Pickup and delivery problems with autonomous vehicles on rings

Manuel Trotta, Claudia Archetti, Dominique Feillet, Alain Quilliot

Pickup and delivery problems with autonomous vehicles on rings

Manuel Trotta[1], Claudia Archetti[2], Dominique Feillet [3], Alain Quilliot [4]

[1] Université Clermont Auvergne, UMR CNRS 6158 LIMOS, F-13541 Gardanne France
manuel.trotta@emse.fr

[2] ESSEC Business School, 3 Avenue Bernard Hirsch, 95000 Cergy, France
archetti@essec.edu

[3] Mines Saint-Etienne, Univ. Clermont Auvergne, CNRS, UMR 6158 LIMOS, Centre CMP, F-13541 Gardanne, France
dominique.feillet@mines-stetienne.fr

[4] LIMOS, ISIMA, Campus des Cézeaux, Aubiére Cedex, France
alain.quilliot@isima.fr

CMP – CAMPUS GEORGES CHARPAK PROVENCE
CENTRE MICROELECTRONIQUE DE PROVENCE

Working Paper EMSE CMP–SFL 2021/2

In this paper we introduce a new class of Pickup and Delivery problems on circles (or rings). We consider m stations arranged in a circle and n transportation requests. Each request i asks for the transportation of a certain quantity qi from a pickup station si to a delivery station ti. A fleet of capacitated vehicles is available at the depot. In the first part of the paper we propose a classification scheme for these problems. In the second part, we address the variants in which the vehicles are allowed to move in a single direction of the circle (either clockwise or counterclockwise) and the objective is to minimize the number of tours on the ring while serving all the requests. We provide a complexity analysis for this class of problems. We develop polynomial time algorithms for the variants that are polynomially solvable and proofs of NP-hardness for the variants that are NP-hard. In addition, for the latter, we provide mathematical formulations and perform computational tests that show the effectiveness of these formulations. Finally, we compare optimal solutions with those obtained using a straightforward greedy algorithm.

MINES Saint-Etienne
Une école de l'IMT

Pickup and delivery problems with autonomous vehicles on rings

Manuel Trotta[a,1,*], Claudia Archetti[b], Dominique Feillet[c], Alain Quilliot[d]

[a] *Université Clermont Auvergne, CNRS, UMR 6158 LIMOS, F-13541 Gardanne France*
[b] *ESSEC Business School, 3 Avenue Bernard Hirsch, 95000 Cergy, France*
[c] *Mines Saint-Etienne, Univ. Clermont Auvergne, CNRS, UMR 6158 LIMOS, Centre CMP, F-13541 Gardanne France*
[d] *LIMOS, Institut Supérieur d'Informatique de Modélisation et leurs Applications and ISIMA, Campus des Cézeaux, Aubiére Cedex, France*

Abstract

In this paper we introduce a new class of Pickup and Delivery problems on circles (or rings). We consider m stations arranged in a circle and n transportation requests. Each request i asks for the transportation of a certain quantity q_i from a pickup station s_i to a delivery station t_i. A fleet of capacitated vehicles is available at the depot. In the first part of the paper we propose a classification scheme for these problems. In the second part, we address the variants in which the vehicles are allowed to move in a single direction of the circle (either clockwise or counterclockwise) and the objective is to minimize the number of tours on the ring while serving all the requests. We provide a complexity analysis for this class of problems. We develop polynomial time algorithms for the variants that are polynomially solvable and proofs of NP-hardness for the variants that are NP-hard. In addition, for the latter, we provide mathematical formulations and perform computational tests that show the effectiveness of these formulations. Finally, we compare optimal solutions with those obtained using a straightforward greedy algorithm.

Keywords: Routing, pickup and delivery problems, ring, computational complexity

1. Introduction

It is a matter of fact that urban mobility, by means of public transport or personal cars, has become a key factor in people's everyday life, making it easier. Nowadays, working activities are more and more concentrated in urban areas. Thus, more and more people commute on urban areas on a daily basis. This clearly causes a number of issues, and public entities are struggling to find ways and policies that help in facing the ever growing demand

*Corresponding author

Email addresses: manuel.trotta@emse.fr (Manuel Trotta), archetti@essec.edu (Claudia Archetti), dominique.feillet@mines-stetienne.fr (Dominique Feillet), alain.quilliot@isima.fr (Alain Quilliot)

[1] This work was sponsored by a public grant overseen by the French National Research Agency as part of the "Investissements d'Avenir" through the IMobS3 Laboratory of Excellence (ANR-10-LABX-0016) and the IDEX-ISITE initiative CAP 20-25 (ANR-16-IDEX-0001).

of service. Among the different solutions proposed in the recent years, the one in which we are interested in this paper is the use of autonomous vehicles, like cars, minibuses and shuttles. The use of these vehicles raises legal, ethical, economic and safety issues. Due to these problems it is not likely that autonomous vehicles will totally replace normal vehicles soon. However, they will probably first be authorized for collective transportation.

There already exist few cases where fully autonomous vehicles are used in public transportation. In May 2019 Groupe Renault, the Transdev Group, IRT SystemX, Institut VEDECOM and the University of Paris-Saclay initiated a new project called Paris-Saclay Autonomous Lab (par (2019)). Its purpose was to develop new autonomous (i.e. driverless) mobility services using dedicated lane and public streets to supplement the existing Saclay Plateau transportation system. An overnight public transportation service using an autonomous Transdev-Lohr i-Cristal shuttle was designed to serve the Saclay Plateau neighborhoods from the Massy station. On December 2018, Keolis and the European Metropolis of Lille launched an electric autonomous shuttle service at the University of Lille in Villeneuve d'Ascq, with a student population of 20,000 and 1,600 researchers (lil (2018)). The service at Lille university had employed two NAVYA electric autonomous shuttles for a period of one year with four dedicated stops on a 1.4 km circle route, and had provided connections to two metro stations. More recently, in Lyon a project has been launched to study the use of electric autonomous shuttles for urban mobility (lyo (2020)).

In Antonialli (2019) a worldwide benchmark on the use of Autonomous Shuttles for Collective Transport (ASCT) has been performed. By the time this research was carried out, a total of 92 experiments were identified, spread over 32 countries around the world and enabled by 20 different autonomous shuttle manufacturers. Results showed a European lead on both the number of experiments and manufacturers, with highlights to the French startups Navya and EasyMile. Regarding the road environment, two distinct scenarios were observed. In the first, shuttles circulate in closed/controlled areas (such as university campuses, parks, hospitals, resorts, airports, and other designated roads); this kind of deployment comprised 52.17% of the projects. In the second scenario (47.83%), shuttles were able to circulate among mixed traffic – for these cases the routes were mainly predetermined in city-centers or areas with a slow-speed circulation for regular vehicles. By analyzing the prevailing business models, the author observed that the vast majority of experiments tackled public transport schemes (96.55%) with daily commuters as the main revenue source for the transport operator. Systems with regular lines comprised the vast majority of models among the sampled projects (91.21%) while demand-responsive transport answered to only 4.40% and a mixed approach comprising both operation models was present in the other 4.40%. Notwithstand-

ing, as more countries and cities begin to allow testing and circulation of AVs, the percentage of on-demand autonomous mobility is likely to increase.

In this work, we investigate on-demand services carried out using autonomous vehicles on circular networks. Circular networks are typical in the closed/controlled areas mentioned above. These problems belong to the class of Vehicle Routing Problems with Pickups and Deliveries (VRPPD). For an overview on the VRPPD the reader is referred to Desaulniers et al. (2002). For a survey on pickup and delivery problems the reader is referred to Parragh et al. (2008, 2007), where the problems are classified as Pickup and Delivery Vehicle Routing Problem (PDVRP), where pickup and delivery points are unpaired, the Pickup and Delivery Problem (PDP), where pickup and delivery points are paired, and the Dial-A-Ride Problem (DARP) which deals with passenger transportation between paired pickup and delivery points. The research on Pickup and Delivery problems is still very active (see Wu et al. (2019), Rüther & Rieck (2020)). The reader is referred to Berbeglia et al. (2007, 2010) for surveys on static and dynamic Pickup and Delivery problems. Toth & Vigo (2014) contains two chapters on the PDP, respectively the PDP for goods transportation (Battarra et al. (2014)) and for people transportation (Doerner & Salazar-González (2014)). For a review article on the DARP the reader is referred to Cordeau & Laporte (2007). For more recent surveys on dial-a-ride problems, see Ho et al. (2018) and Molenbruch et al. (2017). For a high-level classification of dial-a-ride problems, the reader is referred to Gökay et al. (2019).

Concerning problems defined on circles, the exisiting literature is limited. Gendreau et al. (1999) developed a linear time exact algorithm for the Single-Vehicle Pickup and Delivery Problem defined on a cycle graph. Tzoreff et al. (2002) studied the Vehicle Routing Problem with Pickup and Delivery on some special graphs. They developed an optimal algorithm that runs in polynomial time for cycle graphs. Ilani et al. (2015) presented some optimal polynomial-time algorithms for two variants of the Fixed Route DARP with a circular route. The first one considers a fleet of infinite capacity vehicles, while the second one considers the more general case of vehicles with heterogeneous capacities. Dial-a-Ride problems with autonomous vehicles have also been studied in Pimenta et al. (2017) and Baïou et al. (2018).

A number of related problems arise in the field of industrial automation. Atallah & Kosaraju (1988) were probably the first to study the problem of efficiently rearranging parts in the plane with a centrally placed gripper that can rotate. This problem is known as the Stacker Crane Problem (SCP), and they proposed a polynomial time algorithm for the SCP on a circle. Anily & Pfeffer (2013) studied a similar problem, the Uncapacitated Swapping Problem, on a line and on a circle, where the objective is to rearrange objects of different types on a circular graph using an uncapacitated vehicle. It can be seen as a generalization

of the SCP. They proposed a polynomial time algorithm for both cases of a line and a circle.

The contributions of this paper can be summarized as follows. We introduce a new class of problems, the Pickup and Delivery Problems on Rings (PDP-R). We propose a classification scheme for these problems which resembles the classification used for scheduling problems. In this class, we investigate a subclass where vehicles all travel in the same direction along the circle and the objective is to minimize the time at which the last vehicle returns to the depot. The peculiarity of these problems is that they do not involve any routing decision: the vehicles repeatedly turn around the circle until all pickup and delivery services have been carried out. The optimization comes from the efficient assignment of operations to vehicles and the scheduling of these operations. We determine the computational complexity for all variants of this subclass. We develop polynomial time algorithms for the problems that are polynomially solvable and proofs of NP-hardness for the others. In addition, for the latter, we provide efficient mathematical formulations that allow solving large-size instances quickly. Finally, we compare optimal solutions with those obtained using a straightforward greedy algorithm, that could be easily implemented by practitioners.

The rest of the paper is organized as follows. In Section 2, we introduce notation and some basic definitions, and we present the classification scheme. Section 3 is dedicated to problems with unitary requests and no release/due dates. Section 4 is devoted to problems with unitary requests and release/due dates. Section 5 deals with problems having non-unitary requests. Computational experiments and the greedy algorithm are presented in Section 6. In Section 7, we summarize the results and present the perspectives of this work.

2. Problem description, classification and scope of the paper

In this section, we first provide a general description of the PDP-R. Then, we introduce a classification scheme. Finally, we present the variants addressed in this paper.

2.1. General problem description

The setting of the PDP-R is the following. The ring is represented by a directed graph whose set of nodes includes m stations numbered from 0 to $m-1$. To simplify further notation, station 0 is indifferently denoted as 0 or m. The set of arcs consists of $2 \times m$ links $(j, j+1)$ and $(j+1, j)$ between consecutive stations $(0 \leq j \leq m-1)$ (see Figure 1). Travel times $\delta_{j,j+1}$ and $\delta_{j+1,j}$ are defined between two consecutive stations $(j = 0, ..., m-1)$. These travel times are not necessarily symmetric.

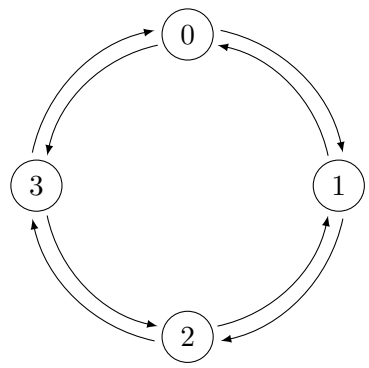


Figure 1: PDP-R with four stations (station 0 is the depot)

We consider a multiset $\mathcal{R}$ of n transportation requests. In the more general case, each request $i \in \mathcal{R}$ is defined by: a pickup station s_i and a target station t_i ($s_i, t_i = 0, ..., m-1,\ s_i \neq t_i$), a quantity q_i to be transported, a release date r_i and a due date d_i. The release date indicates the earliest time for the pickup operation, the due date is the latest time for the delivery. $\mathcal{R}$ is defined as a multiset instead of a set because it can contain identical requests.

The requests are served by a fleet of vehicles that can travel along the ring in either one or both directions. The vehicles are capacitated, so the load they can transport at the same time cannot exceed their capacity. Station 0 is the depot, *i.e.,* a station where the vehicles are located before starting the service and have to return after having served all the requests. Without loss of generality, we assume that travel times are equal to distances, and therefore the words time and distance are interchangeable.

2.2. A classification scheme

One of the objectives of this work is to propose a classification scheme for PDP-R problems. It has been inspired by the three-field classification introduced in Graham et al. (1979) for scheduling problems . It is composed of three fields - α, β and γ - separated by a vertical bar. Each field may be a comma separated list of words that describes one or more features of the problem.

The α field contains information on the vehicles, separated by a comma:

- number of available vehicles: 1 or V (with $V > 1$);
- capacity of the vehicles: 1 or Q (with $Q > 1$) or Q_v (if vehicles have different capacities).

The β field reports optional constraints, i.e., the field is empty in case of default settings. The optional constraints, separated by a comma, are:

- *sd* (single direction): vehicles all follow the same direction, fixed from the beginning (clockwise or counterclockwise);

- md (mixed single-direction): each vehicle follows a single direction, fixed from the beginning, that can be either clockwise or counterclockwise;
- r_i: pickups are subject to release dates;
- d_i: deliveries are subject to due dates;
- u: demands are unitary.

The default settings are: bi-directional, meaning that vehicles travel in both directions; no release and due dates; demands corresponding to generic quantities q_i.

Finally, field γ indicates the objective function:

- C_{max}: makespan, *i.e.*, the completion time of the last request scheduled among all the vehicles;
- $\sum C_i$: total completion time, i.e., the sum of completion times over all vehicles;
- CLT: closing time, *i.e.*, time at which the last vehicle comes back to the depot when all requests are served.

For example, the PDP-R with one vehicle of unit capacity rotating clockwise with no release dates and due dates whose objective is the minimization of the maximum completion time is denoted by $1,1|sd,u|C_{max}$.

2.3. Scope of the paper and related definitions

In this paper, we present theoretical results and computational experiments concerning all the problem variants where the vehicles are allowed to move on the ring in a single direction (sd) and the objective function is the minimization of the closing time (CLT). Without loss of generality, we assume that all vehicles go clockwise.

Due to the circular layout, to go from j_1 to j_2, vehicles must pass through all intermediate stations between j_1 and j_2, that is: stations $j_1+1,\ldots,j_2-1$ if $j_1 < j_2$, stations $j_1+1,\ldots,m-1,0,\ldots,j_2-1$ otherwise. This allows us to define the distance between two stations from distances between consecutive stations:

$$\delta_{j_1,j_2} = \begin{cases} \sum_{k=j_1}^{j_2-1} \delta_{k,k+1} & \text{if } j_1 < j_2 \\ \sum_{k=j_1}^{m-1} \delta_{k,k+1} + \sum_{k=0}^{j_2-1} \delta_{k,k+1} & \text{otherwise} \end{cases} \tag{1}$$

Note that distances δ_{j_1,j_2} satisfy the triangle inequality. We call L the length of a complete tour, starting from the depot and returning back to the depot.

To ease the readability in the remainder of the paper, we introduce a few definitions.

Definition 1. *We say that a request $i \in \mathcal{R}$ covers a segment $[j, j+1]$ if stations j and $j+1$ are within stations s_i and t_i when going clockwise. Equivalently, it means that a vehicle needs to traverse arc $(j, j+1)$ in order to serve i.*

Definition 2. *We say that a request $i \in \mathcal{R}$ covers a station j if it covers both segments $[j-1, j]$ and $[j, j+1]$.*

In particular, a request i *covers* the depot when $0 < t_i < s_i$.

Definition 3. *We say that two requests $i_1, i_2 \in \mathcal{R}$ overlap if at least one segment $[j, j+1]$ of the ring is covered by both requests $(j = 0, ..., m-1)$.*

Definition 4. *If two requests do not overlap they are said to be compatible. Otherwise, they are said to be in conflict (or conflicting).*

Figure 2 shows an example with four stations and three requests identified by their pair (s_i, t_i): $(3, 1)$, $(1, 2)$, $(1, 0)$. Requests $(3, 1)$ and $(1, 2)$ are compatible, while requests $\{(1, 2), (1, 0)\}$ or $\{(1, 0), (3, 1)\}$ are in conflict (requests overlap in both pairs).

