European Journal of Operational Research 301 (2022) 445–457

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Scheduling activities with time-dependent durations and resource consumptions

Steffen Pottel, Asvin Goel*

Kühne Logistics University, Hamburg 20457, Germany

ARTICLE INFO

Article history:
Received 14 May 2021
Accepted 18 November 2021
Available online 24 November 2021

Keywords:
Scheduling
Time-dependency
Resource constraints
Break scheduling
Electric vehicle routing

ABSTRACT

In this paper we study the problem of scheduling a given sequence of activities where each activity consumes a resource with limited availability. Activity durations as well as resource consumptions are assumed to be time-dependent. Because of the interaction of time-dependent activity durations and resource consumptions, scheduling policies based on starting each activity in the sequence as early as possible may fail due to unnecessarily high resource consumptions exceeding the limited availability of the resource. We propose a dynamic discretization discovery algorithm that generates a partially time-expanded network during the search. We propose an acceleration technique allowing to significantly reduce the computational effort if the approach is embedded in an iterative solution procedure that frequently evaluates activity sequences which start with the same activities. Furthermore, we extend our approach to the case where resources can be replenished between subsequent activities. We evaluate our approach as a route evaluation method for the case of routing a fleet of electric vehicles in which travel durations and the energy consumed when travelling from one location to another depend on the time of the day.

1. Introduction

Many operational activities consume some kind of resource throughout their execution. For example, engines used for conducting activities consume energy. With the increasing replacement of internal combustion engines by battery electric propulsion systems, the limited battery capacity has to be considered when scheduling activities and it must be ensured that all activities can be executed without running out of energy (see, e.g., Pelletier, Jabali, & Laporte, 2016). Similarly, the time required by a human operator can be seen as a resource that is consumed throughout the execution of activities and which is limited by work regulations (see, e.g., European Union, 2003; Federal Motor Carrier Safety Administration, 2011)

In this paper we study the problem of scheduling a given sequence of activities where activities consume a limited resource throughout execution. We assume that the start time of each activity is constrained by time windows and that both activity durations and resource consumptions depend on the start time of the activity. This kind of scheduling problem, sometimes also referred to as timing problem, is relevant for the generation of schedules for manually constructed activity sequences as well as for the evaluation of activity sequences generated within optimisation approaches for combined scheduling, sequencing, and allocation of activities to operators.

A major cause of time-dependent activity durations is congestion, which can have a significant impact on the duration required to complete certain activities. In the 15 largest urban areas in the United States, for example, average travel times on freeways are 44 percent higher during peak periods than during free flow times (Schrank, Eisele, & Lomax, 2019). Similarly, in warehouses and manufacturing facilities, congestion can occur due to space limitations and narrow aisles (Zhang, Batta, & Nagi, 2009). Thus the time required for picking activities or moving a forklift may vary over the course of the day.

If electric vehicles are used, the energy stored in the battery is the limited resource. The energy consumption of electric vehicles usually depends on the driving speed which in turn depends on the traffic conditions at the particular time of the day. In other cases, the operator time, e.g. the working time of a truck driver, is the limited resource consumed during the execution of an activity. In such cases the resource consumption is directly related to the activity durations, and thus, also time-dependent.

Time-dependent activity durations and resource consumptions can have various ramifications that have received very little

* Corresponding author.
E-mail address: goel@telematique.eu (A. Goel).

https://doi.org/10.1016/j.ejor.2021.11.032

attention in the scientific literature. Consider, for instance, the case of electric forklifts or vehicles. If activity durations and energy consumptions are time-dependent, the timing of the activities can be of particular importance in order to ensure that the total energy consumption of a sequence of activities stays below the total energy provided by the battery. Similarly, in the case of limited working times, the total working time of an operator may exceed the maximum working time allowed if most activities are conducted during peak-hours. Thus, activities may have to be shifted away from peak-hours in order not to exceed the legal limits on the working time and widely used scheduling policies to schedule activities as early as possible may fail due to unnecessarily high resource consumptions.

In some cases, resources can be regenerated after they have been consumed. For example, an electric battery can be recharged or a human operator can take a rest as required by working regulations. In order to accommodate for such cases, we also study the problem of jointly scheduling activities and replenishments.

The contributions of this paper are: 1) we introduce the time-dependent activity scheduling problem with resource constraints; 2) we propose a dynamic discretization discovery algorithm on time-expanded networks for solving the problem with non-monotonic consumption functions; 3) we extend the problem formulation and solution approach to the case where resources can be replenished between activities; 4) we propose an acceleration technique for iterative problem solving by preloading vertices of the partially expanded network obtained from a previous iteration; 5) we propose new benchmark instances for a time-dependent electric vehicle routing problem as an application example for a combined scheduling, sequencing, and allocation problem with time-dependent activity durations and resource consumptions; 6) we evaluate our scheduling algorithms as route evaluation methods for solution approaches based on iteratively generating and modifying routes for these benchmark instances.

2. Related work

Resources that are consumed throughout the course of activities can be found in various scheduling problems. Often a capacity limit is given on such resources and the total consumption must not exceed the capacity. For example, in route planning problems with battery-powered vehicles (see, e.g., Pelletier et al., 2016), the energy consumption of the vehicles typically depends on the speed and distance travelled. Minimizing route distances helps creating routes that can be conducted without running out of energy. In addition, recharging the battery may be possible in order to increase the range of a vehicle. Schneider, Stenger, & Goeke (2014) study an *electric vehicle routing problem with time windows (EVRPTW)* in which electric vehicles can visit charging stations during the routes to recharge the battery. The energy consumption is assumed to be proportional to the distance travelled and the duration required to recharge the battery is assumed to be proportional to the amount of energy consumed before. In Montoya, Guéret, Mendoza, & Villegas (2017) an *electric vehicle routing problem with nonlinear charging functions (EVRP-NL)* is studied considering that the duration required to recharge a battery does not only depend on the amount of energy to be replenished, but also on the current energy level of the battery. Furthermore, Baum, Dibbelt, Gemsa, Wagner, & Zündorf (2019) find optimal routes for customer locations on road networks and a realistic distribution of charging stations as well as non-linear charging functions allowing partial recharging. The energy consumption for a trip from one location to another, however, is also assumed to be constant.

Another example of a scheduling problem in which the consumption of a limited resource must be considered can be found in the context of the *vehicle routing and truck driver scheduling problem (VRTDSP)* (see, e.g., Goel, 2009; 2018; Goel & Vidal, 2014; Prescott-Gagnon, Desaulniers, Drexl, & Rousseau, 2010). Here, vehicle routes have to be generated in such a way that all customers can be visited within time windows and all routes can be executed without violating government regulations with respect to driving time limits. According to these regulations, truck drivers have to take breaks and rest periods at regular intervals. The remaining time without a break or rest period can be regarded as a resource that is consumed when driving. Although the driving time between two customers is assumed to be a given constant, time windows for customer visits may make it impossible to always drive as long as allowed by the regulations and only take breaks or rests when the legal driving time limits are reached. Instead, determining feasibility of each route requires the solution of a *truck driver scheduling problem (TDSP)* (see, e.g., Archetti & Savelsbergh, 2009; Goel, 2010; 2014; Goel, Archetti, & Savelsbergh, 2012; Goel & Kok, 2012; Goel & Rousseau, 2012). Solving the TDSP involves the search for a schedule with minimal completion time. For a given start time, this objective is equivalent to minimizing schedule durations, however, if the start time can be freely chosen, minimizing schedule durations becomes substantially more difficult (see e.g. Goel, 2012a; 2012b; 2012c; Rancourt, Cordeau, & Laporte, 2013; Tilk & Goel, 2020).

Most of the above-mentioned studies, and those covered in the survey on vehicle routing problems with intermediate stops by Schiffer, Schneider, Walther, & Laporte (2019), assume that activity durations are given as constant values. In many real-life applications, however, the duration of activities depends on the time of the day. Vidal, Crainic, Gendreau, & Prins (2015) review scheduling problems for given activity sequences including time-dependent variants. An example of a combined scheduling and sequencing problem with time-dependent durations is the time-dependent traveling salesperson problem (see, e.g., Cordeau, Ghiani, & Guerriero, 2014; Montero, Méndez-Díaz, & Miranda-Bront, 2017; Nannicini, Delling, Schultes, & Liberti, 2012; Taş, Gendreau, Jabali, & Laporte, 2016). Vu, Hewitt, Boland, & Savelsbergh (2020a) and Vu, Hewitt, & Vu (2020b) apply dynamic discretization discovery, originally proposed by Boland, Hewitt, Marshall, & Savelsbergh (2017), for various variants of the time-dependent traveling salesperson problem. In their proposed dynamic discretization discovery approach, solutions are iteratively generated within a partially time-expanded network and then used to grow the network dynamically when needed. Based on an under-estimation of travel-times, criteria are provided allowing to prove optimality of a solution in the fully expanded network. He, Boland, Nemhauser, & Savelsbergh (2019) also apply dynamic discretization discovery, but for the time-dependent shortest path problem. For the case of piecewise linear travel times satisfying the *first-in-first-out (FIFO)* property, i.e. the property that an earlier start implies an earlier completion, they propose an improvement of the dynamic discretization discovery based on an observation in Foschini, Hershberger, & Suri (2012) allowing to explore only a small fraction of breakpoints of the arc travel time functions.

In the context of time-dependent vehicle routing problems, Hashimoto, Yagiura, & Ibaraki (2008) propose a method to determine an optimal schedule for a given route, where travel times and costs are piecewise linear and lower semi-continuous time-dependent functions. In a similar setting, Visser & Spliet (2020) show how ready time functions can be determined efficiently by composition of piecewise linear travel time functions. Further approaches on vehicle routing problems with time-dependent travel times are surveyed by Gendreau, Ghiani, & Guerriero (2015).

The research combining time-dependent activity durations and the consumption of limited resources is scarce. Kok, Hans, & Schutten (2011) present a mixed-integer program for the *time-dependent*

Table 1
Features related to the evaluation of a given activity sequence.

