Scheduling hours of service for truck drivers with interdependent routes

Carlo S. Sartori*, Pieter Smet, Greet Vanden Berghe

KU Leuven, Department of Computer Science, CODeS, Gebroeders De Smetstraat 1, Gent 9000, Belgium

Abstract

When transporting goods, truck drivers may have to perform load transfers between two vehicles at cross-dock or transfer locations. These transfers create interdependencies between the routes of different truck drivers and require careful temporal synchronization. The situation becomes even more complex when driver schedules must comply with customer time windows, working periods and regulatory resting requirements. Although the scheduling of hours of service for truck drivers is a well-known problem, existing algorithms cannot cope with interdependent routes. Successfully accommodating this real-world characteristic demands the introduction of a new, more general truck scheduling problem: The Truck Driver Scheduling Problem with Interdependent Routes. This paper introduces a mathematical model and a label propagation algorithm to solve this new problem. Experiments indicate that the algorithm can quickly schedule a large number of interdependent routes and outperforms a mathematical programming approach. The algorithm produces valid schedules for multiple drivers and thus becomes a foundation for future vehicle routing algorithms that address interdependent routes and hours of service regulations.

Keywords: Scheduling, Truck driver scheduling, Interdependent routes, Hours of service, Label propagation
2010 MSC: 00-01, 99-00

1. Introduction

During the daily execution of vehicle routes for logistic operations, the drivers' hours of service (HOS) must comply with certain legal requirements.

*Corresponding author. Tel.: +32 9 265 87 03
Email addresses: carlo.sartori@cs.kuleuven.be (Carlo S. Sartori), pieter.smet@cs.kuleuven.be (Pieter Smet), greet.vandenberghe@cs.kuleuven.be (Greet Vanden Berghe)

For instance, most countries stipulate a maximum time for a driver to continuously operate a vehicle, after which a break must be scheduled. This is not arbitrary, as drivers who do not take breaks for long periods compromise the safety of roads. It is, therefore, crucial to take into account such regulations when developing vehicle routing algorithms.

Additionally, recent increases in *e-commerce* activity have brought significant challenges to logistic chains, which must deliver products to ever more demanding customers. This increased pressure on transportation companies has motivated the use of cross-docks (Van Belle et al., 2012), or transfer locations (Mitrović-Minić & Laporte, 2006), where loads are transferred between vehicles to reduce delivery times. Although useful, these operations incur interdependencies between vehicle routes which complicate the development of automated vehicle routing tools.

Goel (2009) introduced the main terminology for the *Truck Driver Scheduling Problem* (TDSP), which is the name given to the scheduling of routes while respecting HOS regulations. There is no singular definition for the TDSP given that different geographical regions enforce different regulations. The two most widely studied regulations in the literature are: the US-TDSP for the United States of America (Goel & Kok, 2012) and the EU-TDSP for the European Union (Goel, 2009). Despite their differences, all TDSP variants stipulate that drivers must take breaks at regular intervals.

In terms of HOS, we consider the EU regulations. Two basic regulatory guidelines dictate the drivers' HOS. Regulation (EC) No. 561/2006 (EU Commission, 2006) concerns driving-related activities. It defines *driving periods* to be at most 4.5 hours of continuous driving time and defines *breaks* to be 45-minute intervals which end a driving period. Additionally, the regulation limits the maximum driving time in a day to nine hours. Meanwhile, directive 2002/15/EC (EU Commission, 2002) defines rules concerning the accumulated working hours for both driving and service tasks (such as loading and unloading) and limits a *working period* to at most six continuous hours. A 30-minute break ends a working period. In this research, we consider only a 45-minute break that ends both working and driving periods as in other TDSP-related papers. The time horizon is a single working day for each driver, thus only breaks must be scheduled. Neither split breaks nor rest periods between days are considered.

Our research sets out to explore whether it is possible to extend the current TDSP methods to schedule HOS for drivers with interdependent routes while respecting specific time windows for their tasks. More specifically: is it possible to quickly produce *feasible* schedules for a set of *fixed and interdependent* truck routes? To answer this question, this paper introduces the *Truck Driver Scheduling Problem with Interdependent Routes* (TDSP-IR).

The remainder of this paper is structured as follows. Section 2 reviews the related literature. The TDSP-IR is formally defined in Section 3 together with a mathematical programming formulation. Section 4 introduces a label

propagation algorithm and its related operations. A complexity analysis of the proposed algorithm is provided in Section 5. Computational results are reported in Section 6, while Section 7 concludes the paper and offers some directions for future research.

2. Related work

The literature relevant to the TDSP-IR can be divided into two main areas. One deals with the family of TDSPs, whereas the other considers vehicle routing problems with synchronization constraints. This section explores both of these areas in view of the TDSP-IR challenges.

2.1. The truck driver scheduling problem

The TDSP literature is incredibly rich and is typically categorized according to the regulations being considered. Since our paper deals with EU regulations, we refrain from reviewing studies that address other sets of rules, such as the US, Canadian, or Australian regulations. For these cases, the interested reader is referred to the research by Goel & Kok (2012), Goel & Rousseau (2012) and Goel et al. (2012).

For the EU regulations, Goel (2009) introduced the first method to check route feasibility in a week-long horizon that requires the scheduling of both breaks and rest periods between days. A *Label Propagation Algorithm* (LPA) performed significantly better on the scheduling problem compared to a naive approach. Goel (2010) presented a detailed discussion concerning the regulations of the EU-TDSP and introduced a new LPA that either finds a feasible schedule or proves none exists. In both studies, the LPA was devised as a feasibility check for Vehicle Routing Problems (VRPs) to ensure that VRP solutions respect the European Union's regulations.

The two aforementioned approaches assumed that breaks or rests can be taken anywhere *en-route*. This is not necessarily true as drivers may have to travel for quite some time before finding an appropriate parking space for their vehicle. For the EU-TDSP, this could lead to breaching the regulations expressed in (EC) No. 561/2006.

Kok et al. (2011) addressed the scheduling of truck drivers under EU regulations, one working day (time horizon of 13 hours) and time-dependent travel times. Similar to our problem settings, they considered breaks to be restricted to customer locations. They solved the scheduling problem with a MILP formulation and showed that execution times were short for practical uses, even for time-dependent travel times.

Goel (2012) proposed a dynamic programming algorithm to create minimum duration schedules for the EU-TDSP when breaks are limited to customer locations, but no polynomial-time bound on the algorithm's worst-case performance was provided. Goel's dynamic programming approach is actually very similar to LPAs in terms of how solutions are enumerated.

Kleff (2019) introduced the first polynomial-time algorithm for the EU-TDSP when breaks are restricted to customer locations. The proposed algorithm is an LPA, but schedules are primarily represented using piecewise linear functions. This is not a straightforward representation for our problem, despite its interesting results for the single-route TDSP.

In a broader study, Goel & Vidal (2013) set out to understand the impact of HOS regulations on VRP solutions. They evaluated route feasibility during heuristic search using LPAs available in the literature. In doing so, it was possible to employ one implementation of a VRP metaheuristic that could create a solution compliant with a specific set of regulations by simply selecting the appropriate LPA. Their approach is important because it shows that studying independent scheduling methods for truck drivers enables one to apply those methods in general VRP algorithms, regardless of the type of metaheuristic or solution representation. The scheduling algorithm works as a "plug-and-play" modular component. Indeed, it is the previous success of applying LPAs to solve TDSPs that has motivated us to investigate their applicability to the TDSP-IR.

An overview of the research related to the TDSP is presented in Table 1. The main characteristics of the scheduling problem are presented in the columns: the objective (whether the algorithm must generate an optimal or a feasible schedule), the regulation(s) incorporated, the number of time windows per location and whether interdependent routes are considered.

Table 1: Summary of TDSP-related research.

Reference	Objective		Regulation				Time windows		
	Opt.	Feas.	US	EU	CAN	AUS	Single	Multiple	Interdep.
Goel (2009)		•		•			•		
Goel (2010)		•		•			•		
Kok et al. (2011)	•			•			•		
Goel & Kok (2012)		•	•					•	
Goel & Rousseau (2012)		•			•		•		
Goel (2012)	•		•	•				•	
Goel et al. (2012)		•				•	•		
Goel & Vidal (2013)		•	•	•	•	•	•		
Kleff (2019)	•	•	•	•				•	
This work		•		•			•		•

Table 1 reveals some interesting patterns. First, most papers studied the decision problem, rather than an optimization problem (such as minimization of the route duration). Second, research has focused mainly on either the US- or the EU-TDSP, probably because these regions have the greatest share of truck-based transport lines. Third, greater attention was given to the simpler case with a single time window per customer location. Finally, no previous research related to the TDSP has considered interdependent routes, even though there are many VRPs which take interdependencies and task synchronization into account, as we will discuss in the following section.

2.2. Vehicle routing problems with task synchronization

A number of researchers have considered interdependent routes in generalizations of established vehicle routing problems. Dohn et al. (2011) described the five most common types of temporal dependencies between tasks in different routes. This paper addresses one of these temporal dependencies, called *minimum difference* (detailed in Section 3.1).

Drexl (2013) reviewed vehicle routing problems with trailers and transshipment including synchronization constraints. He concluded that only a few papers have examined the interplay of HOS and interdependencies, while none tackled the problem we consider in this paper.

Interdependence constraints have also been studied in transfer or transshipment operations. Mitrović-Minić & Laporte (2006) introduced the *Pickup and Delivery Problem with Transshipment* (PDPT). Contrary to usual pickup and delivery problems, the PDPT allows the pickup and the delivery location of a request to be served by different vehicles. An intermediate transfer location may be employed which serves as a temporary storage where vehicles are allowed to drop loads which can then be picked-up later by a second vehicle. A clear interdependence arises here because a vehicle must first unload at a transfer location before another vehicle can pick up the load.

The *Vehicle Routing Problem with Cross-Docking* (Van Belle et al., 2012) is similar to the PDPT. In this version of the VRP, vehicles must collect products from pickup locations, bring these products together at a cross-dock and then deliver those products to another set of customers. Loads may be transferred between vehicles at the cross-dock facility. Similar to the PDPT, synchronization between vehicle routes may be required since vehicles that collect loads at the cross-dock can only leave the facility once all load transferring has finished, thus defining interdependencies between several vehicle routes.

No previous research has considered specialized algorithms to schedule truck drivers' HOS in the PDPT or VRP with cross-docking. This is surprising, given that when interdependencies exist, leaving the scheduling of HOS to a post-routing process can lead to significant cascading effects that render all routes infeasible. This is especially the case when there is a tight, limited route duration. In such situations adding a single 45-minute break to a route can cause disruptions, but not adding the mandatory breaks may create legal issues for logistic companies.

Finally, despite some similarities with scheduling problems encountered in airline, railway and bus transportation (Leung, 2004), the HOS regulations present in truck transportation are different (Goel & Vidal, 2013). For example, arrival and working times in road transportation are defined by time windows, whereas the other cases are generally given by a timetable. Therefore, we consider the study of HOS scheduling in VRPs with interdependent routes and the development of fast algorithms for doing so to address a scientific gap of significant practical relevance.

3. The truck driver scheduling problem with interdependent routes

This section formally defines the Truck Driver Scheduling Problem with Interdependent Routes and presents a MILP formulation. Examples and characteristics of the TDSP-IR are also discussed.

3.1. Problem description

An instance of the TDSP-IR is defined over a set of m fixed vehicle routes. A vehicle route is a sequence of locations $r_k = (\lambda_1, \ldots, \lambda_{|r_k|}),\ k = 1, \ldots, m$, where $|r_k|$ denotes the number of locations in route r_k. These routes are represented as a graph $G = (V, A)$, where V is the set of nodes representing locations, while A is the set of arcs denoting temporal precedence constraints between the nodes in V. The parameters of an instance and the decision variables used throughout this paper are summarized in Table 2.

Arc set A is composed of two sets $A = A_P \cup A_R$ such that $A_P \cap A_R = \emptyset$. Arc set A_R is the set of route arcs of type $(i, j), i, j \in V$, where nodes i and j belong to the same route and node i directly precedes j. Each arc has an associated weight t_{ij}, which is the travel time from node i to j. Meanwhile, set A_P contains precedence arcs between nodes in two different routes, that is, they represent *interdependence constraints* of the type $(u_-, u_+), u_-, u_+ \in V$ such that service at node u_- must end before service at u_+ can begin. These arcs incur the *minimum difference* constraints (Dohn et al., 2011), which are defined by $H_{u_-} + w_{u_-} \leq H_{u_+}$, where H_i is the time at which service begins at node $i \in V$ and w_i is the service duration at node i.

For an instance to be valid, input graph G must be a *directed acyclic graph*: arc set A cannot define cycles in G. This is always true because a cycle creates a precedence between at least one node and itself, which can never be resolved (Masson et al., 2013).

Set $V_O \subset V$ contains vertices o_k and d_k, which are the origin and destination nodes of the k-th route, respectively. Additionally, every node $i \in V$ has an associated time window $[e_i, l_i]$ indicating the earliest time e_i and the latest time l_i service may begin at this location, as well as a service duration w_i. For interdependent nodes $(u_-, u_+) \in A_P$, the start of the time window at u_+ may vary according to the start of service at u_-. Analogously, the end of time window at u_- may change according to the start of service at u_+. Time windows of nodes $o_k, d_k \in V_O$ define the time horizon $\mathcal{H}_k$ of each route r_k, where e_{o_k} is the earliest start time and l_{d_k} is the latest end time. For simplicity, we will assume the time horizons of all routes are the same and denoted as $\mathcal{H}$.

Each route r_k must be completed by a single truck driver. The driver's schedule must not only comply with customer time windows, but also with HOS regulations. In other words, a break of minimum length B must be scheduled before the driver accumulates M_{drive} or M_{work} continuous driving

or working time, respectively. A break is only allowed at customer locations *after* completing the service. However, this could easily be adapted to consider breaks before service begins (Goel, 2012). Once a break has been taken, both accumulated driving and working time are reset to zero and the driver is allowed to drive and work an additional M_{drive} and M_{work} units of time. Note that working time includes both driving times and service duration at nodes. A route must not exceed the maximum duration M_{dur}, which serves as the *shift length* of a driver. Finally, M_{daily} restricts the total driving time in one day. For simplicity, we assume that any route in a TDSP-IR instance does not exceed M_{daily}. In our case, this can be trivially verified in linear time before solving the problem.

Table 2: Notation used throughout the paper.

Instance parameters	
n	total number of nodes $\|V\|$
m	total number of routes (or drivers)
$i \in V$	a node to be visited
$o_k, d_k \in V_O$	origin and destination nodes of the k-th route
$(i,j) \in A_R$	set of route arcs (simple precedence arcs)
$(u_-, u_+) \in A_P$	set of interdependence arcs ($u_-, u_+ \in V$)
$[e_i, l_i]$	time window associated with node $i \in V$
w_i	service duration at node $i \in V$
t_{ij}	travel time associated with arc $(i,j) \in A_R$
$\mathcal{H}$	time horizon of all routes
M_{drive}	maximum continuous driving time
M_{work}	maximum continuous working time
M_{daily}	maximum daily driving time
M_{dur}	maximum route duration
B	minimum duration of a break period
Variables	
H_i	time at which service begins at node i
D_i	accumulated continuous driving time up to node i
W_i	accumulated continuous working time up to node i
x_i	binary variable which equals 1 when a break is scheduled immediately after service at node i and 0 otherwise

The goal of the TDSP-IR is to build a feasible schedule for each route that complies with both customer time windows and HOS regulations. Therefore, a solution method for this problem must provide both the starting time for all routes and the breaks during those routes.

