
# Column Generation for Vehicle Routing Problems With Multiple Synchronization Constraints

Martin Fink, Guy Desaulniers, Markus Frey, Ferdinand Kiermaier, Rainer Kolisch, François Soumis

PII: S0377-2217(18)30598-8
DOI: 10.1016/j.ejor.2018.06.046
Reference: EOR 15231

To appear in: *European Journal of Operational Research*

Received date: 3 November 2016
Revised date: 27 June 2018
Accepted date: 29 June 2018

Please cite this article as: Martin Fink, Guy Desaulniers, Markus Frey, Ferdinand Kiermaier, Rainer Kolisch, François Soumis, Column Generation for Vehicle Routing Problems With Multiple Synchronization Constraints, *European Journal of Operational Research* (2018), doi: 10.1016/j.ejor.2018.06.046

## Highlights

- Ground handling modelled as vehicle routing problem with multiple synchronization constraints.
- Problem covers all five synchronization types, making it a vehicle routing problem with multiple synchronization constraints archetype.
- Two flow formulations to efficiently model all synchronization constraints.
- Branch-and-price heuristic with novel fixing technique finds near-optimal solutions.

# Column Generation for Vehicle Routing Problems With Multiple Synchronization Constraints

Martin Fink[1], Guy Desaulniers[2], Markus Frey[1], Ferdinand Kiermaier[1], Rainer Kolisch[1], François Soumis[2]

[1] TUM School of Management, Technische Universität München, München, Germany, e-mail: martin.fink@tum.de, markus.frey@tum.de, ferdinand.kiermaier@tum.de, rainer.kolisch@tum.de

[2] GERAD and Department of Mathematics and Industrial Engineering, École Polytechnique de Montréal, Montréal, Canada, e-mail: guy.desaulniers@gerad.ca, francois.soumis@gerad.ca

Corresponding author: Markus Frey, Dr.

Corresponding author's e-mail address: markus.frey@tum.de

Corresponding author's institution: TUM School of Management, Technische Universität München, München, Germany

# Column Generation for Vehicle Routing Problems With Multiple Synchronization Constraints

Martin Fink[1], Guy Desaulniers[2], Markus Frey[1], Ferdinand Kiermaier[1], Rainer Kolisch[1], and François Soumis[2]

[1]TUM School of Management, Technische Universität München, München, Germany, e-mail: martin.fink@tum.de, markus.frey@tum.de, ferdinand.kiermaier@tum.de, rainer.kolisch@tum.de
[2]GERAD and Department of Mathematics and Industrial Engineering, École Polytechnique de Montréal, Montréal, Canada, e-mail: guy.desaulniers@gerad.ca, francois.soumis@gerad.ca

July 5, 2018

**Abstract**

Synchronization of workers and vehicles plays a major role in many industries such as logistics, healthcare or airport ground handling. In this paper, we focus on operational ground handling planning and model it as an archetype of vehicle routing problems with multiple synchronization constraints, coined as "abstract vehicle routing problem with worker and vehicle synchronization" (AVRPWVS). The AVRPWVS deals with routing workers to ground handling jobs such as unloading baggage or refuelling an aircraft, while meeting each job's time window. Moreover, each job can be performed by a variable number of workers. As airports span vast distances and due to security regulations, workers use vehicles to travel between locations. Furthermore, each vehicle, moved by a driver, can carry several workers. We propose two mathematical multi-commodity flow formulations based on time-space networks to efficiently model five synchronization types including movement and load synchronization. Moreover, we develop a branch-and-price heuristic that employs both conventional variable branching and a novel variable fixing strategy. We demonstrate that the procedure achieves results close to the optimal solution in short time when compared to the two integer models.

***Keywords:*** Vehicle routing; multiple synchronization constraints; column generation; branch-and-price; variable fixing

## 1 Introduction

Many industries require workers, trucks, trailers or mini-vans to jointly complete tasks and jointly move between locations. Prominent examples for such industries include logistics and transportation, medical, maintenance or airport ground handling services. Despite its importance in the real world, the synchronization of various units is very rare in the vehicle routing problem (VRP) literature (see Lahyani, Khemakhem, and Semet (2015) and Drexl (2012)) especially when it comes to movement and load synchronization.