Paper	Problem	Objective	Time-dependency	Resource consumption		Replenishment duration	
				Proportional	Unrelated	Constant	Dependent
He et al. (2019)	TD-SPP	completion time or duration	✓				
Vu et al. (2020a)	TD-TSP	completion time or duration	✓				
Vu et al. (2020b)	TD-MTDP	duration	✓				
Hashimoto et al. (2008)	TD-VRPTW	costs	✓				
Visser & Spliet (2020)	TD-VRPTW	duration	✓				
Schneider et al. (2014)	EVRPTW	distance		✓	✓		✓
Desaulniers, Errico, Irnich, & Schneider (2016)	EVRPTW	cost		✓	✓		✓
Montoya et al. (2017)	EVRP-NL	travel and charging time		✓	✓		✓
Archetti & Savelsbergh (2009); Goel (2010, 2014); Goel et al. (2012); Goel & Kok (2012); Goel & Rousseau (2012)	TDSP	completion time		✓		✓	
Goel (2012a, 2012b, 2012c)	TDSP	duration		✓		✓	
Kok et al. (2011)	TD-TDSP	duration	✓	✓		✓	
Goel (2009, 2018); Goel & Vidal (2014); Prescott-Gagnon et al. (2010)	VRTDSP	completion time		✓		✓	
Goel (2018); Rancourt et al. (2013); Tilk & Goel (2020)	VRTDSP	duration		✓		✓	
Kleff (2019)	TD-TDRP	completion time	✓	✓		✓	
Kleff (2019)	TD-TDSRP	completion time	✓	✓		✓	
This study	TDASP	completion time	✓	✓	✓		
This study	TDASPR	completion time	✓	✓	✓	✓	✓

truck driver scheduling problem (TD-TDSP), i.e. the problem of finding a schedule of minimal duration for a truck driver who must take breaks as required by hours of service regulations considering that travel times are time-dependent. The problem is modelled using piecewise linear, time-dependent travel times and solved with a general purpose mixed-integer programming solver. Kleff (2019) studies a *time-dependent truck driver routing problem (TD-TDRP)*, i.e. the problem of finding a time-dependent shortest path from an origin to a destination in which breaks can be taken at parking lots in the network. The driving time is limited by European Union regulations. An approach combining forward exploration from the origin to a parking lot and backwards exploration from the destination to a parking lot is presented. Furthermore, Kleff (2019) presents an approach for the *time-dependent truck driver scheduling and routing problem (TD-TDSRP)* in which a route through a road network has to be found visiting a sequence of locations while not exceeding the cumulative driving time without a break. In order to replenish the allowed driving time, breaks with a fix duration can be taken at parking lots along the route. In these works, the resource consumed is the driving time and thus, identical to the duration of the driving activities. This limitation prevents those approaches to be applied to, for instance, electric vehicle routing problems in which driving times and energy consumptions may not be proportional to each other. In such cases, it may be beneficial to avoid areas prone to congestion during peak hours. While avoiding congested areas may reduce travel times, it typically increases the distance and travel speed and, thus, the energy consumed.

Table 1 provides an overview over the related work on scheduling problems including features related to time-dependency, the consumption of a limited resource, and the replenishment of that resource. For those works in which activities consume a limited resource, the table distinguishes between works in which it is assumed that resource consumptions are proportional or unrelated to activity durations, and between works in which the resource can be fully replenished with a constant or a consumption-dependent duration. The *time-dependent activity scheduling problem (TDASP)* proposed in the next section, combines time-dependency with resource consumptions that may be unrelated to activity durations, whereas the *time-dependent activity scheduling problem with replenishments (TDASPR)* presented in Section 4 extends the TDASP by replenishments with consumption-dependent durations. Thus, the TDASPR considers all of the features of Table 1. The TDASP and TDASPR focus on the scheduling of a given activity sequence and can be used as route evaluation procedures within time-dependent variants of the electric vehicle routing problem with or without recharging, the vehicle routing and truck driver scheduling problem, and others.

3. Problem description

Let us consider a sequence of activities $(1, 2, \ldots, n)$ to be conducted in the given order and without preemption. The time-dependent duration and resource consumption of each activity i are given by functions $\tau_i(t_i)$ and $\rho_i(t_i)$, respectively, where t_i refers to the start time of the activity. Each activity i must start within a given time interval $[e_i, l_i]$ and the cumulative resource consumption must not exceed a given quantity Q. Defining the completion time functions by $\theta_i(t_i) = t_i + \tau_i(t_i)$ for all $i \in \{1, \ldots, n\}$, the *time-dependent activity scheduling problem (TDASP)* can be represented by minimize

$$\theta_n(t_n) \tag{1}$$

subject to

$$\theta_i(t_i) \leq t_{i+1} \text{ for all } 1 \leq i < n \tag{2}$$

$$\sum_{i=1}^{n} \rho_i(t_i) \leq Q \tag{3}$$

$$t_i \in [e_i, l_i] \text{ for all } 1 \leq i \leq n \tag{4}$$

The objective (1) is to minimize the completion time. Constraint (2) demands that the start time of any activity must not be before the completion time of the previous activity. Constraint (3) requires that the cumulative resource consumption does not exceed the available amount. Lastly, the domain of the variables is given by (4).

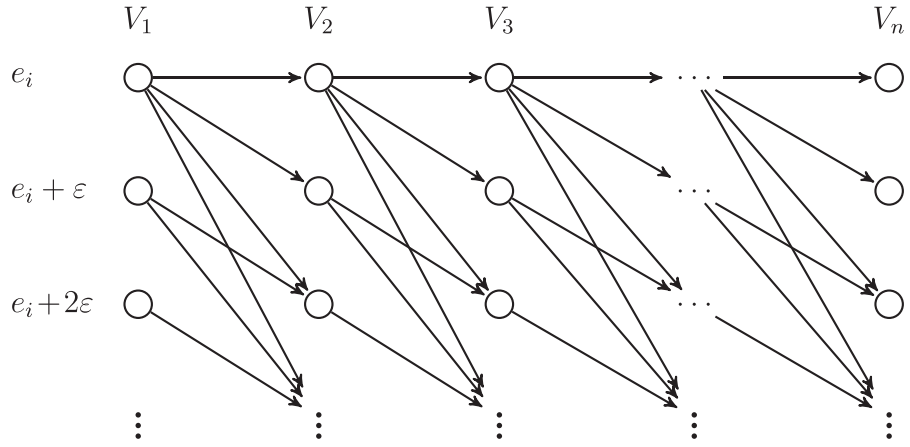


Fig. 1. A time-expanded network.

Throughout this paper we assume that the completion time functions $\theta_i(\cdot)$ are non-decreasing for all $i \in \{1, \dots, n\}$, meaning that completion times satisfy the FIFO property, i.e. $\theta_i(t_i) \leq \theta_i(t_i')$ holds if $t_i < t_i'$. Without loss of generality we assume in the remainder of this paper that time windows are restricted in such a way that $\theta_i(e_i) \leq e_{i+1}$ and $\theta_i(l_i) \leq l_{i+1}$ for all $1 \leq i < n$.

If resource consumption functions are non-decreasing, as for example, in scheduling problems with deterioration (see, e.g., Mosheiov, 1994), an optimal activity schedule can be easily found by starting each activity as early as possible. In general, however, the resource consumption is non-monotonic and, e.g., depends on congestion levels which vary throughout the day. In such cases it may be possible to reduce the total resource consumption by delaying the start time of some of the activities. In the following sections we present solution approaches for cases in which resource consumption functions are not monotonic.

3.1. Time-expanded networks

An approximation of the solution of the TDASP can be obtained by discretizing time and modelling the problem as a time-expanded network. Let us assume we are given a parameter $\varepsilon > 0$ for which e_i and l_i are a multiple of ε for each activity i. Then, we can create for each activity i and each time point $t_i \in \{e_i, e_i + \varepsilon, \dots, l_i\}$ a vertex (i, t_i) in the time-expanded network and subsume all created vertices of activity i in a set $V_i = \{(i, e_i), (i, e_i + \varepsilon), \dots, (i, l_i)\}$. In addition, we introduce the auxiliary variable $q_{(i,t)}$ for each vertex (i, t), which will be used to estimate the resource consumption of activity i about time t. The edges of the time-expanded network are obtained by connecting every vertex (i, t_i) with each vertex $(i+1, t_{i+1})$ if and only if $t_{i+1} \geq \theta_i(t_i)$. Fig. 1 illustrates such a time-expanded network with vertex sets V_i for each $1 \leq i \leq n$.

For the fully time-expanded network of this section, let us set $V = \bigcup_{i=1}^{n} V_i$ and $q_{(i,t)} = \rho_i(t)$ for all $(i, t) \in V$. Then we can solve the TDASP by calling the function $\texttt{solve}\big(V, \big(q_{(i,t)}\big)_{(i,t)\in V}\big)$ shown in Algorithm 1 . Note that we could have simply used the values $\rho_i(t)$ and $\bigcup_{i=1}^{n} V_i$ instead of $q_{(i,t)}$ and V in the algorithm, respectively. However, the used notation allows us in the following sections to use different values of $q_{(i,t)}$ and V to accelerate the search.

The algorithm uses labels representing the cumulative resource consumption prior to the start of an activity at a particular time. The algorithm starts with initializing a set of vertices $\bar{V}$ for which the label is permanently set. It tentatively sets all labels associated with the first activity to zero, and all other labels to infinity. As long as there is a vertex in $V \setminus \bar{V}$ which has a label allowing to conduct the activity without exceeding the capacity, the algorithm

Algorithm 1: Solving the TDASP in a time-expanded network.