To better illustrate the problem definition, consider the instance in Figure 1 with $m = 3$ routes. Arc weights represent travel times while service durations are $w_i = 3,\ \forall\ i \in V$. The only time windows relevant to the example are those at nodes e and f. All other time windows are assumed to be $[0, \mathcal{H}]$. Interdependencies are $(a_-, a_+), (c_-, c_+) \in A_P$.

Suppose the HOS regulations are $M_{\text{drive}} = 27, M_{\text{work}} = 36, M_{\text{daily}} = 50$, $M_{\text{dur}} = 53$ and $B = 5$, while the time horizon is $\mathcal{H} = 70$. A schedule for each route that complies with time windows, interdependence constraints, M_{drive},

M_{work} and M_{daily} regulations, but exceeds M_{dur} for two routes is shown in Figure 2(a). Gray squares denote the service duration at each node, blue rectangles (D) denote driving periods, red rectangles (B) denote scheduled breaks, while white rectangles (I) denote idle or waiting periods. All three schedules begin their routes at time $t = 0$, which incurs unnecessary waiting time. As a result, schedules for routes r_1 and r_2 are *infeasible* because they have a total duration of $57 > M_{\text{dur}}$ and $58 > M_{\text{dur}}$, respectively. Note that the schedule in r_1 has unnecessary waiting time at a_+ even though this location has no time window. This is due to the start time of service at a_-.

Figure 1: A TDSP-IR instance. Solid arcs belong to set A_R and dotted arcs to set A_P.

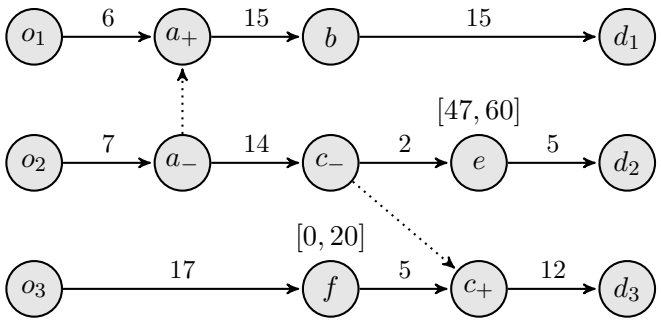
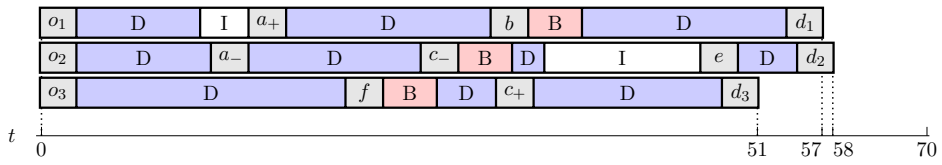
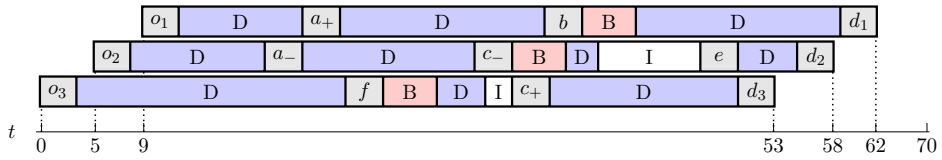


Figure 2: Example of two sets of schedules for a TDSP-IR instance.



(a) Infeasible schedules beginning at time $t = 0$ (some durations exceed $M_{\text{dur}} = 53$).



(b) Feasible schedules with postponed starting times (all durations within $M_{\text{dur}} = 53$).

Modifying the start time of r_2 to reduce the waiting time at e increases the time spent waiting at a_+ and the duration of r_1 even further. Since route r_1 has no tight time windows, it is possible to completely remove the idle time by postponing the start time of its service as much as possible.

However, waiting in r_2 cannot be completely eliminated because the time window at f forbids route r_3 to be postponed. Figure 2(b) presents three feasible schedules with postponed starting times. All three routes have a duration of 53.

3.2. Mathematical programming formulation

The TDSP-IR can be modeled as a MILP using the variables presented in Table 2. The set of constraints is then written as follows:

$$H_j \geq H_i + t_{ij} + w_i + x_i B, \quad \forall\ (i,j) \in A_R \tag{1}$$
$$e_i \leq H_i \leq l_i, \quad \forall\ i \in V \tag{2}$$
$$H_{u_-} + w_{u_-} \leq H_{u_+}, \quad \forall\ (u_-, u_+) \in A_P \tag{3}$$
$$D_j \geq D_i + t_{ij} - x_i M_{\text{drive}}, \quad \forall\ (i,j) \in A_R \tag{4}$$
$$D_j \geq t_{ij}, \quad \forall\ (i,j) \in A \tag{5}$$
$$W_j \geq W_i + t_{ij} + w_j - x_i M_{\text{work}}, \quad \forall\ (i,j) \in A_R \tag{6}$$
$$W_j \geq t_{ij} + w_j, \quad \forall\ (i,j) \in A_R \tag{7}$$
$$0 \leq D_i \leq M_{\text{drive}}, \quad \forall\ i \in V \tag{8}$$
$$0 \leq W_i \leq M_{\text{work}}, \quad \forall\ i \in V \tag{9}$$
$$H_{d_k} - H_{o_k} \leq M_{\text{dur}}, \quad \forall\ o_k, d_k \in V_O \tag{10}$$
$$x_i \in \{0,1\}, \quad \forall\ i \in V \tag{11}$$

Constraints (1) update the start of service variables along a route in accordance with the travel times, service times and breaks. Time window compliance for every customer location is ensured by Constraints (2). Constraints (3) guarantee that for precedence pairs $(u_-, u_+) \in A_P$, service at node u_- always ends before service at node u_+ begins. For continuous driving and working time, Constraints (4)–(7) update the values according to the sequence of nodes and are responsible for resetting the corresponding variables whenever a break takes place. Constraints (8) and (9) ensure compliance with maximum driving and working HOS. Constraints (10) guarantee that the length of a route never exceeds the maximum duration. Finally, Constraints (11) restrict x variables to assume only binary values.

Computational experiments indicated that the MILP formulation was too slow for use in iterative algorithms for vehicle routing problems. Therefore, we turned our attention to the development of a specialized approach to solve the TDSP-IR.

4. Label propagation algorithm

Based on previous research for the TDSP, we propose a Label Propagation Algorithm (LPA): an iterative technique that propagates labels, here representing a set of partial driver schedules, until a valid solution is either

found or proven not to exist. For the single-route TDSP, all iterations propagate labels from one node to the next within the same route. However, in the TDSP-IR, iterations propagate information across multiple routes until all of the routes are scheduled.

This section begins by providing an overview of the algorithm, highlighting its main components. A label is then formally defined in Section 4.2. Operations over labels and their enumeration process are detailed in Sections 4.3 and 4.4, respectively. Section 4.5 describes the feasibility conditions of labels. Section 4.6 presents an example of the LPA's execution. Finally, dominance criteria are presented in Section 4.7.

4.1. Algorithm overview

An overview of the LPA is shown in Algorithm 1, which receives as input a TDSP-IR instance with graph G. The output is a boolean value indicating whether the instance is feasible with respect to all the TDSP-IR constraints.

Algorithm 1 LPA

Input: TDSP-IR instance with graph G.
Output: Boolean value indicating whether the input instance is feasible.

```
1: T_G ← toposort(G)
2: L^0 ← initial_label(G)
3: S ← push(S, L^0)
4: feasible ← false
5: while S ≠ ∅ and not feasible do
6:     S ← label_enumeration(S, T_G)
7:     if L ∈ S is finished then
8:         feasible ← true
9:     end if
10: end while
11: return feasible
```

Line 1 generates a *topological order* $\mathcal{T}_G$ (Cormen et al., 2009) of the input graph G. This order transforms the propagation across multiple routes into a sequential propagation, as though there was only one route. Such an order is necessary to define the precise sequence of propagations that respects the interdependent tasks in set A_P. In other words, order $\mathcal{T}_G$ ensures that whenever a node u_+ from a pair $(u_-, u_+) \in A_P$ is to be processed, the corresponding u_- has already been processed. Since G is a directed acyclic graph, it is always possible to obtain a topological order. An example of this order for the graph in Figure 1 is:

$$\mathcal{T}_G = (a_-, c_-, e, d_2, a_+, b, d_1, f, c_+, d_3)$$

Note that origin nodes are not included because they are assumed to be the starting location for each route. Order $\mathcal{T}_G$ is fixed and decided in this step of the algorithm. Procedure `toposort` may be any valid topological sort

algorithm, however certain characteristics may improve the performance of the LPA (see Section 4.4).

Line 2 creates an initial label $\mathcal{L}^0$, which defines the beginning of the propagation. This label is stored in stack $\mathcal{S}$ by pushing it to the top (line 3). The main loop spanning lines 5–10 calls the necessary procedure to propagate labels. Procedure `label_enumeration` generates labels by taking a label from the top of $\mathcal{S}$, propagating it over multiple iterations and storing all the newly generated labels back into $\mathcal{S}$. This procedure is fully described in Section 4.4.

If some label in $\mathcal{S}$ reaches the end of the propagation – that is, reaches the last node in $\mathcal{T}_G$ – then it means a feasible solution exists (line 7). The variable `feasible` is updated to `true` and the algorithm terminates. However, should stack $\mathcal{S}$ become empty before any label reaches the last node in $\mathcal{T}_G$, then the value of `feasible` remains `false`, indicating infeasibility.

4.2. Label definition

A label $\mathcal{L}$ stores information for a set of m partial schedules, one for each route in the TDSP-IR. The variables in Table 2 are stored in each label $\mathcal{L}$ and we denote them by $\mathcal{L}(H_i), \mathcal{L}(D_i), \mathcal{L}(W_i), \mathcal{L}(x_i)$: the earliest start time of service at node i, the continuous driving time up until node i, the continuous working time up until node i and whether a break is scheduled immediately after servicing node i. Recall that nodes are unique and always belong to a single route.

Labels also store slack values $\mathcal{L}(S_{ki})$, which correspond to the accumulated waiting time from every origin $o_k, \forall k \in \Phi(i)$ up to every node i. This is based on the definition of total waiting time offered by Masson et al. (2013). Here, $\Phi(i)$ denotes all the route indices k for which o_k precedes node i in G. For example, in the graph in Figure 1, $\Phi(e) = \{2\}$ because only o_2 precedes node e. Meanwhile, $\Phi(b) = \{1, 2\}$ because o_1 clearly precedes node b, while the path (o_2, a_-, a_+, b) shows that o_2 also precedes b. The slack is necessary to indicate how much each route in the TDSP-IR can be postponed without affecting any services. The slack is used in the dominance conditions (Section 4.7) to avoid the removal of labels which contain a feasible solution with respect to maximum route duration.

Moreover, labels store push values $\mathcal{L}(P_k)$. They indicate how much the start time of service at $o_k, k = 1, \ldots, m$, that is $\mathcal{L}(H_{o_k})$, may be postponed without causing route r_k, or any other dependent routes, to become infeasible with respect to time windows.

To keep track of the propagation across multiple routes, every label stores values $\mathcal{L}(i_k), k = 1, \ldots, m$ denoting the last node scheduled in route r_k. Meanwhile, $\mathcal{L}(p)$ denotes the next node to be scheduled for this label following the order $\mathcal{T}_G$. To clarify, consider the topological order of Figure 1: $\mathcal{T}_G = (a_-, c_-, e, d_2, a_+, b, d_1, f, c_+, d_3)$. A label with $\mathcal{L}(p) = 3$ holds values $\mathcal{L}(i_1) = o_1$, $\mathcal{L}(i_2) = c_-$ and $\mathcal{L}(i_3) = o_3$. Then, $\mathcal{T}_G[\mathcal{L}(p)]$ provides the next

node to be reached in the propagation of label $\mathcal{L}$ in the linear sequence. Thus, for the example in Figure 1, the next propagation will be from c_- to e in route r_2 given that $\mathcal{T}_G[\mathcal{L}(p)] = \mathcal{T}_G[3] = e$.

The initial label $\mathcal{L}^0$ is defined as:

$$\begin{aligned}
&\mathcal{L}^0(p) = 1 \\
&\mathcal{L}^0(i_k) = o_k, && k = 1, \ldots, m \\
&\mathcal{L}^0(H_{o_k}) = \mathcal{L}^0(D_{o_k}) = \mathcal{L}^0(W_{o_k}) = 0, && k = 1, \ldots, m \\
&\mathcal{L}^0(S_{ko_k}) = 0, && k = 1, \ldots, m \\
&\mathcal{L}^0(P_k) = l_{o_k}, && k = 1, \ldots, m \\
&\mathcal{L}^0(x_i) = 0, && \forall\ i \in V
\end{aligned}$$

4.3. Propagation operations

A propagation refers to a single move from a node i to j in one route r_k. This one move creates exactly two labels from $\mathcal{L}$. The first, denoted $\mathcal{L}^{\text{direct}}$, is a direct trip from i to j. The second, denoted $\mathcal{L}^{\text{break}}$, contains a break after service at node i before continuing on to j. The difference between the two labels is that $\mathcal{L}^{\text{direct}}(x_i) = 0$ and $\mathcal{L}^{\text{break}}(x_i) = 1$. All the other values in these labels are computed according to standard operations that depend on the value of variable x_i.

Equations (12)–(18) describe the propagation operations. Value $\tilde{e}_j$ is the earliest start time of service at j. It is $\tilde{e}_j = \max\{e_{u_+}, \mathcal{L}(H_{u_-}) + w_{u_-}\}$ if $j = u_+$ for $(u_-, u_+) \in A_P$ and $\tilde{e}_j = e_j$ otherwise. The first definition is used to guarantee compliance with the interdependence constraints.

$$\mathcal{L}(i_k) = j \tag{12}$$
$$\mathcal{L}(p) = \mathcal{L}(p) + 1 \tag{13}$$
$$\mathcal{L}(H_j) = \max\{\tilde{e}_j, \mathcal{L}(H_i) + w_i + \mathcal{L}(x_i) \cdot B + t_{ij}\} \tag{14}$$
$$\mathcal{L}(D_j) = \mathcal{L}(x_i) \cdot \mathcal{L}(D_i) + t_{ij} \tag{15}$$
$$\mathcal{L}(W_j) = \mathcal{L}(x_i) \cdot (\mathcal{L}(W_i) + w_i) + t_{ij} \tag{16}$$

Slack values are also propagated from i to j. Due to the interdependencies, slack values propagate from all origins $o_z, z \in \Phi(i)$ which precede node i (and necessarily node j). Slack value $\mathcal{L}(S_{zi})$ is propagated from i to j by:

$$\mathcal{L}(S_{zj}) = \mathcal{L}(S_{zi}) + \max\{0, \tilde{e}_j - (\mathcal{L}(H_i) + w_i + \mathcal{L}(x_i) \cdot B + t_{ij})\}, \forall z \in \Phi(i) \tag{17}$$

If $j = u_+$ in a pair $(u_-, u_+) \in A_P$, then we must also consider the slack from origins that precede the corresponding node u_-:

$$\mathcal{L}(S_{zu_+}) = \min\{\mathcal{L}(S_{zu_+}), \mathcal{L}(S_{zu_-}) + \max\{0, \mathcal{L}(H_{u_+}) - (\mathcal{L}(H_{u_-}) + w_{u_-})\}\}, \forall z \in \Phi(u_-) \tag{18}$$

Meanwhile, the push value is updated for each route where the origin precedes j by:

$$\mathcal{L}(P_z) = \min\{\mathcal{L}(P_z), \mathcal{L}(S_{zj}) + l_j - \mathcal{L}(H_j)\}, \forall z \in \Phi(j) \tag{19}$$

One label propagation has time complexity $\mathcal{O}(m)$ since it must consider all $z \in \Phi(j)$, which can contain at most m elements. The memory consumption of a label depends on both n and m. Variables $\mathcal{L}(H_i)$, $\mathcal{L}(D_i)$, $\mathcal{L}(W_i)$ and $\mathcal{L}(x_i)$ are stored for every node in G, thus requiring $\mathcal{O}(n)$ memory. Meanwhile $\mathcal{L}(P_k)$ requires $\mathcal{O}(m)$ memory and $\mathcal{L}(S_{ki})$ requires $\mathcal{O}(nm)$ memory. Hence a label requires $\mathcal{O}(nm)$ memory.