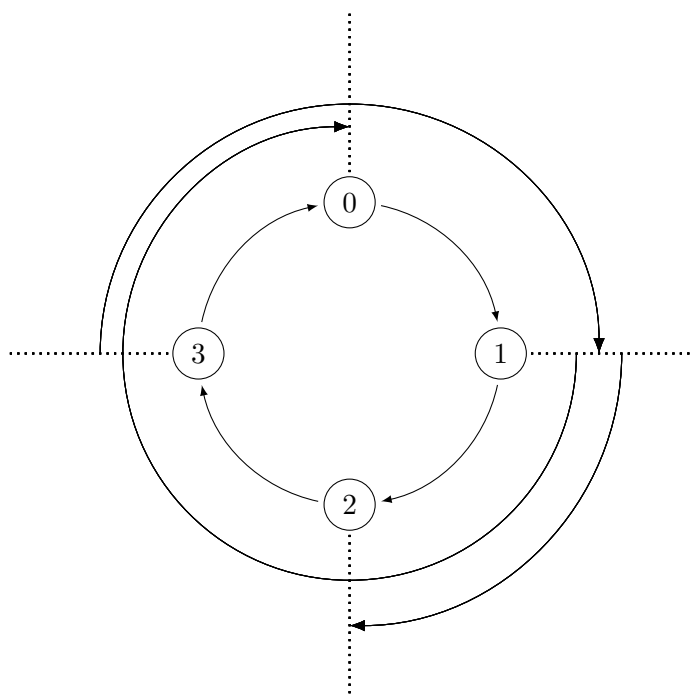


Figure 2: An example of compatible and overlapping requests

Definition 5. *A tour is a complete rotation of a vehicle around the ring, starting from station 0, passing through all stations j $(j = 1, ..., m-1)$ and getting back to 0.*

Definition 6. *A schedule is an assignment of the requests to the vehicles that specifies, for each vehicle, the tour in which the requests are executed. A schedule is also a solution for the problem.*

Definition 7. *A request is said active at a given time when the service of this request is started but not finished, i.e., the request has been picked up at station s_i but not yet delivered to station t_i.*

3. Problems with unitary requests and no release/due dates

In this section, we first prove that problem $1,1|sd,u|CLT$ is polynomially solvable. In this case, the fleet is limited to a single vehicle of unit capacity and requests have unitary demands. No release dates nor due dates are considered.

We then investigate other classes of problems with unitary demands and no release/due dates. Remember that, as all other problems addressed in this paper, we consider that vehicles are subject to the *sd* constraint (single-direction) and that the objective is *CLT* (closing time). We show that these problems are NP-hard (Section 3.2) and propose a mixed integer linear programming formulation (Section 3.3).

3.1. Problem $1,1|sd,u|CLT$

A single vehicle of capacity one has to serve the multiset $\mathcal{R}$ of n unitary requests. It is constrained to travel clockwise. The objective is to serve all requests and return to the depot as early as possible. Equivalently, it consists in minimizing the number of tours traveled by the vehicle to serve all requests.

We propose a polynomial-time algorithm that solves the problem. The basic idea is to construct a graph whose vertices represent the stations and whose arcs represent the requests, and then find an Eulerian tour in this graph.

We introduce the following notation. T_{OPT} is the optimal number of tours. We denote $z(\mathcal{R})$ the optimal closing time. We express z as a function of $\mathcal{R}$ because we introduce dummy requests in the algorithm, as explained later. Note that $z(\mathcal{R}) = L \times T_{OPT}$. Requests are defined by their origin and their destination; we represent them by pair (s_i, t_i). Then, $\mathcal{R} = ((s_i, t_i) : 1 \leq i \leq n)$. We define for all stations j $(j = 0, ..., m-1)$:

- $N(j, j+1)$: the number of requests that cover segment $[j, j+1]$;
- $N^*(j)$: the number of requests that cover station j;
- $N^{sup} = \max(N(j, j+1) : 0 \leq j \leq m-1)$. N^{sup} is the maximum number of pairwise overlapping requests.

Note that $N^*(j) \leq N(j, j+1)$ and, thus, also, $N^*(j) \leq N^{sup}$ for $j = 0, ..., m-1$.

The pseudo-code of the algorithm for solving problem $1,1|sd,u|CLT$ is provided in Algorithm 1.

Algorithm 1 Solution algorithm for problem $1,1|sd,u|CLT$

1: **for all** $j \in \{0,\dots,m-1\}$ **do**
2: compute $N(j,j+1)$ and $N^*(j)$
3: **end for**
4: compute N^{sup}
5: $\mathcal{R}^{(1)} \leftarrow \mathcal{R}$
6: **for all** $j \in \{0,\dots,m-1\}$ **do**
7: add $N^{sup} - N(j,j+1)$ copies of $(j,j+1)$ to $R^{(1)}$
8: **end for**
9: $\mathcal{R}^{(2)} \leftarrow \mathcal{R}^{(1)}$
10: **if** $N^*(0) = N^{sup}$ **then**
11: **for all** $j \in \{0,\dots,m-1\}$ **do**
12: add $(j,j+1)$ to $\mathcal{R}^{(2)}$
13: **end for**
14: **end if**
15: $\mathcal{G} \leftarrow (\{0,\dots,m-1\},\mathcal{R}^{(2)})$
16: $\{(C_1,n_1^{inf})...,(C_P,n_P^{inf})\} \leftarrow decomposeEulerian(\mathcal{G})$
17: $\Lambda \leftarrow C_1$
18: **if** $P > 1$ **then**
19: **for all** $p \in \{2,\dots,P\}$ **do**
20: add $(n_{p-1}^{inf},n_p^{inf})$ to Λ
21: add C_p to Λ
22: **end for**
23: add (n_P^{inf},n_1^{inf}) to Λ
24: **end if**
25: **return** Λ

In Lines 1 to 4, we first compute values $N(j,j+1)$ and $N^*(j)$ for $j = 0,\dots,m-1$. Then, we compute N^{sup}.

Lines 5 to 8 complete the multiset of requests so that the number of requests covering every segment $[j,j+1]$ is exactly N^{sup}. To this end, artifical requests $(j,j+1)$ are added. We denote $\mathcal{R}^{(1)}$ the new multiset. We will see in Lemma 2 that adding these requests does not change the optimal solution value: $z(\mathcal{R}^{(1)}) = z(\mathcal{R})$.

If the number of requests covering station 0 is equal to N^{sup}, we modify again the request multiset in Lines 9 to 14. The new multiset is called $\mathcal{R}^{(2)}$. In addition to $\mathcal{R}^{(1)}$, it contains a

request $(j, j+1)$ for every segment $[j, j+1]$. Again, we will see in Lemma 4 that $z(\mathcal{R}^{(2)}) = z(\mathcal{R}^{(1)})$.

The next step consists in introducing a directed multigraph $\mathcal{G}$ (Line 15). In this graph, the vertex set is the set of stations. An arc is added between two stations j_1 and j_2 for each request (j_1, j_2) in the extended multiset of requests. A cost δ_{j_1,j_2} is defined for each arc (j_1, j_2). We will prove in Lemma 5 that graph $\mathcal{G}$ is semi-Eulerian.

Semi-Eulerian graphs can be exhaustively decomposed into a set of Eulerian circuits with disjoint vertices (see Wahlström (2018), in which these kinds of graphs are called *balanced*). Procedure *decomposeEulerian*$(\mathcal{G})$ at Line 16 executes this decomposition. It results in a set of P circuits C_1 to C_P. In addition, it computes station n_p^{inf} of minimal index in every circuit C_p. Without loss of generality, we assume $n_1^{inf} < \cdots < n_P^{inf}$.

If $P = 1$, that is, if graph $\mathcal{G}$ is Eulerian, the procedure returns the schedule defined by circuit C_1 (Lines 17 and 25). Note that the schedule corresponds to the sequence in which requests are served which, in turn, is the sequence in which the corresponding arcs are traversed in the Eulerian graph. Otherwise, in Lines 17 to 24, the algorithm connects the P circuits using arcs $(n_p^{inf}, n_{p+1}^{inf})$ to obtain a single circuit Λ. Then, it returns the schedule obtained from this circuit (Line 25). We prove in Theorem 1 that in both cases the schedules are optimal.

We now prove the various lemmas that are needed to prove the main result in Theorem 1. In what follows, multisets $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$, graph $\mathcal{G}$ and circuit Λ are those obtained from Algorithm 1. Two illustrative examples follow.

Lemma 1. $T_{OPT} \geq N^{sup}$.

Proof. Since we consider a vehicle of capacity one, conflicting requests must be served in different tours. The N^{sup} requests that cover the same segment must then be active on N^{sup} different tours. It follows that the number of tour in any feasible solution is at least N^{sup}. □

Lemma 2. *Multiset $\mathcal{R}^{(1)}$ is such that $z(\mathcal{R}^{(1)}) = z(\mathcal{R})$.*

Proof. We call $\Lambda^*(\mathcal{R})$ the optimal schedule when the request set is $\mathcal{R}$. From Lemma 1, we know that the vehicle performs at least N^{sup} tours in $\Lambda^*(\mathcal{R})$. Furthermore, we know that every segment $[j, j+1]$ is covered exactly $N(j, j+1)$ times in request multiset $\mathcal{R}$. It means that the vehicle is not active on at least $N^{sup} - N(j, j+1)$ tours in schedule $\Lambda^*(\mathcal{R})$ when it traverses segment $[j, j+1]$. A feasible schedule for request multiset $\mathcal{R}^{(1)}$ can be constructed from $\Lambda^*(\mathcal{R})$ by serving the new requests $(j, j+1)$ when the vehicle is not active. This schedule has the same cost than $\Lambda^*(\mathcal{R})$, that is, $z(\mathcal{R})$. It follows that $z(\mathcal{R}^{(1)}) \leq z(\mathcal{R})$. Trivially, as $\mathcal{R} \subseteq \mathcal{R}^{(1)}$, we have that $z(\mathcal{R}^{(1)}) \geq z(\mathcal{R})$. Thus, $z(\mathcal{R}^{(1)}) = z(\mathcal{R})$. □

Lemma 3. *If $N^*(0) = N^{sup}$, $T_{OPT} \geq N^{sup} + 1$.*

Proof. If $N^*(0) = N^{sup}$, N^{sup} requests cover the depot and have to be active at the depot on different tours. Given a feasible solution, the first active request at the depot cannot be finished before tour number 2, the second before tour number 3, and, by a simple induction, active request number N^{sup} before tour number $N^{sup} + 1$. It implies $T_{OPT} \geq N^{sup} + 1$. □

Lemma 4. *Multisets $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ are such that $z(\mathcal{R}^{(2)}) = z(\mathcal{R}^{(1)})$.*

Proof. If $N^*(0) < N^{sup}$, $\mathcal{R}^{(2)} = \mathcal{R}^{(1)}$. Otherwise, we can follow exactly the same proof as in Lemma 2. With request multiset $\mathcal{R}^{(2)}$, we know, thanks to Lemma 3, that the vehicle performs at least $N^{sup} + 1$ tours. So, compared to $\mathcal{R}^{(1)}$, an additional request can be added for each segment $[j, j + 1]$ without increasing the solution cost. □

Lemma 5. *Graph $\mathcal{G}$ is semi-Eulerian.*

Proof. For each node j of graph $\mathcal{G}$, let $d_G^+(j)$ be the number of outgoing arcs and $d_G^-(j)$ be the number of ingoing arcs. To show that the graph is semi-Eulerian, we prove $d_G^+(j) = d_G^-(j)$ for $j = 0, \ldots, m - 1$. We know that every segment $[j, j + 1]$ is covered by the same number N of requests from $\mathcal{R}^{(2)}$ ($N = N^{sup}$ or $N^{sup} + 1$, depending on condition $N^*(0) < N^{sup}$). Let us consider $j \in \{0, \ldots, m - 1\}$, $d_G^+(j) + N^*(j) = N = d_G^-(j) + N^*(j)$, which proves the lemma. □

Lemma 6. *If $N^*(0) = N^{sup}$, graph $\mathcal{G}$ is Eulerian.*

Proof. If $N^*(0) = N^{sup}$, $\mathcal{R}^{(2)}$ contains at least once each request $(j, j+1)$ for $j = 0, \ldots, m-1$. These requests create a directed circuit in $\mathcal{G}$ containing all the nodes. Graph $\mathcal{G}$ is then strongly connected. As Lemma 5 states that $\mathcal{G}$ is semi-Eulerian, then $\mathcal{G}$ is Eulerian. □

Theorem 1. *Schedule Λ is optimal. Three cases can be distinguished:*

1. *If $N^*(0) = N^{sup}$, $T_{OPT} = N^{sup} + 1$*
2. *If $N^*(0) < N^{sup}$ and graph $\mathcal{G}$ is Eulerian (algorithm decomposeEulerian($\mathcal{G}$) returns a single circuit), $T_{OPT} = N^{sup}$*
3. *If $N^*(0) < N^{sup}$ and graph $\mathcal{G}$ is not Eulerian (algorithm decomposeEulerian($\mathcal{G}$) returns several circuits), $T_{OPT} = N^{sup} + 1$*

Proof. We prove the three cases separately:

1. If $N^*(0) = N^{sup}$, $\mathcal{G}$ is Eulerian. Every segment $[j, j + 1]$ is covered exactly $N^{sup} + 1$ times in $\mathcal{R}^{(2)}$, so the sum of request lengths is $(N^{sup} + 1) \times L$. Circuit Λ contains an arc

for each request in $\mathcal{R}^{(2)}$. Its total length is also $(N^{sup}+1)\times L$. Thus, $T_{OPT} \leq N^{sup}+1$. From Lemma 1, we have $T_{OPT} \geq N^{sup}+1$. This demonstrates that $T_{OPT} = N^{sup}+1$ and that schedule Λ is optimal.

2. If $N^*(0) < N^{sup}$ and graph $\mathcal{G}$ is Eulerian, we can apply exactly the same proof, except that every segment is now covered N^{sup} times. We obtain $T_{OPT} = N^{sup}$ and schedule Λ is optimal.

3. If $N^*(0) < N^{sup}$ and graph $\mathcal{G}$ is not Eulerian, we first show that $T_{OPT} \geq N^{sup}+1$. Assume $T_{OPT} \leq N^{sup}$. From Lemma 1 it means $T_{OPT} = N^{sup}$, that is, the optimal schedule exactly covers the arcs in $\mathcal{G}$. However, this contradicts the fact that $\mathcal{G}$ is not connected. Thus, $T_{OPT} \geq N^{sup}+1$. We now show that schedule Λ is optimal. The total length of this schedule is $N^{sup}\times L + \sum_{1\leq p\leq P-1} \delta_{n_p^{inf},n_{p+1}^{inf}} + \delta_{n_P^{inf},n_1^{inf}} = N^{sup}\times L + L = (N^{sup}+1)\times L$. Thus, the number of tours is $N^{sup}+1$ and it is optimal.

□

Using Hierholzer's algorithm , procedure $decomposeEulerian(\mathcal{G})$ can be implemented with a complexity $O(N^{sup}\times m)$ (see, for example, Jungnickel (2013)). The different loops of the algorithm (to compute values $N(j,j+1)$ and $N^*(j)$, to construct $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$, to obtain Λ) all have either the same complexity or a lower complexity. The overall complexity of the algorithm is thus $O(N^{sup}\times m)$. Seeing that $N^{sup}\leq n$, it proves that problem $1,1|sd,u|CLT$ is polyniomally solvable.

We illustrate the algorithm with two simple examples.

Example 1

Let us consider a ring with $m=5$ equidistant stations (numbered from 0 to 4) and four requests: $\mathcal{R} = \{(4,2),(2,3),(1,3),(3,1)\}$ (see Figure 3). On this example:

- $N(0,1) = N(1,2) = N(2,3) = N(4,0) = 2$, $N(3,4) = 1$, $N^{sup} = 2$ and $N^*(0) = 2$
- $\mathcal{R}^{(1)} = \mathcal{R}\cup\{(3,4)\}$ (see Figure 4)
- $\mathcal{R}^{(2)} = \mathcal{R}^{(1)}\cup\{(0,1),(1,2),(2,3),(3,4),(4,0)\}$ (see Figure 4)

Based on multiset $\mathcal{R}^{(2)}$, we obtain graph $\mathcal{G}$ represented on Figure 5. Applying Hierholzer's algorithm provides a single optimal circuit (the graph is Eulerian), *e.g.*: $0\to1\to2\to3\to1\to3\to4\to2\to3\to4\to0$. This circuit gives the schedule $(2,3),(3,1),(1,3),(4,2)$ which is completed in three tours on the ring.