Data: $(e_i, l_i, \tau_i, \theta_i, \rho_i)_{i\in\{1,\dots,n\}}, Q$
Function $\texttt{TEN}::\texttt{solve}\big(V, \big(q_{(i,t)}\big)_{(i,t)\in V}\big)$
  $\bar{V} \leftarrow \emptyset$
  foreach $(i, t) \in V$ **do**
    $\ell_{(i,t)} \leftarrow \begin{cases} 0 & \text{if } i = 1 \\ \infty & \text{else} \end{cases}$
  end
  while *exists* $(i, t) \in V \setminus \bar{V}$ *with* $\ell_{(i,t)} + q_{(i,t)} \leq Q$ **do**
    $(i, t) \leftarrow \operatorname{argmin}^{\text{lex}}_{(i,t)\in V\setminus\bar{V}:\ell_{(i,t)}+q_{(i,t)}\leq Q} \Big\{\big(\theta_i(t) + \sum_{j=i+1}^{n} \min_{t'\in[e_j,l_j]} \tau_j(t'),\ i, t\big)\Big\}$
    $\bar{V} \leftarrow \bar{V} \cup \{(i, t)\}$
    if $i = n$ **then break**
    foreach $t' \in \{t' \mid (i+1, t') \in V \setminus \bar{V}, \theta_i(t) \leq t'\}$ **do**
      if $\ell_{(i,t)} + q_{(i,t)} < \ell_{(i+1,t')}$ **then**
        $\ell_{(i+1,t')} \leftarrow \ell_{(i,t)} + q_{(i,t)}$
        $p_{(i+1,t')} \leftarrow (i, t)$
      end
    end
  end
  return $\texttt{solution}\big(\bar{V}, (q_v, \ell_v, p_v)_{v\in\bar{V}}\big)$
end

selects such a vertex $(i, t) \in V \setminus \bar{V}$ according to a lexicographical ordering on a lower bound on the completion time of the last activity, the activity index, and the time associated with the vertex. The lower bound on the completion time of the last activity is determined using an under-estimating heuristic having an effect similar to the label selection in an A*-Algorithm for shortest path problems (Hart, Nilsson, & Raphael, 1968).

The lexicographical ordering is necessary to ensure that all relevant predecessors are considered before selecting a vertex. The selected vertex is inserted in $\bar{V}$. If the selected vertex belongs to the last activity, the algorithm terminates. Otherwise, non-permanent labels of the next activity are updated if their value can be reduced. Whenever a label is updated, the predecessor p_v for the vertex is updated so that the solution can be reconstructed by a function $\texttt{solution}\big(\bar{V}, (q_v, \ell_v, p_v)_{v\in\bar{V}}\big)$. If a solution is found, this function returns the vertices of the solution obtained by selecting the earliest vertex (n, t) with $\ell_{(n,t)} + q_{(n,t)} \leq Q$ and iterating

through its predecessors. If no solution is found, the function returns $\{(1,\infty),\ldots,(n,\infty)\}$. For brevity reasons we omit a detailed description of this trivial function.

Lemma 1. *Algorithm 1 finds an optimal solution of the TDASP restricted to the time-expanded network with vertex set V if and only if such a solution exists.*

Proof. Assume that vertex (i,t) is selected in an iteration of the algorithm. Due to the FIFO property, each potential predecessor of (i,t) must have been selected in an earlier iteration. Thus, $\ell_{(i,t)}$ represents the minimal cumulative resource consumption before conducting activity i at time t. If (n,t) is selected and $\ell_{(n,t)}+q_{(n,t)}\leq Q$, then t is the earliest time for which $\ell_{(n,t)}+q_{(n,t)}\leq Q$. Thus, the solution obtained by backtracking the predecessors is optimal. If the algorithm terminates without having selected a vertex (n,t) with $\ell_{(n,t)}+q_{(n,t)}\leq Q$, then there is no path $(1,t_1),\ldots,(n,t_n)$ with $(i,t_i)\in V$ for $i\in\{1,\ldots,n\}$, $\theta_i(t_i)\leq t_{i+1}$ for $i\in\{1,\ldots,n-1\}$, and $\sum_{i=1}^{n} q_{(i,t_i)}\leq Q$. □

With n activities and an upper bound $m=\max_{i\in\{1,\ldots,n\}}\left\{\frac{l_i-e_i}{\varepsilon}\right\}$ on the number of vertices per activity, Algorithm 1 requires at most nm iterations. Assuming that all vertices in V are stored in a sorted list, the selection of vertex (i,t) in Step 7 can be conducted in constant time. In each iteration the labels of at most m vertices are updated. Thus, the computational effort of Algorithm 1 is bounded by $O(nm^2)$.

3.2. Dynamic discretization discovery

The difficulty in solving the TDASP with discretized times is that the number of vertices in the time-expanded network can become very large for small values of ε, i.e. the parameter influencing the granularity of the time discretization. In order to reduce the number of vertices we now show how to solve the TDASP with discretized times on a partially expanded network similar to the dynamic discretization discovery approach by Vu et al. (2020a) for the time-dependent traveling salesman problem with time windows. However, as we do not assume that resource consumption functions are monotonic, our dynamic discretization discovery approach cannot be based on the properties exploited by Vu et al. (2020a).

In the remainder of this section we will refer to a partially expanded network if the set of vertices V is a subset of the vertices of the fully expanded network and the set of edges is obtained by connecting every vertex (i,t_i) with each vertex $(i+1,t_{i+1})$ if and only if $t_{i+1}\geq\theta_i(t_i)$. Note that in contrast to Vu et al. (2020a), we do not under-estimate activity durations. Thus, we guarantee temporal validity of all paths through our partially expanded networks when resource consumptions are not considered. Our approach will generate and modify such partially expanded networks with the following properties.

Property 1. *For each $1\leq i\leq n$ we have $(i,e_i)\in V$ and $(i,l_i)\in V$.*

Property 2. *For each vertex $(i,t)\in V$ with $e_{i+1}<\varepsilon\left\lceil\frac{\theta_i(t)}{\varepsilon}\right\rceil<l_{i+1}$, we have $\left(i+1,\varepsilon\left\lceil\frac{\theta_i(t)}{\varepsilon}\right\rceil\right)\in V$.*

Property 3. *Each vertex $(i,t)\in V$ with $t<l_i$ is assigned a consumption value of*

$$q_{(i,t)}=\min_{\bar{t}\in\{t,t+\varepsilon,\ldots,t'-\varepsilon\}}\rho_i(\bar{t})$$

where $t'>t$ is the smallest value for which $(i,t')\in V$. Each vertex $(i,l_i)\in V$ is assigned a consumption value of $q_{(i,l_i)}=\rho_i(l_i)$.

Property 1 guarantees that for each activity i, a vertex representing the earliest and latest start time is included in the partially expanded network. For every vertex, its immediate successor is included in the partially expanded network if Property 2 is satisfied. Because of Property 3, each vertex (i,t) is assigned a value $q_{(i,t)}$, which is a lower bound on the actual resource consumption of the activity for any start time between t and $t'-\varepsilon$, where t' refers to the next vertex of activity i.

With these properties we can make the following observation.

Lemma 2. *The optimal solution of a TDASP restricted to a partially expanded network satisfying Properties 1 to 3 is a lower bound on the optimal solution of the fully expanded TDASP with discretized times.*

Proof. Assume that vertices $(1,t_1),\ldots,(n,t_n)$ correspond to an optimal solution of the fully expanded TDASP with discretized times. Because of Properties 1 and 2, we can find a path $(1,t'_1),\ldots,(n,t'_n)$ in the partially expanded network, where for each $1\leq i\leq n$ the value t'_i is the largest while not exceeding t_i such that (i,t'_i) is in the partially expanded network. Because of Property 3, we have $q_{(i,t'_i)}\leq\rho_i(t_i)$ for each $1\leq i\leq n$ and $\sum_{i=1}^{n} q_{(i,t'_i)}\leq\sum_{i=1}^{n}\rho_i(t_i)\leq Q$. Therefore, $(1,t'_1),\ldots,(n,t'_n)$ is feasible for the TDASP restricted to the partially expanded network and has a completion time that is smaller or equal to the solution of the fully expanded TDASP with discretized times. □

Lemma 3. *Assume that the path $(1,t_1),\ldots,(n,t_n)$ corresponds to an optimal solution of the TDASP restricted to a partially expanded network satisfying Properties 1 to 3. Furthermore, assume that $q_{(i,t_i)}=\rho_i(t_i)$ for each $1\leq i\leq n$. Then, the path corresponds to an optimal solution of the fully expanded TDASP with discretized times.*

Proof. If $q_{(i,t_i)}=\rho_i(t_i)$ for each $1\leq i\leq n$, then $\sum_{i=1}^{n}\rho_i(t_i)=\sum_{i=1}^{n} q_{(i,t_i)}\leq Q$. Therefore, $(1,t_1),\ldots,(n,t_n)$ is feasible for the TDASP with discretized times. Because of Lemma 2 we can conclude that the path corresponds to an optimal solution of the TDASP with discretized times. □

We can use Algorithm 2 to create an initial partially expanded network. According to Property 1, the algorithm adds vertices corresponding to each activity at the start and end of the time window. These vertices are added in the order $(n,l_n),(n,e_n),\ldots,(1,l_1),(1,e_1)$ and their immediate successors are recursively added according to Property 2. When adding a vertex (i,t), the consumption values of the new vertex and the preceding vertex (i,t') are set as required by Property 3.

For adding a new vertex (i,t) to V and maintaining a sorted list of all vertices, as assumed in the complexity analysis for Algorithm 1, at most $O(nm)$ time is required. To determine $q_{(i,t)}$, the function `DDD::addRecursive` has to retrieve the time-dependent consumption value of ρ_i at most $\frac{l_i-e_i}{\varepsilon}\leq m$ times. If k denotes an upper bound on the number of operations required to retrieve a value of ρ_i, the computational effort required to determine $q_{(i,t)}$ is $O(mk)$. As `DDD::addRecursive` recurses at most n times, the running time of `DDD::addRecursive` and `DDD::initialize` are both bounded by $O(n^2m+nmk)$ when maintaining a sorted list of vertices.