4.4. Label enumeration

Algorithm 2 describes the procedure `label_enumeration` used in Algorithm 1 to generate labels, that is, to propagate them iteratively. This function is the main component of the LPA in terms of enumerating solutions. It begins by taking one label from the top of stack $\mathcal{S}$ and pushing it to queue $\mathcal{Q}$. In line 2, $r(i)$ provides the index of the route to which node i belongs: an index k in the range $1, \ldots, m$. Index k is used so that Algorithm 2 propagates labels only for node sequences in r_k.

Algorithm 2 `label_enumeration`

Input: stack $\mathcal{S}$, topological order $\mathcal{T}_G$
Output: updated stack $\mathcal{S}$

1: $\mathcal{L}' \leftarrow$ `top`$(\mathcal{S})$
2: $k \leftarrow r(\mathcal{T}_G[\mathcal{L}'(p)])$
3: $\mathcal{Q} \leftarrow$ `push`$(\mathcal{Q},\ \mathcal{L}')$
4: **while** $\mathcal{Q} \neq \emptyset$ **do**
5: $\quad\mathcal{L} \leftarrow$ `front`$(\mathcal{Q})$
6: $\quad$**if** $r(\mathcal{T}[\mathcal{L}(p)]) = k$ **then**
7: $\quad\quad\mathcal{L}^{\text{direct}},\ \mathcal{L}^{\text{break}} \leftarrow$ `propagate`$(\mathcal{L},\ \mathcal{T}_G)$
8: $\quad\quad\mathcal{Q} \leftarrow$ `push`$(\mathcal{Q},\ \mathcal{L}^{\text{direct}})$
9: $\quad\quad\mathcal{Q} \leftarrow$ `push`$(\mathcal{Q},\ \mathcal{L}^{\text{break}})$
10: $\quad\quad\mathcal{Q} \leftarrow$ `apply_dominance`$(\mathcal{Q})$
11: $\quad$**else**
12: $\quad\quad\mathcal{S} \leftarrow$ `push`$(\mathcal{S},\ \mathcal{L})$
13: $\quad$**end if**
14: **end while**
15: **return** $\mathcal{S}$

The main loop of the algorithm (lines 4–14) is repeated so long as there are labels in $\mathcal{Q}$. In each iteration, a label $\mathcal{L}$ is removed from the queue. Line 6 checks whether the next propagation of $\mathcal{L}$ considers the same route r_k as the previous ones. If this is true, then two labels $\mathcal{L}^{\text{direct}}$ and $\mathcal{L}^{\text{break}}$ are created from $\mathcal{L}$ in accordance with the operations described in Section 4.3.

Both labels are pushed to the queue (lines 8–9) followed by a dominance test (explained in Section 4.7) to remove redundant labels from $\mathcal{Q}$ (line 10). However, if a route which differs from r_k would be modified in the next propagation, then label $\mathcal{L}$ is added to stack $\mathcal{S}$ instead of being propagated (line 12). Once $\mathcal{Q}$ becomes empty, the algorithm terminates and returns $\mathcal{S}$. Note that a label is only pushed to either $\mathcal{Q}$ (lines 8–9) or $\mathcal{S}$ (line 12) if it is *feasible* (see Section 4.5). An infeasible or dominated label is always excluded from the propagation.

While Algorithm 1 applies Depth-First Search (DFS) by employing a stack, Algorithm 2 applies Breadth-First Search (BFS) by employing a queue. The idea is to benefit from the problem structure, trying to propagate labels in one route using a BFS (Algorithm 2), which has been observed to perform better than its DFS-counterpart (Goel, 2010). On the other hand, our preliminary experiments showed that a DFS strategy performed better when moving across routes in the propagation by decreasing memory usage when solving large problem instances. Section 4.7 shows how this DFS-BFS search strategy reduces the complexity of dominance tests.

The performance of this search strategy is, however, highly dependent on the structure of the topological order. The order in $\mathcal{T}_G$ should contain *sequences of nodes belonging to the same route that are as long as possible* in order to benefit from the BFS phase (lines 5–10). Otherwise, if neighboring nodes in $\mathcal{T}_G$ always belong to a different route, then the LPA becomes a full DFS search, which tends to be inefficient for large instances. For this reason, we describe how to obtain an efficient topological order in Appendix A.

None of the aforementioned measures reduces the theoretical complexity of the LPA, however. They were simply observed to perform well.

4.5. Feasibility conditions

A label is considered *feasible* if it respects a number of conditions. Three of these are Conditions (20)–(22), which must be respected for all route indices $k = 1, \ldots, m$ considering their last scheduled node $\mathcal{L}(i_k)$, which we will refer to as i_k for simplicity.

$$\mathcal{L}(H_{i_k}) \leq l_{i_k} \tag{20}$$

$$\mathcal{L}(D_{i_k}) \leq M_{\text{drive}} \tag{21}$$

$$\mathcal{L}(W_{i_k}) \leq M_{\text{work}} \tag{22}$$

However, these conditions do not guarantee compliance with maximum route duration. Unfortunately, the presence of interdependencies makes the computation of route duration nontrivial. To guarantee compliance with maximum route duration constraints, we associate every label $\mathcal{L}$ with a *Simple Temporal Problem* (STP, Dechter et al. (1991)). Let us denote the STP associated with label $\mathcal{L}$ as STP($\mathcal{L}$). A solution to STP($\mathcal{L}$) provides valid start times for all drivers in the TDSP-IR instance. If no solution can

be found, the (partial) schedules in $\mathcal{L}$ are infeasible with respect to maximum route duration.

In the STP, a set of variables represents the moment in time when a given action takes place. An auxiliary variable is created to represent the origin of the time horizon $t = 0$. For simplicity, we will refer to this special variable as 0. For the TDSP-IR, we associate a variable h_i with every node $i \in V$ and variable $h_0 = 0$ with the beginning of the time horizon. Based on Masson et al. (2014), we define the associated STP($\mathcal{L}$) as:

$$h_i - h_j \leq -w_i - t_{ij} - \mathcal{L}(x_i) \cdot B, \quad \forall (i,j) \in A_R \tag{23}$$

$$h_{u_-} - h_{u_+} \leq -w_{u_-}, \quad \forall (u_-, u_+) \in A_P \tag{24}$$

$$h_0 - h_i \leq -e_i, \quad \forall i \in V \tag{25}$$

$$h_i - h_0 \leq l_i, \quad \forall i \in V \tag{26}$$

$$h_{d_k} - h_{o_k} \leq M_{\text{dur}}, \quad \forall o_k, d_k \in V_O \tag{27}$$

Due to break scheduling, every label produces a unique STP by means of Constraint (23). Dechter et al. (1991) showed that every STP can be written as a distance graph, where vertices are variables and arcs are the constraints between these variables. Thus, it is possible to solve the problem using shortest path algorithms that accommodate negative cycles. For example, the Bellman-Ford algorithm leads to a time complexity of $\mathcal{O}(n^2)$, where n is the number of nodes in the TDSP-IR instance. Dechter et al. (1991) showed other properties of the STP: (i) if there is a negative cycle then the problem is infeasible and (ii) if there is no negative cycle, then the STP is called *consistent* and its shortest paths provide a valid schedule. For the TDSP-IR, the resulting shortest distances in a consistent STP create valid starting times for all truck drivers' routes.

In summary, a label $\mathcal{L}$ is *feasible* if it respects Constraints (20)–(22) and STP($\mathcal{L}$) is consistent. This is ensured for every new label in the LPA to avoid propagating partial solutions which may already be proven infeasible. However, a complete TDSP-IR solution is only obtained once a label $\mathcal{L}$ reaches the last node in $\mathcal{T}_G$, in which case the starting time of service in each location is set as $\mathcal{L}(H_i) = h_i, \forall i \in V$.

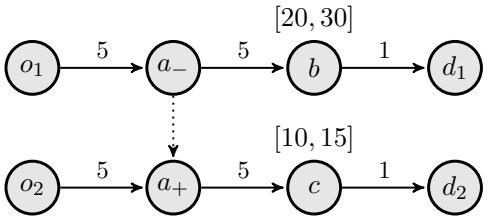
4.6. An example of the label propagation

The example in Figure 3 illustrates the execution of the algorithm, the feasibility conditions and the interdependent routes. Suppose values $M_{\text{dur}} = 16$, $M_{\text{drive}} = 7$ and $B = 1$. For simplicity, working time is ignored. Let us assume the propagation order of the LPA is $\mathcal{T}_G = (a_-, b, d_1, a_+, c, d_2)$. Figure 4 also depicts the execution of the label propagation using the proposed LPA by showing the labels relative to the two routes at every iteration.

Breaks at the origin nodes have no effect on the solution, so these are ignored. The label $\mathcal{L}^0$ starting in o_1 and o_2 (Figure 4(a)) is propagated to a_-

following the order $\mathcal{T}_G$, generating one label $\mathcal{L}$ which arrives at $\mathcal{L}(H_{a_-}) = 5$ (Figure 4(b)). A break must then be scheduled after a_- since a direct trip from a_- to b would exceed the maximum continuous driving time M_{drive}. This results in a single feasible label reaching node b at $\mathcal{L}(H_b) = 20$ (Figure 4(c)). Note the required waiting time enforced by the time window in b. Then, two labels would reach destination d_1, one with a break at a_- and another with an additional break at b (Figure 4(d)). For simplicity, we will ignore the second label because it has no impact on our analysis. This leads to a single label $\mathcal{L}$ reaching d_1 at time $\mathcal{L}(H_{d_1}) = 21$, break $\mathcal{L}(x_{a_-}) = 1$, nodes $\mathcal{L}(i_1) = d_1$, $\mathcal{L}(i_2) = o_2$ and position $\mathcal{L}(p) = 4$. Thus far STP($\mathcal{L}$) remains consistent because route r_1 can be postponed up to $\mathcal{L}(H_{o_1}) = 5$, making its total duration $\mathcal{L}(H_{d_1}) - \mathcal{L}(H_{o_1}) = 16 = M_{\text{dur}}$.

Figure 3: A simple TDSP-IR instance with two routes.

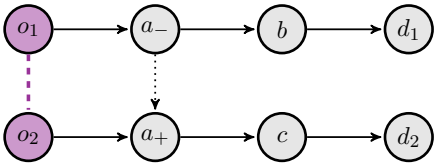


Next, by following the topological order, $\mathcal{T}_G[\mathcal{L}(p)] = a_+$, the propagation begins in route r_2 (Figure 4(e)). At this point in Algorithm 2, label $\mathcal{L}$ would be pushed to $\mathcal{S}$ (line 12), returning to Algorithm 1 where a new call to `label_enumeration` would be made to propagate in r_2.

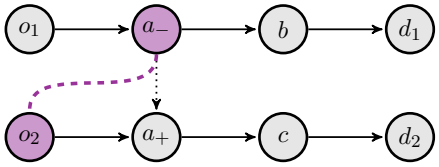
One label reaches a_+ with $\mathcal{L}(H_{a_+}) = 5$ (note that this label is simply the propagation of the label upon d_1 in r_2). Once again, a break is needed at a_+ for feasibility reasons, resulting in a single label $\mathcal{L}$ reaching node c at $\mathcal{L}(H_c) = 11$ (Figure 4(f)). Although this label might initially appear feasible, STP($\mathcal{L}$) is not consistent. This is due to the increase in elapsed time from a_+ to c which restricts the start time of service in route r_1 to be at most $\mathcal{L}(H_{o_1}) = 4$, so that service at c begins within the time window. However, the duration of r_1 then is at least $\mathcal{L}(H_{d_1}) - \mathcal{L}(H_{o_1}) = 17 > M_{\text{dur}}$ and thus an infeasible label.

In conclusion, the instance depicted in Figure 3 has no solution, but this is not immediately obvious given that it depends not only on the interaction between the two routes but also on the breaks that are scheduled in each route. As the number of nodes, routes and interdependencies increase, it becomes more challenging to decide the feasibility of an instance, thus requiring a dedicated solution method such as the proposed LPA.

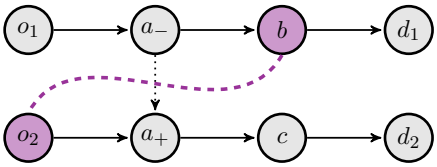
Figure 4: How labels propagate across the routes in the graph of Figure 3. The dashed purple line represents one label and its position with regard to each of the routes after every propagation.



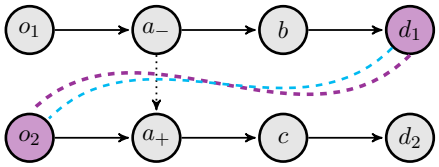
(a) The algorithm starts with $\mathcal{L}^0$ at the two origin locations: o_1 and o_2 respectively in routes r_1 and r_2.



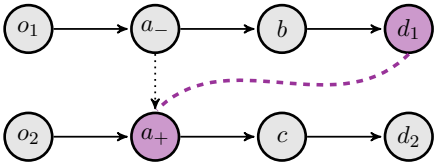
(b) By following $\mathcal{T}_G$, the first propagation changes the label position relative to only one route.



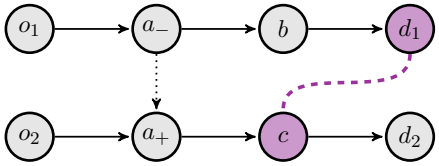
(c) A new propagation in route r_1 again modifies the label for this route only. For r_2, the label has not been changed yet.



(d) Here, the blue line represents the second label that would be created in the propagation (but for simplicity purposes it will be ignored).



(e) The propagation in route r_2 begins, following the order $\mathcal{T}_G$.



(f) The propagation reaches node c and proves the instance infeasible.

