[15, 32]$		$[0, 12]$	$[15, 32]$	
2	10	70	70	0	0	V_1	$[22, 40]$	$[2, 20]$	$[22, 40]$		$[2, 20]$	$[22, 35]$	
3	20	90	60	0	0	V_1	$[25, 39]$	$[5, 19]$	$[25, 39]$		$[5, 19]$	$[25, 35]$	
4	20	110	0	1	0	U_1	$[20, 55]$			$[0, 35]$			$[5, 35]$

Table 2. An example with 3 customers besides the depot. Column 8 reports the time windows after the elimination of service times and travel times. Columns $9 - 11$ report the time windows after the elimination of rest periods. Columns $12 - 14$ report the tightened time windows.

Subsets U_k are much easier to consider, because they can be feasibly traversed in one way, since no rest periods must be inserted in them. Hence, each subset U_k is simply replaced by a dummy customer with a single time window that is the intersection

of the time windows of the real customers in U_k.

Now, the residual problem is to find an optimal path through the multiple time windows of the dummy customers originated by subsets U_k and V_k for $k = 1, \ldots, \pi$. In the remainder index j is used to indicate dummy customers; their number m is upper bounded by n and by $2\pi + 1$.

An algorithm to solve this problem is the following. Consider the "columns", made by sequences of consecutive customers with overlapping time windows. As a special case, a column can be made by a single customer. Once all columns have been listed, one searches for a shortest path on a weighted acyclic digraph with a node for each column; the weight of each arc can be determined easily, as shown hereafter.

The algorithm to list all columns is illustrated in Algorithm 5 and it uses the following data-structures:

- Data-structures to describe customers:
  - $ptop[j]$: vector with the index of the column terminating at customer j;
  - $pbot[j]$: vector with the index of the column starting at customer j.
- Data-structures to describe columns:
  - $top[c]$: vector with the last customer of each column c;
  - $bot[c]$: vector with the first customer of each column c;
  - $s[c]$: start time of column c;
  - $e[c]$: end time of column c.
- Data-structures to describe time windows:
  - $vertex[w]$: customer of time window w;
  - $dir[w]$: binary datum, where 1 means that w starts, 0 indicates that w ends;
  - $time[w]$: start time or end time of time window w.

Other indices that are used in Algorithm 5:

- $j = 0, \ldots, m$: customer index;
- nc: number of columns;
- $c = 1, \ldots, nc$: column index;
- nw: number of time windows;
- $w = 1, \ldots, nw$: time window index;
- t: current point in time.

The sub-routine $FindColumn(j)$ takes in input a customer index j and finds the column containing j among those already started and not yet terminated when the sub-routine is called.

Algorithm 5 The algorithm that lists all columns.

for $j = 1, \ldots, m$ **do**
  $ptop[i] \leftarrow 0$
  $pbot[i] \leftarrow 0$
$nc \leftarrow 0$
/* Populate and sort the vectors $vertex$, dir and $time$ with start and end time of all time windows */
for $w = 1, \ldots, nw$ **do**
  $j \leftarrow vertex[w]$
  $t \leftarrow time[w]$
  if $dir[w] = 1$ **then**
    $WOpen(j)$
  else
    $WClose(j)$

Algorithm 6 Procedure $WOpen(j)$

$nc \leftarrow nc + 1$
// Close the column above j, if any //
if $(j < m) \wedge (pbot[j+1] \neq 0)$ **then**
  $c \leftarrow pbot[j+1]$
  $e[c] \leftarrow t$
  $pbot[j+1] \leftarrow 0$
  $top[nc] \leftarrow top[c]$
  $Succ[c] \leftarrow nc$ (1)
else
  $top[nc] \leftarrow j$
  if $tail[j] \neq 0$ **then**
    $Succ[tail[j]] \leftarrow nc$ (2)
// Close the column under j, if any //
if $(j > 0) \wedge (ptop[j-1] \neq 0)$ **then**
  $c \leftarrow ptop[j-1]$
  $e[c] \leftarrow t$
  $ptop[j-1] \leftarrow 0$
  $bot[nc] \leftarrow bot[c]$
  $Succ[c] \leftarrow nc$ (3)
else
  $bot[nc] \leftarrow j$
  if $tail[j-1] \neq 0$ **then**
    $Succ[tail[j-1]]] \leftarrow nc$ (4)
// Open a new column including j //
$s[nc] \leftarrow t$
$ptop[top[nc]] \leftarrow nc$
$pbot[bot[nc]] \leftarrow nc$
// Update tails //
$tail[j] \leftarrow 0$
$tail[j-1] \leftarrow 0$
$tail[top[nc]] \leftarrow nc$