Algorithm 3 solves the TDASP with discretized times by dynamically creating a time-expanded network. The algorithm starts by initializing the network using Algorithm 2. Then it iterates until a solution of the TDASP with discretized times is found. Each iteration begins with finding a solution of the TDASP restricted to the partially expanded network using Algorithm 1. If there is an activity i in the solution that begins at a time t_i such that $\rho_i(t_i)>q_{(i,t_i)}$, then the total consumption associated with the solution may exceed the capacity. Therefore, a new vertex is added to the partially expanded network V between (i,t_i) and the next vertex of activ-

Algorithm 2: Initializing a partial time-expanded network.

Data: $(e_i, l_i, \tau_i, \theta_i, \rho_i)_{i\in\{1,\dots,n\}}, Q, \varepsilon$
Function DDD::initialize()
  $V \leftarrow \emptyset$
  for $i \leftarrow n$ **to** 1 **do**
    $\big(V, (q_v)_{v\in V}\big) \leftarrow$ DDD::addRecursive$\big(V, (q_v)_{v\in V}, (i, l_i)\big)$
    $\big(V, (q_v)_{v\in V}\big) \leftarrow$ DDD::addRecursive$\big(V, (q_v)_{v\in V}, (i, e_i)\big)$
  end
  return $\big(V, (q_v)_{v\in V}\big)$
end
Function DDD::addRecursive$\big(V, (q_v)_{v\in V}, (i,t)\big)$
  if $(i,t) \in V$ **then return** $\big(V, (q_v)_{v\in V}\big)$
  $V \leftarrow V \cup \{(i,t)\}$
  if $t = l_i$ **then**
    $q_{(i,t)} \leftarrow \rho_i(t)$
  else
    $t' \leftarrow \min\{t' \mid (i,t') \in V, t' > t\}$
    $q_{(i,t)} \leftarrow \min_{\bar{t}\in\{t, t+\varepsilon, \dots, t'-\varepsilon\}} \rho_i(\bar{t})$
    if $t > e_i$ **then**
      $t' \leftarrow \max\{t' \mid (i,t') \in V, t' < t\}$
      $q_{(i,t')} \leftarrow \min_{\bar{t}\in\{t', t'+\varepsilon, \dots, t-\varepsilon\}} \rho_i(\bar{t})$
    end
  end
  if $i = n$ **then**
    return $\big(V, (q_v)_{v\in V}\big)$
  else
    return DDD::addRecursive$\Big(V, (q_v)_{v\in V}, \big(i+1, \max\{e_{i+1}, \varepsilon\lceil\frac{\theta_i(t)}{\varepsilon}\rceil\}\big)\Big)$
  end
end

ity i after time t_i. The algorithm starts with time t representing the time of the vertex following (i, t_i) and iteratively reduces t until $\min_{\bar{t}\in\{t_i, t_i+\varepsilon, \dots, t-\varepsilon\}} \rho_i(\bar{t}) > q_{(i,t_i)}$. Then, the function DDD::addRecursive is used to add the new vertex and increase the value of $q_{(i,t_i)}$ to $\min_{\bar{t}\in\{t_i, t_i+\varepsilon, \dots, t-\varepsilon\}} \rho_i(\bar{t})$.

If a feasible solution exists, the algorithm either terminates with the optimal solution, or adds at least one vertex to the partially expanded network in each iteration. Therefore, the algorithm is guaranteed to terminate with the optimal solution. In the worst case, the algorithm may have to fully expand the network. However, in general, the algorithm can be expected to require only a fraction of the vertices.

In each iteration of Algorithm 3 (except the last) at least one vertex is added to the time-expanded network. Thus, the number of iterations of Algorithm 3 is at most nm. In each iteration, Algorithm 1 is called to solve the subproblem in at most $O(nm^2)$ time. The effort for checking the condition in Step 11 of Algorithm 3 is bounded by $O(mk)$, where k denotes an upper bound on the number of operations required to retrieve a value of ρ_i. With appropriate data structures, we can reduce this effort to constant time, but building up the data structure requires $O(mk)$ operations. As the condition is checked at most $\log(m)$ times, the effort for determining t is dominated by $O(mk)$. Concludingly, at most $O(nm^2)$ time is required for solving the subproblem, at most $O(mk)$ time is required for determining t, and at most $O(n^2m + nmk)$ is required for adding the new vertex in each iteration of Algorithm 3. Thus, the total effort is bounded by $O(n^2m^3 + n^3m^2 + n^2m^2k)$.

Algorithm 3: Dynamic discretization discovery algorithm.

Data: $(e_i, l_i, \tau_i, \theta_i, \rho_i)_{i\in\{1,\dots,n\}}, Q, \varepsilon$
Function DDD::solve()
  $\big(V, (q_v)_{v\in V}\big) \leftarrow$ DDD::initialize()
  repeat
    $\big\{(1,t_1), \dots, (n,t_n)\big\} =$ TEN::solve$\big(V, (q_v)_{v\in V}\big)$
    if $t_n = \infty$ **then break**
    if *exists* $1 \le i \le n$ *with* $(i,t_i) \in V$ *and* $\rho_i(t_i) > q_{(i,t_i)}$ **then**
      select $(i,t_i) \in V$ with $\rho_i(t_i) > q_{(i,t_i)}$
      $t \leftarrow \min\{t' \mid t' > t_i, (i,t') \in V\}$
      repeat
        $t \leftarrow \varepsilon\lceil\frac{t_i+t}{2\varepsilon}\rceil$
      until $\min_{\bar{t}\in\{t_i, t_i+\varepsilon, \dots, t-\varepsilon\}} \rho_i(\bar{t}) > q_{(i,t_i)}$
      $\big(V, (q_v)_{v\in V}\big) \leftarrow$ DDD::addRecursive$\big(V, (q_v)_{v\in V}, (i,t)\big)$
    end
  until $\rho_i(t_i) = q_{(i,t_i)} \forall 1 \le i \le n$
  return $\big\{(1,t_1), \dots, (n,t_n)\big\}$
end

4. Replenishments

So far we assumed that the resource level is gradually reduced through the consumption of the activities. In many cases, however the resource levels can be increased again through replenishments. For example, electric batteries can be recharged or workers are allowed to resume after taking a sufficiently long break. In this section we consider an extension of the TDASP in which the resource can be replenished between the end of an activity and the beginning of the next activity. We assume that the time required to fully replenish a resource after conducting activity i can be determined by a consumption-dependent and non-decreasing replenishment function $\Delta_i(Q_i)$, where Q_i is the cumulative resource consumption after completing activity i.

The resulting *time-dependent activity scheduling problem with replenishments (TDASPR)* is to minimize

$$\theta_n(t_n) \tag{5}$$

subject to

$$\theta_i(t_i) + y_i\Delta_i(Q_i) \le t_{i+1} \text{ for all } 1 \le i < n \tag{6}$$

$$Q_1 = \rho_1(t_1) \tag{7}$$

$$Q_{i+1} = (1-y_i)Q_i + \rho_{i+1}(t_{i+1}) \text{ for all } 1 \le i < n \tag{8}$$

$$t_i \in [e_i, l_i], Q_i \in [0, Q], y_i \in \{0,1\} \text{ for all } 1 \le i \le n. \tag{9}$$

In this problem Q_i is a variable representing the cumulative resource consumption between the last replenishment and completion of activity i, and y_i is a binary variable representing whether the resource is replenished after conducting activity i. The objective function is the same as in the case without replenishments. Constraint (6) is analogous to Constraint (2) but includes the replenishment duration if necessary. Constraints (7) and (8) ensure that the cumulative resource consumption is correctly determined and Constraint (9) restricts the domains of the variables.

For the case of constant replenishment durations, a mixed integer programming formulation is provided in Appendix A. The special case where replenishments must or must not be taken after some of the activities is covered in Appendix B.

4.1. Replenishments in time-expanded networks

In the following we show how dynamic discretization discovery can be used to solve the TDASPR. For this, we replace Algorithm 1 by Algorithm 4 which dynamically adds new vertices

Algorithm 4: Solving the TDASPR in a partially expanded network.

Data: $(e_i, l_i, \tau_i, \theta_i, \rho_i, \Delta_i)_{i\in\{1,\dots,n\}}, Q, \varepsilon$
1 **Function** TEN-Δ::solve$\left(V, \left(q_{(i,t)}\right)_{(i,t)\in V}\right)$
2 $\quad\bar{V} \leftarrow \emptyset$
3 $\quad$**foreach** $(i,t) \in V$ **do**
4 $\quad\quad \ell_{(i,t)} \leftarrow \begin{cases} 0 & \text{if } i = 1 \\ \infty & \text{else} \end{cases}$
5 $\quad$**end**
6 $\quad$**while** *exists* $(i,t) \in V \setminus \bar{V}$ *with* $\ell_{(i,t)} + q_{(i,t)} \le Q$ **do**
7 $\quad\quad (i,t) \leftarrow \operatorname{argmin}^{\text{lex}}_{(i,t)\in V\setminus\bar{V}:\ell_{(i,t)}+q_{(i,t)}\le Q} \left\{\left(\theta_i(t) + \sum_{j=i+1}^{n} \min_{t'\in[e_j,l_j]} \tau_j(t'),\ i,t\right)\right\}$
8 $\quad\quad \bar{V} \leftarrow \bar{V} \cup \{(i,t)\}$
9 $\quad\quad$**if** $i = n$ **then break**
10 $\quad\quad t^* \leftarrow \varepsilon\left\lceil \frac{\theta_i(t)+\Delta_i(\ell_{(i,t)}+q_{(i,t)})}{\varepsilon} \right\rceil$
11 $\quad\quad$**if** $t^* \le l_{i+1}$ **then**
12 $\quad\quad\quad$ DDD::addRecursive$\left(V, (q_v)_{v\in V}, (i+1, t^*)\right)$
13 $\quad\quad$**end**
14 $\quad\quad$**foreach** $t' \in \{t' \mid (i+1,t') \in V \setminus \bar{V}, \theta_i(t) \le t' < t^*\}$ **do**
15 $\quad\quad\quad$**if** $\ell_{(i,t)} + q_{(i,t)} < \ell_{(i+1,t')}$ **then**
16 $\quad\quad\quad\quad \ell_{(i+1,t')} \leftarrow \ell_{(i,t)} + q_{(i,t)}$
17 $\quad\quad\quad\quad p_{(i+1,t')} \leftarrow (i,t)$
18 $\quad\quad\quad$**end**
19 $\quad\quad$**end**
20 $\quad\quad$**foreach** $t' \in \{t' \mid (i+1,t') \in V \setminus \bar{V}, t' \ge t^*\}$ **do**
21 $\quad\quad\quad \ell_{(i+1,t')} \leftarrow 0$
22 $\quad\quad\quad p_{(i+1,t')} \leftarrow (i,t)$
23 $\quad\quad$**end**
24 $\quad$**end**
25 $\quad$**return** solution$\left(\bar{V}, (q_v, \ell_v, p_v)_{v\in\bar{V}}\right)$
26 **end**

corresponding to a fully replenished resource to the partially expanded network.