In this paper, we propose a branch-and-price procedure for vehicle routing problems with synchronization constraints, namely we focus on column generation for the abstract vehicle routing problem with worker and vehicle synchronization (AVRPWVS). As area of application, the AVRPWVS deals with workers and vehicles in an airport ground handling setting and aims

at minimizing job delays as well as worker and vehicle routing costs. At an airport, ground handling operations encompass jobs for airplanes stationed at parking positions, e.g. unloading bags, cleaning or refuelling the aircraft. Jobs need to be conducted in a certain sequence, e.g. unloading of inbound and transfer baggage needs to be finished before outbound baggage can be loaded. Furthermore, to ensure punctual flight operations, jobs have to be finished within time windows and can only be started once all required workers have arrived at the plane's parking position. Workers use vehicles to drive from one parking position to another due to large distances and security regulations at airports. Therefore, workers need to synchronize with vehicles in time and space.

The AVRPWVS is a special case of the vehicle routing problem with worker and vehicle synchronization (VRPWVS) presented in Fink et al. (2016) and is an archetype of the class of vehicle routing problems with multiple synchronization constraints (VRPMS) meaning that the AVRPWVS covers all five synchronization types introduced by Drexl (2012); for a detailed discussion see Section 3.1. Another example for an archetype of VRPMSs is the Vehicle Routing Problem with Trailers and Transshipments (VRPTT) introduced by Drexl (2007).

To the best of our knowledge, this paper represents the first column generation approach for a VRPMS archetype and makes two major contributions: First, we present two multi-commodity flow formulations based on non-trivial network representations with time discretization that efficiently model all five synchronization constraints. Second, we present a novel branch-and-price heuristic (BAPH) that combines branch-and-price with heuristically fixing sums of specially modified columns. The BAPH can easily be extended to an exact branch-and-price procedure.

The paper is structured as follows. In Section 2, we outline the AVRPWVS. We provide an overview of related literature in Section 3, and propose two multi-commodity flow formulations of the AVRPWVS in Sections 4 and 5. We describe the BAPH in Section 6 and present computational results (Section 7). Finally, we summarize our work and propose areas for future research in Section 8.

## 2 Problem description

In this paper, we chose the AVRPWVS as an example for a VRPMS archetype, but it should be noted that our research can also be a starting point for other branch-and-price approaches for VRPMSs. In the following, we outline an application of the AVRPWVS to ground handling services (see Fink et al. (2016)).

At airports, all aircraft are associated with a set of ground handling jobs such as refuelling or baggage loading. Each plane can only depart when all ground handling jobs have been completed. The number of jobs and their processing times depend on the aircraft type, airline, destination and number of passengers.

Each job requires a certain minimum number of workers to be started. To shorten a job's processing time, additional workers can be assigned to the job. A job only starts when all assigned workers have arrived at the respective aircraft's parking position. This is due to security reasons and due to the required alignment of workers prior to starting a job. Processing a job cannot be interrupted, nor can the number of assigned workers be altered during the job's processing. To meet the security regulations on the apron and to increase worker utilization, a worker must not wait longer than a given upper bound, e.g. 10 minutes, at a job's location both prior to the job's execution and after the job's finish time. If the worker has not been assigned to a new job until workers' maximum waiting time is reached, the worker has to return

to the depot. Each worker's availability is determined by the worker's shift. Additionally, each worker is located at a depot at the beginning and at the end of the shift.

Workers use vehicles to move from one location, such as a gate or depot, to another location. Hence, both workers and vehicles are passive units on their own. They can only move when they team up. This movement synchronization is not required if the distance between two job locations is zero, i.e. the jobs are located at the same parking position. The number of workers that can be seated on a vehicle is limited by the vehicle capacity and it should be noted that every worker can either drive or be a passenger on any vehicle. Once a vehicle arrives at a plane's parking position, workers either embark, disembark or remain seated on the vehicle. A vehicle can only move if at least one worker at the same position as the vehicle uses the vehicle. At the beginning of the day, all vehicles are located at the depot.

The goal of operational ground handling planning is to start each job of a flight as soon as possible to reduce the risk that the succeeding jobs get delayed which may lead to a late departure of the airplane incurring high penalty costs charged by the airport for changing departure slots. A delay also leads to a reduced service quality for passengers as connecting flights may not be reached (see Balakrishnan and Chandran (2010)). The slot changing cost as well as the magnitude of service quality reduction for passengers depend upon the aircraft size. In addition to minimizing the overall flight job delay as our major goal, the available workers and vehicles should be used as efficiently as possible. Consequently, we aim at a minimal travel and waiting time for workers and a minimal travel time for vehicles.