Algorithm 7 Procedure $WClose(j)$

$c \leftarrow FindColumn(j)$
$h \leftarrow top[c]$
$k \leftarrow bot[c]$
// Close the column with j //
$e[c] \leftarrow t$
$ptop[h] \leftarrow 0$
$pbot[k] \leftarrow 0$
if $h > j$ **then**
  // Open a new column above j //
  $nc \leftarrow nc + 1$
  $top[nc] \leftarrow h$
  $bot[nc] \leftarrow j + 1$
  $s[nc] \leftarrow t$
  $ptop[h] \leftarrow nc$
  $pbot[j+1] \leftarrow nc$
  $Succ[c] \leftarrow nc$ (5)
  $tail[h] \leftarrow nc$
if $k < j$ **then**
  // Open a new column under j //
  $nc \leftarrow nc + 1$
  $top[nc] \leftarrow j - 1$
  $bot[nc] \leftarrow k$
  $s[nc] \leftarrow t$
  $ptop[j-1] \leftarrow nc$
  $pbot[k] \leftarrow nc$

A numerical example is shown in detail in Figure 2 and Table 3.

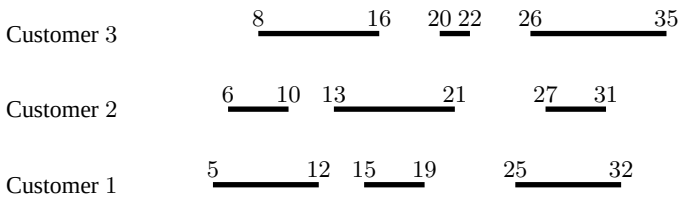

Fig. 2. Multiple time windows of three (dummy) customers.

w	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$vertex$	1	2	3	2	1	2	1	3	1	3	2	3	1	3	2	2	1	3
$time$	5	6	8	10	12	13	15	16	19	20	21	22	25	26	27	31	32	35
dir	1	1	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0

Table 3. List of all columns obtained from the multiple time windows shown in Figure 2.

The datum $tail[j]$ for each customer j, when different from 0, is the index of the column that is the tail of an arc in the digraph. The corresponding head of the arc is identified when a column is found containing customer j or $j+1$. When this occurs the new column is identified as the head of the arc. All arcs found in this way are stored in the data-structure $Succ$ which has a component for each column, since in the acyclic digraph each node has at most one successor.

Figures 3 and 4 show the different cases that can occur when executing $WOpen(j)$ and $WClose(j)$, respectively.

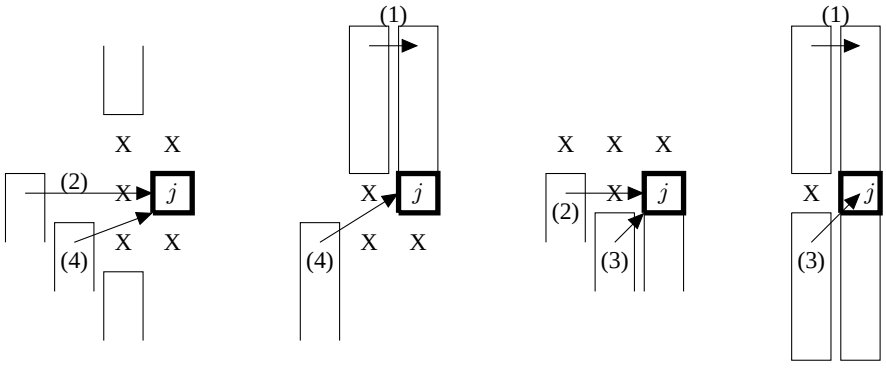

Fig. 3. Linking columns with $WOpen(j)$.