4.7. Dominance conditions

A label $\mathcal{L}$ dominates another label $\mathcal{L}'$ if Conditions (28)–(34) are satisfied. A precondition to decide dominance is that both labels must be propagated up to the same node in $\mathcal{T}_G$. In other words, $\mathcal{L}(p) = \mathcal{L}'(p)$. This implies $\mathcal{L}(i_k) = \mathcal{L}'(i_k),\ k = 1, \ldots, m$. For simplicity purposes, throughout the remainder of this section we will adopt i_k to denote the last scheduled node in route r_k for either $\mathcal{L}$ or $\mathcal{L}'$. All conditions must simultaneously be true *for all* routes $r_k, k = 1, \ldots, m$.

$$\mathcal{L}(H_{i_k}) \leq \mathcal{L}'(H_{i_k}) \tag{28}$$
$$\mathcal{L}(D_{i_k}) \leq \mathcal{L}'(D_{i_k}) \tag{29}$$

$$\mathcal{L}(W_{i_k}) \leq \mathcal{L}'(W_{i_k}) \tag{30}$$
$$\mathcal{L}(P_k) \geq \mathcal{L}'(P_k) \tag{31}$$
$$\mathcal{L}(S_{zi_k}) \geq \mathcal{L}'(S_{zi_k}), \quad \forall\ z \in \Phi(i_k) \tag{32}$$
$$\mathcal{L}(H_{u_-}) \leq \mathcal{L}'(H_{u_-}), \quad \forall\ u_- \in U_-(\mathcal{L}) \tag{33}$$
$$\mathcal{L}(S_{zu_-}) \geq \mathcal{L}'(S_{zu_-}), \quad \forall\ u_- \in U_-(\mathcal{L}),\ z \in \Phi(u_-) \tag{34}$$

where $\Phi(i)$ is the set defined in Section 4.2. Meanwhile, $U_-(\mathcal{L})$ is the set of all u_- locations that belong to a precedence pair $(u_-, u_+) \in A_P$ for which u_- has been scheduled in the partial solution $\mathcal{L}$, while u_+ has not been scheduled. When considering the example in Figure 3, $U_-(\mathcal{L}^0) = \emptyset$ for the initial label, but after propagating to a_- it becomes $U_-(\mathcal{L}) = \{a_-\}$ and remains like this until a_+ is processed, whereupon the set once again becomes $U_-(\mathcal{L}) = \emptyset$.

Conditions (28)–(31) are similar to the conditions for the TDSP (Goel, 2010). Condition (32) verifies slack with respect to all origins $o_z, z \in \Phi(i_k)$ that precede node i_k. This ensures that any postponement of r_z's start time has at most as much impact on the service time at i_k in $\mathcal{L}$ as in $\mathcal{L}'$.

Conditions (33) and (34) compare the values of labels at interdependencies which have not been fully scheduled yet. The intuition behind Condition (33) is that arriving earlier at u_- provides a greater degree of freedom to (i) postpone service at u_- without impacting u_+ (considering Equation 12) and (ii) choose a start time for service at u_+ without incurring additional waiting time at u_+'s route (considering Equation 13). Meanwhile, Condition (34) ensures that slack up to the interdependence points in $\mathcal{L}$ allow for a greater degree of freedom than in $\mathcal{L}'$ to postpone a route's starting time of service without affecting other routes connected together by these interdependence points. Conditions (33) and (34) are both used to avoid the removal of feasible solutions such as those illustrated in the examples in Appendices B.1 and B.2.

To improve efficiency, once a route r_k has been fully scheduled ($\mathcal{L}(i_k) = d_k$) we may drop Conditions (29) and (30) for route r_k. This is always true because Conditions (20)–(22) ensure label $\mathcal{L}$ is feasible in terms of working and driving time for route r_k. Since $\mathcal{L}(i_k) = d_k$, no further propagations will be performed in r_k and so values $\mathcal{L}(D_{i_k})$ and $\mathcal{L}(W_{i_k})$ will remain fixed. Thus, it is safe to drop Conditions (29) and (30) from the dominance tests for route r_k.

Then, Conditions (28)–(34) define label dominance. This is formally stated in Lemma 1.

Lemma 1. *Given two labels $\mathcal{L}$ and $\mathcal{L}'$ with partial schedules for which Conditions (28)–(34) are valid, then either $\mathcal{L}$ produces a feasible solution or neither $\mathcal{L}$ nor $\mathcal{L}'$ can produce a feasible solution.*

Proof. Appendix C provides the proof.

Finally, we will verify the asymptotic time complexity of the dominance test. Conditions (28)–(31) require a constant number of operations for each route index k, thus $\mathcal{O}(m)$ operations in total. Conditions (32), (33) and (34) require $\mathcal{O}(m^2)$, $\mathcal{O}(|A_P|)$ and $\mathcal{O}(m|A_P|)$ operations, respectively. Therefore, a single dominance test has time complexity $\mathcal{O}(m^2+m|A_P|)$. In the proposed LPA, dominance tests are only applied in Algorithm 2. This allows the quadratic term $\mathcal{O}(m^2)$ in Condition (32) to be reduced to $\mathcal{O}(m)$ since only slack values from origins o_z which precede nodes in r_k are changed in a call to Algorithm 2. Slack values to any nodes in routes r_z, $r_z \neq r_k$ remain unchanged for all labels during one procedure call and do not need to be tested. The complexity of the dominance test reduces to $\mathcal{O}(m|A_P|)$.

5. Complexity analysis

In this section, we turn our attention towards analyzing the worst-case time complexity of the LPA. We begin by presenting an analysis for the general definition of the TDSP-IR in Section 5.1. Then, in Section 5.2, we show how the complexity of the LPA may be reduced by dropping a single constraint from the original problem.

5.1. Complexity of the algorithm

For the full time complexity of the LPA, we first note that a topological sort may be obtained in strictly polynomial time (Cormen et al., 2009). Since other operations are asymptotically worse, we ignore the complexity of the sorting throughout the remainder of the analysis.

It is not clear how many labels can be dominated overall, or even in each single propagation. The presence of multiple interdependencies makes this analysis nontrivial. Thus, the best bound is the trivial one which considers the worst case: that no label is dominated and all labels must be considered while still comparing them all with one another to verify dominance.

There may be $\mathcal{O}(2^n)$ possible solutions for the TDSP-IR. Every propagation takes $\mathcal{O}(m)$ time and an STP consistency check takes $\mathcal{O}(n^2)$, thus in total $\mathcal{O}(2^n(m+n^2))$. We may have to compare each solution against all other solutions every iteration to verify dominance, which requires $\mathcal{O}(2^{2n})$ dominance tests. Each dominance test takes $\mathcal{O}(m|A_P|)$ using the DFS-BFS propagation scheme, resulting in $\mathcal{O}(2^{2n}m|A_P|)$. Hence, considering all steps, the asymptotic complexity of the LPA is $\mathcal{O}(2^n(m+n^2)+4^nm|A_P|)$, which can be simplified to $\mathcal{O}(4^nm|A_P|)$. Since $n \geq |A_P| \geq m-1$, we may also denote the LPA's final complexity as $\mathcal{O}(4^nn^2)$.

5.2. Complexity when disregarding maximum route duration

Abandoning compliance with maximum route duration may be interesting for applications where drivers are allowed to work for long shifts. For

example, if all routes are performed within a time horizon of $\mathcal{H} = 13$h (Kok et al., 2011; Goel, 2012) and the maximum duration permitted per route is $M_{\text{dur}} = 13$h, then the LPA only needs to decide where to schedule the breaks in each route to comply with customer time windows and interdependencies.

In this case, setting departure times at origin locations to $H_{o_k} = 0$ is always feasible. Variables $\mathcal{L}(S_{ki})$ and $\mathcal{L}(P_k)$ must be verified in dominance tests due to the presence of maximum route duration constraints. When these constraints can be ignored, it is possible to remove Conditions (31), (32) and (34) from the dominance tests.

The impact of this change cannot be underestimated. Eliminating the aforementioned conditions enables one to turn each label propagation into an $\mathcal{O}(1)$ time operation. Label dominance is reduced to verifying Conditions (28)–(30) and (33) with a total complexity of $\mathcal{O}(m+|A_P|)$. The structure of the LPA further reduces this complexity to $\mathcal{O}(|A_P|)$ (by applying dominance only in Algorithm 2). Additionally, it is not necessary to verify consistency with the STP, since no duration exists. The overall complexity is thus reduced to $\mathcal{O}(4^n|A_P|)$.

6. Computational experiments

Computational experiments are devised to evaluate the scalability of the LPA. Since there are no TDSP-IR instances available, we generated instances using an insertion heuristic for the PDPT executed over modified instances from Sampaio et al. (2020). The modifications are detailed in Appendix D.1. The resulting TDSP-IR instances had characteristics which lie within the following ranges: $4 \leq n \leq 300$, $1 \leq m \leq 19$ and $0 \leq |A_P| \leq 34$.

In accordance with EU regulations, parameter values are set as follows: $M_{\text{drive}} = 4$h30, $M_{\text{work}} = 6$h00, $M_{\text{daily}} = 9$h00 and $B = 0$h45. The time horizon $\mathcal{H} = 13$h is the same across all instances and all routes. To show the impact of maximum route duration when solving the problem, we ran experiments varying the shift length of the truck drivers so that they were assigned a short shift of $M_{\text{dur}} = 8$h21 (respecting the Belgian law specifying 7h36 of work plus a 45-minute break), a medium shift of $M_{\text{dur}} = 10$h or a long shift of $M_{\text{dur}} = 13$h (or no maximum duration).

Results of the LPA are compared against those of the MILP formulation, which was executed without an objective function and seeks only to generate a feasible schedule. The entire approach was implemented in `C++` and compiled using GNU `g++` version 7.5 in an Ubuntu 20.04 LTS operating system. Experiments were carried out on an Intel Xeon E5-2660 at 2.6 GHz with 160 GB of RAM. The MILP was implemented using the `C++` API of Gurobi 9 with default settings and single-thread mode.

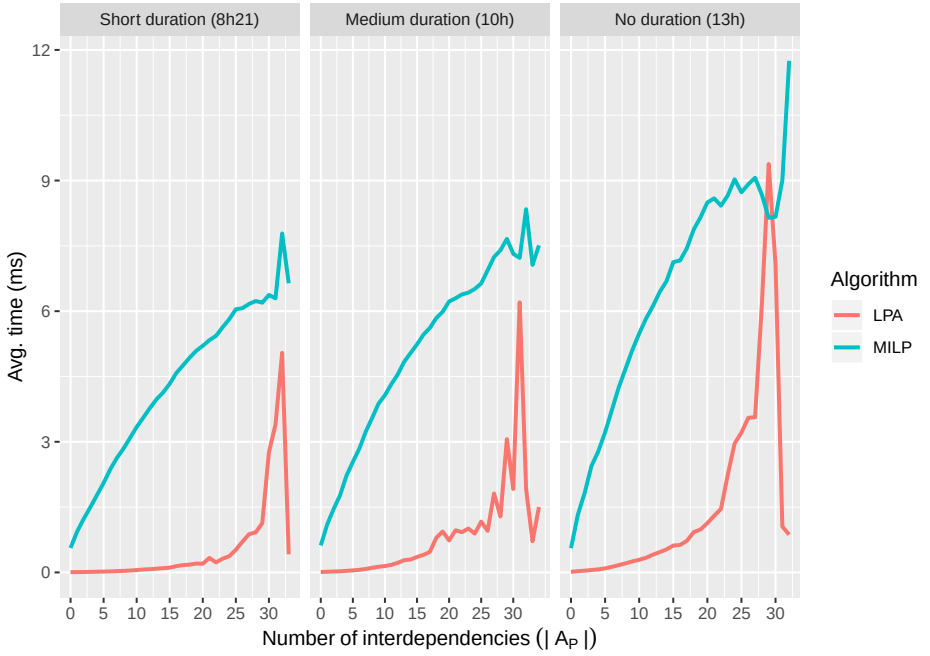
First, we inspect the performance of the algorithms over all instances in Section 6.1. The results considering only feasible TDSP-IR instances are re-

ported in Section 6.2. Finally, an analysis of a naive algorithm's performance is presented in Section 6.3.

6.1. Performance over all instances

Figure 5 illustrates the execution times in milliseconds (ms) for both the LPA and the MILP for each shift length. The execution times are the average per run according to the total number of interdependencies in a TDSP-IR instance: the cardinality of set A_P, which influences many of the asymptotic operation complexities for the LPA and is directly related to the number of routes and nodes (see Appendix D.2). Over all PDPT instances, a total of 67 million executions of the LPA and MILP were performed. Appendix D.3 provides detailed results for these experiments.

Figure 5: Average computation times of the LPA and MILP.



The experiments indicate that the LPA almost always outperforms the MILP. This is particularly true for instances with $|A_P| \leq 25$, for which differences in average computation times significantly favor the LPA. For instances with $|A_P| \geq 25$ the execution times present a greater variation for the LPA, often incurring an increase in average computation times. This is not surprising, since an increase in the cardinality of set A_P is associated with an increase in the number of nodes or the number of routes in the TDSP-IR instance.

Additionally, varying the maximum route duration has an impact on the average execution time. Both the MILP and LPA require more time to

solve TDSP-IR instances as the maximum duration increases. At first, this may seem counter-intuitive given that we showed in Section 5.2 that the worst-case complexity of the LPA reduces when the duration constraints are dropped. However, the number of feasible positions to schedule breaks increases with duration, since routes become less temporally constrained. Similarly, the number of nodes that can feasibly fit into a TDSP-IR instance is larger, further increasing the number of possible break locations and therefore the difficulty of solving these instances.

In terms of total execution time to solve all 67 million instances, the LPA required 148 minutes, while the MILP took 2840 minutes – almost 20 times longer than the LPA. This further confirms that if the purpose of these scheduling algorithms is to iteratively solve multiple instances of the TDSP-IR in a VRP heuristic similar to (Goel & Vidal, 2013), using the LPA will provide a major advantage concerning the VRP heuristic's observed computation time when compared to using the MILP approach.

Even though we report results for instances with $|A_P| \geq 25$, we should consider how realistic instances containing multiple routes connected by 25–30 interdependent tasks are. This is not only a theoretical challenge, but one of major practical consequences given that any delay or disruption with respect to one customer's service may initiate significant cascading effects that render all routes infeasible. Indeed, our own preliminary experiments with real-world data have shown that a Belgian transportation company rarely has more than six routes connected by $|A_P| \leq 10$ interdependencies.

In summary, for applications that impose a limit of $|A_P| \leq 25$ (for practical reasons), our results show the LPA is a safe choice. However, even when this limit may not exist the LPA is expected to present better runtime performance on average than the MILP approach.