Whenever a vertex (i,t) with $i < n$ is selected in Algorithm 4, the algorithm determines the earliest replenishment time t^* after conducting activity i at time t. If $t^* \le l_{i+1}$, the vertex $(i+1, t^*)$ is added to V using algorithm DDD::addRecursive. Then, all labels for vertices with a time prior to t^* are updated if their value can be reduced and all labels for vertices with a later time are set to zero.

Lemma 4. *Assume that Algorithm 4 is applied to a partially expanded network satisfying Properties 1 to 3. Then, the solution of the algorithm is a lower bound on the solution in the fully expanded network.*

Proof. Let us first note that the solution returned by Algorithm 4 is the same as the solution returned by Algorithm 4 without Line 9. For the ease of argumentation, we therefore base our proof on a variant of Algorithm 4 without Line 9. Let $(1,t_1), \dots, (n,t_n)$ denote an optimal solution in the fully expanded network. Let $\bar{V}$ denote the set of permanently labelled vertices after execution of this variant of the algorithm, including the vertices that may have been added to V during the course of the algorithm. Let $(1,\bar{t}_1), \dots, (n,\bar{t}_n) \in \bar{V}$ denote the vertices returned by the algorithm. Analogously to Lemma 1 we can show that $\bar{t}_n$ is smaller or equal to the completion time of any other path through vertices in $\bar{V}$ not exceeding the capacity between replenishments.

Let $(1,t'_1), \dots, (n,t'_n)$ denote the vertices in $\bar{V}$ for which t'_i is the largest time not exceeding t_i for each $i \in \{1,\dots,n\}$. These vertices can be found because of Property 1. As θ_i is non-decreasing for each $i \in \{1,\dots,n\}$, Property 2 guarantees that $\theta_i(t'_i) \le t'_{i+1}$ for $i \in \{1,\dots,n-1\}$.

Assume that, in the optimal solution $(1,t_1), \dots, (n,t_n)$, the first replenishment is conducted just before starting activity j. Because of Property 3, we have $\sum_{i=1}^{j-1} q_{(i,t'_i)} \le \sum_{i=1}^{j-1} \rho_i(t_i) \le Q$. Because Δ_{j-1} is non-decreasing, we have $\theta_{j-1}(t'_{j-1}) + \Delta_{j-1}\left(\sum_{i=1}^{j-1} q_{(i,t'_i)}\right) \le \theta_{j-1}(t_{j-1}) + \Delta_{j-1}\left(\sum_{i=1}^{j-1} \rho_i(t_i)\right)$. Therefore, there is a vertex $(j,t) \in \bar{V}$ with $t \le t_j$ and $\ell_{(j,t)} = 0$. By definition we have $t \le t'_j \le t_j$ and we can conclude that $\ell_{(j,t'_j)} = 0$. Analogously we can show that $\sum_{i=j}^{k-1} q_{(i,t'_i)} \le \sum_{i=j}^{k-1} \rho_i(t_i) \le Q$ and $\ell_{(k,t'_k)} = 0$ if the next replenishment in the solution $(1,t_1), \dots, (n,t_n)$ is conducted just before starting activity k. We can apply this argument repeatedly to conclude that the capacity is not exceeded along the path $(1,t'_1), \dots, (n,t'_n)$ and that $\ell_{(n,t'_n)} + q_{(n,t'_n)} \le Q$. Thus, $\bar{t}_n \le t'_n \le t_n$. □

Lemma 5. *Assume that the path $(1,t_1), \dots, (n,t_n)$ corresponds to a solution created by Algorithm 3 using Algorithm 4. Furthermore, assume that $q_{(i,t_i)} = \rho_i(t_i)$ for each $1 \le i \le n$. Then, the path corresponds to an optimal solution of the fully expanded TDASPR with discretized times.*

Proof. For each $1 \le i \le n$ let $\ell_{(i,t_i)}$ denote the corresponding label created by the algorithm. Let i be any activity index with $\ell_{(i,t_i)} = 0$. For any activity index $j > i$ such that no replenishment is conducted between i and j, we have $Q \ge \ell_{(j,t_j)} + q_{(j,t_j)} = \sum_{k=i}^{j} q_{(k,t_k)} = \sum_{k=i}^{j} \rho_k(t_k)$. Therefore, the duration of any replenishment is sufficiently long and the capacity is never exceeded between replenishments. Thus, $(1,t_1), \dots, (n,t_n)$ is a feasible path in the fully expanded network. Because of Lemma 4 we can conclude that the path corresponds to an optimal solution of the TDASPR with discretized times. □

5. Preloading of vertices

If the TDASP or TDASPR is used for the evaluation of activity sequences obtained by a local-search based method for combined scheduling and sequencing problems, it is likely that many similar sequences have to be evaluated. We can avoid redundant calculations by using the results of previous sequence evaluations using long-term global memory structures (see e.g. Goel & Vidal, 2014) or the incumbent solution. Assume we have already solved the problem for a sequence of activities $(1,\dots,n)$ and we want to solve the problem for a sequence $(1,\dots,n,n+1,\dots,m)$. It is likely that some of the vertices that are generated when scheduling activities $(1,\dots,n)$ are also generated when scheduling activities $(1,\dots,n,n+1,\dots,m)$. We can reduce the computational effort associated to finding such vertices again by preloading them before starting the solution process for sequence $(1,\dots,n,n+1,\dots,m)$. On the other hand, preloading all of the vertices generated when determining a schedule for activities $(1,\dots,n)$ may create an unnecessary computational overhead, because only a share of the vertices can be expected to be required to find a solution for activities $(1,\dots,n,n+1,\dots,m)$.

To balance the tradeoff between salvaging previous computational effort and avoiding unnecessary overhead by adding too many vertices, we propose to preload all vertices of the solution path for sequence $(1, \dots, n)$ to our partially expanded network after initialization and before starting the solution process. In order to ensure that $q_{(i,t)} = \rho_i(t)$ for all vertices preloaded, we also add the immediate successors. As all vertices are added using the function `DDD::addRecursive`, Properties 1 to 3 remain satisfied. Consequently, the optimality conditions for all vertices on the previous solution path are satisfied.

The benefit of adding these vertices after initialization of the partially expanded network is that the total number of vertices generated by the dynamic discretization discovery may be drastically reduced for those cases, where the solution for sequence $(1, \dots, n, n+1, \dots, m)$ overlaps with the solution for sequence $(1, \dots, n)$. In some cases, however, preloading may cause a small overhead because vertices may be recursively added that are not needed for the solution process.

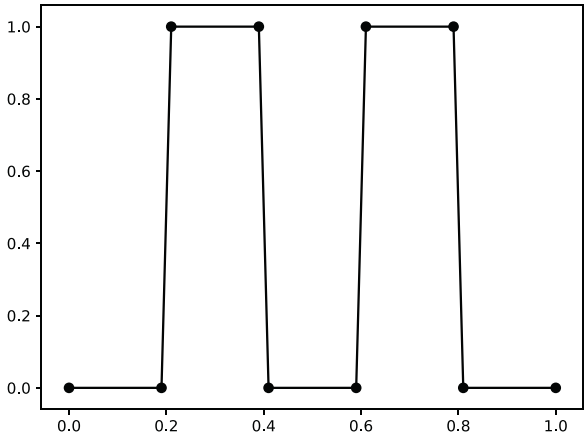


Fig. 2. Time-dependent congestion $\delta(t)$.

6. Evaluation

To evaluate our approach we use instances for an application example motivated by an industry project on last-mile deliveries with electric vehicles, where a homogeneous fleet of electric delivery vehicles is used to deliver parcels within metropolitan areas, i.e. regions consisting of densely populated city centres and less-populated surrounding territories. Traffic conditions depend both on the particular area as well as the time of the day with morning and afternoon peak hours. With increasing distance to the city centres, the impact of peak hours on travel times decreases. Thus, the time-dependency of travel times is not homogeneous and delays caused by congestion can be avoided by circumventing city centres during peak hours. Current electric vehicles have minimal consumption per unit distance at moderate speeds while the consumption increases towards very low or high velocities. Therefore, the energy consumption of the vehicles is related to travel speeds, which in turn depend on the congestion level. All vehicles leave the depot fully charged and have to return to the depot before running out of energy. They can recharge the battery at the depot as well as at public charging infrastructure.

Due to confidentiality reasons, we are not allowed to evaluate our approaches on instances obtained from data of our project partners. Therefore, we generate artificial instances for the time-dependent vehicle routing problem with replenishments having the characteristics described above. Our instances are based on the well-known instances for the vehicle routing problem with time windows by Solomon (1987). These instances have customer locations distributed in the Euclidean plane and are subdivided into instance sets with clustered customer locations (C1 and C2), randomly distributed customer locations (R1 and R2), and mixed locations (RC1 and RC2).