Once all columns have been listed, an acyclic digraph is defined with two nodes for each column (start and end of the corresponding time window). Columns that con-

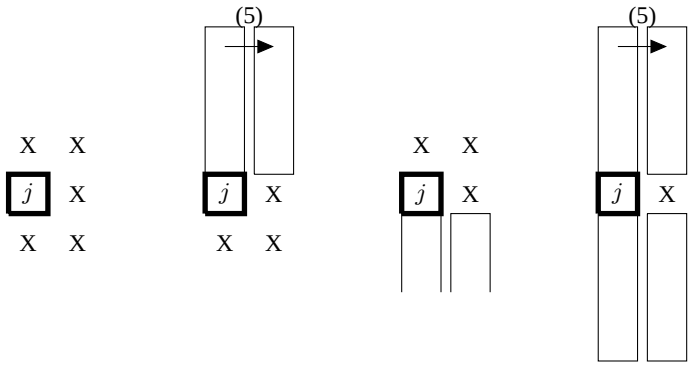

Fig. 4. Linking columns with $WClose(j)$.

tain the initial depot 0 have a single node corresponding to the end of their time window, while those containing the final depot n have a single node coorresponding to the beginning ot their time window. The arc between the two nodes of a same column $c = 1, \ldots, n-1$ has weight $e[c] - s[c]$. The weight of any arc from the end node of a column c to the start node of $Succ[c]$ is set equal to $s[Succ[c]] - e[c]$. Each column c has at most one leaving arc, connecting it to the first successive column that contains customer $top[c]$ or $top[c] + 1$.

4.1 Complexity

The number of dummy customers m is not larger than the number of real customers n. The number of time windows nw is also $O(n)$ by construction. Therefore the number of columns is also $O(n)$, because the number of columns increases by one or two units at every start time or end time of a time window. The number of arcs in the digraph is twice the number of columns and hence $O(n)$. The search for an optimal path in a weighted acyclic digraph has linear complexity in the number of arcs, i.e. $O(n)$. The procedure $FindColumns$ has complexity $O(\log n)$ if the list of the columns already started and not yet terminated is implemented as a binary tree, so that $O(\log n)$ is the complexity for inserting a new column (and this is done $O(n)$ times), as well as for deleting a column (this is also done $O(n)$ times) and also for finding the column containing a given customer (this is also done $O(n)$ times).

It remains to evaluate the complexity for sorting the time windows endpoints. If time windows are already sorted for each customer, it is necessary to merge $O(u)$ ordered lists, each one with $O(v)$ elements, where $uv = n$. Merging two sorted lists of $O(v)$ elements takes $O(v)$. Joining list of the same length we have $\log_2 u$ levels, each one with complexity $uv/2$, which yields $O(uv \log u)$ complexity. The worst case is with few windows per customer: when u tends to n and v tends to 1, the complexity tends to $O(n \log n)$.

If time windows are initially unsorted, then it is also needed to sort them, which takes $O(n \log n)$.

Therefore the worst-case time complexity of the algorithm is $O(n \log n)$ (which improves upon the complexity of the dynamic programming algorithm developed in [1]).

5 Extensions

The analysis of this version of the TPRP is a starting point to efficiently solve more complex variations.

Fixed start time. In this problem variation the driver is not allowed to select the optimal start time of the route, but he is forced to start at time 0. This problem is just a special case of the previous one, with $a_0 = b_0 = 0$.

Rest periods with different duration. In this problem variation there are two different types of rest periods: short rest periods represent breaks during the day, while long rest periods represent overnight breaks. Thsi problem variation can be solved in the same way as the basic case: it is only required to replace δ with δ_k in the rest period elimination at pre-processing. Also the computation of λ and μ values must be slightly modified: instead of multiplying the number of rest periods by δ, one must sum up the duration of the rest period, whose sequence is known.

Multiple time windows. An important variation is when customers may have two or more associated time windows. This can be solved as the general case of the TPRP with possibly disjoint multiple time windows for each customer.

Time-dependent travel time. Another variation of the TPRP that is relevant in practice and remains to be addressed as a further development is that with time-dependent travel times. This allows to model different traffic conditions, intuitively making it profitable to place rest periods in correspondence of peak hours.

Acknowledgements The authors acknowledge their fruitful collaboration with Giulia Novello, who developed a dynamic programming algorithm for the TPRP, and WorkWaveFleet that provided the problem description and several real or realistic instances.

References

1. Novello, G.: Mathematical optimization algorithms for the insertion of breaks in truck driver duties, Master thesis, University of Milan (2022).
2. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Timing problems and algorithms: Time decisions for sequences of activities, Networks, 65 (2) 102-128 (2015).