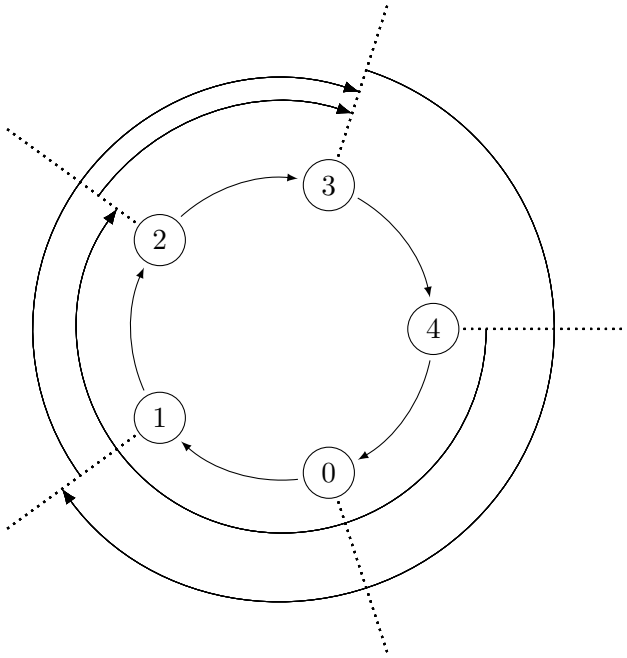
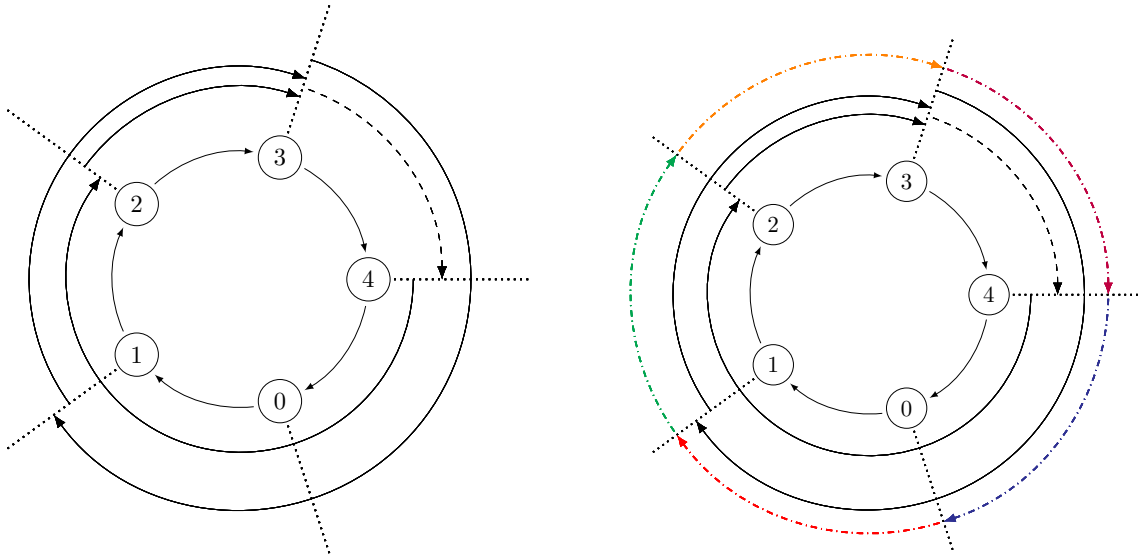


Figure 3: Example 1: Ring and initial requests ($\mathcal{R}$)



(a) Request multiset $\mathcal{R}^{(1)}$

(b) Request multiset $\mathcal{R}^{(2)}$

Figure 4: Addition of new requests. The dashed arc denotes the request added in lines 6-8, while the dash-dotted arcs denote the requests added in lines 9-14 of Algorithm 1.

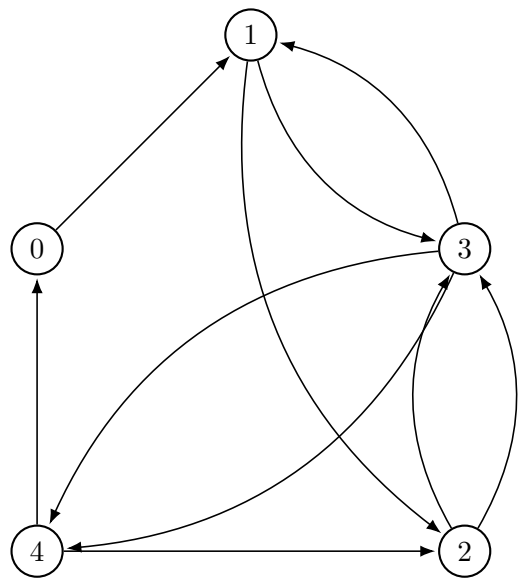


Figure 5: Graph $\mathcal{G}$ of Example 1

Example 2

We consider the same ring and a new request set: $R = \{(0,4),(4,0)(1,3),(3,1)\}$ (see Figure 6):

- $N(0,1) = N(1,2) = N(2,3) = N(3,4) = N(4,0) = 2$, $N^{sup} = 2$ and $N^*(0) = 1$
- no new requests are added: $\mathcal{R}^{(2)} = \mathcal{R}^{(1)} = \mathcal{R}$

Graph $\mathcal{G}$ is depicted on Figure A.13. Applying Hierholzer's algorithm provides two Eulerian circuits $C_1 = 0 \to 4 \to 0$ and $C_2 = 1 \to 3 \to 1$. These two circuits are reconnected with arcs $(0,1)$ and $(1,0)$ to form $\Lambda = 0 \to 4 \to 0 \to 1 \to 3 \to 1 \to 0$. The vehicle performs 3 tours and the schedule is $(0,4),(4,0),(1,3),(3,1)$.

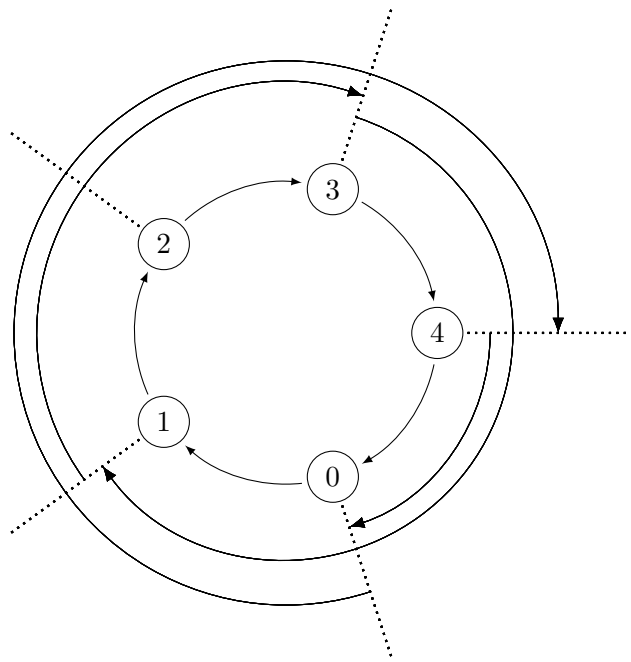


Figure 6: Example 2: Ring and initial requests ($\mathcal{R}$)

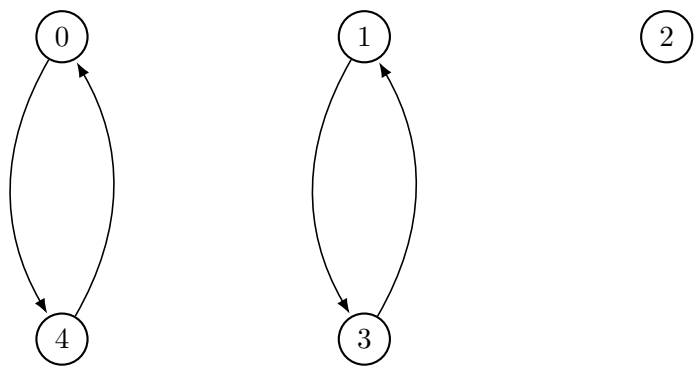


Figure 7: Graph $\mathcal{G}$ of Example 2

3.2. Problems $\alpha|sd, u|CLT$, with $\alpha = V, 1$ or $1, Q$ or V, Q or V, Q_v

In this section we demonstrate that different generalizations of problem $1, 1|sd, u|CLT$ are NP-hard.

We first show that the problem with multiple vehicles of capacity 1 is NP-hard (problem $V, 1|sd, u|CLT$). Because of the length of the proof, the proof is omitted here and can be found in the Appendix. We then show that this problem is equivalent to the problem faced when the fleet is composed of a single vehicle of non-unitary capacity (problem $1, Q|sd, u|CLT$), which demonstrates that $1, Q|sd, u|CLT$ is also NP-hard. These results show that $V, Q|sd, u|CLT$ and $V, Q_v|sd, u|CLT$, that generalize the two others, are NP-hard.

Theorem 2. *Problem $V, 1|sd, u|CLT$ is NP-hard.*

Proof. See proof in Appendix. □

Theorem 3. *Problem* $1, Q|sd, u|CLT$ *is NP-hard.*

Proof. We consider an instance of $1, Q|sd, u|CLT$ and the equivalent instance of $V, 1|sd, u|CLT$ where the vehicle of capacity Q is replaced by V vehicles of capacity 1, with $V = Q$. To prove that $1, Q|sd, u|CLT$ is NP-hard, we show that any feasible solution $\Lambda^{1,Q}$ of the former problem can be transformed to a same-cost feasible solution $\Lambda^{V,1}$ of the latter, and vice-versa.

Considering a feasible schedule $\Lambda^{1,Q}$, we assign a number v_i between 1 and Q to all requests $i \in \mathcal{R}$, so that two requests active at the same time have different numbers. This numbering always exists as, when a request starts, at most $Q - 1$ other requests are active and so, a number between 1 and Q is available. Then, we build schedule $\Lambda^{V,1}$ by starting all requests as in $\Lambda^{1,Q}$ and by assigning every request to vehicle number v_i. The schedule is feasible because two requests with the same number are not active at the same time and because $v_i \leq V$ for all requests. Both schedules $\Lambda^{1,Q}$ and $\Lambda^{V,1}$ are feasible for their respective problem and have the same closing time. Starting with a feasible schedule $\Lambda^{V,1}$, the reverse transformation can be applied exactly the same, with the same conclusion. The two problems having the same set of feasible solutions with equivalent costs, they have the same complexity. It permits to conclude that problem $1, Q|sd, u|CLT$ is NP-hard. □

3.3. An integer linear programming formulation for problem $V, Q|sd, u|CLT$

As problem $V, Q|sd, u|CLT$ is NP-hard, we propose a mathematical formulation that could allow solving it. The formulation makes use of the following notation, in addition to the one introduced in previous sections.

- K: maximum tour number at which a request should be started
- $\mathcal{R}[j, j+1]$: set of requests in $\mathcal{R}$ that cover segment $[j, j+1]$.
- c_{ik}: cost of inserting request $i \in \mathcal{R}$ if it is started in tour number k

$$c_{ik} = \begin{cases} k & \text{if } s_i < t_i \\ k+1 & \text{otherwise} \end{cases}$$

To calculate K, finding a feasible solution is enough. Requests can be ordered in the increasing order of their pickup station, and started in consecutive tours. Then, the last request would start at tour n. We set $K = n$.

We introduce the following decision variables:

$$x_{ik} = \begin{cases} 1 & \text{if request } i \text{ is started in tour k} \\ 0 & \text{otherwise} \end{cases} \qquad (1 \leq i \leq n, 1 \leq k \leq K)$$

$$CLT = \text{closing time.}$$

Note that that the closing time corresponds to the maximum number of tours traversed by each vehicle multiplied by the length of the ring L. As L is a constant, then minimizing CLT corresponds to minimizing the the maximum number of tours traversed by each vehicle. Thus, in the following we refer to CLT as the latter number.

The mixed integer linear program is then:

$$\min CLT \tag{2}$$

s.t.:

$$\sum_{\{i \in \mathcal{R}[j,j+1], s_i \leq j\}} x_{ik} + \sum_{\{i \in \mathcal{R}[j,j+1], s_i \geq j+1\}} x_{ik-1} \leq V \times Q \quad (0 \leq j \leq m-1,\ 1 \leq k \leq K) \tag{3}$$

$$\sum_{k \in \mathcal{K}} x_{ik} = 1 \quad (1 \leq i \leq n) \tag{4}$$

$$CLT \geq \sum_{k \in \mathcal{K}} c_{ik} x_{ik} \quad (1 \leq i \leq n) \tag{5}$$

$$x_{ik} \in \{0, 1\} \quad (1 \leq i \leq n,\ 1 \leq k \leq K) \tag{6}$$

$$CLT \geq 0 \tag{7}$$

The objective function minimizes the closing time, expressed in number of tours. Constraints (3) make sure that the number of active requests never exceeds the total capacity of the V vehicles. For each segment $[j, j+1]$ and each tour k, this number is evaluated by counting the active requests that started in the tour before station j and those that started in the preceding tour after station $j+1$. Constraints (4) make sure that every request is served. Constraints (5) compute the maximal tour number CLT. Constraints (6)-(7) define decision variables.

4. Problems with unitary requests and release/due dates

In this section, we analyze the class of problems where each request $i \in R$ is unitary and is subject to a release date r_i and a due date d_i. Note that in the context of the single-direction (sd) constraint, release dates and due dates can be expressed as the smallest and the highest tour number in which the request can be served.

4.1. Problem $1,1|sd,u,r_i,d_i|CLT$

This section is devoted to the problem where a single vehicle of capacity one has to serve the requests. The problem is the same as in Section 3.1 with the addition of release dates and due dates. We show that, when having release dates and due dates, the problems

becomes NP-hard. We proceed by reduction from a variant of the list coloring problem, called (γ, μ)-coloring problem, introduced Bonomo et al. (2009) and shown to be NP-hard.

Definition 8. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and functions $\gamma, \mu : \mathcal{V} \to \mathbb{N}$ such that $\gamma(v) \leq \mu(v)$ for every $v \in \mathcal{V}$, we say that $\mathcal{G}$ is (γ, μ)-colorable if there exists a function $f : \mathcal{V} \to \mathbb{N}$ of $\mathcal{G}$ such that $\gamma(v) \leq f(v) \leq \mu(v)$ for every $v \in \mathcal{V}$, and $f(v) \neq f(v')$ for every $(v, v') \in \mathcal{E}$.*

In Bonomo et al. (2009) it is shown that this problem generalizes the graph coloring problem and is NP-hard, also in the case where the underlying graph is an interval graph.

Theorem 4. *Problem $1, 1|sd, u, r_i, d_i|CLT$ is NP-hard.*

Proof. The main argument in the proof is that the constraints implied by release dates and due dates can be equivalently expressed as constraints on the lowest and largest tour number on which a request can be started. The starting tour of a request can then be interpreted as a color, bounded by these two limits. Given that fact, we first show how to transform an instance of the (γ, μ)-coloring problem on an interval graph into an instance of $1, 1|sd, u, r_i, d_i|CLT$.

We consider an instance I_c of the (γ, μ)-coloring problem on an interval graph. $\mathcal{A} = \{(a_1, b_1), ..., (a_n, b_n)\}$ is a set of n intervals on a real line with $a < b$ for each $(a, b) \in \mathcal{A}$. An interval graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is constructed from $\mathcal{A}$ by introducing a vertex v_i for each interval $(a_i, b_i) \in \mathcal{A}$ and by adding an edge (v_i, v_j) each time intervals (a_i, b_i) and (a_j, b_j) overlap. Let γ and μ be two functions that map vertices in $\mathcal{V}$ into natural numbers, such that $\gamma(v) \leq \mu(v)$ for each $v \in \mathcal{V}$. Instance I_c consists in deciding if there exists a coloring function $f : \mathcal{V} \to \mathbb{N}$ such that $\gamma(v) \leq f(v) \leq \mu(v)$ for each $v \in \mathcal{V}$ and $f(v) \neq f(v')$ for $(v, v') \in \mathcal{E}$.

We build an instance I_s of $1, 1|sd, u, r_i, d_i|CLT$ as follows. Let $\mathcal{D}$ denote the set of interval extremities: $\mathcal{D} = \{a_i, b_i : i = 1, ..., n\}$. We sort $\mathcal{D}$ in increasing order and introduce a station in the ring for every element in $\mathcal{D}$, in this order. We complete the ring with station 0. This way, the number m of stations is at most two times the number n of intervals plus the depot ($m \leq 2n + 1$). We define unitary distances between successive stations, which gives a tour length $L = |\mathcal{D}| + 1$. We introduce a request in $\mathcal{R}$ for every interval in $\mathcal{A}$. Given interval (a_i, b_i), request i is defined as follows: s_i is the station obtained from extremity a_i, t_i is the station obtained from extremity b_i, r_i is set to $L \times \gamma(v_i)$ and $d_i = L \times (\mu(v_i) + 1))$. Basically, this means that request i has to be active between the beginning of tour number $\gamma(v_i)$ and the end of tour number $\mu(v_i)$. We note that no request covers the depot, so two requests will be scheduled in the same tour if and only if they do not overlap. Note also that this construction is polynomial.

We claim that $\mathcal{H}$ is (γ, μ) colorable if and only if I_s admits a feasible schedule. A feasible schedule for I_s is a function $f : \mathcal{R} \to \mathbb{N}$ that assigns each request $i \in \mathcal{R}$ to a tour (that is the tour in which the request is served) so that requests do not overlap and that the tour number of request i lies between the lowest possible tour number and highest possible tour number given by the release and due dates, that is, $\gamma(v_i)$ and $\mu(v_i)$. More formally, function f has to be such that:

- $f(i_1) = f(i_2) \implies i_1$ and i_2 do not overlap $\forall i_1, i_2 \in \mathcal{R}$
- $\gamma(v_i) \leq f(i) \leq \mu(v_i)$ $\forall i \in \mathcal{R}$

Assume first that $\mathcal{H}$ is (γ, μ)-colorable. This means that a function $g : V \to \mathbb{N}$ is a coloring for $\mathcal{H}$. Let $f : R \to \mathbb{N}$ be a function such that $f(i) = g(v_i)$, $v_i \in V$. It is easy to see that f is a feasible schedule for I_s.