The delivery company can reduce costs by minimizing the number of vehicles required and the respective labour costs. As all drivers need to pick the parcels assigned to their routes in the morning, the start time of each route is fixed. Thus, labour costs are related to the time between the fixed start time and the completion of the route. Therefore, we use a hierarchical objective aiming at minimizing the total number of vehicles first and, as a second criterion, the total completion time.

In order to include time-dependent travel times, we introduce a time-dependent congestion factor $\delta(t) \in [0, 1]$ for each point in time as shown in Fig. 2. The time values on the horizontal axis in the figure are given relative to the duration of the time horizon which differs throughout the instances. Low levels of $\delta(t)$ indicate free-flow traffic conditions during off-peak hours. High values of $\delta(t)$ indicate high congestion levels during morning and afternoon peak-hours.

We assume a city centre in each quadrant of the Euclidean plane where congestion is maximal. In order to consider that the level of congestion decreases with increasing distance to the city centres, we calculate a location-dependent congestion factor $\gamma(x, y) \in [0, 1]$ by the Gaussian function based on the distance to the city centres. Fig. 3 shows the location-dependent congestion factors, the customer locations (shown as $\bullet$), and the location of the depot (shown as $\times$) for the different subsets of instances proposed by Solomon (1987). Low values of $\gamma(x, y)$ indicate low sensitivity to congestion and are illustrated in light color. High values of $\gamma(x, y)$ indicate high sensitivity to congestion and are illustrated in dark color. We can see that all instances comprise customers located in areas prone to congestion and others in areas where congestion is negligible. Even if two customers are located in areas without significant congestion, the direct path from one to the other may traverse congested areas during peak-hours.

For each integer valued point (x, y), we calculate the time required to travel a unit distance at a time t by

$$\tau_{(x,y)}(t) = \frac{\tau^{\text{free}}_{(x,y)}}{1 - \min\left\{0.8, \gamma(x, y)\right\} \cdot \delta(t)}, \tag{10}$$

where $\tau^{\text{free}}_{(x,y)}$ is the congestion-agnostic time necessary to travel a unit distance at point (x, y). The factor $\min\left\{0.8, \gamma(x, y)\right\}$ in (10) is used to generate homogeneous travel times in an area around the city centres. With this, the maximum travel time close to a city centre is five times the free flow travel time.

In order to determine time-dependent travel time and consumption functions for any pair of locations, we determine a time-dependent shortest path for different start times equally distributed over the planning horizon, assuming that the vehicle can only move one unit left, right, up, or down, and $\sqrt{2}$ units in diagonal direction. The travel time for all paths is determined by cumulating the time-dependent travel times $\tau_{(x,y)}(t)$ (or $\sqrt{2} \cdot \tau_{(x,y)}(t)$ for diagonal movements) for each integer valued point (x, y) along the path considering the respective time t at which the point is reached. A time-dependent travel time function is then determined by fitting a piecewise linear function to the duration of the time-dependent shortest paths. For each of the time-dependent shortest paths, we determine the energy consumed along the path by accumulating the speed-dependent energy consumption (Galvin, 2017) for each point traversed along the path. Then, the time-dependent consumption function is obtained by fitting a piecewise linear function to the energy consumption of the

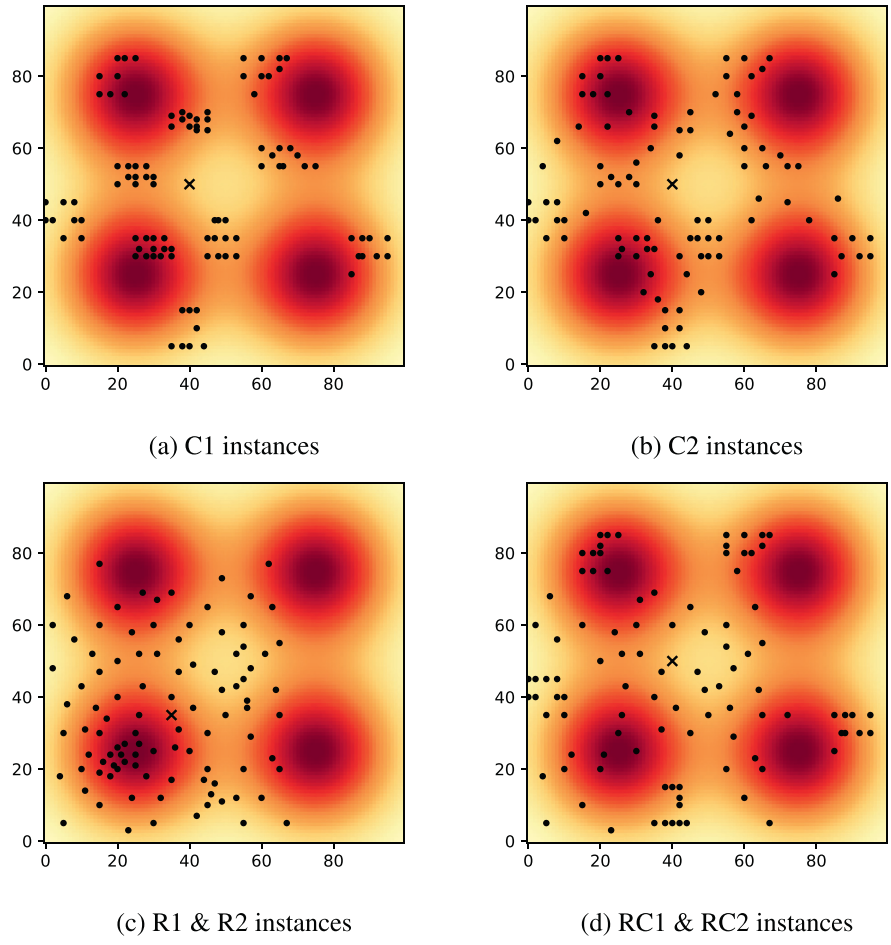


Fig. 3. Location-dependent congestion $\gamma(x, y)$ and customer locations.

time-dependent shortest paths. The source code of the instance generator can be found online at https://github.com/SteffenPottel/td_vrptw_instancegenerator.

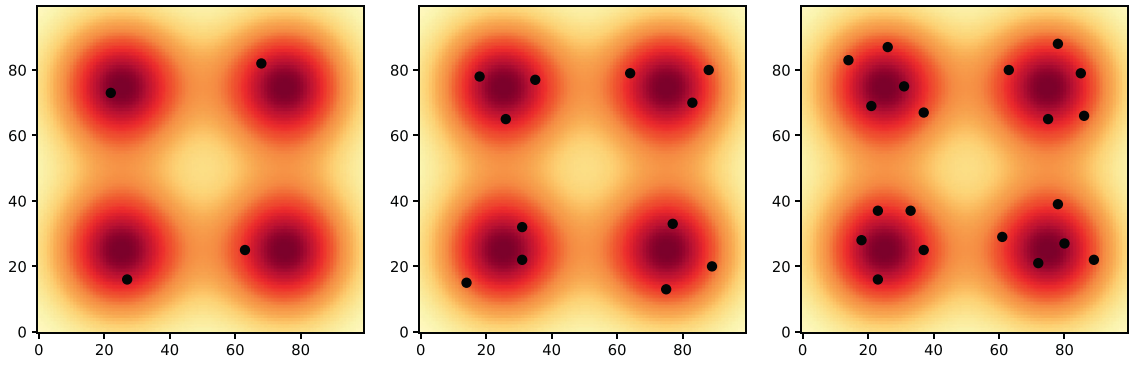
To heuristically solve these instances, we can use any local search-based approach for the vehicle routing problem in which routes are iteratively modified and each route change invokes a route evaluation that can be conducted by solving a TDASP. During the course of such an algorithm many thousands of routes are evaluated. For our experiments we decided to use an adaptation of the well-known savings algorithm (Clarke & Wright, 1964), where the savings of concatenating any pair of routes are determined using the approaches presented in this paper. As the savings algorithm is deterministic and the computational effort is mainly caused by the effort to calculate the savings, i.e., the effort to solve the TDASP, the results obtained in our experiments are not subject to any random bias and can be expected to be good indicators on the general performance of our approaches for solving the TDASP. Of course, it is likely that the overall solution quality could be significantly improved by using more sophisticated solution approaches, e.g., Adaptive Large Neighbourhood Search (Ropke & Pisinger, 2006) or Hybrid Genetic Search (Goel & Vidal, 2014). As solving the TDASP is computationally expensive, it is important that such algorithms avoid redundant route evaluations as much as possible and that solving the TDASP for unavoidable route evaluations is as fast as possible. The former can be achieved, e.g., by using global memory structures for storing and retrieving the outcome of previously conducted route evaluations (see Goel & Vidal, 2014). As our experiments will show, the latter can be achieved using the approaches presented in this paper.

In a first set of experiments we compare the impact of the choice of the discretization parameter ε and demonstrate the effectiveness of our dynamic discretization discovery approaches in comparison with applying Algorithm 1 on the fully expanded network. We implemented our algorithms in C++ and ran the experiments on a single core of an Intel Core i7-7700HQ CPU with 2.80GHz. Average results over all instances are shown in Table 2 and detailed results can be found in Tables 5 to 7 in Appendix C.