6.2. Performance over feasible instances

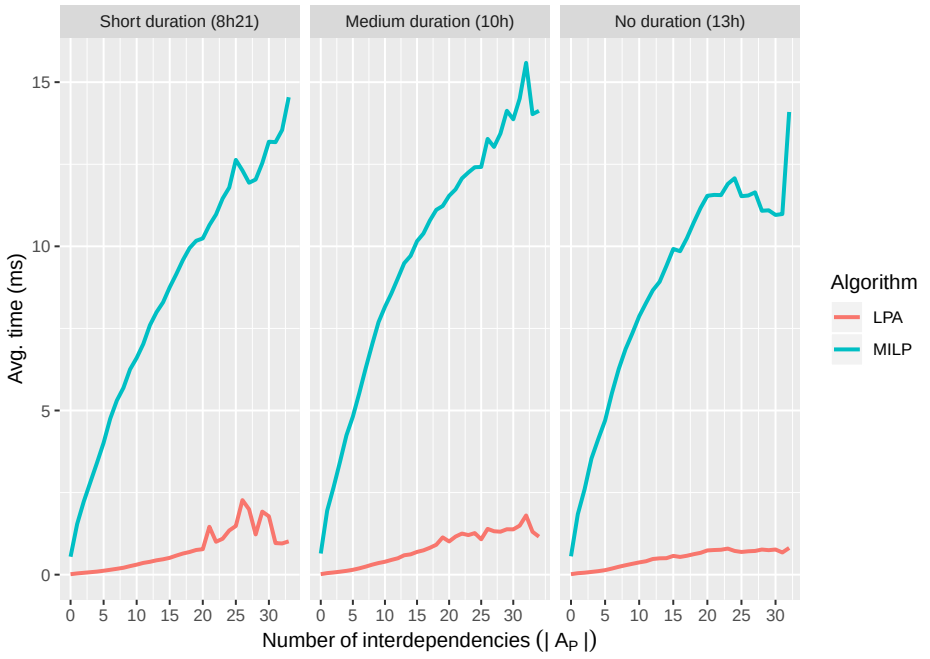
Some intriguing behavior is encountered when exploring the performance of the two algorithms for only the feasible TDSP-IR instances. The average execution time for these instances is shown in Figure 6. The LPA always requires substantially shorter computation times to prove the feasibility compared to proving the infeasibility of an instance. By contrast, the MILP demonstrates the opposite behavior. As a result, the execution time of the LPA exhibits slower growth, while execution times for the MILP exhibit a more steep growth compared to those in Figure 5. Moreover, the graphs show that route duration has less of an impact on the performance for feasible TDSP-IR instances, although some variation can still be observed. Appendix D.3 provides detailed results for these experiments.

In terms of the total computation time, the LPA solved all feasible TDSP-IR instances in 56 minutes, while the MILP required 1031 minutes. Therefore, the LPA remains almost 20 times faster than the MILP.

The computational effort required by the LPA depends on the number of propagated labels. Meanwhile, the number of labels that need to be propagated to prove infeasibility depends not only on the size of the instance, but also on the topological order. If two nodes which create a conflict are at the beginning of $\mathcal{T}_G$, then infeasibility is detected quickly. However, when these nodes are located towards the end of $\mathcal{T}_G$, the number of propagations can grow significantly, increasing the computational burden. Indeed, all instances which require long computation times of the LPA contain hundreds of nodes and $|A_P| \geq 18$ interdependencies. Thus, there is a large probability that conflicts will take many propagations to be detected due to the long topological order in these instances.

Note that identifying the conflicting nodes and deciding the best topological order in advance are nontrivial tasks and remain as an open question. Meanwhile, proving feasibility only requires finding one solution and can be achieved much faster by the LPA, as demonstrated by Figure 6.

Figure 6: Average execution times of the LPA and MILP for feasible TDSP-IR instances.



6.3. Comparison to a naive scheduling method

To understand both the importance of considering the interdependencies when scheduling breaks and the TDSP-IR instances' difficulty, we have implemented a naive algorithm inspired by Goel (2009). In contrast to the LPA, which is a multi-label algorithm, the naive approach uses only one

label and schedules each route independently with breaks as late as possible. Once a schedule for each route has been generated, feasibility is verified using the STP described in Section 4.5.

Table 3 shows the percentage of feasible solutions found by the naive method compared to both the LPA and the MILP (both of which find 100% of the solutions). The accuracy ratio is based on all feasible instances for which $|A_P| \leq q$, for a given value $q \geq 0$. The single-route TDSP is encountered when $q = 0$, while $q \geq 1$ refers strictly to TDSP-IR instances. Results are grouped according to their maximum duration M_{dur} as this parameter has some influence over the degree of difficulty of the instances. Given that the execution times of the naive algorithm were always below 0.1 ms, we do not report them in the table.

Table 3: Ratio (%) of feasible solutions found by the scheduling algorithms.

		Duration M_{dur}		
Algorithm	$\lvert A_P \rvert$	8h21	10h	13h
LPA/MILP	all	100.00	100.00	100.00
Naive	$= 0$	81.59	84.62	86.81
	≤ 1	77.67	80.16	82.07
	≤ 2	75.21	76.56	78.61
	≤ 3	73.31	74.05	75.56
	≤ 4	71.61	71.28	73.10
	≤ 5	69.98	68.95	70.46
	≤ 30	52.06	44.63	39.67

The ratios for the single-route case ($|A_P| = 0$) are similar to those reported by Goel (2010) for routes containing between 6 and 7 customers. The average number of locations in our instances was 12, but the scheduling was also arguably simpler by considering only one working day. Therefore, the results for the single-route case are within the expected ratios, and even in this simple case the naive approach misses many feasible solutions.

Once instances with one interdependent task are considered ($|A_P| = 1$), the ratios of the naive method already drop by 4%. Further increases in the size of A_P always decrease the solution ratios of the naive approach. When accounting for the more complex and large instances ($A_P \leq 30$), the ratios drop to no more than 52%, but reach as little as 39% of the feasible solutions found. These ratios clearly demonstrate how problematic it is to ignore interdependencies when scheduling the drivers' HOS.

Additionally, for small sizes of $|A_P|$ it is easier for the naive algorithm to find solutions for longer time horizons due to routes which are less restricted when it comes to positioning breaks. However, once the size exceeds the $|A_P| \geq 4$ threshold, the naive algorithm finds fewer feasible solutions when the time horizon increases as this is also related to an increase in the number of nodes and routes requiring the careful scheduling of breaks.

Finally, the results in Table 3 demonstrate the importance of considering the interplay between interdependencies and the scheduling of breaks for truck drivers. If the algorithms described in this paper are to be used in VRP solvers, it is undesirable to employ feasibility tests that miss 20% of the feasible solutions ($A_P = 0$), let alone 50% of them. Therefore, it is worth considering that the LPA presented a good trade-off between execution time and feasibility ratio and can thus be used to quickly answer decision questions regarding TDSP-IR instances for real-world applications.

7. Conclusion and future work

Research concerning the Truck Driver Scheduling Problem had, until now, mainly focused on the scheduling of independent routes. This sets limitations for real-world applications since solution techniques for problems such as the Pickup and Delivery Problem with Transshipments could not effectively guarantee compliance with hours of service regulations due to the presence of interdependent routes.

The research reported in this paper has successfully closed this gap by introducing a Mixed Integer Programming formulation and a Label Propagation Algorithm capable of scheduling interdependent routes while respecting a set of EU regulations for truck drivers. One major advantage of the proposed Label Propagation Algorithm is that it can be easily applied to any vehicle routing solver that addresses minimum difference interdependencies.

Many interesting scientific challenges remain open. An obvious question relates to the theoretical complexity of the Truck Driver Scheduling Problem with Interdependent Routes. Is the problem solvable in polynomial time? Additionally, are there speedup techniques for the search or faster ways to prove an instance infeasible? Is it possible to quickly solve an optimization version of the problem? What is the impact of including split breaks, rest periods and other EU-related regulations?

From a broader perspective, one may consider the impact that hours of service regulations have on the use of transfer locations and cross-docks in logistic applications. It should be expected that within a longer time horizon, it is preferable for routes of limited duration to utilize transfers to partially serve more requests in order to make the most of otherwise useless trips. However, this remains an open research topic.

Acknowledgments

This research is part of the KU Leuven Institute for AI (Leuven.AI). The project received financial support from the Strategic Basic Research project Data-driven logistics (S007318N) funded by the Research Foundation Flanders (FWO), and from the Flemish Government under the Artificiële

Intelligentie (AI) Vlaanderen programme. Editorial consultation provided by Luke Connolly (KU Leuven).

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT press.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*, 61 – 95.

Dohn, A., Rasmussen, M. S., & Larsen, J. (2011). The vehicle routing problem with time windows and temporal dependencies. *Networks*, *58*, 273–289.

Drexl, M. (2013). Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research*, *227*, 275 – 283.

EU Commission (2002). Directive 2002/15/ec on the organisation of the working time of persons performing mobile road transport activities. Official J. Eur. Commun. L 80 3539. Available online at: `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32002L0015`.

EU Commission (2006). Regulation (ec) no. 561/2006 on the harmonisation of certain social legislation relating to road transport and amending council regulations (eec) no 3821/85 and (ec) no. 2135/98 and repealing council regulation (eec) no. 3820/85. Official J. Eur. Union L 102, 11.04.2006. Available online at: `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32002L0015`.

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

Goel, A. (2010). Truck driver scheduling in the european union. *Transportation Science*, *44*, 429–441.

Goel, A. (2012). The minimum duration truck driver scheduling problem. *EURO Journal on Transportation and Logistics*, *1*, 285–306.

Goel, A., Archetti, C., & Savelsbergh, M. (2012). Truck driver scheduling in australia. *Computers & Operations Research*, *39*, 1122 – 1132.

Goel, A., & Kok, L. (2012). Truck driver scheduling in the united states. *Transportation Science*, *46*, 317–326.

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

Goel, A., & Vidal, T. (2013). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science*, *48*, 391–412.

Kleff, A. (2019). *Scheduling and Routing of Truck Drivers Considering Regulations on Drivers' Working Hours*. Ph.D. thesis Karlsruher Institut für Technologie (KIT). doi:10.5445/IR/1000097855.

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

Leung, J. Y. (2004). *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press.

Masson, R., Lehud, F., & Pton, O. (2013). Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, *41*, 211 – 215.

Masson, R., Lehud, F., & Pton, O. (2014). The dial-a-ride problem with transfers. *Computers & Operations Research*, *41*, 12 – 23.

Mitrović-Minić, S., & Laporte, G. (2006). The pickup and delivery problem with time windows and transshipment. *INFOR: Information Systems and Operational Research*, *44*, 217–227.

Sampaio, A., Savelsbergh, M., Veelenturf, L. P., & Van Woensel, T. (2020). Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers. *Networks*, *76*, 232–255.

Van Belle, J., Valckenaers, P., & Cattrysse, D. (2012). Cross-docking: State of the art. *Omega*, *40*, 827–846.

APPENDICES

A. Dedicated topological order

This appendix describes the topological sorting algorithm applied in the `toposort` function from Algorithm 1. The idea is to benefit from the structure of the problem to achieve shorter execution times for the LPA, while having no proven impact on the theoretical worst-case complexity of the algorithm.

A.1. Impact of topological order

To understand the impact of a specific topological order, consider the instance with $m = 2$ routes depicted in Figure 7. Assume that $M_{\text{drive}} = 7$, $B = 1$ and consider $\mathcal{T}_G^1 = (a, b_-, c, e, b_+, f, d_2, d_1)$. For simplicity, route duration and working times are ignored.

Algorithm 1 starts with initial label $\mathcal{L}^0$ and calls Algorithm 2 which in turn uses the `label_enumeration` function. This function follows $\mathcal{T}_G^1$ and propagates labels in route r_1 until node c is reached. The next propagation (to e) refers to route r_2, in which case Algorithm 2 terminates and returns to Algorithm 1 for the next iteration. Thus, upon scheduling c there are three *non-dominated* labels in the expansion – $\mathcal{L}^1$, $\mathcal{L}^2$ and $\mathcal{L}^3$ – which have no break scheduled, a break after a, and a break after b_-, respectively. Table 4 reports the values of these labels. Note that the expansion up to c generated four labels, but the label with two breaks (after a and b_-) is dominated by $\mathcal{L}^3$ (and thus removed from the search).

As for the other dominance tests, label $\mathcal{L}^1$ cannot dominate any other label because it requires more driving time. Label $\mathcal{L}^2$ has less available slack and push than the others, but visits c earlier than $\mathcal{L}^3$. Finally, label $\mathcal{L}^3$ visits c later than any other label, but with the least driving time.

Continuing the propagation, each of the three labels previously expanded up to c in route r_1 are now expanded from $\mathcal{L}^1(i_2) = \mathcal{L}^2(i_2) = \mathcal{L}^3(i_2) = o_2$ until the destination node d_2. The propagation generates three non-dominated labels (break at e, break at b_+, and break at both) *for each* one of the three labels up to d_1 (enumerating the valid combination of breaks in every route). Note that no break at all is infeasible. Thus, the total number of labels upon reaching d_2 is nine.

Now consider a different topological order, such that d_1 is reached before the propagation in r_2 begins, that is, $\mathcal{T}_G^2 = (a, b_-, c, d_1, e, b_+, f, d_2)$. The same three non-dominated labels are generated up to node c. Expanding them up to d_1 generates a total of six new labels (for each of the three labels up to c, it either breaks after c or has a direct trip to d_1). However, expanding $\mathcal{L}^1$ and $\mathcal{L}^2$ without a break after c are both infeasible, thus only four feasible and consistent labels remain. Let us denote these labels $\mathcal{L}^4$

(break at c), $\mathcal{L}^5$ (break at b_-), $\mathcal{L}^6$ (break at a and c), and $\mathcal{L}^7$ (break at b_- and c). Table 5 reports their values.

Figure 7: Instance to demonstrate the differences incurred by topological order.

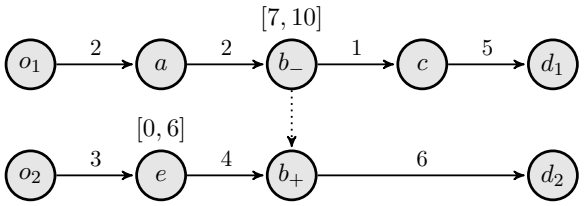


Table 4: Three labels in the expansion with $\mathcal{T}_G^1$ up to c.

Lab.	p	i_1	i_2	H_c	D_c	P_1	S_{1c}
$\mathcal{L}^1$	4	c	o_2	8	5	6	3
$\mathcal{L}^2$	4	c	o_2	8	3	5	2
$\mathcal{L}^3$	4	c	o_2	9	1	6	3

Table 5: Four labels in the expansion with $\mathcal{T}_G^2$ up to d_1.

Lab.	p	i_1	i_2	H_{d_1}	D_{d_1}	P_1	S_{1d_1}
$\mathcal{L}^4$	5	d_1	o_2	14	1	6	3
$\mathcal{L}^5$	5	d_1	o_2	14	6	6	3
$\mathcal{L}^6$	5	d_1	o_2	14	5	5	2
$\mathcal{L}^7$	5	d_1	o_2	15	5	6	3

Looking closely at dominance tests, we note that label $\mathcal{L}^4$ dominates $\mathcal{L}^5$, $\mathcal{L}^6$ and $\mathcal{L}^7$. Therefore, when the propagation from o_2 until d_2 begins for route r_2, only one non-dominated label remains ($\mathcal{L}^4$). As with the first topological order, upon reaching d_2 three non-dominated schedules were generated per label that reached, in this case, d_1. Since there was only one label at d_1, the total number of labels upon reaching d_2 is three. The reduction is therefore two-thirds when compared to the first topological order.

A.2. Procedure for the dedicated topological order

To obtain a topological order $\mathcal{T}_G$, we propose the following procedure. Associate with each route r_k a value δ_k that contains the total number of u_+ nodes belonging to route r_k (recall that by definition u_+ belongs to a pair $(u_-, u_+) \in A_P$). Furthermore, let $|r_k|$ denote the number of locations in the k-th route and ρ_k a penalty value, which is initialized with a value of zero for all routes.