On the other hand, assume that I_s admits a feasible schedule f. Let $g : \mathcal{V} \to \mathbb{R}$ be a function such that $g(v_i) = f(i)$, $(a_i, b_i) \in \mathcal{A}$. It is easy to see that g is a coloring function for $\mathcal{H}$. In fact, $g(v_{i_1}) = g(v_{i_2})$ means that requests i_1 and i_2 do not overlap, *i.e.,* edge $(v_{i_1}, v_{i_2}) \notin \mathcal{E}$. In addition, from the definition of r_i and d_i, $\gamma(v_i) \leq g(v_i) \leq \mu(v_i)$.

It shows that solving the (γ, μ)-coloring problem on an interval graph amounts to finding a feasible solution to an instance of problem $1, 1|sd, u, r_i, d_i|CLT$. This proves that problem $1, 1|sd, u, r_i, d_i|CLT$ is NP-hard. □

Given the result of Theorem 4, it follows that all problems with release and due dates, monodirectional and where the objective function is the minimization of CLT are NP-Hard.

In the following section, we present a mathematical formulation for problem $V, Q|sd, u, r_i, d_i|CLT$ which can be used to solve also problems $\alpha|sd, u, r_i, d_i|CLT$, with $\alpha = 1, 1$ or $1, Q$ or $V, 1$, and that could be easily adapted to solve problems with heterogeneous fleet of vehicles.

4.2. Integer linear programming formulation for problem $V, Q|sd, u, r_i, d_i|CLT$

In addition to notation defined in previous sections, we define $\mathcal{K}_i$ as the set of tour numbers in which request i can be started

$$\mathcal{K}_i = \{k \in \mathbb{N} : (k-1)L + \delta_{0,s_i} \geq r_i \text{ and } (k-1)L + \delta_{0,s_i} + \delta_{s_i,t_i} \leq d_i\}$$

Also, we tighten K, that is, the tour number upper bound for the last started request, by taking account of due dates:

$$K = \max_{1 \leq i \leq n} \left(\left\lceil \frac{d_i}{L} \right\rceil \right)$$

Decision variables are the same as in model (2)-(7). The formulation is given by (2)-(3), (5)-(7) and the following modification of constraints (4):

$$\sum_{k \in \mathcal{K}_i} x_{ik} = 1 \qquad (1 \leq i \leq n) \tag{8}$$

Constraints (8) make sure that every request is served and that release and due dates are respected.

5. Problems with non-unitary requests

In this section, we consider the extension of previous problems to non-unitary requests. Note that in this context, the case $Q = 1$ does not make sense. Remembering that $1, Q|sd, u|CLT$ is NP-hard (see Section 3.2), we can conclude that all problems investigated in this section are NP-hard.

5.1. Alternative complexity proof for problem $1, Q|sd|CLT$

Though theorems 2 and 3 show that problem $1, Q|sd|CLT$ is NP-hard, the proof appears overly complex when requests are not necessarily unitary. In this section, we propose a simpler proof based on a reduction from the Bin Packing Problem (BPP).

The BPP is defined as follows. We are given a finite set $\mathcal{O}$ of n objects and K bins. Let $W \in \mathbb{Z}^+$ be the bin capacity and w_o be the weight of each object $o \in \mathcal{O}$, with $w_o \leq W$. The problem consists in determining whether a partition of $\mathcal{O}$ in K disjoint subsets $\mathcal{O}_1, \ldots, \mathcal{O}_K$ exists such that the sum of item weights in each subset $\mathcal{O}_k$ is at most W. This problem is NP-complete.

Theorem 5. $1, Q|sd|CLT$ *is NP-hard*

Proof. We consider the decision version of problem $1, Q|sd|CLT$, *i.e.*, the problem of determining, given an integer C, whether a feasible schedule exists with $CLT \leq C$.

Let us consider an instance I_B for the BPP, with a set $\mathcal{O}$ of objects and K bins. We construct an instance I_S for $1, Q|sd|CLT$ as follows. We define a ring with two stations $\mathcal{S} = \{0, 1\}$. For each object $o \in \mathcal{O}$, we introduce a request i in $\mathcal{R}$, with $s_i = 0$, $t_i = 1$ and $q_i = w_o$. The capacity Q of the vehicle is set to bin capacity W. We finally set $C = K$. In instance I_S, all requests overlap, hence, a schedule Λ is feasible if and only if:

- in each tour, the sum of the demands of the requests started in the tour is not greater than Q,
- the schedule contains at most C tours.

We show that I_B is feasible if and only if I_S is. Assume first I_B feasible. We build a schedule Λ for I_S by executing the requests of the K bins in K separate tours. Conversely, let us assume I_S feasible. We consider a feasible schedule Λ. We build a solution for I_B by assigning all requests starting in the same tour to the same bin. This shows that solving the BPP can be reduced to solving the decision version of $1, Q|sd|CLT$, with a polynomial reduction, which proves that $1, Q|sd|CLT$ is NP-hard. □

5.2. An integer linear programming model

In this section, we present a mathematical formulation for problem $V, Q|sd, r_i, d_i|CLT$. This formulation can also be used to solve problems $1, 1|sd, r_i, d_i|CLT$, $1, Q|sd, r_i, d_i|CLT$ and $V, 1|sd, r_i, d_i|CLT$.

This formulation extends the formulations presented in Section 3.3 and 4.2. We use the notation introduced in these sections. Tour assignment variables takes a third index to take into account the vehicle that serves a request:

$$x_{ikv} = \begin{cases} 1 & \text{if request } i \text{ is started in tour } k \text{ by vehicle } v \\ 0 & \text{otherwise} \end{cases}$$

The mixed integer linear program is then:

$$\min CLT \tag{9}$$

s.t.:

$$\sum_{\{i \in \mathcal{R}[j,j+1], s_i \leq j\}} q_i x_{ikv} + \sum_{\{i \in \mathcal{R}[j,j+1], s_i \geq j+1\}} q_i x_{ik-1v} \leq Q \quad (0 \leq j \leq m-1,\ 1 \leq k \leq K,\ 1 \leq v \leq V) \tag{10}$$

$$\sum_{v \in \{1,\ldots,V\}} \sum_{k \in \mathcal{K}_i} x_{ikv} = 1 \qquad (1 \leq i \leq n) \tag{11}$$

$$CLT \geq \sum_{v \in \{1,\ldots,V\}} \sum_{k \in \mathcal{K}_i} c_{ik} x_{ikv} \qquad (1 \leq i \leq n) \tag{12}$$

$$x_{ikv} \in \{0, 1\} \qquad (1 \leq i \leq n,\ 1 \leq k \leq K,\ 1 \leq v \leq V) \tag{13}$$

$$CLT \geq 0 \tag{14}$$

The objective function minimizes the closing time, expressed as number of tours, over all vehicles. Constraints (10) ensure that vehicle capacities are satisfied. They disaggregate constraints (3) for each vehicle. Constraints (11) make sure that every requests is served, at

an acceptable time. Constraints (12) compute CLT. Constraints (13)-(14) define variables domain.

We propose the following symmetry breaking constraints to strengthen formulation (9)–(14). The first set of constraints ranks vehicles according to the number of satisfied requests: the smallest the vehicle index, the largest the number of satisfied requests

$$\sum_{i=1}^{n} \sum_{k \in \mathcal{K}_i} x_{ikv} \geq \sum_{i=1}^{n} \sum_{k \in \mathcal{K}_i} x_{ikv+1} \qquad (1 \leq v \leq V-1). \tag{15}$$

The second set imposes that the first requests are served by the first vehicles: the i first requests have to be assigned to vehicles in set $\{1, \ldots, i\}$:

$$\sum_{u \in \{1,\ldots,j\}} \sum_{v \in \{1,\ldots,j\}} \sum_{k \in \mathcal{K}_i} x_{ukv} = j \qquad (1 \leq j \leq \min(V, n)). \tag{16}$$

Note that (15) and (16) cannot be used simultaneously.

We evaluated computationally the impact of these constraints and noticed a slight improvement, in terms of computing times, over the formulation without symmetry breaking constraints. In particular, the most effective constraints are the ones given by inequalities (16). Thus, the computational results presented in the following include these constraints.

6. Computational tests

We now present the set of experiments we made in order to evaluate the efficacy of the formulations presented above. The formulations are solved through CPLEX 12.9.0 on a computer equipped with an Intel Core i7-9700 processor and 32GB of RAM. For all instances a time limit of 30 minutes was set. As shown in the results, all instances were solved (either to optimality or by proving infeasibility) within this time limit.

6.1. Instance sets

As far as we know, the problems investigated in this paper are new and no benchmark instances exist. We propose to generate random instances as follows. All random numbers are chosen from uniform distributions.

We consider a ring with $m = 10$ stations. The number of requests is chosen in the set $\{20, 40, 80, 160\}$, and we generated 5 instances for each number of requests. The pickup station of a request is randomly taken in $\{0, \ldots, m-1\}$ and the target station in $\{0, \ldots, m-1\} \setminus \{s_i\}$. For each instance, distances between consecutive stations are randomly drawn in $[1, 10]$. This gives a total of 20 combinations of the parameters mentioned above.

Every instance is replicated several times with different fleet of vehicles and demand values. We consider the following values for V and Q: $V \in \{1, 2, 3\}$, $Q \in \{1, 2, 4, 8\}$. When

demands are not unitary, they are generated in interval $[1, Q]$. For each of the 20 combinations mentioned above, this gives 12 fleet/demand compositions for unitary and non-unitary demands, respectively.

When needed, release dates and due dates are defined based on the following observations. The expected length of a request is approximately $\frac{L}{2}$. The expected demand is approximately $\frac{Q}{2}$ when demands are non-unitary or exactly 1 when they are. So, every request approximately generates an expected workload $\frac{L}{2} \times \frac{Q}{2}$ in unit-of-distance $\times$ unit-of-demand for non-unitary demands and an expected workload $\frac{L}{2}$ otherwise. The total expected workload is thus $\frac{L}{2} \times \frac{Q}{2} \times n$ or $\frac{L}{2} \times n$, respectively. The fleet brings a total capacity $CLT \times V \times Q$ in the same unit. With a capacity fully exploited, CLT should thus approach $\frac{nL}{4V}$ when demands are not unitary, $\frac{nL}{2QV}$ otherwise. We denote as C this value.

Then:

- Release dates are randomly generated in intervals $[0, 1.5C]$ (tight) or $[0, 2C]$ (wide)
- Due dates are generated in intervals $[r_i + L, r_i + L + C]$ (tight) or $[r_i + L, r_i + L + 1.5C]$ (wide)

For the 20×24 combinations above, this gives four alternatives for release and due dates: t-t, t-w, w-t, w-w, where t and w stand for tight and wide, respectively.

For each instance constructed this way, two variants are finally considered: the original instance and the instance where due dates are relaxed. The reason behind the latter is to investigate the impact of due dates on the problem tractability. Also, it might correspond to a realistic situation where, for example, goods have to be moved between stations to prepare the planning for the next day. In this case, goods are ready at a given time at the pickup station (the release date) but there is no restriction on the time at which they should be available at the delivery station.

To calculate value K in the case with release dates and no due date, it suffices to notice that at the beginning of tour $\lceil \frac{2C}{L} \rceil + 1$ all requests are ready to be processed (release dates are passed). Then, the reasoning of Section 3.3 can be applied to set $K = \lceil \frac{2C}{L} \rceil + n$.

6.2. Greedy algorithm

In order to evaluate the benefits of solving the problems to optimality, we designed the following simple greedy algorithm that is easy to implement and that would be easy to understand for practitioners. We then compared the solutions provided by the greedy algorithm with the optimal solutions. The greedy algorithm works as follows.

When a vehicle reaches a station that is the pickup station of at least one request and this request can be started, it is started. If several requests can be started at the same time, ties are broken by giving priority to requests according to the following criteria, taken hierarchically:

- earliest due date first (due date is considered infinite if no due date is defined);
- largest demand first;
- longest distance to reach the delivery node first.

Note that, in case of instances with unit demands and no release/due dates, only the third criterion is used.

6.3. Computational results for $V, Q|sd, u|CLT$

In Table 1 we report results for problem $V, Q|sd, u|CLT$. All results are averaged over the 5 instances with the same value of n, V and Q, considering feasible instances only. Column *CPU(s)* reports the solution time of the formulation in Section 3.3, in seconds. Column *feas* reports the number of feasible instances. Column *grSuccess* reports the number of instances for which the greedy algorithm found a feasible solution. Column T_{OPT} reports the value of the optimal solution (closing time expressed in number of tours). Column *gapGr* gives the average percentage gap of the solutions found with the greedy algorithm with respect to the optimal solution.

n	V	Q	CPU(s)	feas	grSuccess	CLT	gapGr(%)	n	V	Q	CPU(s)	feas	grSuccess	CLT	gapGr(%)
20	1	1	0.1	5	5	13.0	4.5	80	1	1	2.5	5	5	46.4	1.3
20	1	2	0.0	5	5	6.8	5.4	80	1	2	1.2	5	5	23.4	1.6
20	1	4	0.0	5	5	3.8	11.7	80	1	4	0.6	5	5	12.0	1.7
20	1	8	0.0	5	5	2.6	20.0	80	1	8	0.2	5	5	6.4	12.9
20	2	1	0.1	5	5	6.8	8.7	80	2	1	2.5	5	5	23.4	1.6
20	2	2	0.0	5	5	3.8	11.7	80	2	2	1.3	5	5	12.0	5.0
20	2	4	0.0	5	5	2.6	20.0	80	2	4	0.6	5	5	6.4	6.2
20	2	8	0.0	5	5	2.0	0.0	80	2	8	0.4	5	5	3.4	25.0
20	3	1	0.1	5	5	4.8	9.0	80	3	1	3.8	5	5	15.8	3.8
20	3	2	0.0	5	5	3.0	6.7	80	3	2	1.7	5	5	8.2	9.7
20	3	4	0.0	5	5	2.2	10.0	80	3	4	0.8	5	5	4.4	19.0
20	3	8	0.0	5	5	2.0	0.0	80	3	8	0.3	5	5	3.0	6.7
40	1	1	0.3	5	5	25.4	0.7	160	1	1	17.6	5	5	87.4	0.2
40	1	2	0.2	5	5	13.2	6.1	160	1	2	8.2	5	5	44.2	0.9
40	1	4	0.1	5	5	7.0	2.9	160	1	4	5.3	5	5	22.2	2.8
40	1	8	0.1	5	5	4.0	10.0	160	1	8	2.0	5	5	11.4	5.5
40	2	1	0.4	5	5	13.2	4.7	160	2	1	30.9	5	5	44.2	0.5
40	2	2	0.2	5	5	7.0	9.0	160	2	2	19.9	5	5	22.2	1.9
40	2	4	0.1	5	5	4.0	5.0	160	2	4	6.6	5	5	11.4	5.5
40	2	8	0.1	5	5	2.6	20.0	160	2	8	3.3	5	5	6.0	16.7
40	3	1	0.4	5	5	9.0	4.2	160	3	1	39.9	5	5	29.4	2.1
40	3	2	0.2	5	5	5.0	8.0	160	3	2	18.8	5	5	15.0	5.4
40	3	4	0.1	5	5	3.0	13.3	160	3	4	9.9	5	5	7.8	10.4
40	3	8	0.1	5	5	2.0	10.0	160	3	8	4.3	5	5	4.0	25.0

Table 1: Problem $V, Q|sd, u|CLT$

We can observe that the formulation in Section 3.3 is extremely effective. In fact, the computational time is always smaller than 40 seconds. The computational time increases with the number of requests, as expected, and decreases with the vehicle capacity. Also, the solution value decreases as the capacity increases, as expected. The performance of the greedy algorithm highly depends on vehicle capacity: the larger the capacity, the worse is the performance. This might be due to the fact that, when increasing the capacity, the chances of making a bad choice of assignment of requests to vehicles increases. On the other side, we see that the gap decreases with solution values. This was expected: in fact, the relative impact of bad choices made by the greedy algorithm decreases.