The first column of Table 2 gives the discretization parameter ε. The second and third columns show the average number of vehicles and the total completion time which are identical for both dynamic discretization discovery approaches and Algorithm 1 applied on the fully expanded network. For the dynamic discretization discovery approach (DDD) and its variant with vertex preloading (DDD-PL), the average computation time in seconds (CPU), the percentage of vertices of the fully expanded network (Vertices (%)), and the average number of vertices per route evaluation (Vertices ($\varnothing$)) are given. For Algorithm 1 applied on the fully time-expanded network (TEN) the average computation time (CPU) over those instances which are solved within the run time limit of 7200 seconds is given. As we can see, only a small percentage of the vertices of the fully expanded network are generated by our dynamic discretization discovery approaches for $\varepsilon = 0.1$ and $\varepsilon = 1.0$. With $\varepsilon = 5.0$, this percentage appears relatively high, however, this is mainly because short customer time windows in some of the instances result in a small number of vertices in the fully expanded network. Overall, the number of vertices per route evaluations only grows slowly when reducing ε and thereby increasing the number of vertices in the fully expanded network. On the other hand,

Table 2
Averaged results over all instances without service stations and different values of ε.

	DDD					DDD-PL			TEN
ε	Veh.	Compl.	CPU	Vertices (%)	Vertices (⌀)	CPU	Vertices (%)	Vertices (⌀)	CPU
0.1	14.0	7124.6	389.0	0.75	66.7	385.3	0.48	43.6	–
1.0	14.0	7110.7	34.5	4.91	45.9	27.5	3.63	33.1	*1713.9*
5.0	14.9	7432.4	13.1	17.60	33.4	9.4	15.07	27.2	*372.7*



(a) One station per city (b) Three stations per city (c) Five stations per city

Fig. 4. Location of the charging stations.

the computational effort required by Algorithm 1 applied on the fully expanded network grows quickly, thus, rendering this approach impractical for small values of ε if used within an iterative procedure with many thousands of route evaluations. We have to note, that although increasing the discretization parameter ε cannot lead to better optimal solutions, a small decrease in the completion time can be observed when going from $\varepsilon = 0.1$ to $\varepsilon = 1.0$. This unexpected results is caused by different savings values calculated for different values of ε. Even small deviations in the savings can result in different rankings of the savings. Thus different routes and solutions are generated for different values of ε.

In additional experiments, we also considered the possibility to recharge the battery during the route. In these experiments we use $\varepsilon = 0.1$. Given the poor performance of Algorithm 1 applied on the fully expanded network with $\varepsilon = 0.1$, we compare the performance of our dynamic discretization discovery approaches in these experiments with the multi-purpose mixed integer programming (MIP) solver CPLEX (with default parameters) using the MIP formulation provided in Appendix A. For simplicity of the formulation, the replenishment duration is set to a constant value. Using a non-constant replenishment function would render the MIP formulation more complicated and would also slow done the respective solution process, whereas non-constant replenishment functions only have a minor effect on the performance of the dynamic discretization discovery approaches. For these experiments, we derived additional sets of instances from the original instances. In the first additional set of instances, vehicles are allowed to recharge the battery at the depot. In the other additional sets of instances, vehicles can recharge the battery at the depot and at public charging infrastructure close to one of the four city centres. The number of charging stations available ranges from one per city to five per city and their locations are illustrated in Fig. 4.

In order to allow recharging of the battery at the depot or at public charging stations we allow vehicles to take a detour via a charging station whenever the increase in the travel distance does not exceed $\sqrt{2}$ times the direct distance between delivery locations. If multiple charging stations satisfy this criterion, then the charging station with the smallest detour is chosen. It must be noted that the limitation to only consider charging stations with smallest detour may theoretically lead to a deterioration in the overall solution quality. However, the further away the charging stations are from each other, the less likely this effect is because short trips are more likely to be used in good solutions than longer trips.

Tables 3 and 4 provide an overview over the results of these experiments. Detailed results of the experiments can be found in Tables 8 to 12 in Appendix C. The first column in Tables 3 and 4 indicates the solver used for route evaluations. The dynamic discretization discovery approach is denoted by DDD, the dynamic discretization discovery approach with preloading of vertices is denoted by DDD-PL, and the approach using the mixed integer programming solver is denoted by MIP. The second column lists the number of charging stations available. The third and fourth columns show the average computation time (in seconds) until termination of the savings algorithm, and the computation time (in seconds) per route evaluation. The fifth and sixth columns report the average number of vehicles and the average total completion time. The next two columns indicate the total number of replenishments, i.e., the total number of visits of a service station (or the depot) in order to recharge the battery, and the number of routes with at least one replenishment. The last column indicates the number of instances for which the savings algorithm terminated within the run time limit of 7200 seconds. As we can see in Tables 3 and 4, route evaluations conducted within the savings algorithm are magnitudes faster if the dynamic discretization discovery approaches are used compared to the case where CPLEX is used to solve the MIP formulation of the TDASPR. In the latter case, the savings algorithm fails to terminate for many of the instances within the run time limit of 7200 seconds.

Instances belonging to sets C1, R1, and RC1, have a smaller freight capacity and a shorter planning horizon compared to instances belonging to sets C2, R2, and RC2. Therefore, more vehicles are used for instances of sets C1, R1, and RC1 than for instances of sets C2, R2, and RC2. Thus routes for instances in sets C1, R1, and

Table 3
Results averaged over 29 instances of type C1, R1, and RC1.

Solver	Charging	Avg. CPU	CPU per Evaluation	Avg. Veh.	Avg. Compl.	Replenishments	Routes w. Repl.	Terminated
DDD	none	82.4	0.000182	18.0	5949.4	0	0	29
DDD	Depot	93.9	0.000161	18.0	5948.2	6	6	29
DDD	Depot + 1 per city	134.8	0.000203	18.1	5958.3	9	9	29
DDD	Depot + 3 per city	135.3	0.000186	18.0	5954.8	10	10	29
DDD	Depot + 5 per city	146.9	0.000198	18.0	5960.5	10	10	29
DDD-PL	none	72.6	0.000162	18.0	5949.4	0	0	29
DDD-PL	Depot	75.8	0.000128	18.0	5948.2	6	6	29
DDD-PL	Depot + 1 per city	84.0	0.000127	18.1	5958.3	9	9	29
DDD-PL	Depot + 3 per city	81.1	0.000113	18.0	5954.8	10	10	29
DDD-PL	Depot + 5 per city	88.4	0.000121	18.0	5960.5	10	10	29
MIP	none	2508.0	0.005232	18.1	5950.6	0	0	29
MIP	Depot	3299.9	0.005505	18.4	5752.5	4	4	28
MIP	Depot + 1 per city	3621.5	0.005620	18.4	5753.5	7	7	28
MIP	Depot + 3 per city	3345.9	0.005407	19.3	5622.5	5	5	23
MIP	Depot + 5 per city	3282.6	0.005342	19.5	5753.8	6	6	22

Table 4
Results averaged over 27 instances of type C2, R2, and RC2.

Solver	Charging	Avg. CPU	CPU per Evaluation	Avg. Veh.	Avg. Compl.	Replenishments	Routes w. Repl.	Terminated
DDD	none	718.2	0.001021	9.6	8386.9	0	0	27
DDD	Depot	934.5	0.000802	6.7	6936.7	188	124	27
DDD	Depot + 1 per city	1254.9	0.000853	6.8	6884.8	190	122	27
DDD	Depot + 3 per city	2021.6	0.000739	6.2	6430.8	189	113	26
DDD	Depot + 5 per city	2306.4	0.000761	6.3	6858.1	172	105	23
DDD-PL	none	721.2	0.001020	9.6	8386.9	0	0	27
DDD-PL	Depot	640.4	0.000552	6.7	6936.7	188	124	27
DDD-PL	Depot + 1 per city	739.2	0.000513	6.8	6884.8	190	122	27
DDD-PL	Depot + 3 per city	1030.3	0.000351	6.2	6654.4	197	117	27
DDD-PL	Depot + 5 per city	1045.7	0.000305	6.2	6854.9	194	116	26
MIP	none	3729.6	0.005853	10.0	9056.6	0	0	21
MIP	Depot	4877.2	0.005443	7.1	8100.1	55	40	8
MIP	Depot + 1 per city	5542.7	0.005701	6.2	7588.0	29	25	5
MIP	Depot + 3 per city	6725.1	0.005562	4.0	10778.2	1	1	1
MIP	Depot + 5 per city	6230.4	0.005494	4.0	10746.7	3	3	1

RC1 visit fewer customers and the approach requires significantly less computational effort than for instances of sets C2, R2, and RC2.

With fewer stops per route, the likelihood of running out of electric energy is much smaller. In fact, the solutions for instances of sets R1 and RC1 have no recharging stops at all, because these instances have a very short planning horizon, leaving little time to sensibly recharge batteries. For instances in sets C2, R2, and RC2, the freight capacity and planning horizons are larger. Therefore, less vehicles are used and many more routes make use of the possibility to recharge the batteries.

With an increasing number of charging stations, the number of alternative routes via charging stations increases and so does the computational effort required for evaluating these alternatives. We can see that preloading of vertices helps stabilizing the run time of the algorithm and leads to significantly shorter computation times for instances with many charging possibilities. With preloading of vertices, the dynamic discretization discovery approach is able to (heuristically) solve almost all of the instances except for instance r211 with five charging stations per city within the run time limit. Without run time limit, instance r211 with five charging stations per city can be solved in 8027.5 seconds.

An interesting observation from an application point of view is that the possibility to recharge the battery during a route can significantly reduce the number of vehicles required as well as the total completion time. Table 4 indicates that for instances in sets C2, R2, and RC2, the number of routes required can be reduced by approximately one third if vehicles can recharge batteries while they are on route. This demonstrates the importance of considering replenishments within scheduling. Furthermore, it can be seen that the number of vehicles required is not significantly reduced with an increasing number of charging stations. In fact, the possibility to recharge the battery at the depot already accounts for the main benefit in terms of the number of vehicles required. Additional public charging infrastructure may furthermore reduce the detour required to reach a charging station and thus can contribute to smaller total completion times and less routes requiring replenishments.