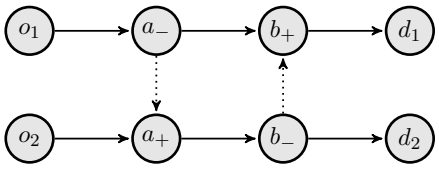
Our procedure begins by sorting routes in ascending order of δ_k. Ties are broken by descending order of $|r_k|$. Then, the procedure takes the first route after sorting, denoted r_z (the route with the smallest δ_z), and adds as many nodes as possible from this route to the topological order $\mathcal{T}_G$. Nodes

from r_z are no longer included either when d_z is added to the order (the route ends) or when a node u_+ is reached for which the corresponding u_- has not been added to the order. In the latter case, ρ_z is increased by one unit. Whenever a u_- node is included in $\mathcal{T}_G$, take the route r_y of the corresponding u_+ node and decrease the values δ_y and ρ_y by one unit. Once nodes of a route cannot be added to the order, re-sort routes in ascending order of ρ_k, then ascending order of δ_k, and finally descending order of $|r_k|$. When a destination d_k is added to $\mathcal{T}_G$, route r_k is no longer considered.

In this way, whenever a route r_z has no unresolved dependencies ($\delta_z = 0$) it is preferred for inclusion in the topological order. By adding as many nodes of only one route as possible, many routes will be completed one after the other, thereby avoiding back-and-forth propagations between different routes. Meanwhile, penalties ρ are used to avoid cases in which a mutual dependency could cause an infinite loop in the sorting function, such as in Figure 8.

This example shows the reasoning behind penalty ρ_k when using the specific `toposort` described in the paper. A mutual dependency is depicted in Figure 8, where $\delta_1 = 1$ and $\delta_2 = 1$. Sorting these routes has two possible outcomes. The first case begins with r_1, which can add nodes to the order up to a_- (inclusive), reducing $\delta_2 = 0$. Re-sorting the routes makes r_2 the next to be processed. Finally, adding all nodes in r_2 and then completing r_1 ends the topological order. Alternatively, if r_2 is the first route, the procedure can only add o_2 before reaching an unresolved dependency. Re-sorting routes without penalty ρ_k does not change anything since no δ_k value was modified. If a penalty is included in the sorting and increased once a_+ is reached, route r_1 will be preferred after the re-sorting step.

Figure 8: An instance depicting mutual dependencies.



The proposed sorting is clearly more complex than a simple topological sorting. However, its computational cost is compensated for by the observed performance gains. In terms of runtime complexity, the initial sorting of routes takes $\mathcal{O}(m \log m)$ with a standard sorting algorithm. In the worst case, the algorithm may have to re-sort routes after each node insertion, thus leading to an overall complexity of $\mathcal{O}(nm \log m)$.

B. Detailed examples

Sections B.1 and B.2 present examples to help justify label dominance Conditions (33) and (34). The variables used are those listed in Table 2

unless otherwise stated.

B.1. Example for dominance Condition (33)

To help clarify Condition (33), consider Figure 9. Suppose $M_{\text{drive}} = 15$, $M_{\text{dur}} = 30$, $B = 1$, and $\mathcal{T}_G = (a, b, c_-, e, d_1, f, c_+, d_2)$. A break is required in r_1 at some point before c_-. When scheduling r_1, the algorithm reaches node e with two labels $\mathcal{L}^1$ and $\mathcal{L}^2$. For $\mathcal{L}^1(x_a) = 1$ (break after a), whereas for $\mathcal{L}^2(x_b) = 1$ (break after b). The variables for both labels when $\mathcal{L}^1(i_1) = \mathcal{L}^2(i_1) = e$ are detailed in Table 6.

Figure 9: A TDSP-IR instance highlighting the necessity for Condition (33).

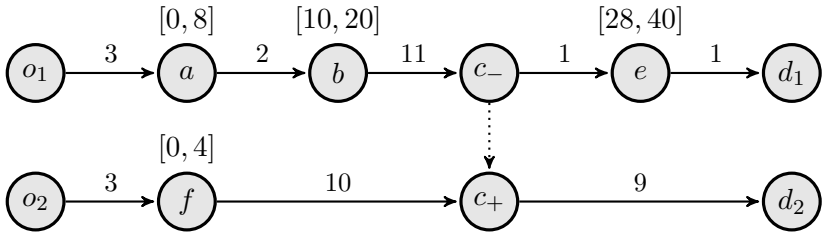


Table 6: Two non-dominated labels at node e for the example in Figure 9

Lab.	p	i_1	i_2	H_e	D_e	P_1	S_{1e}
$\mathcal{L}^1$	4	e	o_2	28	14	5	10
$\mathcal{L}^2$	4	e	o_2	28	12	5	10

All variables are identical, except for D_e. A break after b appears to be the best schedule since it serves e at the same time for both $\mathcal{L}^1$ and $\mathcal{L}^2$ and keeps the same P_1 and S_{1e} values, but with *less* driving time. Without further investigation, one could simply say that if $\mathcal{L}^1$ is feasible, then $\mathcal{L}^2$ must also be feasible, and it is therefore only necessary to propagate $\mathcal{L}^2$.

However, let us consider what happens when propagation continues until the end of $\mathcal{T}_G$. Normally, the propagation from e to d_1 would generate two labels for each label upon e, making for a total of 4 labels. Since it is unnecessary to schedule a break after e, we ignore the labels for which $x_e = 1$ and consider only labels $\mathcal{L}^1$ and $\mathcal{L}^2$ with a direct trip from e to d_1. After that, further propagations continue in r_2.

A break is required after c_+ to comply with driving regulations M_{drive}. Considering label $\mathcal{L}^1$, route r_2 will have a waiting time of 7 at c_+ because $\mathcal{L}^1(H_{c_-}) = 21$. Meanwhile, for $\mathcal{L}^2$ there is a waiting time of 8 units because $\mathcal{L}^2(H_{c_-}) = 22$. Note that including the break after c_+ as required, that is setting $\mathcal{L}^1(x_{c_+}) = \mathcal{L}^2(x_{c_+}) = 1$ changes the arrival times $\mathcal{L}^1(H_{d_2}) = 31$ and $\mathcal{L}^2(H_{d_2}) = 32$. Due to node f, the driver in route r_2 cannot depart from o_2 later than at time $H_{o_1} = 1$ and therefore the duration of route r_2 in label $\mathcal{L}^1$

is feasible, while in $\mathcal{L}^2$ it is not. In other words, while STP($\mathcal{L}^1$) is consistent, STP($\mathcal{L}^2$) has a negative cycle and is thus inconsistent.

The problem in this example is that $\mathcal{L}^1(H_e) = \mathcal{L}^2(H_e)$. In other words, the two drivers begin service at node e at the same time, due to the waiting time at e. The only difference between variables in $\mathcal{L}^1$ and $\mathcal{L}^2$ is the start time of service at node c_-. However, upon scheduling e, the information from c_- would not be considered anymore in dominance tests unless we include Condition (33).

B.2. Example for dominance Condition (34)

By way of example, consider Figure 10. Suppose $M_{\text{drive}} = 12$, $M_{\text{dur}} = 15$, $B = 1$ and $\mathcal{T}_G = (a_-, d_1, b, a_+, c_-, e_-, d_2, c_+, e_+, f, d_3)$. It is self-evident that no break is required in route r_1, while a break is required in r_2 at either b, a_+, c_- or e_-. Let $\mathcal{L}^1$ be the label with a break scheduled after b, $\mathcal{L}^2$ the label with a break at a_+, $\mathcal{L}^3$ the label with a break at c_-, and $\mathcal{L}^4$ the label with a break at e_-. Consider their expansion up to d_2. Table 7 reports the values for each label.

Figure 10: A TDSP-IR instance showing the need for Condition (34).

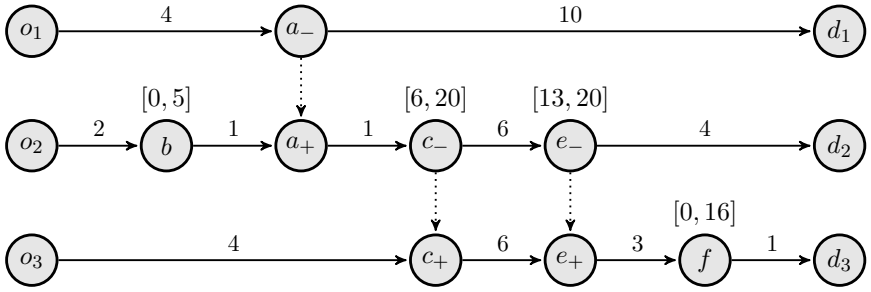


Table 7: Labels active upon reaching node d_2.

Lab.	p	i_1	i_2	i_3	H_{d_2}	D_{d_2}	P_1	P_2	S_{2c_-}	S_{2d_2}	S_{1d_2}
$\mathcal{L}^1$	7	d_1	d_2	o_3	17	12	9	3	1	2	2
$\mathcal{L}^2$	7	d_1	d_2	o_3	17	11	8	3	1	2	1
$\mathcal{L}^3$	7	d_1	d_2	o_3	17	10	8	3	2	2	1
$\mathcal{L}^4$	7	d_1	d_2	o_3	18	4	9	3	2	3	2

At first, the only valid dominance is $\mathcal{L}^3$ over $\mathcal{L}^2$, because $\mathcal{L}^3$ requires less driving time up to d_2 while the remaining values are the same as for $\mathcal{L}^2$. Thus, $\mathcal{L}^2$ is no longer considered. As for the other labels, $\mathcal{L}^3$ does not dominate $\mathcal{L}^1$ because it has less slack S_{1d_2}. Label $\mathcal{L}^4$ cannot dominate the other two labels because it arrives later at d_2.

Due to time windows, all labels serve requests c_- and e_- at the same time: $\mathcal{L}^1(H_{c_-}) = \mathcal{L}^3(H_{c_-}) = \mathcal{L}^4(H_{c_-}) = 6$ and $\mathcal{L}(H_{e_-})^1 = \mathcal{L}(H_{e_-})^3 =$

$\mathcal{L}(H_{e_-})^4 = 13$. However, if we ignore values D_{d_2} (because route r_2 has been concluded), there are two new dominances: $\mathcal{L}^1$ over $\mathcal{L}^3$ and $\mathcal{L}^1$ over $\mathcal{L}^4$. Note that $\mathcal{L}^1$ over $\mathcal{L}^3$ is only valid if Condition (34) is not included in the dominance test. Moreover, label $\mathcal{L}^4$ was only removed by ignoring variable D_{d_2}, which would otherwise be propagated unnecessarily until the end.

Consider the expansion of r_3, which can only have a break scheduled after c_+ (a break after e_+ is infeasible due to the time window at f). After expanding label $\mathcal{L}^1$ (the only one remaining), we discover that STP($\mathcal{L}^1$) is not consistent, because route r_2 needs to be postponed such that service at $\mathcal{L}^1(H_{c_-}) = 7$. However, this would require service at $\mathcal{L}^1(H_f) = 16 > l_f$. Therefore, the conclusion would be that this instance is infeasible.

If we include Condition (34), label $\mathcal{L}^3$ is not removed. Indeed, following the same steps as before shows that $\mathcal{L}^3$ is consistent because route r_2 is feasible with $\mathcal{L}^3(H_{c_-}) = 6$.

C. Proof of Lemma 1

This Appendix presents the detailed proof that the label dominance Conditions (28)–(34) only remove labels for which another label exists that can produce a feasible schedule, should one exist. The variables used are those in the main body of the paper, unless otherwise stated. For simplicity, we denote the latest scheduled node in any route r_k ($\mathcal{L}(i_k) = \mathcal{L}'(i_k)$) as i_k.

Proof. Let us define two labels $\mathcal{L}$ and $\mathcal{L}'$. As an initial hypothesis, suppose $\mathcal{L}$ dominates $\mathcal{L}'$. Then, we must show that any future modification in $\mathcal{L}'$ will impact all the m schedules at least as much as $\mathcal{L}$.

Consider any route index $k = 1, \ldots, m$. Conditions (28)–(31) assure that: $\mathcal{L}(H_{i_k}) \leq \mathcal{L}'(H_{i_k}), \mathcal{L}(D_{i_k}) \leq \mathcal{L}'(D_{i_k}), \mathcal{L}(W_{i_k}) \leq \mathcal{L}'(W_{i_k})$, and $\mathcal{L}(P_k) \geq \mathcal{L}'(P_k)$. Suppose $\mathcal{L}(H_{i_k}) = \mathcal{L}'(H_{i_k})$. If $\mathcal{L}'$ can create a feasible schedule for r_k, then $\mathcal{L}$ must also create a feasible schedule for this route since it requires fewer resources (D_{i_k} and W_{i_k}) to reach the same location (i_k) at the same time (H_{i_k}), while having more opportunities to postpone the start time (P_k). This is also true when $\mathcal{L}(H_{i_k}) < \mathcal{L}'(H_{i_k})$ since it is possible to purposefully postpone the service at i_k in label $\mathcal{L}$ such that $\mathcal{L}(H_{i_k}) = \mathcal{L}'(H_{i_k})$.

For Condition (32), we verify that all slack from origins o_z, $z \in \Phi(i_k)$ up to i_k are at least as large for $\mathcal{L}$ as they are for $\mathcal{L}'$ to ensure compliance with maximum route duration. This guarantees that any postponement to the start time of a route in $\mathcal{L}$ will impact other routes at least as much as the same postponement in $\mathcal{L}'$.

Suppose the start time of service at origin o_z has to be shifted forward in time by α_z. In other words, the start times at o_z, $\mathcal{L}(H_{o_z})$ and $\mathcal{L}'(H_{o_z})$ will be modified. This shift causes the following variable updates:

$$\mathcal{L}(H_{o_z}) \leftarrow \mathcal{L}(H_{o_z}) + \alpha_z$$

$$\mathcal{L}'(H_{o_z}) \leftarrow \mathcal{L}'(H_{o_z}) + \alpha_z$$

which implies $\mathcal{L}(H_{i_k})$ in route r_k will be shifted because origin o_z precedes node i_k ($z \in \Phi(i_k)$). The incurred shift is given by:

$$\mathcal{L}(H_{i_k}) \leftarrow \mathcal{L}(H_{i_k}) + \max\{0, \alpha_z - \min\{\mathcal{L}(P_z), \mathcal{L}(S_{zi_k})\}\}$$
$$\mathcal{L}'(H_{i_k}) \leftarrow \mathcal{L}'(H_{i_k}) + \max\{0, \alpha_z - \min\{\mathcal{L}'(P_z), \mathcal{L}'(S_{zi_k})\}\}$$

Define $\mu_{i_k} = \max\{0, \alpha_z - \min\{\mathcal{L}(P_z), \mathcal{L}(S_{zi_k})\}\}$ as the shift in $\mathcal{L}(H_{i_k})$ (analogously for $\mathcal{L}'$ we define μ'_{i_k}). Then, if $\mu_{i_k} \leq \mu'_{i_k}$, it follows that $\min\{\mathcal{L}(P_z), \mathcal{L}'(S_{zi_k})\} \geq \min\{\mathcal{L}'(P_z), \mathcal{L}(S_{zi_k})\}$. From the initial hypothesis that $\mathcal{L}$ dominates $\mathcal{L}'$, it is true that $\mathcal{L}(P_k) \geq \mathcal{L}'(P_k)$ and that $\mathcal{L}(S_{zi_k}) \geq \mathcal{L}'(S_{zi_k})$, thus the inequality $\min\{\mathcal{L}(P_z), \mathcal{L}(S_{zi_k})\} \geq \min\{\mathcal{L}'(P_z), \mathcal{L}'(S_{zi_k})\}$ holds. Since α_z is a constant value, the resulting shift in i_k must respect the relation $\mathcal{L}(H_{i_k}) + \mu_{i_k} \leq \mathcal{L}'(H_{i_k}) + \mu'_{i_k}$. Alternatively, $\mu_{i_k} > \mu'_{i_k}$ would imply that $\min\{\mathcal{L}(P_z), \mathcal{L}(S_{zi_k})\} > \min\{\mathcal{L}'(P_z), \mathcal{L}'(S_{zi_k})\}$. Thus, either Condition (31) or Condition (32) fails and no dominance can be established, contradicting our initial hypothesis.