6.4. Computational results for $V, Q|sd, u, r_i, d_i|CLT$

Tables 2 and 3 report results for problem $V, Q|sd, u, r_i, d_i|CLT$. The meaning of the column headings is the same as above. "-" indicates that either the instance is infeasible or the greedy algorithm failed in finding a solution. In Table 2, three cases are considered for release and due dates: tight-tight (t-t), tight-wide (t-w) and tight release dates without due

dates (t). The three remaining cases are reported in Table 3.

n	V	Q	t-t					t-w					t				
			CPU(s)	feas	grSuccess	CLT	gapGr	CPU(s)	feas	grSuccess	CLT	gapGr	CPU(s)	feas	grSuccess	CLT	gapGr
20	1	1	0.0	3	0	17.3	-	0.0	4	1	17.0	6.7	0.0	5	5	17.4	7.0
20	1	2	0.0	5	0	9.2	-	0.0	5	0	9.2	-	0.0	5	5	9.2	8.7
20	1	4	0.0	5	0	5.0	-	0.0	5	1	4.8	20.0	0.0	5	5	5.0	12.0
20	1	8	0.0	5	0	3.0	-	0.0	5	2	2.8	25.0	0.0	5	5	3.0	26.7
20	2	1	0.0	5	0	8.6	-	0.0	5	0	9.2	-	0.0	5	5	8.6	9.4
20	2	2	0.0	5	2	4.8	22.5	0.0	5	2	4.6	20.0	0.0	5	5	4.6	22.0
20	2	4	0.0	5	0	3.0	-	0.0	5	0	3.2	-	0.0	5	5	3.0	20.0
20	2	8	0.0	5	0	2.0	-	0.0	5	0	2.0	-	0.0	5	5	2.0	50.0
20	3	1	0.0	5	2	6.2	18.3	0.0	4	2	6.5	0.0	0.0	5	5	6.2	10.7
20	3	2	0.0	5	0	3.6	-	0.0	5	2	3.6	12.5	0.0	5	5	3.6	23.3
20	3	4	0.0	5	0	2.8	-	0.0	5	1	2.8	0.0	0.0	5	5	2.8	10.0
20	3	8	0.0	5	0	2.0	-	0.0	5	0	2.0	-	0.0	5	5	2.0	30.0
40	1	1	0.0	5	0	34.2	-	0.0	5	2	31.2	1.7	0.1	5	5	34.0	3.5
40	1	2	0.0	5	0	16.4	-	0.0	5	0	16.6	-	0.0	5	5	16.4	7.4
40	1	4	0.0	5	0	9.2	-	0.0	5	1	8.6	0.0	0.0	5	5	9.0	4.7
40	1	8	0.0	5	0	5.0	-	0.0	5	0	4.8	-	0.0	5	5	5.0	8.0
40	2	1	0.0	5	0	16.6	-	0.0	5	0	16.4	-	0.1	5	5	16.6	3.8
40	2	2	0.0	5	1	9.0	0.0	0.0	5	3	8.8	7.9	0.0	5	5	9.0	2.2
40	2	4	0.0	5	0	4.8	-	0.0	5	0	5.0	-	0.0	5	5	4.8	13.0
40	2	8	0.0	5	0	3.0	-	0.0	5	0	3.0	-	0.0	5	5	3.0	33.3
40	3	1	0.0	5	0	11.0	-	0.0	5	1	11.6	8.3	0.1	5	5	11.0	7.3
40	3	2	0.0	5	0	6.2	-	0.0	5	0	5.8	-	0.1	5	5	6.2	16.2
40	3	4	0.0	5	0	4.0	-	0.0	5	0	4.0	-	0.1	5	5	4.0	0.0
40	3	8	0.0	5	0	3.0	-	0.0	5	0	3.0	-	0.1	5	5	3.0	0.0
80	1	1	0.1	5	1	61.4	3.2	0.1	4	0	62.0	-	0.2	5	5	61.4	1.3
80	1	2	0.0	5	0	31.4	-	0.0	5	0	31.6	-	0.1	5	5	31.4	2.6
80	1	4	0.0	5	1	16.6	6.3	0.0	5	1	15.6	6.3	0.1	5	5	16.6	4.9
80	1	8	0.0	5	0	9.0	-	0.0	5	1	8.8	0.0	0.1	5	5	9.0	8.9
80	2	1	0.1	5	1	31.4	0.0	0.1	5	0	31.0	-	0.3	5	5	31.4	2.6
80	2	2	0.0	5	0	15.8	-	0.0	5	1	16.2	6.3	0.2	5	5	15.8	5.1
80	2	4	0.0	5	2	8.6	6.3	0.0	5	0	8.8	-	0.2	5	5	8.6	7.2
80	2	8	0.0	5	0	5.0	-	0.0	5	0	5.0	-	0.1	5	5	5.0	16.0
80	3	1	0.1	5	1	21.4	4.8	0.2	5	1	21.4	4.5	0.4	5	5	21.4	3.8
80	3	2	0.0	5	1	11.0	0.0	0.0	5	1	11.0	9.1	0.3	5	5	11.0	7.3
80	3	4	0.0	5	0	6.0	-	0.0	5	0	6.0	-	0.2	5	5	6.0	16.7
80	3	8	0.0	5	0	4.0	-	0.0	5	0	4.0	-	0.2	5	5	4.0	15.0
160	1	1	0.6	5	0	120.6	-	0.8	4	0	121.8	-	1.9	5	5	120.6	1.5
160	1	2	0.1	5	1	61.6	1.6	0.1	5	1	61.2	1.6	0.5	5	5	61.6	1.0
160	1	4	0.0	5	1	30.8	3.3	0.1	5	0	31.0	-	0.3	5	5	30.8	2.0
160	1	8	0.0	5	0	16.0	-	0.0	5	2	16.0	3.1	0.3	5	5	16.0	5.0
160	2	1	0.7	5	0	62.2	-	0.7	5	0	61.0	-	2.2	5	5	62.2	1.0
160	2	2	0.1	5	0	31.2	-	0.2	5	1	31.6	0.0	0.9	5	5	31.2	2.6
160	2	4	0.0	5	1	16.2	6.3	0.1	5	0	16.2	-	0.7	5	5	16.2	3.8
160	2	8	0.0	5	0	9.0	-	0.0	5	0	9.0	-	0.7	5	5	9.0	4.4
160	3	1	0.5	5	0	41.6	-	0.6	5	1	40.8	2.5	2.1	5	5	41.6	0.5
160	3	2	0.1	5	1	20.8	4.8	0.2	5	1	21.2	4.8	1.5	5	5	20.8	4.8
160	3	4	0.1	5	0	11.2	-	0.1	5	1	11.0	9.1	1.2	5	5	11.2	5.5
160	3	8	0.0	5	0	6.0	-	0.0	5	0	6.0	-	1.1	5	5	6.0	16.7

Table 2: Problem $V, Q|rd, u, r_i, d_i|CLT$ (part 1)

n	V	Q	w-t					w-w					w				
			CPU(s)	feas	grSuccess	CLT	gapGr	CPU(s)	feas	grSuccess	CLT	gapGr	CPU(s)	feas	grSuccess	CLT	gapGr
20	1	1	0.0	5	2	20.6	5.0	0.0	5	1	19.4	5.6	0.0	5	5	20.6	4.9
20	1	2	0.0	5	1	11.2	8.3	0.0	5	1	10.6	0.0	0.0	5	5	10.8	9.3
20	1	4	0.0	5	1	5.8	16.7	0.0	5	1	5.8	16.7	0.0	5	5	5.8	14.0
20	1	8	0.0	5	0	4.0	-	0.0	5	0	4.0	-	0.0	5	5	3.8	11.7
20	2	1	0.0	5	1	10.6	10.0	0.0	5	4	10.8	7.0	0.0	5	5	10.6	4.0
20	2	2	0.0	5	1	6.0	16.7	0.0	5	3	5.8	12.2	0.0	5	5	6.0	10.0
20	2	4	0.0	5	0	4.0	-	0.0	5	0	3.8	-	0.0	5	5	4.0	10.0
20	2	8	0.0	5	0	3.0	-	0.0	5	0	2.6	-	0.0	5	5	3.0	0.0
20	3	1	0.0	5	1	7.8	0.0	0.0	5	1	8.0	0.0	0.0	5	5	7.8	7.9
20	3	2	0.0	5	0	4.4	-	0.0	5	1	4.4	0.0	0.0	5	5	4.4	20.0
20	3	4	0.0	5	0	2.6	-	0.0	5	0	3.0	-	0.0	5	5	2.6	20.0
20	3	8	0.0	5	0	2.0	-	0.0	5	0	2.0	-	0.0	5	5	2.0	50.0
40	1	1	0.0	5	0	39.2	-	0.0	5	0	41.4	-	0.0	5	5	39.2	2.5
40	1	2	0.0	5	0	20.6	-	0.0	5	1	20.8	4.8	0.0	5	5	20.6	4.9
40	1	4	0.0	5	1	11.0	9.1	0.0	5	2	10.8	5.0	0.0	5	5	11.0	3.6
40	1	8	0.0	5	0	6.0	-	0.0	5	1	6.0	16.7	0.0	5	5	6.0	13.3
40	2	1	0.0	5	1	20.8	4.8	0.0	5	1	20.4	4.8	0.1	5	5	20.8	3.9
40	2	2	0.0	5	0	10.6	-	0.0	5	2	10.6	9.5	0.0	5	5	10.6	4.0
40	2	4	0.0	5	2	6.0	8.3	0.0	5	1	6.0	0.0	0.0	5	5	6.0	13.3
40	2	8	0.0	5	0	4.0	-	0.0	5	0	4.0	-	0.0	5	5	4.0	0.0
40	3	1	0.0	5	1	14.6	0.0	0.0	5	1	14.4	0.0	0.1	5	5	14.6	2.8
40	3	2	0.0	5	0	8.0	-	0.0	5	1	7.8	12.5	0.1	5	5	8.0	2.5
40	3	4	0.0	5	0	4.4	-	0.0	5	1	4.8	0.0	0.1	5	5	4.4	10.0
40	3	8	0.0	5	0	3.0	-	0.0	5	0	3.0	-	0.1	5	5	3.0	6.7
80	1	1	0.1	5	2	80.2	0.6	0.1	5	4	80.6	1.2	0.1	5	5	80.2	0.8
80	1	2	0.0	5	1	40.4	0.0	0.0	5	4	41.0	1.2	0.1	5	5	40.4	1.5
80	1	4	0.0	5	0	20.4	-	0.0	5	1	20.8	0.0	0.1	5	5	20.4	4.9
80	1	8	0.0	5	0	10.6	-	0.0	5	1	11.0	9.1	0.1	5	5	10.6	7.6
80	2	1	0.1	5	1	41.0	4.9	0.1	5	2	40.8	1.2	0.3	5	5	41.0	2.0
80	2	2	0.0	5	1	21.0	0.0	0.0	5	2	20.8	2.5	0.2	5	5	21.0	2.9
80	2	4	0.0	5	0	10.8	-	0.0	5	0	11.0	-	0.1	5	5	10.8	7.5
80	2	8	0.0	5	0	6.0	-	0.0	5	1	6.0	16.7	0.1	5	5	6.0	16.7
80	3	1	0.1	5	0	27.6	-	0.1	5	2	28.0	1.8	0.4	5	5	27.6	2.2
80	3	2	0.0	5	1	14.8	0.0	0.0	5	2	14.6	3.6	0.2	5	5	14.8	0.0
80	3	4	0.0	5	1	7.8	12.5	0.0	5	0	8.0	-	0.2	5	5	7.8	12.9
80	3	8	0.0	5	0	5.0	-	0.0	5	0	5.0	-	0.2	5	5	5.0	4.0
160	1	1	0.3	5	1	160.2	0.0	0.3	5	2	160.6	0.9	0.6	5	5	160.2	0.5
160	1	2	0.1	5	0	80.6	-	0.1	5	1	80.8	0.0	0.4	5	5	80.6	0.7
160	1	4	0.0	5	0	40.8	-	0.1	5	3	41.0	1.6	0.3	5	5	40.8	1.0
160	1	8	0.0	5	0	20.8	-	0.0	5	1	21.0	4.8	0.3	5	5	20.8	3.9
160	2	1	0.2	5	0	81.0	-	0.3	5	2	80.2	1.3	1.2	5	5	81.0	0.7
160	2	2	0.1	5	1	41.0	0.0	0.1	5	1	40.8	0.0	0.8	5	5	41.0	1.0
160	2	4	0.0	5	1	21.0	4.8	0.0	5	0	20.4	-	0.8	5	5	21.0	2.9
160	2	8	0.0	5	0	11.0	-	0.0	5	1	11.0	9.1	0.7	5	5	11.0	7.3
160	3	1	0.2	5	0	54.6	-	0.3	5	2	54.0	1.9	1.5	5	5	54.6	0.4
160	3	2	0.1	5	0	27.8	-	0.1	5	2	28.0	1.8	1.3	5	5	27.8	1.5
160	3	4	0.0	5	0	14.8	-	0.1	5	0	15.0	-	1.1	5	5	14.8	1.4
160	3	8	0.0	5	0	8.0	-	0.0	5	0	8.0	-	1.1	5	5	8.0	5.0

Table 3: Problem $V, Q|rd, u, r_i, d_i|CLT$ (part 2)

The main observation from these tables is that the model remains extremely efficient, even when release dates and due dates are considered. Even more, solution times are much smaller than those reported in Table 1, with almost all instances solved in less than one second. This might be due to the fact that release and due dates reduce the solution space. Still, the instances are not trivial to solve, as witnessed by the results related to the greedy algorithm. Indeed, while almost all instances admit feasible solutions, the greedy algorithm fails in finding the optimal solution for most of them. In addition, the greedy algorithm has difficulties even in finding a feasible solution, especially for the case in which release and due dates are tight. This shows that a solution approach smarter than the simple greedy algorithm can provide large advantages.

6.5. Computational results for $V, Q|sd, r_i, d_i|CLT$

Tables 5 and 6 report results for problem $V, Q|sd, r_i, d_i|CLT$. Contrary to previous results, demands are not forced to be unitary. optimal solutions are the ones obtained from solving the formulation in Section 5.2. Columns are the same as in the former section. In addition, the case with no release dates and due dates is considered in table 4.

n	V	Q	CPU(s)	feas	grSuccess	CLT	gapGr(%)	n	V	Q	CPU(s)	feas	grSuccess	CLT	gapGr(%)
20	1	1	0.1	5	5	13.0	1.5	80	1	1	2.5	5	5	46.4	1.7
20	1	2	0.1	5	5	9.8	8.2	80	1	2	2.9	5	5	35.6	2.8
20	1	4	0.1	5	5	8.8	11.6	80	1	4	15.9	5	5	30.4	9.9
20	1	8	0.1	5	5	8.2	16.9	80	1	8	22.9	5	5	27.6	11.9
20	2	1	0.1	5	5	6.8	2.9	80	2	1	2.5	5	5	23.4	1.6
20	2	2	0.1	5	5	5.4	10.9	80	2	2	3.9	5	5	18.2	5.6
20	2	4	0.1	5	5	5.2	13.0	80	2	4	5.4	5	5	15.6	9.0
20	2	8	0.1	5	5	4.8	13.0	80	2	8	24.4	5	5	14.2	12.8
20	3	1	0.1	5	5	4.8	9.0	80	3	1	3.9	5	5	15.8	3.8
20	3	2	0.1	5	5	3.8	23.3	80	3	2	5.8	5	5	11.8	7.0
20	3	4	0.1	5	5	3.6	26.7	80	3	4	6.0	5	5	10.4	13.5
20	3	8	0.1	5	5	3.2	26.7	80	3	8	20.2	5	5	9.8	16.6
40	1	1	0.3	5	5	25.4	1.6	160	1	1	17.6	5	5	87.4	0.7
40	1	2	0.2	5	5	19.0	3.2	160	1	2	102.9	5	5	65.4	1.2
40	1	4	1.2	5	5	17.0	12.9	160	1	4	227.7	5	5	54.8	10.6
40	1	8	0.5	5	5	14.6	11.0	160	1	8	292.3	5	5	51.6	12.5
40	2	1	0.4	5	5	13.2	1.5	160	2	1	30.8	5	5	44.2	1.8
40	2	2	0.3	5	5	9.8	10.4	160	2	2	29.7	5	5	33.2	3.0
40	2	4	0.4	5	5	8.6	6.9	160	2	4	213.1	5	5	27.4	11.8
40	2	8	0.7	5	5	8.8	11.4	160	2	8	453.7	5	5	25.4	11.0
40	3	1	0.4	5	5	9.0	4.2	160	3	1	39.9	5	5	29.4	2.0
40	3	2	0.4	5	5	7.2	14.0	160	3	2	50.3	5	5	22.2	3.6
40	3	4	0.4	5	5	6.0	14.7	160	3	4	86.4	5	5	19.8	10.1
40	3	8	0.3	5	5	5.0	20.3	160	3	8	162.7	5	5	18.4	12.1