7. Conclusions

In this paper we introduce a time-dependent activity scheduling problem in which activities consume a limited resource during execution and activity durations and resource consumptions are time-dependent. Moreover, we study the case in which the resource can be replenished between conducting subsequent activities. We propose a dynamic discretization discovery algorithm which is based on partially time-expanded networks which are dynamically filled with additional vertices. The dynamic discretization discovery algorithm can be used for general duration and consumption functions and only requires the first-in-first-out property for activity durations. For the case that the dynamic discretization discovery is embedded in an iterative solution procedure that frequently evaluates activity sequences that start with the same activities, we propose to preload the partially expanded network with vertices generated in previous iterations. This preloading of vertices can significantly reduce the computational effort required.

We evaluate our approaches on a case of determining routes for a fleet of electric delivery vehicles for last-mile deliveries. Our

experimental results indicate that the dynamic discretization discovery algorithm is magnitudes faster than the commercial solver CPLEX using a mixed integer programming formulation of the problem. On average, our dynamic discretization discovery algorithm with preloading of vertices generates only 0.48% of the nodes of the fully time-expanded network. Furthermore, our experiments show that for our instances, which are motivated by the real-life case of last-mile delivery operations, the possibility to recharge batteries en-route can significantly reduce the number of vehicles required and the total completion time.

In some applications multiple resources may have to be considered simultaneously, e.g., if the energy of a battery-electric vehicle and the working time of the driver are both constrained. In principle, our algorithms can be adapted to consider multiple resources with their respective limit. While most of the adaptation would be straight forward, Algorithm 1, and Algorithm 4 respectively, would require multi-dimensional resource labels $\ell_{(i,t)}$ indicating the cumulative resource consumption of each resource. Many pareto-optimal resource labels may exist at a single vertex of the time-expanded network. For example, we may have resource labels with low energy consumption but a high value of working time and vice versa. Preliminary computational experiments indicated that the number of pareto-optimal labels can grow significantly resulting in a very high computational effort for solving the TDASP and TDASPR with multiple constrained resources. Future research would be required to find ways of reducing this computational effort e.g., by reducing the number of labels that actually need to be considered at each vertex.

Acknowledgment

This research is supported by the German Research Foundation (DFG) under grant GO 1841/5-1 and by the Federal Ministry of Transport and Digital Infrastructure (BMVI) for the project ZUKUNFT.DE (funding code 03EMF0101K).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.11.032

References

Archetti, C., & Savelsbergh, M. W. P. (2009). The trip scheduling problem. *Transportation Science, 43*(4), 417–431. https://doi.org/10.1287/trsc.1090.0278.

Baum, M., Dibbelt, J., Gemsa, A., Wagner, D., & Zündorf, T. (2019). Shortest feasible paths with charging stops for battery electric vehicles. *Transportation Science, 53*(6), 1501–1799.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2017). The continuous-time service network design problem. *Operations Research, 65*(5), 1303–1321. https://doi.org/10.1287/opre.2017.1624.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research, 12*(4), 519–643. https://doi.org/10.1287/opre.12.4.568.

Cordeau, J.-F., Ghiani, G., & Guerriero, E. (2014). Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. *Transportation Science, 48*, 46–58.

Desaulniers, G., Errico, F., Irnich, S., & Schneider, M. (2016). Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research, 64*(6), 1388–1405. https://doi.org/10.1287/opre.2016.1535.

European Union (2003). Directive 2003/88/EC of the European Parliament and of the Council of 4 November 2003 concerning certain aspects of the organisation of working time. *Official Journal of the European Communities, L 299*, 9–19.

Federal Motor Carrier Safety Administration (2011). Hours of service of drivers. *Federal Register, 76*(248), 81134–81188.

Foschini, L., Hershberger, J., & Suri, S. (2012). On the complexity of time-dependent shortest paths. *Algorithmica, 68*(4), 1075–1097.

Galvin, R. (2017). Energy consumption effects of speed and acceleration in electric vehicles: Laboratory case studies and implications for drivers and policymakers. *Transportation Research Part D: Transport and Environment, 53*, 234–248. https://doi.org/10.1016/j.trd.2017.04.020.

Gendreau, M., Ghiani, G., & Guerriero, E. (2015). Time-dependent routing problems: A review. *Computers and Operations Research, 64*, 189–197. https://doi.org/10.1016/j.cor.2015.06.001.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science, 43*(1), 17–26. https://doi.org/10.1287/trsc.1070.0226.

Goel, A. (2010). Truck driver scheduling in the European Union. *Transportation Science, 44*(4), 429–441. https://doi.org/10.1287/trsc.1100.0330.

Goel, A. (2012a). A mixed integer programming formulation and effective cuts for minimising schedule durations of Australian truck drivers. *Journal of Scheduling, 15*, 733–741.

Goel, A. (2012b). The minimum duration truck driver scheduling problem. *EURO Journal on Transportation and Logistics, 1*(4), 285–306. https://doi.org/10.1007/s13676-012-0014-9.

Goel, A. (2012c). The Canadian minimum duration truck driver scheduling problem. *Computers & Operations Research, 39*(10), 2267–2456.

Goel, A. (2014). Hours of service regulations in the United States and the 2013 rule change. *Transport Policy, 33*, 48–55.

Goel, A. (2018). Legal aspects in road transport optimization in Europe. *Transportation Research Part E: Logistics and Transportation Review, 114*, 144–162. https://doi.org/10.1016/j.tre.2018.02.011.

Goel, A., Archetti, C., & Savelsbergh, M. (2012). Truck driver scheduling in Australia. *Computers & Operations Research, 39*(5), 1122–1132. https://doi.org/10.1016/j.cor.2011.05.021.

Goel, A., & Kok, L. (2012). Efficient scheduling of team truck drivers in the European Union. *Flexible Services and Manufacturing Journal, 24*(1), 81–96.

Goel, A., & Rousseau, L. M. (2012). Truck driver scheduling in Canada. *Journal of Scheduling, 15*(6), 783–799.

Goel, A., & Vidal, T. (2014). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science, 48*(3), 391–412. https://doi.org/10.1287/trsc.2013.0477.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136.

Hashimoto, H., Yagiura, M., & Ibaraki, T. (2008). An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization, 5*(2), 434–456.

He, E., Boland, N., Nemhauser, G., & Savelsbergh, M. (2019). Dynamic discretization discovery algorithms for time-dependent shortest path problems. *Optimization Online, 7082*.

Kleff, A. (2019). *Scheduling and routing of truck drivers considering regulations on drivers working hours*. Karlsruhe: Karlsruher Institut für Technologie (KIT) Ph.D. thesis.. doi:10.5445/IR/1000097855

Kok, A. L., Hans, E. W., & Schutten, J. M. J. (2011). Optimizing departure times in vehicle routes. *European Journal of Operational Research, 210*(3), 579–587.

Montero, A., Méndez-Díaz, I., & Miranda-Bront, J. J. (2017). An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research, 88*, 280–289.

Montoya, A., Guéret, C., Mendoza, J. E., & Villegas, J. G. (2017). The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological, 103*, 87–110. https://doi.org/10.1016/j.trb.2017.02.004. Green Urban Transportation

Mosheiov, G. (1994). Scheduling jobs under simple linear deterioration. *Computers and Operations Research, 21*(6), 653–659. https://doi.org/10.1016/0305-0548(94)90080-9.

Nannicini, G., Delling, D., Schultes, D., & Liberti, L. (2012). Bidirectional a* search on time-dependent road networks. *Networks, 59*(2), 240–251.

Pelletier, S., Jabali, O., & Laporte, G. (2016). 50Th Anniversary invited article-goods distribution with electric vehicles: Review and research perspectives. *Transportation Science, 50*(1), 3–22. https://doi.org/10.1287/trsc.2015.0646.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., & Rousseau, L. M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science, 44*(4), 455–473. https://doi.org/10.1287/trsc.1100.0328.

Rancourt, M.-E., Cordeau, J.-F., & Laporte, G. (2013). Long-haul vehicle routing and scheduling with working hour rules. *Transportation Science, 47*(1), 81–107. https://doi.org/10.1287/trsc.1120.0417.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science, 40*(4), 455–472. https://doi.org/10.1287/trsc.1050.0135.

Schiffer, M., Schneider, M., Walther, G., & Laporte, G. (2019). Vehicle routing and location routing with intermediate stops: A review. *Transportation Science, 53*(2), 319–343. https://doi.org/10.1287/trsc.2018.0836.

Schneider, M., Stenger, A., & Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science, 48*(4), 500–520. https://doi.org/10.1287/trsc.2013.0490.

Schrank, D., Eisele, B., & Lomax, T. (2019). *2019 Urban mobility report*. Texas A&M Transportation Institute. URL https://static.tti.tamu.edu/tti.tamu.edu/documents/mobility-report-2019.pdf

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 35*(2), 166–324. https://doi.org/10.1287/opre.35.2.254.

Taş, D., Gendreau, M., Jabali, O., & Laporte, G. (2016). The traveling salesman problem with time-dependent service times. *European Journal of Operational Research, 248*(2), 372–383.

Tilk, C., & Goel, A. (2020). Bidirectional labeling for solving vehicle routing and truck driver scheduling problems. *European Journal of Operational Research, 283*(1), 108–124. https://doi.org/10.1016/j.ejor.2019.10.038.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2015). Timing problems and algorithms: Time decisions for sequences of activities. *Networks, 65*(2), 102–128.

Visser, T. R., & Spliet, R. (2020). Efficient move evaluations for time-dependent vehicle routing problems. *Transportation Science, 54*(4), 1091–1112. https://doi.org/10.1287/trsc.2019.0938.

Vu, D. M., Hewitt, M., Boland, N., & Savelsbergh, M. (2020a). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science, 54*(3), 703–720. https://doi.org/10.1287/trsc.2019.0911.

Vu, D. M., Hewitt, M., & Vu, D. D. (2020b). Solving the time dependent minimum tour duration and delivery man problems with dynamic discretization discovery. *Optimization online, 7998*.

Zhang, M., Batta, R., & Nagi, R. (2009). Modeling of workflow congestion and optimization of flow routing in a manufacturing/warehouse facility. *Management Science, 55*(2), 267–280.