Condition (33) similarly checks the impact that a postponement on a service u_- in route r_k may have on route r_z ($r_k \neq r_z$) which has not been scheduled up to the corresponding u_+ (for $(u_-, u_+) \in A_P$). In other words, there is no information concerning the slack, push, or service times at u_+. Since service time $\mathcal{L}(H_{u_+})$ may be defined by either its direct predecessors or by $\mathcal{L}(H_{u_-})$, two distinct scenarios arise when analyzing this condition.

First, suppose that upon evaluating node u_+ the algorithm finds the arrival time at u_+, here denoted by E_{u_+}, to be $E_{u_+} < \mathcal{L}(H_{u_-})$. This means that there is a waiting time at u_+, which is computed by $(\mathcal{L}(H_{u_-}) + w_{u_-}) - E_{u_+}$. Clearly, for the same value E_{u_+}, the amount of waiting time increases proportional to $\mathcal{L}(H_{u_-})$. Thus, unnecessary waiting time in route r_z is minimized whenever $\mathcal{L}(H_{u_-})$ is minimized. By contrast, suppose $E_{u_+} > \mathcal{L}(H_{u_-})$. In this case, no waiting occurs at node u_+ and the proof focuses on the slack from origin o_k to node u_+ in route r_z. The increase in slack *may* include the term $\mathcal{L}(H_{u_+}) - (\mathcal{L}(H_{u_-}) + w_{u_-})$ (Equation 18), which is maximized whenever $\mathcal{L}(H_{u_-})$ is minimum for the same value $\mathcal{L}(H_{u_+})$. However, this is not yet known when verifying Condition (33).

Hence, to guarantee minimum mandatory waiting time or maximum slack (both which contribute to ensuring maximum route duration), labels with minimum service time at $u_- \in U_-(\mathcal{L})$ nodes must be propagated until the corresponding u_+ node is scheduled to generate complete information. Otherwise, a dominance could fail as per the example illustrated in Appendix B.1.

Condition (34) complements Condition (33) with the basic idea that if the labels have everything else equal, then it is necessary to look at the slack up to any given interdependent task u_- for which we do not know

the start time of service at u_+. Essentially, for any origin o_z in routes r_z that precede u_- (that is, index $z \in \Phi(u_-)$), the impact of postponing H_{o_z} on the start time of service at u_- should be minimal in order to establish dominance. Thus, if service in o_z, H_{o_z} is postponed by α_z, we have the following expressions:

$$\mathcal{L}(H_{u_-}) \leftarrow \mathcal{L}(H_{u_-}) + \max\{0, \alpha_z - \min\{\mathcal{L}(P_z), \mathcal{L}(S_{zu_-}\}\}$$
$$\mathcal{L}'(H_{u_-}) \leftarrow \mathcal{L}'(H_{u_-}) + \max\{0, \alpha_z - \min\{\mathcal{L}'(P_z), \mathcal{L}'(S_{zu_-}\}\}$$

The proof continues in a similar way to what has been done for the general slack rule in Condition (32), except that in this case we look at slack up to all $u_- \in U_-(\mathcal{L})$ nodes, instead of looking at the latest scheduled node in each route. This clearly leads to a valid requirement that whenever Condition (34) is respected then, all else being equal, any postponement in routes preceding nodes u_- will impact the respective u_+ service in $\mathcal{L}$ no more than in $\mathcal{L}'$.

In conclusion, whenever Conditions (28)–(34) are satisfied for labels $\mathcal{L}$ and $\mathcal{L}'$, either $\mathcal{L}$ produces a feasible schedule or neither $\mathcal{L}$ nor $\mathcal{L}'$ can produce a feasible solution. Therefore, if $\mathcal{L}$ dominates $\mathcal{L}'$, then $\mathcal{L}'$ can be removed from the set of labels without incurring any loss of feasible solutions.

□

D. Supplementary results

This appendix provides additional information for the computational results, including the specific modifications to standard PDPT instances in the literature, additional graphs and detailed tables.

D.1. Modifications to Sampaio et al. (2020) instances

The instances for the Pickup and Delivery Problem with Transshipment proposed by Sampaio et al. (2020) were modified in our experiments to better fit our experimental settings. The original instances considered a time horizon of no more than $\mathcal{H} = 5$h, which is short for our purposes. However, given that the instances were large (100–200 nodes) and readily available, we opted to modify them instead of creating new instances for the TDSP-IR experiments.

The modifications were restricted to: time horizon, time windows, and service times for customer locations. The time horizon of all instances was increased to $\mathcal{H} = 13$h to fit our scenario. This extension of the time horizon forced us to also modify the time windows of the requests so that they could be served throughout the entire period. We employed the following procedure: given a request (p, d) for a pickup node p and delivery node d,

associated with time windows $[e_p, l_p]$ and $[e_d, l_d]$, we shift the original time windows by

$$\begin{aligned} e_p &= e_p + R \\ l_p &= l_p + R \\ e_d &= e_d + R \\ l_d &= l_d + R \end{aligned}$$

where R is a random integer in the interval $[0, 600]$. In this way, the requests of the original instances are spread over the longer time horizon, while still taking into account their original values.

Service times of 5 minutes were added to every customer location (and transfer operation) since the original instances assumed a value of zero for them. In our case, they are directly related to the working hours.

Finally, following Sampaio et al. (2020) we assume that every unit of travel time equals 1 minute. Thus, all times in the instances are represented as minutes for simplicity.

D.2. Graphs

Figure 11: Growth of average number of nodes in G with respect to the number of interdependencies in TDSP-IR instances.

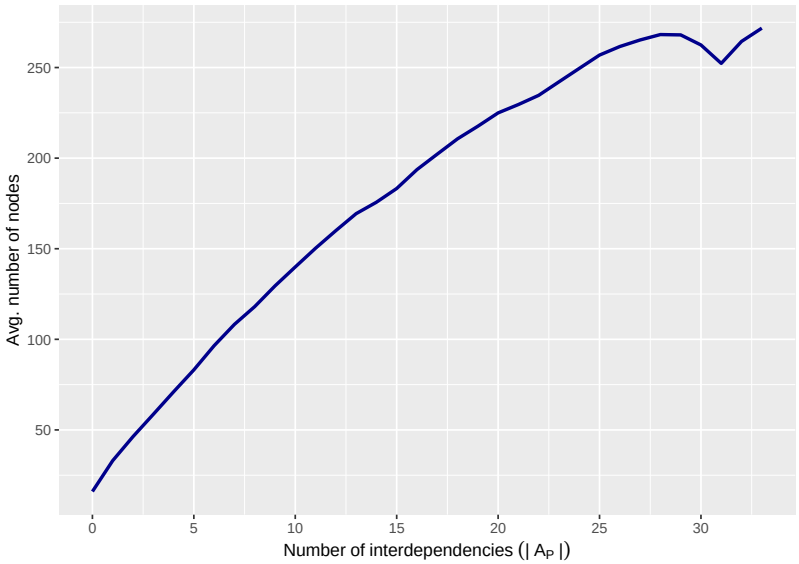
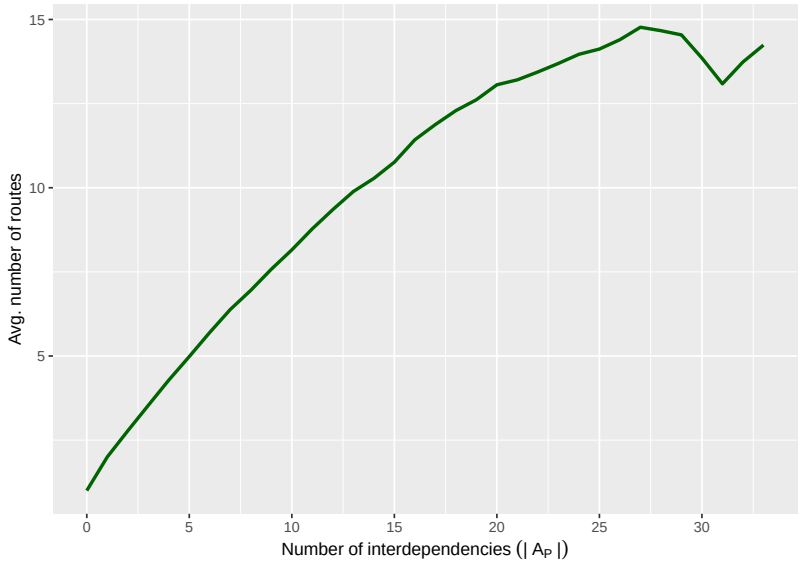


Figure 12: Growth of average number of routes in G with respect to the number of interdependencies in TDSP-IR instances.



D.3. Tables

The tables in this appendix present detailed execution times for both the LPA and the MILP. Tables 8, 9 and 10 report results over all TDSP-IR instances and for maximum route durations of $M_{\text{dur}} = 8\text{h}21$, $M_{\text{dur}} = 10\text{h}$ and $M_{\text{dur}} = 13\text{h}$, respectively. Similarly, Tables 11, 12 and 13 report results only over the feasible TDSP-IR instances. For each table, we report the minimum, average (and standard deviation), and maximum recorded running times of the scheduling algorithms according to the number of interdependencies $|A_P|$.

Table 8: Detailed runtimes over all instances for each algorithm ($M_{\text{dur}} = 8\text{h}21$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.00	0.01 ± 0.01	1.58	0.15	0.56 ± 0.23	24.73
1	0.00	0.01 ± 0.01	1.56	0.20	0.94 ± 0.33	22.85
2	0.00	0.01 ± 0.02	1.84	0.27	1.23 ± 0.45	28.12
3	0.00	0.01 ± 0.02	0.98	0.34	1.50 ± 0.56	23.50
4	0.00	0.02 ± 0.03	1.94	0.41	1.78 ± 0.68	26.28
5	0.00	0.02 ± 0.04	2.48	0.42	2.05 ± 0.82	27.44
6	0.00	0.02 ± 0.06	3.85	0.50	2.37 ± 0.96	28.90
7	0.00	0.03 ± 0.08	3.91	0.55	2.63 ± 1.05	30.39
8	0.00	0.04 ± 0.10	11.74	0.60	2.84 ± 1.13	26.89
9	0.00	0.04 ± 0.17	18.29	0.62	3.09 ± 1.23	30.69
10	0.00	0.05 ± 0.22	25.62	0.71	3.34 ± 1.30	29.84
11	0.00	0.07 ± 0.36	52.85	0.79	3.55 ± 1.38	32.23
12	0.00	0.07 ± 0.37	75.30	0.85	3.77 ± 1.48	32.41
13	0.00	0.09 ± 0.72	434.82	0.84	3.98 ± 1.56	33.11
14	0.00	0.10 ± 0.67	90.32	0.83	4.13 ± 1.61	33.84
15	0.00	0.11 ± 0.76	318.41	0.89	4.33 ± 1.71	35.41
16	0.00	0.14 ± 0.93	197.39	0.90	4.58 ± 1.81	33.61
17	0.00	0.17 ± 1.18	467.68	1.05	4.75 ± 1.88	33.44
18	0.00	0.18 ± 1.09	124.55	1.14	4.93 ± 1.94	35.74
19	0.00	0.20 ± 1.91	502.41	1.17	5.09 ± 2.00	36.52
20	0.00	0.20 ± 1.45	172.63	1.13	5.20 ± 1.96	34.75
21	0.00	0.33 ± 2.76	325.98	1.20	5.33 ± 2.04	29.73
22	0.00	0.23 ± 1.60	156.20	1.22	5.43 ± 2.09	34.82
23	0.00	0.31 ± 2.44	234.93	1.26	5.63 ± 2.17	36.90
24	0.00	0.37 ± 3.17	303.73	1.41	5.81 ± 2.19	36.99
25	0.00	0.52 ± 7.56	671.35	1.41	6.04 ± 2.42	38.65
26	0.00	0.71 ± 8.71	936.48	1.50	6.07 ± 2.25	36.79
27	0.00	0.88 ± 6.44	218.54	1.53	6.17 ± 2.22	34.52
28	0.00	0.92 ± 9.48	658.02	1.56	6.23 ± 2.30	28.33
29	0.00	1.13 ± 10.65	636.57	1.60	6.20 ± 2.31	28.45
30	0.00	2.76 ± 24.43	1237.14	1.63	6.37 ± 2.60	24.62
31	0.00	3.40 ± 20.44	583.35	1.69	6.29 ± 2.46	17.04
32	0.00	5.04 ± 14.18	49.12	5.35	7.78 ± 3.56	13.98
33	0.00	0.42 ± 1.86	14.59	5.60	6.64 ± 3.00	29.64

Table 9: Detailed runtimes over all instances for each algorithm ($M_{\text{dur}} = 10\text{h}$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.00	0.01 ± 0.01	1.55	0.14	0.62 ± 0.38	25.87
1	0.00	0.01 ± 0.02	1.61	0.20	1.10 ± 0.61	25.95
2	0.00	0.02 ± 0.03	1.60	0.27	1.46 ± 0.82	27.88
3	0.00	0.03 ± 0.04	1.05	0.30	1.77 ± 1.00	27.13
4	0.00	0.04 ± 0.05	1.90	0.41	2.22 ± 1.28	29.56
5	0.00	0.05 ± 0.07	2.45	0.46	2.54 ± 1.42	32.47
6	0.00	0.06 ± 0.11	7.14	0.49	2.84 ± 1.62	30.52
7	0.00	0.08 ± 0.16	19.08	0.63	3.23 ± 1.84	66.65
8	0.00	0.11 ± 0.33	39.02	0.65	3.55 ± 2.06	32.71
9	0.00	0.13 ± 0.36	30.58	0.71	3.88 ± 2.24	35.63
10	0.00	0.15 ± 0.46	34.43	0.75	4.07 ± 2.37	33.29
11	0.00	0.17 ± 0.56	74.56	0.78	4.32 ± 2.48	35.45
12	0.00	0.22 ± 1.12	119.38	0.87	4.55 ± 2.59	37.96
13	0.00	0.28 ± 1.50	186.71	0.93	4.84 ± 2.74	35.39
14	0.00	0.30 ± 1.47	141.59	0.81	5.04 ± 2.78	53.60
15	0.00	0.35 ± 1.75	183.17	0.97	5.24 ± 2.90	41.79
16	0.00	0.40 ± 1.92	247.39	1.02	5.46 ± 3.02	52.75
17	0.00	0.47 ± 3.53	701.92	1.08	5.61 ± 3.09	39.60
18	0.00	0.80 ± 10.00	1993.29	1.07	5.84 ± 3.20	41.61
19	0.00	0.94 ± 9.13	1294.59	1.08	5.99 ± 3.24	39.04
20	0.00	0.74 ± 4.75	395.94	1.25	6.22 ± 3.32	38.58
21	0.00	0.97 ± 7.55	563.25	1.34	6.30 ± 3.39	38.63
22	0.00	0.92 ± 6.56	781.82	1.37	6.38 ± 3.38	34.62
23	0.00	1.01 ± 11.32	1514.07	1.39	6.43 ± 3.45	37.91
24	0.00	0.89 ± 6.58	416.41	1.42	6.51 ± 3.45	30.25
25	0.00	1.17 ± 8.17	321.05	1.47	6.64 ± 3.38	39.08
26	0.00	0.95 ± 6.56	263.35	1.52	6.94 ± 3.57	38.46
27	0.00	1.81 ± 10.93	500.18	1.55	7.24 ± 3.54	29.91
28	0.00	1.29 ± 8.84	599.67	1.62	7.40 ± 3.51	35.81
29	0.00	3.06 ± 21.02	814.51	1.71	7.66 ± 3.86	39.63
30	0.00	1.92 ± 12.13	318.29	1.79	7.32 ± 3.31	39.94
31	0.00	6.20 ± 26.94	624.27	1.80	7.22 ± 3.42	17.41
32	0.00	1.92 ± 7.47	196.63	1.92	8.34 ± 4.10	24.21
33	0.00	0.72 ± 3.17	43.17	2.03	7.06 ± 2.77	15.96
34	0.00	1.50 ± 5.18	41.19	2.11	7.51 ± 2.95	16.11