Table 4: Problem $V, Q|sd|CLT$

n	V	Q	t-t					t-w					t				
			CPU(s)	feas	grSuccess	T_{OPT}	gapGr	CPU(s)	feas	grSuccess	T_{OPT}	gapGr	CPU(s)	feas	grSuccess	T_{OPT}	gapGr
20	1	1	0.0	0	0	-	-	0.0	1	0	13.0	-	0.0	5	5	13.6	7.5
20	1	2	0.0	0	0	-	-	0.0	4	0	11.3	-	0.0	5	5	11.0	13.6
20	1	4	0.0	1	1	9.0	11.1	0.0	4	0	10.0	-	0.0	5	5	10.6	16.9
20	1	8	0.0	2	0	9.0	-	0.0	3	1	10.0	20.0	0.0	5	5	10.0	16.6
20	2	1	0.0	0	0	-	-	0.0	1	0	8.0	-	0.0	5	5	7.0	17.9
20	2	2	0.0	1	0	6.0	-	0.0	4	0	6.0	-	0.0	5	5	5.6	21.5
20	2	4	0.0	2	0	6.0	-	0.0	3	0	5.7	-	0.0	5	5	5.8	24.7
20	2	8	0.0	4	0	5.8	-	0.0	5	0	5.0	-	0.0	5	5	5.4	26.7
20	3	1	0.0	0	0	-	-	0.0	1	0	4.0	-	0.0	5	5	5.2	17.3
20	3	2	0.0	3	0	4.3	-	0.0	5	0	4.6	-	0.0	5	5	4.4	15.0
20	3	4	0.0	2	0	4.0	-	0.0	5	0	4.2	-	0.0	5	5	4.0	32.3
20	3	8	0.0	4	0	4.3	-	0.0	5	1	4.2	0.0	0.0	5	5	4.2	9.0
40	1	1	0.0	0	0	-	-	0.0	1	0	24.0	-	0.2	5	5	26.8	4.8
40	1	2	0.0	1	0	21.0	-	0.1	3	0	20.7	-	0.1	5	5	20.2	9.3
40	1	4	0.0	3	0	18.7	-	0.1	3	0	19.0	-	0.2	5	5	19.2	12.4
40	1	8	0.0	4	0	17.8	-	0.0	4	0	17.8	-	0.1	5	5	18.0	14.5
40	2	1	0.0	0	0	-	-	0.0	2	0	13.0	-	0.2	5	5	13.6	7.6
40	2	2	0.0	1	0	10.0	-	0.1	4	0	10.0	-	0.2	5	5	10.2	13.8
40	2	4	0.0	4	0	9.3	-	0.1	5	0	9.8	-	0.2	5	5	9.4	13.9
40	2	8	0.1	4	0	10.5	-	0.0	5	0	10.0	-	0.2	5	5	10.4	12.3
40	3	1	0.0	0	0	-	-	0.0	4	0	9.0	-	0.3	5	5	9.2	13.2
40	3	2	0.0	3	0	7.3	-	0.0	4	0	8.0	-	0.2	5	5	7.4	16.4
40	3	4	0.0	4	1	6.8	16.7	0.0	4	0	7.0	-	0.1	5	5	6.8	14.9
40	3	8	0.0	4	0	6.5	-	0.0	5	0	6.4	-	0.1	5	5	6.4	22.4
80	1	1	0.0	0	0	-	-	0.2	4	0	47.5	-	1.3	5	5	47.8	4.2
80	1	2	0.4	3	0	39.3	-	1.2	5	0	37.2	-	2.2	5	5	38.2	10.0
80	1	4	0.2	3	0	34.3	-	12.2	5	1	35.2	11.4	2.7	5	5	34.6	11.6
80	1	8	0.4	5	0	32.6	-	0.8	5	0	33.6	-	2.2	5	5	32.2	11.7
80	2	1	0.0	0	0	-	-	0.2	3	0	24.3	-	2.0	5	5	24.0	4.2
80	2	2	0.5	5	0	19.4	-	2.0	5	0	18.8	-	2.4	5	5	18.8	14.9
80	2	4	7.4	5	0	17.2	-	1.2	5	0	17.6	-	9.2	5	5	17.0	11.7
80	2	8	0.2	5	0	17.2	-	0.8	5	0	17.2	-	1.8	5	5	17.2	6.9
80	3	1	0.1	0	0	-	-	0.3	4	0	16.3	-	2.4	5	5	16.2	5.0
80	3	2	1.0	5	0	13.2	-	0.4	5	0	12.4	-	1.4	5	5	13.2	9.1
80	3	4	0.2	5	0	12.0	-	1.7	5	0	12.0	-	1.3	5	5	12.0	13.3
80	3	8	0.5	5	1	12.0	9.1	0.6	5	0	11.6	-	1.6	5	5	11.8	15.3
160	1	1	0.4	0	0	-	-	7.8	5	0	88.4	-	45.6	5	5	88.6	3.8
160	1	2	179.5	5	0	69.4	-	38.5	5	0	69.2	-	346.4	5	5	69.2	6.5
160	1	4	15.3	5	0	64.8	-	22.0	5	0	63.4	-	27.2	5	5	64.6	7.1
160	1	8	18.6	5	0	63.4	-	8.6	5	0	63.4	-	17.4	5	5	63.4	7.0
160	2	1	0.3	0	0	-	-	4.1	5	0	44.8	-	28.5	5	5	44.2	2.2
160	2	2	87.7	5	0	34.4	-	25.9	5	0	34.4	-	44.9	5	5	34.4	6.9
160	2	4	365.6	5	0	32.3	-	9.2	5	0	33.4	-	44.5	5	5	32.0	10.6
160	2	8	12.0	5	0	32.0	-	71.6	5	0	32.2	-	20.5	5	5	31.8	8.8
160	3	1	0.3	0	0	-	-	4.0	5	0	29.6	-	49.8	5	5	29.8	6.0
160	3	2	51.5	5	0	23.2	-	17.2	5	0	23.0	-	26.5	5	5	23.2	9.5
160	3	4	20.4	5	0	22.4	-	24.8	5	0	22.0	-	362.2	5	5	22.2	10.8
160	3	8	92.8	5	0	21.8	-	5.7	5	0	21.8	-	56.4	5	5	21.8	8.3

Table 5: Problem $V, Q|3d, r_i, d_i|CLT$ (part 1)

n	V	Q	w-t					w-w					w				
			CPU(s)	feas	grSuccess	T_{OPT}	gapGr	CPU(s)	feas	grSuccess	T_{OPT}	gapGr	CPU(s)	feas	grSuccess	T_{OPT}	gapGr
20	1	1	0.0	0	0	-	-	0.0	2	0	14.0	-	0.0	5	5	14.0	11.4
20	1	2	0.0	2	1	12.0	9.1	0.0	4	0	12.5	-	0.0	5	5	12.2	18.0
20	1	4	0.0	2	0	11.0	-	0.0	1	0	15.0	-	0.0	5	5	11.8	11.9
20	1	8	0.0	2	0	11.5	-	0.0	4	1	11.8	10.0	0.0	5	5	11.4	13.9
20	2	1	0.0	0	0	-	-	0.0	3	0	7.0	-	0.0	5	5	8.4	7.9
20	2	2	0.0	4	0	7.0	-	0.0	4	1	6.5	33.3	0.0	5	5	6.4	15.9
20	2	4	0.0	2	0	7.0	-	0.0	4	1	6.8	16.7	0.0	5	5	6.6	15.4
20	2	8	0.0	5	0	6.4	-	0.0	5	0	6.2	-	0.0	5	5	6.4	12.4
20	3	1	0.0	1	0	5.0	-	0.0	2	0	5.5	-	0.0	5	5	5.6	14.0
20	3	2	0.0	3	0	4.3	-	0.0	5	0	4.6	-	0.0	5	5	4.6	18.0
20	3	4	0.0	4	0	5.0	-	0.0	5	3	4.6	15.0	0.0	5	5	5.0	22.0
20	3	8	0.0	5	1	4.4	25.0	0.0	5	0	4.8	-	0.0	5	5	4.4	23.0
40	1	1	0.0	1	0	26.0	-	0.0	2	0	26.0	-	0.1	5	5	27.0	6.2
40	1	2	0.0	4	0	22.8	-	0.0	5	0	22.6	-	0.1	5	5	21.6	7.4
40	1	4	0.0	4	0	23.5	-	0.0	5	0	23.4	-	0.1	5	5	22.0	10.1
40	1	8	0.0	5	0	21.2	-	0.0	5	0	21.8	-	0.1	5	5	20.8	10.7
40	2	1	0.0	1	0	13.0	-	0.0	3	0	13.7	-	0.2	5	5	14.0	8.8
40	2	2	0.0	4	0	11.5	-	0.0	5	0	11.4	-	0.1	5	5	11.2	10.8
40	2	4	0.0	4	0	11.8	-	0.0	5	0	11.4	-	0.1	5	5	11.6	12.3
40	2	8	0.0	4	1	11.0	10.0	0.0	5	0	11.2	-	0.1	5	5	11.0	14.6
40	3	1	0.0	1	0	9.0	-	0.0	5	0	9.6	-	0.2	5	5	9.0	13.7
40	3	2	0.0	5	0	8.4	-	0.0	5	0	8.4	-	0.1	5	5	8.2	14.7
40	3	4	0.0	5	0	8.2	-	0.0	5	0	8.0	-	0.1	5	5	8.0	12.5
40	3	8	0.0	5	0	7.8	-	0.0	5	0	8.0	-	0.1	5	5	7.8	17.9
80	1	1	0.1	2	0	47.5	-	0.2	4	0	48.8	-	0.8	5	5	48.0	7.5
80	1	2	0.1	5	0	43.6	-	0.3	5	0	42.6	-	0.4	5	5	43.4	6.5
80	1	4	0.1	5	0	41.2	-	0.2	5	0	41.8	-	0.3	5	5	41.2	6.8
80	1	8	0.1	5	0	41.8	-	0.1	5	0	41.2	-	0.3	5	5	41.8	4.7
80	2	1	0.1	4	0	24.0	-	0.3	5	0	24.8	-	1.3	5	5	23.8	7.5
80	2	2	0.1	5	0	22.2	-	0.2	5	1	22.0	4.5	0.7	5	5	22.2	5.7
80	2	4	0.1	5	1	21.4	4.5	0.1	5	0	21.4	-	0.4	5	5	21.2	5.7
80	2	8	0.1	5	0	21.4	-	0.1	5	0	21.4	-	0.4	5	5	21.4	5.6
80	3	1	0.1	2	0	16.5	-	0.2	5	0	17.0	-	1.4	5	5	16.8	8.4
80	3	2	0.1	5	0	15.0	-	0.1	5	0	14.8	-	0.5	5	5	15.0	9.3
80	3	4	0.1	5	0	15.0	-	0.1	5	0	15.2	-	0.4	5	5	15.0	5.3
80	3	8	0.1	5	0	15.0	-	0.1	5	0	14.8	-	0.5	5	5	15.0	4.0
160	1	1	2.7	5	0	91.4	-	4.5	5	0	91.2	-	17.0	5	5	91.2	6.4
160	1	2	1.7	5	0	83.6	-	3.6	5	0	83.6	-	5.0	5	5	83.6	3.3
160	1	4	0.4	5	0	81.6	-	0.5	5	0	81.4	-	1.7	5	5	81.6	2.5
160	1	8	0.3	5	0	82.0	-	0.4	5	0	81.4	-	1.4	5	5	81.8	1.7
160	2	1	1.7	5	0	46.4	-	3.5	5	0	45.8	-	12.3	5	5	46.4	5.2
160	2	2	0.8	5	0	42.4	-	2.1	5	0	41.8	-	2.6	5	5	42.2	4.3
160	2	4	0.5	5	0	42.0	-	0.9	5	0	41.8	-	2.4	5	5	42.0	3.3
160	2	8	0.4	5	0	40.8	-	0.6	5	1	41.2	2.4	3.2	5	5	40.8	3.9
160	3	1	1.7	5	0	30.2	-	2.8	5	0	30.8	-	21.1	5	5	30.0	9.3
160	3	2	0.6	5	0	27.6	-	1.2	5	0	28.6	-	4.7	5	5	27.6	5.8
160	3	4	1.0	5	0	28.2	-	0.4	5	0	28.4	-	3.3	5	5	28.2	1.4
160	3	8	0.3	5	0	27.6	-	0.4	5	1	27.8	3.7	3.4	5	5	27.6	2.9

Table 6: Problem $V, Q|s_{ij}d, r_i, d_i|CLT$ (part 2)

These tables confirm the good-quality of the formulation but also exhibit larger computing times. Largest instances regularly require a few minutes for getting the optimal solution, with or without release dates and due dates. The reason is related to the larger number of variables due to the disaggregated capacity constraints. These instances are also specially difficult for the greedy algorithm. In fact, it fails in finding a feasible solution for most of the instances and shows larger gaps compared to instances with unitary demands, when feasible solution are found. The additional complexity implied by the packing of non-unitary requests makes the greedy algorithm not effective.

7. Conclusions and perspectives

In this paper we introduced a new class of Pickup and Delivery problems where the stations are located on rings. We first proposed a classification scheme. Then, we investigated the complexity of the variants in which the vehicles are allowed to move in a single direction and the objective is the minimization of the maximum number of tours. We described a polynomial time algorithm for some variants and we proved the NP-hardness of the remaining variants.

For the NP-hard variants, we proposed MILP formulations and computational tests to evaluate these formulations. Experiments on a large number of instance show the impressive efficiency of our formulations. All instances, with up to 160 requests, could be solved in a few minutes. Comparisons with a simple and practically relevant greedy algorithm also confirmed the intrinsic difficulty of the problems/instances and the usefulness of applying exact solution schemes.

Different future research directions are possible. In parallel to this work, we started addressing other variants of these problems, e.g., problems where vehicles are allowed to move in different directions or problems with alternative objective functions. Another interesting direction would be to consider different network topologies such as lines or other geometric shapes that can be encountered in practice. Also, autonomous vehicles are bound to use electric engines. A future step of our research should be to investigate the issues implied by the limited autonomy of electric vehicles (range anxiety, recharging policies...).

References

(2018). Keolis deploys electric autonomous shuttles at two university campuses in France. https://www.keolis.com/en/media/newsroom/press-releases/keolis-deploys-electric-autonomous-shuttles-two-university-campuses. [Online; accessed 04-October-2019].

(2019). Paris-Saclay Autonomous Lab: new autonomous, electric and shared mobility services. https://www.transdev.com/en/press-release/paris-saclay-autonomous-lab/. [Online; accessed 04-October-2019].

(2020). Projet AVENUE : Navettes autonomes en milieu urbain. http://www.lgi.centralesupelec.fr/en/node/328. [Online; accessed 22-January-2021].

Anily, S., & Pfeffer, A. (2013). The uncapacitated swapping problem on a line and on a circle. *Discrete Applied Mathematics*, *161*, 454–465.

Antonialli, F. (2019). International benchmark on experimentations with autonomous shuttles for collective transport. In *27th International Colloquium of Gerpisa*.

Atallah, M. J., & Kosaraju, S. R. (1988). Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, *17*, 849–869.

Baïou, M., Colares, R., & Kerivin, H. (2018). The stop number minimization problem: Complexity and polyhedral analysis. In *International Symposium on Combinatorial Optimization* (pp. 64–76). Springer.

Battarra, M., Cordeau, J.-F., & Iori, M. (2014). Chapter 6: pickup-and-delivery problems for goods transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition* (pp. 161–191). SIAM.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., & Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *Top*, *15*, 1–31.

Berbeglia, G., Cordeau, J.-F., & Laporte, G. (2010). Dynamic pickup and delivery problems. *European journal of operational research*, *202*, 8–15.

Bonomo, F., Durán, G., & Marenco, J. (2009). Exploring the complexity boundary between coloring and list-coloring. *Annals of Operations Research*, *169*, 3.

Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of operations research*, *153*, 29–46.

Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., & Soumis, F. (2002). Vrp with pickup and delivery. In *The Vehicle Routing Problem* (pp. 225–242). SIAM.

Doerner, K. F., & Salazar-González, J.-J. (2014). Chapter 7: Pickup-and-delivery problems for people transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition* (pp. 193–212). SIAM.

Gendreau, M., Laporte, G., & Vigo, D. (1999). Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research*, *26*, 699–714.

Gökay, S., Heuvels, A., & Krempels, K.-H. (2019). A high-level category survey of dial-a-ride problems. In *VEHITS* (pp. 594–600).

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics* (pp. 287–326). Elsevier volume 5.

Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421.

Ilani, H., Shufan, E., & Grinshpoun, T. (2015). A fixed route dial-a-ride problem. *Proceedings of the 7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015), 25 - 28 Aug 2015, Prague, Czech Republic*, (pp. 313–324).

Itai, A., Perl, Y., & Shiloach, Y. (1982). The complexity of finding maximum disjoint paths with length constraints. *Networks*, *12*, 277–286.

Jungnickel, D. (2013). Algorithms and complexity. In *Graphs, Networks and Algorithms* (pp. 35–63). Springer.

Molenbruch, Y., Braekers, K., & Caris, A. (2017). Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, *259*, 295–325.

Parragh, S. N., Doerner, K., & Hartl, R. F. (2008). A survey on pickup and delivery models part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, *58*, 21–51.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2007). A survey on pickup and delivery problems. *Part II: Transportation between pickup and delivery locations, to appear: Journal für Betriebswirtschaft*, .

Pimenta, V., Quilliot, A., Toussaint, H., & Vigo, D. (2017). Models and algorithms for reliability-oriented dial-a-ride with autonomous electric vehicles. *European Journal of Operational Research*, *257*, 601–613.

Rüther, C., & Rieck, J. (2020). A grouping genetic algorithm for multi depot pickup and delivery problems with time windows and heterogeneous vehicle fleets. In *European Con-*

ference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar) (pp. 148–163). Springer.

Toth, P., & Vigo, D. (2014). *Vehicle routing: problems, methods, and applications*. SIAM.

Tzoreff, T. E., Granot, D., Granot, F., & Sošić, G. (2002). The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathematics*, *116*, 193–229.

Wahlström, M. (2018). Euler digraphs. In *Classes of Directed Graphs* (pp. 173–205). Springer.

Wu, J., Zheng, L., Huang, C., Cai, S., Feng, S., & Zhang, D. (2019). An improved hybrid heuristic algorithm for pickup and delivery problem with three-dimensional loading constraints. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 1607–1612). IEEE.

Appendix A. Complexity of problem $V, 1|sd, u|CLT$

In this Appendix, we prove that problem $V, 1|sd, u|CLT$ is NP-hard. We consider the decision version of the problem and prove that it is NP-complete. In the decision version of $V, 1|sd, u|CLT$, one searches for a feasible solution with route lengths not longer than a given threshold CLT.

The proof is in two steps:

1. We introduce the Eulerian Path Partition Problem (EPP) and prove that it is NP-complete. This is done through a polynomial-time reduction from the well-known 3-SAT problem, which is NP-complete.
2. We then propose a polynomial-time reduction from EPP to the decision version of $V, 1|sd, u|CLT$. Given that EPP is NP-complete, this proves that $V, 1|sd, u|CLT$ is NP-hard.

The EPP is defined as follows.

Definition 9. *Let us consider a directed acyclic graph $G = (X, E)$, together with 2 nodes $s \in X$ and $p \in X$ and 2 integer numbers K and T. We suppose that, for any node $x \in X$, a path from s to x and a path from x to p exist in G. The EPP consists in determining if a partition of E into K arc-disjoint paths can be found, with exactly T arcs in each path. In this case, we say that G is $(K, T) - EPP$.*

Appendix A.1. Reduction of 3-SAT to EPP

We will follow an approach very similar to the construction procedure which was proposed in Itai et al. (1982). We consider a 3-SAT instance $Z = \{z_1, ..., z_N\}$ (variable set) and $C = \{c_1, ..., c_S\}$ (3-clause set), with N variables and S clauses. We suppose that for any $j = 1, ..., N$, the number of occurrences of z_j in C is equal to the number of occurrences of $\neg z_j$ and we denote it by $u(j)$. We call this assumption the *Well-Balanced Hypothesis*. We know from Lemma 3.1 Itai et al. (1982) that the resulting restriction of 3-SAT remains NP-Complete. We set $U = \sum_{j=1}^{N} u(j)$ and see that the total number of literals in the clauses is $3S = 2U$. In what follows, we assume that each occurrence of any literal y_i in a clause (variable z_j or its negation) is associated with a number in $\{1, \ldots, u(j)\}$.

We build a graph $H = (V, F)$ such that the 3-SAT instance is feasible if and only if a graph which is slightly modified with respect to H is $(3S + 2U, 11)$-EPP. H is acyclic and is composed of six layers. The construction is illustrated on Figures A.8 and A.9 for an instance with 3 variables and 2 clauses $c_1 = (z_1 \vee \neg z_2 \vee z_3)$, $c_2 = (\neg z_1 \vee z_2 \vee \neg z_3)$. Node set V is given by Table A.7. Each line defines a given category of nodes, at a given layer, and introduces

both a name and a notation for the nodes of the category. The total number of nodes in each category is reported in the last column. All nodes are visible in Figure A.8. One can notice that layers 3 and 4 contain exactly $2U$ nodes, that is, the number of literals in the clauses.

Layer	Category	Symbol		Number
1	Source node	s		1
2	Clause nodes	$c_1, ..., c_S$		S
	Low layer variable nodes	(j, u)	$(j \in \{1, \ldots, N\}, u \in \{1, \ldots, u(j)\})$	U
3	First middle nodes	(j, u, ϵ)	$(j \in \{1, \ldots, N\}, u \in \{1, \ldots, u(j)\}, \epsilon \in \{0, 1\})$	$2U$
4	Second middle nodes	$(j, u, \epsilon)^*$	$(j \in \{1, \ldots, N\}, u \in \{1, \ldots, u(j)\}, \epsilon \in \{0, 1\})$	$2U$
5	Bottleneck node	Q		1
	Top layer variable nodes	$(j, u)^*$	$(j \in \{1, \ldots, N\}, u \in \{1, \ldots, u(j)\})$	U
6	Sink node	p		1

Table A.7: Node set V

Arc set F includes clause-related and variable-related arcs. Figure A.8 reports all these arcs for our illustrative example.

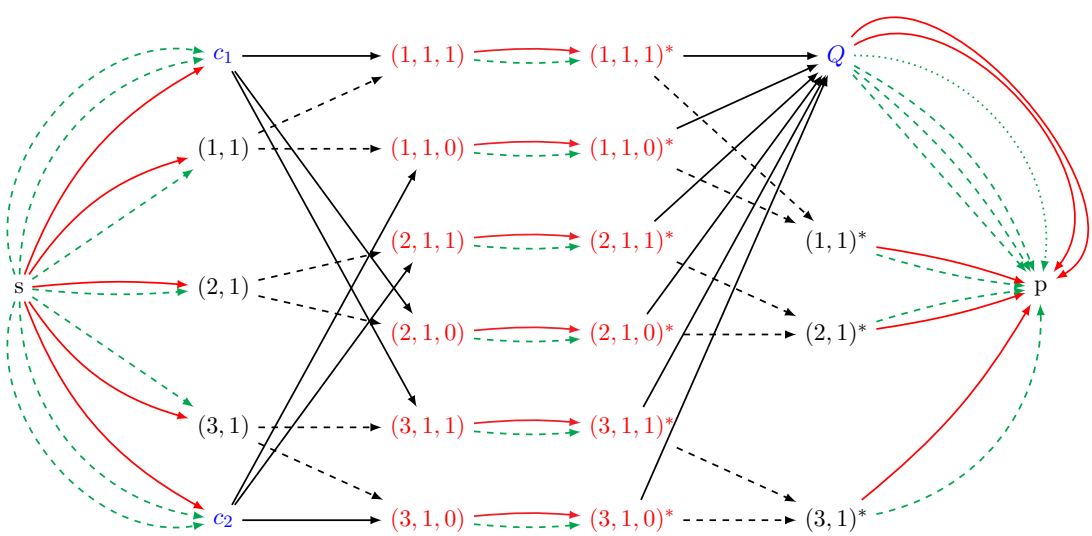


Figure A.8: A Graph $H = (V, F)$, with 2 clauses c_1, c_2 and 3 variables z_1, z_2, z_3

Clause-related arcs are defined in Table A.8 and are shown in figures A.8 and A.9, with information on arc lengths on the latter. The structure of the table is similar to that of Table A.7, with an additional column for arc lengths. Given a clause c_k and a literal y_i

in this clause, notation $lit(c_k, i)$ in the table indicates a triplet composed of: the index of the associated variable, the occurrence number of the literal, 0 or 1 if the variable is in its negative or positive form in the literal, respectively. For example, the three literals associated with clause $c_1 = (z_1 \vee \neg z_2 \vee z_3)$ are $lit(c_1, 1) = (1, 1, 1)$, $lit(c_1, 2) = (2, 1, 0)$, and $lit(c_1, 3) = (3, 1, 1)$. Consequently, the three c-variable arcs for the clause are arcs $(c_1, (1, 1, 1))$, $(c_1, (2, 1, 0))$ and $(c_1, (3, 1, 1))$. Note that first middle nodes have exactly one in-going clause-related arc each. Note also that the notation introduced in the table does not always enable to distinguish between parallel arcs (c-default arcs, good bottleneck arcs, bad bottleneck arcs are not distinguished). We keep this notation to ease readability. Clause-related arcs represent exactly $3S$ arcs between every successive layers except between layers 3 and 4 where no clause-related arcs are introduced (remembering that $3S = 2U$).

Layers	Category	Symbol		Length	Number
$(1, 2)$	c-id arcs	$(s, c_k)^{Id}$	$(k \in \{1, \ldots, S\})$	5	S
	first c-def arcs	$(s, c_k)^{Def}$	$(k \in \{1, \ldots, S\})$	1	S
	second c-def arcs	$(s, c_k)^{Def}$	$(k \in \{1, \ldots, S\})$	1	S
$(2, 3)$	first c-variable arcs	$(c_k, (j, u, \epsilon))$	$(k \in \{1, \ldots, S\}, (j, u, \epsilon) = lit(c_k, 1))$	1	S
	second c-variable arcs	$(c_k, (j, u, \epsilon))$	$(k \in \{1, \ldots, S\}, (j, u, \epsilon) = lit(c_k, 2))$	1	S
	third c-variable arcs	$(c_k, (j, u, \epsilon))$	$(k \in \{1, \ldots, S\}, (j, u, \epsilon) = lit(c_k, 3))$	1	S
$(4, 5)$	bottleneck arcs	$((j, u, \epsilon)^*, Q)$	$(j \in \{1, \ldots, N\}, u \in \{1, \ldots, u(j)\}, \epsilon \in \{0, 1\})$	1	$2U$
$(5, 6)$	good bottleneck arcs	$(Q, p)^{good}$	S copies	3	S
	bad bottleneck arcs	$(Q, p)^{bad,Id}$	$U - S$ copies	7	$U - S$
	bad bottleneck arcs	$(Q, p)^{bad,Def}$	$3S - U$ copies	6	$3S - U$

Table A.8: Clause-related arcs in Set F

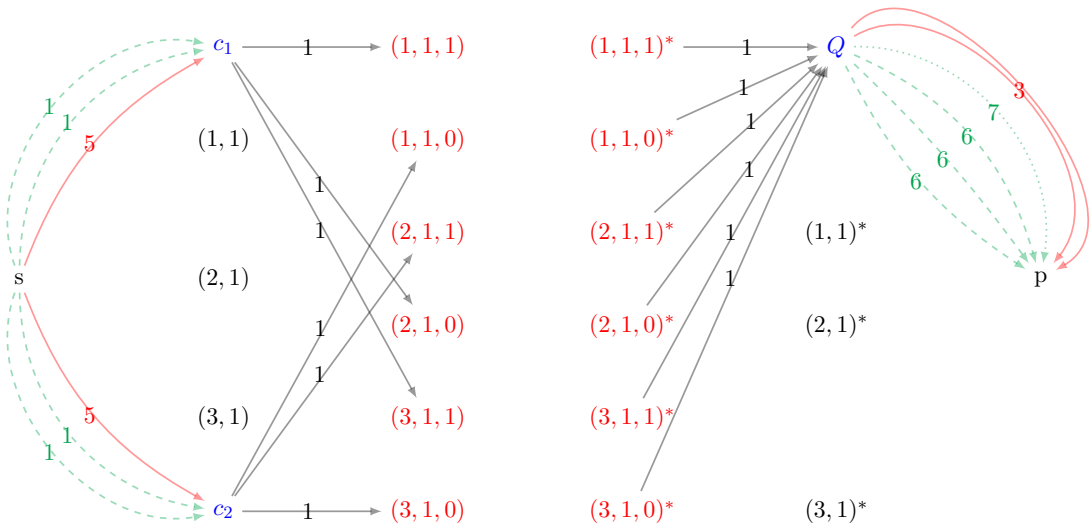


Figure A.9: Arc lengths for clause-related arcs of graph $H = (V, F)$ of Figure A.8

Variable-related arcs are defined in Table A.9. This table reads as Table A.8. Arcs can be seen on Figure A.8 for our example. Top-second arcs induce what we call a *Saw Pattern*. This pattern cannot clearly be observed on Figure A.8 because $u(j) = 1$ for every variable z_j. We illustrate the pattern on figure A.10, with a variable having three occurrences in the clauses. In the definition of these arcs, notation $u+\epsilon$ is assumed to give value 1 when $u = u(j)$ and $\epsilon = 1$. The Saw Pattern is needed to have consistency among different occurrences of the same variable. As shown in Figure A.8, it defines two complementary perfect matchings between the three *second middle nodes* having the same value for ϵ and the three *top layer variable nodes*. Similarly to clause-related arcs, variable-related arcs involve exactly $2U$ arcs (that is, $3S$) between every successive layers except between layers 3 and 4, with $4U$ arcs. Considering the two sets of arcs, the graph contains exactly $4U$ arcs between every successive layers.

Layers	Category	Symbol		Length	Number
$(1,2)$	low-first-id arcs	$(s,(j,u))^{Id}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\})$	7	U
	low-first-def arcs	$(s,(j,u))^{Def}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\})$	3	U
$(2,3)$	low-second arcs	$((j,u)),(j,u,\epsilon))$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\}, \epsilon \in \{0,1\})$	1	$2U$
$(3,4)$	middle-id arcs	$((j,u,\epsilon),(j,u,\epsilon)^*)^{Id}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\}, \epsilon \in \{0,1\})$	1	$2U$
	middle-def arcs	$((j,u,\epsilon),(j,u,\epsilon)^*)^{Def}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\}, \epsilon \in \{0,1\})$	2	$2U$
$(4,5)$	top-second	$((j,u,\epsilon)^*,(j,u+\epsilon)^*)$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\}, \epsilon \in \{0,1\})$	1	$2U$
$(5,6)$	top-first-id arcs	$((j,u)^*,p)^{Id}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\})$	1	U
	top-first-def arcs	$((j,u)^*,p)^{Def}$	$(j \in \{1,\dots,N\}, u \in \{1,\dots,u(j)\})$	4	U

Table A.9: Variable-related arcs in Set F

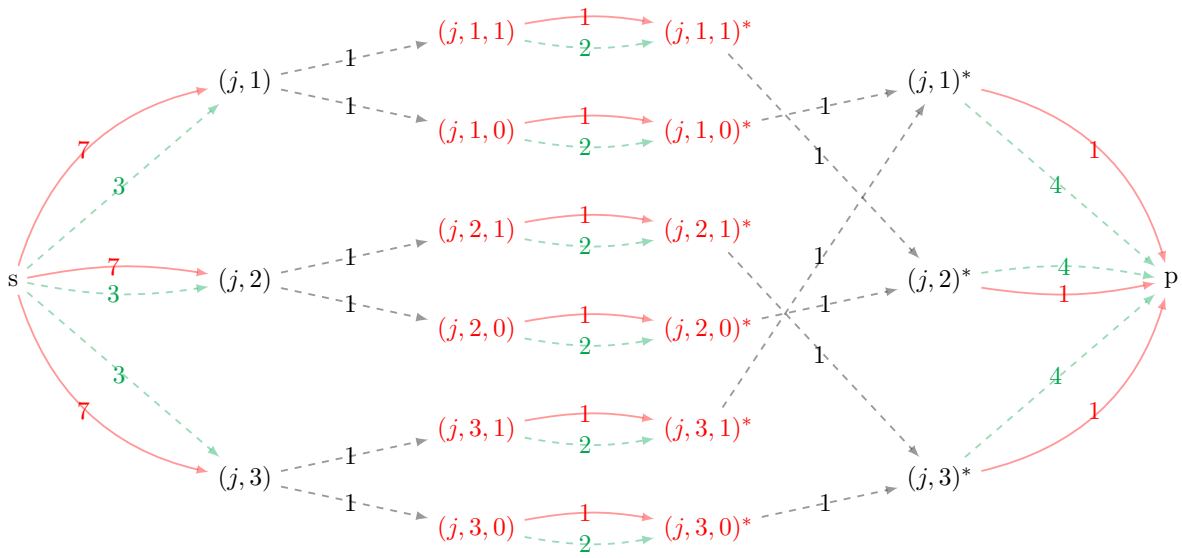


Figure A.10: Variable-related arcs for a given variable z_j with $u(j) = 3$, with the Saw Pattern between layers 4 and 5.

From graph $H = (V, F)$, we define a second graph $H^* = (V^*, F^*)$ by replacing every arc of length $h > 1$ with a chain of h arcs of length 1. This new graph is useful to meet the definition of EPP in which the length of a path is given by its number of arcs. Then, finding a partition of F^* in K arc-disjoint paths with T arcs in each path is equivalent to finding a partition of F in K arc-disjoint paths with length T.

Theorem 6. *3-SAT instance (Z, C) is feasible iff $H^* = (V^*, F^*)$ is $(3S + 2U, 11)$-EPP.*

Proof. Let us first assume that our 3-SAT instance is positive, that is, it admits a feasible solution $z = (\epsilon_1, \ldots, \epsilon_N)$. Then we show how we can partition the arcs of H^* into $3S + 2U$ paths with 11 arcs or equivalently the arcs of H into $3S + 2U$ paths with length 11. The construction is illustrated in Figures A.11 and A.12, pursuing with the same two clauses c_1 and c_2, and considering assignment $z = \{1, 1, 1\}$. The partition is as follows:

- We first generate a path for every occurrence u of literal z_j and a path for every occurrence u of literal $\neg z_j$ in the clauses, for a total of $2 \times U$ paths:
  - **(j,u)-identifier** path: $s \xrightarrow{Id} (j, u) \rightarrow (j, u, \neg\epsilon_j) \xrightarrow{Id} (j, u, \neg\epsilon_j)^* \rightarrow (j, u{+}(\neg\epsilon_j))^* \xrightarrow{Id} p$; (the red paths in fig. A.11)
  - **(j,u)-default** path: $s \xrightarrow{Def} (j, u) \rightarrow (j, u, \epsilon_j) \xrightarrow{Def} (j, u, \epsilon_j)^* \rightarrow (j, u + \epsilon_j)^* \xrightarrow{Def} p$; (the green paths in fig. A.11)

  These paths cover all variable-related arcs, except middle-id arcs $((j, u, \epsilon_j), (j, u, \epsilon_j)^*)^{Id}$ and middle-def arcs $((j, u, \neg\epsilon_j), (j, u, \neg\epsilon_j)^*)^{Def}$. They all have a length equal to 11. Figure A.11 shows these paths.