Table 10: Detailed runtimes over all instances for each algorithm ($M_{\text{dur}} = 13\text{h}$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.00	0.01 ± 0.01	1.62	0.15	0.56 ± 0.45	22.43
1	0.00	0.03 ± 0.02	2.01	0.20	1.32 ± 0.79	21.94
2	0.00	0.04 ± 0.03	1.07	0.28	1.82 ± 1.04	27.16
3	0.00	0.06 ± 0.04	3.43	0.37	2.44 ± 1.42	27.52
4	0.00	0.07 ± 0.06	0.68	0.41	2.78 ± 1.64	29.01
5	0.00	0.09 ± 0.08	1.98	0.42	3.22 ± 1.83	28.15
6	0.00	0.13 ± 0.11	2.49	0.45	3.73 ± 2.13	30.74
7	0.00	0.17 ± 0.15	5.60	0.61	4.25 ± 2.40	32.07
8	0.00	0.21 ± 0.21	13.28	0.67	4.68 ± 2.61	32.87
9	0.00	0.25 ± 0.26	18.67	0.78	5.11 ± 2.74	34.03
10	0.00	0.29 ± 0.29	18.55	0.76	5.48 ± 2.93	33.77
11	0.00	0.34 ± 0.38	22.83	0.85	5.83 ± 3.06	35.35
12	0.00	0.41 ± 0.65	28.49	0.96	6.11 ± 3.20	34.06
13	0.00	0.47 ± 0.99	77.45	1.03	6.44 ± 3.28	35.45
14	0.00	0.53 ± 1.17	79.47	0.95	6.69 ± 3.46	36.87
15	0.00	0.62 ± 1.64	84.49	1.07	7.13 ± 3.68	37.36
16	0.00	0.63 ± 1.57	116.94	1.07	7.16 ± 3.63	36.97
17	0.00	0.73 ± 1.83	143.59	1.10	7.45 ± 3.76	37.89
18	0.00	0.93 ± 2.68	90.30	1.15	7.88 ± 3.89	37.93
19	0.00	0.99 ± 2.61	258.38	1.23	8.16 ± 4.01	38.55
20	0.00	1.13 ± 3.25	201.66	1.27	8.50 ± 4.08	38.98
21	0.00	1.30 ± 4.46	258.34	1.28	8.59 ± 4.10	38.18
22	0.00	1.46 ± 6.59	369.20	1.40	8.42 ± 4.13	37.90
23	0.00	2.24 ± 11.12	399.18	1.48	8.66 ± 4.21	38.33
24	0.00	2.96 ± 16.62	414.53	1.49	9.03 ± 4.16	38.50
25	0.00	3.21 ± 21.10	1150.30	1.47	8.73 ± 3.88	34.96
26	0.00	3.55 ± 35.45	1126.43	1.48	8.91 ± 3.74	33.25
27	0.00	3.56 ± 21.23	689.95	1.58	9.06 ± 3.62	19.21
28	0.00	6.17 ± 33.18	526.44	1.83	8.67 ± 3.33	34.24
29	0.00	9.38 ± 59.35	958.84	1.87	8.14 ± 3.70	33.57
30	0.00	7.07 ± 52.51	732.06	1.94	8.17 ± 3.58	15.83
31	0.00	1.05 ± 1.71	12.47	1.91	9.00 ± 3.17	14.83
32	0.79	0.87 ± 0.09	0.99	6.20	11.75 ± 3.80	14.83

Table 11: Detailed runtimes over feasible instances for each algorithm ($M_{\text{dur}} = 8\text{h}21$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.01	0.01 ± 0.01	1.58	0.24	0.55 ± 0.40	24.73
1	0.01	0.04 ± 0.01	1.56	0.37	1.54 ± 0.65	21.59
2	0.01	0.06 ± 0.02	1.84	0.42	2.23 ± 0.72	28.12
3	0.02	0.07 ± 0.02	0.93	0.43	2.82 ± 0.78	23.50
4	0.02	0.09 ± 0.03	1.94	0.85	3.41 ± 0.96	26.28
5	0.02	0.12 ± 0.04	2.48	1.10	4.02 ± 1.07	21.64
6	0.04	0.15 ± 0.06	3.85	1.66	4.75 ± 1.19	28.90
7	0.05	0.18 ± 0.06	2.92	2.13	5.30 ± 1.20	30.39
8	0.05	0.21 ± 0.08	3.22	2.43	5.69 ± 1.20	14.82
9	0.07	0.26 ± 0.14	6.56	2.79	6.25 ± 1.34	30.69
10	0.08	0.30 ± 0.20	6.64	2.91	6.60 ± 1.36	29.84
11	0.06	0.36 ± 0.27	10.36	2.98	7.03 ± 1.41	32.23
12	0.09	0.39 ± 0.24	28.03	3.96	7.60 ± 1.38	32.41
13	0.10	0.44 ± 0.36	18.43	3.84	7.99 ± 1.43	33.11
14	0.12	0.47 ± 0.25	6.73	3.53	8.30 ± 1.50	33.84
15	0.15	0.51 ± 0.36	33.08	4.18	8.75 ± 1.59	35.41
16	0.15	0.58 ± 0.32	18.95	4.46	9.16 ± 1.57	33.61
17	0.16	0.64 ± 0.49	33.23	4.94	9.59 ± 1.55	33.44
18	0.16	0.69 ± 0.42	16.22	5.26	9.95 ± 1.57	35.74
19	0.16	0.75 ± 0.91	88.27	5.38	10.17 ± 1.48	36.52
20	0.25	0.78 ± 0.62	33.98	5.57	10.24 ± 1.55	34.75
21	0.29	1.46 ± 4.84	132.71	5.83	10.65 ± 1.56	21.89
22	0.30	1.00 ± 1.68	75.24	6.10	10.97 ± 1.55	34.82
23	0.31	1.09 ± 1.66	78.37	6.61	11.46 ± 1.75	36.90
24	0.37	1.35 ± 2.21	31.36	6.68	11.79 ± 1.86	36.99
25	0.46	1.48 ± 4.63	108.52	7.62	12.63 ± 2.03	38.65
26	0.48	2.27 ± 10.02	144.44	7.89	12.31 ± 1.92	36.79
27	0.52	1.99 ± 5.70	142.83	8.69	11.94 ± 1.67	34.52
28	0.54	1.23 ± 1.40	28.64	8.74	12.04 ± 1.63	19.93
29	0.64	1.92 ± 4.18	62.36	8.70	12.54 ± 1.81	19.38
30	0.64	1.78 ± 3.36	45.40	9.34	13.19 ± 1.82	17.30
31	0.79	0.96 ± 0.10	1.63	10.18	13.17 ± 1.20	17.04
32	0.88	0.95 ± 0.05	1.01	12.07	13.54 ± 0.83	13.98
33	0.97	1.01 ± 0.04	1.08	14.29	14.54 ± 0.19	14.78

Table 12: Detailed runtimes over feasible instances for each algorithm ($M_{\text{dur}} = 10\text{h}$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.01	0.02 ± 0.01	1.55	0.24	0.64 ± 0.55	25.87
1	0.01	0.05 ± 0.02	1.61	0.37	1.95 ± 0.87	25.95
2	0.01	0.07 ± 0.02	1.60	0.40	2.67 ± 0.98	27.88
3	0.02	0.09 ± 0.03	0.97	0.54	3.44 ± 1.09	27.13
4	0.03	0.12 ± 0.04	1.90	0.66	4.24 ± 1.31	29.56
5	0.04	0.15 ± 0.06	2.45	1.03	4.81 ± 1.39	32.47
6	0.05	0.19 ± 0.09	2.12	1.09	5.53 ± 1.50	30.52
7	0.05	0.24 ± 0.11	4.84	1.95	6.29 ± 1.63	30.85
8	0.05	0.30 ± 0.15	6.52	2.09	7.00 ± 1.74	32.71
9	0.05	0.35 ± 0.17	7.97	2.57	7.69 ± 1.79	35.63
10	0.05	0.39 ± 0.18	7.57	2.96	8.16 ± 1.88	33.29
11	0.09	0.45 ± 0.25	17.45	3.28	8.56 ± 1.92	35.45
12	0.11	0.50 ± 0.28	26.89	3.49	9.02 ± 1.93	37.96
13	0.12	0.59 ± 0.50	19.00	3.64	9.49 ± 1.90	35.39
14	0.11	0.62 ± 0.64	75.03	3.52	9.71 ± 1.93	35.82
15	0.12	0.69 ± 0.65	42.45	3.76	10.16 ± 2.02	41.79
16	0.13	0.74 ± 0.73	56.32	4.11	10.39 ± 2.05	40.69
17	0.15	0.82 ± 1.09	47.25	4.33	10.79 ± 2.09	39.60
18	0.24	0.91 ± 2.28	182.04	5.38	11.11 ± 2.18	41.61
19	0.23	1.14 ± 3.65	135.63	4.80	11.23 ± 2.11	39.04
20	0.26	1.01 ± 2.48	225.64	5.38	11.54 ± 2.07	38.58
21	0.28	1.16 ± 2.36	97.66	6.44	11.73 ± 1.89	38.63
22	0.31	1.25 ± 2.73	105.53	6.56	12.07 ± 1.97	34.62
23	0.32	1.21 ± 2.09	71.89	6.54	12.25 ± 2.00	37.91
24	0.35	1.27 ± 2.21	93.63	6.64	12.41 ± 2.15	24.59
25	0.43	1.08 ± 1.07	35.25	7.07	12.42 ± 2.52	39.08
26	0.43	1.39 ± 1.96	34.30	7.99	13.27 ± 2.72	38.46
27	0.46	1.32 ± 3.36	80.57	7.56	13.03 ± 2.08	23.61
28	0.53	1.31 ± 2.51	40.84	8.49	13.44 ± 1.84	35.81
29	0.62	1.38 ± 3.97	84.23	9.50	14.13 ± 2.01	39.63
30	0.68	1.38 ± 2.39	42.78	9.11	13.87 ± 1.61	39.94
31	0.70	1.49 ± 2.66	42.15	9.48	14.50 ± 1.58	17.41
32	0.79	1.80 ± 4.92	57.72	12.02	15.59 ± 1.34	24.21
33	0.87	1.31 ± 2.14	16.15	12.16	14.03 ± 0.65	15.96
34	0.88	1.16 ± 1.48	15.84	12.59	14.13 ± 0.83	16.11

Table 13: Detailed runtimes over feasible instances for each algorithm ($M_{\text{dur}} = 13\text{h}$).

	LPA			MILP		
$\lvert A_P \rvert$	min	avg ± sd	max	min	avg ± sd	max
0	0.00	0.01 ± 0.01	1.62	0.15	0.56 ± 0.45	22.43
1	0.00	0.03 ± 0.02	2.01	0.20	1.32 ± 0.79	21.94
2	0.00	0.04 ± 0.03	1.07	0.28	1.82 ± 1.04	27.16
3	0.00	0.06 ± 0.04	3.43	0.37	2.44 ± 1.42	27.52
4	0.00	0.07 ± 0.06	0.68	0.41	2.78 ± 1.64	29.01
5	0.00	0.09 ± 0.08	1.98	0.42	3.22 ± 1.83	28.15
6	0.00	0.13 ± 0.11	2.49	0.45	3.73 ± 2.13	30.74
7	0.00	0.17 ± 0.15	5.60	0.61	4.25 ± 2.40	32.07
8	0.00	0.21 ± 0.21	13.28	0.67	4.68 ± 2.61	32.87
9	0.00	0.25 ± 0.26	18.67	0.78	5.11 ± 2.74	34.03
10	0.00	0.29 ± 0.29	18.55	0.76	5.48 ± 2.93	33.77
11	0.00	0.34 ± 0.38	22.83	0.85	5.83 ± 3.06	35.35
12	0.00	0.41 ± 0.65	28.49	0.96	6.11 ± 3.20	34.06
13	0.00	0.47 ± 0.99	77.45	1.03	6.44 ± 3.28	35.45
14	0.00	0.53 ± 1.17	79.47	0.95	6.69 ± 3.46	36.87
15	0.00	0.62 ± 1.64	84.49	1.07	7.13 ± 3.68	37.36
16	0.00	0.63 ± 1.57	116.94	1.07	7.16 ± 3.63	36.97
17	0.00	0.73 ± 1.83	143.59	1.10	7.45 ± 3.76	37.89
18	0.00	0.93 ± 2.68	90.30	1.15	7.88 ± 3.89	37.93
19	0.00	0.99 ± 2.61	258.38	1.23	8.16 ± 4.01	38.55
20	0.00	1.13 ± 3.25	201.66	1.27	8.50 ± 4.08	38.98
21	0.00	1.30 ± 4.46	258.34	1.28	8.59 ± 4.10	38.18
22	0.00	1.46 ± 6.59	369.20	1.40	8.42 ± 4.13	37.90
23	0.00	2.24 ± 11.12	399.18	1.48	8.66 ± 4.21	38.33
24	0.00	2.96 ± 16.62	414.53	1.49	9.03 ± 4.16	38.50
25	0.00	3.21 ± 21.10	1150.30	1.47	8.73 ± 3.88	34.96
26	0.00	3.55 ± 35.45	1126.43	1.48	8.91 ± 3.74	33.25
27	0.00	3.56 ± 21.23	689.95	1.58	9.06 ± 3.62	19.21
28	0.00	6.17 ± 33.18	526.44	1.83	8.67 ± 3.33	34.24
29	0.00	9.38 ± 59.35	958.84	1.87	8.14 ± 3.70	33.57
30	0.00	7.07 ± 52.51	732.06	1.94	8.17 ± 3.58	15.83
31	0.00	1.05 ± 1.71	12.47	1.91	9.00 ± 3.17	14.83
32	0.79	0.87 ± 0.09	0.99	6.20	11.75 ± 3.80	14.83