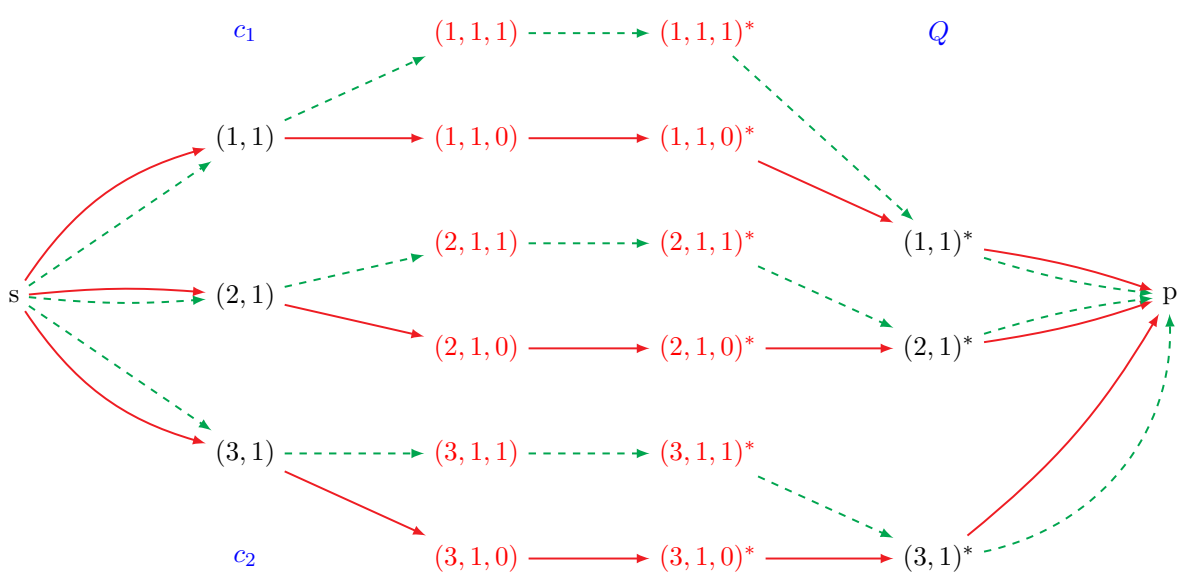


Figure A.11: $(j, u)-$identifier and $(j, u)-$default paths in graph H

- We then generate three paths for every clause c_k, for a total of $3 \times S$ paths. A clause involves three literals that can be matched to a variable z_j and an occurrence u. Each path visits, in sequence, nodes s, c_k, (j, u, ϵ), $(j, u, \epsilon)^*$, Q and p, where $\epsilon = 1$ if the literal is z_j and $\epsilon = 0$ if the literal is $\neg z_j$ (see definition of clause-variables arcs). The arcs traversed depend on the type of path as follows.

– **c-identifier** path: among the three literals, at least one confirms the validity of the clause, i.e., its assignment is such that the clause is satisfied; we arbitrarily choose one such literal (in case there is more than one literal that satisfies the clause) and we generate path $s \xrightarrow{Id} c_k \to (j,u,\epsilon) \xrightarrow{Id} (j,u,\epsilon)^* \to Q \xrightarrow{Good} p$; (in red in figure A.12)

– two **c-default** paths: a path is generated for every of the two remaining literals; the literals can satisfy the clause or not; the path is $s \xrightarrow{Def} c_k \to (j,u,\epsilon) \xrightarrow{Id} (j,u,\epsilon)^* \to Q \xrightarrow{Bad,Id} p$ if the literal satisfies the clause, $s \xrightarrow{Def} c_k \to (j,u,\epsilon) \xrightarrow{Def} (j,u,\epsilon)^* \to Q \xrightarrow{Bad,Def} p$ otherwise (with the same definition as above for ϵ). (in green in figure A.12)

Figure A.12 shows the $c-$identifier and the $c-$default paths.

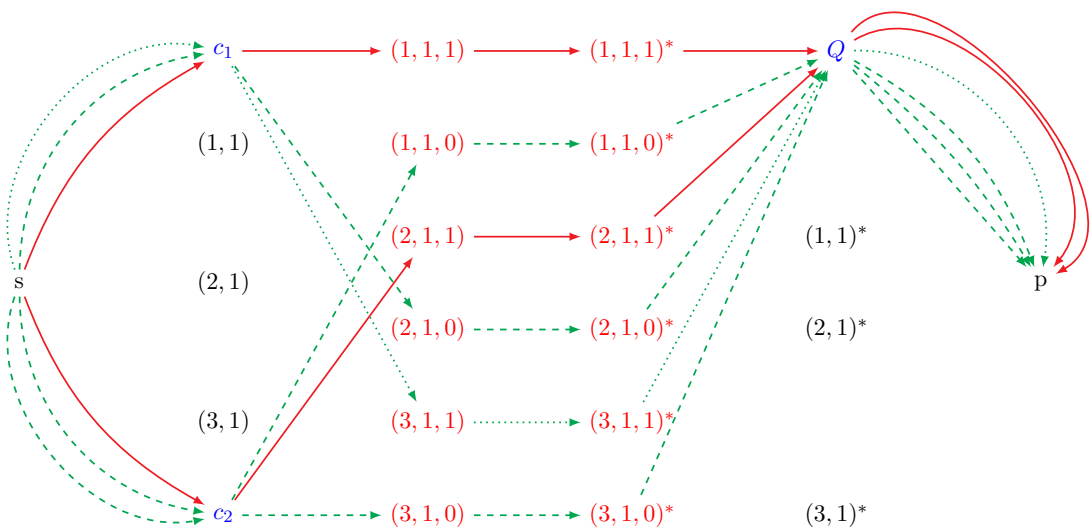


Figure A.12: $c-$identifier and $c-$default paths in graph H

All paths have a length equal to 11. They cover all remaining middle-id and middle-def arcs. Indeed, an arc $((j,u,\epsilon),(j,u,\epsilon)^*)^{Id}$ is traversed when the literal is true in the clause, that is, when $\epsilon = \epsilon_j$. They also cover all clause-related arcs. Especially, for every clause, the c-id and the two c-def arcs are covered, as well as one good bottleneck arc and two bad bottleneck arcs. Furthermore, thanks to the well-balanced assumption, exactly U literals satisfy their clause and U literals do not. So, U literals cover the U bad bottlenecks arcs with cost 6, the other bottleneck arcs being covered by the other literals (S for the good bottleneck arcs, U-S for the remaining bad bottleneck arcs).

All together, the $3S + 2U$ paths partition F in paths of length 11, which proves that H^* is

$(3S + 2U, 11)$-EPP.

Conversely, let us now assume that H^* is $(3S + 2U, 11)$-EPP. We will show that 3-SAT is positive. Indeed, if H^* is $(3S + 2U, 11)$-EPP, it means that there exists a partition of the arcs of H into $M = 3S + 2U$ paths with length 11, which we denote by $\Gamma_1, ..., \Gamma_M$. Then we see that those paths, which all contain one first middle node (j, u, ϵ), can be split into 4 categories:

1. Those who reach node (j, u, ϵ) with a sub-path $s \xrightarrow{Id} (j, u) \rightarrow (j, u, \epsilon)$ of length 8. To have a length 11, they must have the form of **(j,u)-identifier** paths. Every such path contains arc $((j, u, \epsilon), (j, u, \epsilon)^*)^{Id}$.
2. Those who reach node (j, u, ϵ) with a sub-path $s \xrightarrow{Def} (j, u) \rightarrow (j, u, \epsilon)$ of length 4. They must be **(j,u)-default** paths. Every such path contains arc $((j, u, \epsilon), (j, u, \epsilon)^*)^{Def}$.
3. Those who reach node (j, u, ϵ) with a sub-path $s \xrightarrow{Id} c_k \rightarrow (j, u, \epsilon)$ of length 6. They must finish with **good bottleneck** arcs and be **c-identifier** paths.
4. Those who reach node (j, u, ϵ) with a sub-path $s \xrightarrow{Def} c_k \rightarrow (j, u, \epsilon)$ of length 2. They should finish with **bad bottleneck** arcs. They are **c-default** paths. Furthermore, $U - S$ of them contain bad bottleneck arcs of size 7, and, so, also contain an arc $((j, u, \epsilon), (j, u, \epsilon)^*)^{Id}$.

Every node (j, u, ϵ) is traversed by two paths:

- One path Γ_m which is either a **(j,u)-identifier** path or a **(j,u)-default** path. We call this path the **representative path** of (j, u, ϵ). We see that if the **representative path** of (j, u, ϵ) is a **(j,u)-default** path then the **representative path** of $(j, u, \neg\epsilon)$ is a **(j,u)-identifier** path and vice-versa. Indeed, one path starts with sequence $s \xrightarrow{Id} (j, u)$, the other with sequence $s \xrightarrow{Def} (j, u)$.

- One path Γ_{m^*} which is either a **c-default** or a **c-identifier** path for some clause c_k.

We now derive from paths $\Gamma_1, ..., \Gamma_M$, an assignment of $\{0, 1\}$ values to variables z_j. For every clause c_k, we consider the node (j, u, ϵ) traversed by the c-identifier path. We give value ϵ to variable z_j. After this first step, it is possible that not all variables are assigned. We complete with random values for other variables. We prove that these values are consistent, i.e., the resulting assignment makes 3-SAT positive.

We use the following property of the Saw Pattern that we call the Saw Pattern Property. Given a variable z_j, we know that paths $\Gamma_1, ..., \Gamma_M$ contain $u(j)$ (j, u)-default paths and that each of these (j, u)-default paths reaches one of the $u(j)$ nodes $(j, u)^*$. The Saw Pattern imposes that the latter are reached from nodes (j, u, ϵ_u) with all ϵ_u equal (using the arcs of

one of the two perfect-matchings that compose the Saw Pattern). This also implies that, if variable z_j gets value ϵ, then all arcs $((j,u,\neg\epsilon),(j,u,\neg\epsilon)^*)^{Id}$ have to be in (j,u)-identifier paths and all arcs $((j,u,\epsilon),(j,u,\epsilon)^*)^{Def}$ have to be in (j,u)-default paths. For example, in Figure A.10, the first occurrence of variable z_j is negative (*i.e.* takes value 0), if and only if node $(j,1,1)$ is reached with a (j,u)-identifier path, that must therefore finish with arc $((j,2)^*,p)$ of length 1. It means that arc $((j,2)^*,p)$ of length 4 must be in a (j,u)-default path together with arc $((j,2,0),(j,2,0)^*)^{Def}$, forcing also arc $((j,2,1),(j,2,1)^*)^{Id}$ to be in a (j,u)-identifier path and thus forcing the second occurrence of variable j to get value 0. The same happens for the third occurrence.

Assume now that a variable z_j receives two values ϵ and ϵ' from two clauses c and c'. It means that arcs $((j,u,\epsilon),(j,u,\epsilon)^*)^{Id}$ and $((j,u',\epsilon'),(j,u',\epsilon')^*)^{Id}$ are in the c-identifier path and the c'-identifier path, respectively. It implies that arcs $((j,u,\epsilon),(j,u,\epsilon)^*)^{Def}$ and $((j,u',\epsilon'),(j,u',\epsilon')^*)^{Def}$ are in the (j,u)-default path and (j,u')-default paths, respectively. Equality $\epsilon=\epsilon'$ follows from the Saw Pattern Property.

By construction, we also know that all clauses are satisfied for these values, which concludes the proof.

□

Appendix A.2. Reduction of EPP to $V,1|sd,u|CLT$

In this section we prove that EPP can be reduced to $V,1|sd,u|CLT$.

Theorem 7. *EPP can be reduced to $V,1|sd,u|CLT$.*

Proof. Let $\mathcal{I}_{EPP}=(G(X,E),s,p,K,T)$ be a non trivial instance of EPP, i.e., the problem of determining whether there exists K disjoint paths of length T from s to t in G. Non trivial means that G has the following properties:

1. $d^-(x)=d^+(x)$ for any node $x\neq s,p$, where $d^-(x)$ denotes the *in-degree* of x and $d^+(x)$ denotes its *out-degree*
2. $d^+(s)=K$
3. $|E|=KT$

This implies that $d^+(s)=d^-(p)$. It is easy to see that any instance that does not meet these requirements is trivially not $(K,T)-EPP$. Note that it is enough to consider non-trivial instances in the reduction because if non-trivial instances could be solved with the reduction then all instances would be solved.

We build an instance $\mathcal{I}$ of $V,1|sd,u|CLT$ as follows. We introduce a ring with $m=|X|+1$ stations. We first define station 0, the depot. The other stations have a one-to-one

correspondence with the vertices in X as follows. We first sort the nodes in X in a topological order (we can do that because G is a directed acyclic graph) and reverse this order. Node p is then the first node and node s the last. We then associate one station with each node following this order, thus having station 1 representing node p and station $|X|$ representing node s. We denote $stat(x)$ the station associated with node $x \in X$. We define unitary distances between successive stations, which gives a tour length $L = |X| + 1$. For every arc $(x, y) \in E$, we introduce a request in $\mathcal{R}$ defined by $s_i = stat(x)$, $t_i = stat(y)$. This request is denoted $r(x, y)$. Since we considered nodes in a reversed topological order, all requests cover the depot. We complete $\mathcal{R}$ by adding K requests $(0, stat(s))$ and K requests $(stat(p), 0)$. This way, $|\mathcal{R}| = KT + 2K$. We finally set $V = K$ and $CLT = L \times (T + 1)$. Note that this construction is polynomial.

We illustrate this construction on a simple example. We consider the graph G of figure A.13.

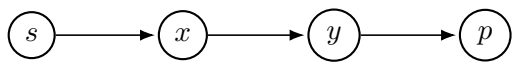
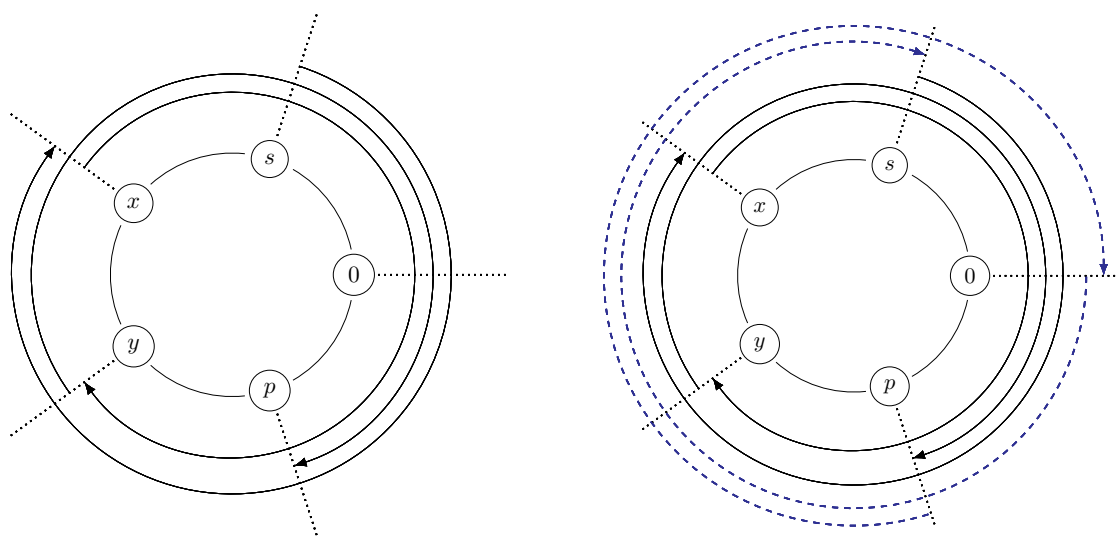


Figure A.13: Graph G

The resulting ring and set of requests are shown in Figure A.14.



(a) Ring and requests from arcs in E

(b) Ring and complete set of requests

Figure A.14: Ring and requests constructed from G

Assume first that $\mathcal{I}_{EPP}$ is positive. This means that set E can be partitioned into K arc-disjoint paths $\mathcal{P}_k$, each of length T. We build a feasible schedule for $\mathcal{I}$ in the following way. Every vehicle k $(1 \leq k \leq V)$ starts from the depot by serving request $(0, stat(s))$. Then, it serves all requests $r(x, y)$ for $(x, y) \in \mathcal{P}_k$, in the order defined by this path. It finally serves

request $(stat(p), 0)$. This sequence starts from the depot, finishes at the depot and is such that the ending station of every request is the starting station of the next request. It thus defines a feasible schedule without any intermediate empty traveling between requests. The schedule traverses exactly T times the depot, once for every request $r(x, y)$ with $(x, y) \in \mathcal{P}_k$. So, the number of tours is $T+1$. All $KT+2K$ requests are covered by the V vehicles, which proves that $\mathcal{I}$ is positive.

Assume now that $\mathcal{I}$ is feasible. This means that there exists a schedule with the $K(T+2)$ requests assigned to the K vehicles, each vehicle making $T+1$ tours. We first show that vehicles never travel empty in this schedule.

Segment $[0, 1]$ of the ring is covered by the KT requests $r(x, y)$ with $(x, y) \in E$ and K requests $(0, stat(s))$, that is, $K(T+1)$ requests. We show that all other segments are covered by the same number of requests. We can easily check that at every station $stat(x)$ exactly K requests stop, either because $d^-(x) = K$ or because $x = s$ and the K requests $(0, stat(s))$ stop (remember that $d^-(s) = 0$ so no other request stops). Similarly, exactly K requests start from every station $stat(x)$, either because $d^+(x) = K$ or because $x = p$ and the K requests $(stat(p), 0)$ start. This shows that every segment is covered by exactly $K(T+1)$ requests which, in turn, means that none of these segments is traveled empty in a schedule composed of $K(T+1)$ tours.

We now consider the schedule of a given vehicle k. It executes $T+1$ tours so it covers at most T requests $r(x, y)$ with $(x, y) \in E$. As the KT requests $r(x, y)$ with $(x, y) \in E$ are shared by the K vehicles, every vehicle exactly fulfills T of these requests. Also, every vehicle starts and ends at the depot, and given that the vehicle never travels empty, this means that it starts with a request $(0, stat(s))$ and ends with a request $(stat(p), 0)$. Given the total number K of $(0, stat(s))$ requests and $(stat(p), 0)$ requests, each vehicle exactly fulfills one request $(0, stat(s))$ and one request $(stat(p), 0)$.

One can conclude that every vehicle starts from the depot with a request $(0, stat(s))$, continues with T requests $r(x, y)$ with $(x, y) \in E$ and finishes with request $(stat(p), 0)$, without any segment traveled empty in between. It also means that the arcs $(x, y) \in E$ associated with the requests in the vehicle schedule form a path in G, starting from s, ending at p and composed of T arcs. It proves that $\mathcal{I}_{EPP}$ is positive.

$\square$