## 3 Literature review

Excluding worker qualifications and different vehicle capacities, the AVRPWVS is a special case of the VRPWVS and belongs to the class of NP-hard problems as it generalizes the NP-hard vehicle routing problem with time windows (VRPTW) (see Lenstra and Kan (1981)). The major properties that the AVRPWVS shares with the VRPWVS are the multiple synchronization constraints (VRPMSs, see Drexl (2012)) which marks it as a VRPMS archetype. The five known synchronization types and how the AVRPWVS covers them are described in Section 3.1. Related column generation approaches for VRPs with movement synchronization en route and at the depot are discussed in Section 3.2. For non-column generation approaches for the VRPMS archetype with passive load synchronization constraints see Drexl (2014) who developed a branch-and-cut algorithm.

### 3.1 The five synchronization types in the AVRPWVS

Drexl (2012) introduced five synchronization types to characterize VRPMSs. In the following, we briefly describe each synchronization type and how they appear in the AVRPWVS. When we refer to the term *unit*, we mean a worker or vehicle. We define a *task* (used as synonym for the term job) as a service that has to be done (e.g. loading bags off an aircraft).

*Resource synchronization* entails that the supply of a given resource used by all units is limited. In the AVRPWVS, vehicle capacity is limited for example when workers travel on vehicles from the start depot to a parking position. When multiple units coordinate their operation in time or space, *operation synchronization* is required. For example, prior to workers starting from the depot to a parking position, they need to embark on one out of many vehicles. *Movement synchronization* prevails when two different types of units require each other to travel from one location to another. For instance, workers and vehicles need to synchronize in

time and space when they travel from the depot to a parking position. *Task synchronization* is required when more than one unit is employed for processing the task and all units processing the task have to start and finish at the same time. In the AVRPWVS, each job is completed exactly once by a given number of workers. *Load synchronization* ensures that, whenever any type of load is transferred between different units, no load gets lost and that the right amount of load is delivered to a customer. As described in Fink et al. (2016), we extend load synchronization by differentiating between active load (e.g. workers) and passive load (e.g. baggage). The AVRPWVS covers active load synchronization because the amount of workers that change vehicles is synchronized. Our modelling approaches, see Sections 4 and 5, consider both operation and load synchronization implicitly: embarking and disembarking of workers is not explicitly modelled, neither is the changing of vehicles by workers. However, we explicitly model resource, movement and task synchronization.

### 3.2 Related column generation approaches for VRPs with movement synchronization

In the following, we briefly outline related column generation approaches for VRPs with movement synchronization. For a general introduction to branch-and-price, see Barnhart et al. (1998) or Desrosiers and Lübbecke (2005).

Tilk et al. (2015) present an exact branch-and-price procedure for the active-passive vehicle-routing problem which was first introduced by Meisel and Kopfer (2014). This problem entails pickup-and-delivery requests that require a joint operation of active vehicles and passive vehicles, which in turn can be combined at each customer location. It is the first exact procedure for a VRP with movement synchronization en route. However, the active-passive vehicle routing problem does not entail load synchronization and is thus no VRPMS archetype.

Visser (2015) analyzes a simultaneous vehicle routing and crew scheduling problem arising in the distribution of goods from a depot to customers using trailers, trucks and drivers. A truck and driver combination can change trailers only at the depot. The author investigates the performance of construction heuristics and column generation methods with exact and heuristic pricing. To obtain an integral solution, the author solves the master problem as an integer linear program once column generation terminates.

Hollis, Forbes, and Douglas (2006) consider a combined vehicle and crew routing and scheduling problem to solve the urban mail distribution problem of Australia Post. They model the problem as a pick-up-and-delivery problem (PDP) with a time horizon of 24 hours, multiple depots and a heterogeneous set of drivers that needs to be synchronized with a heterogeneous set of vehicles at several locations. Therefore, movement synchronization en route prevails. The authors solve the problem in two steps. First, they determine feasible vehicle routes by solving a path-based MIP model with a heuristic column generation approach and second, they generate worker and vehicle schedules that fit to the routes generated by the heuristic column generation approach.

Xiang, Chu, and Chen (2006) describe a dial-a-ride problem with movement synchronization of workers with vehicles only at depots. The authors solve the problem using a heuristic and demonstrate that it finds solutions with a small gap to a lower bound obtained by an algorithm based on column generation.

# 4 Multi-commodity flow formulation I

In the following, we introduce the notation (see Section 4.1) for the time-space network representation of the problem (see Section 4.2) and a first multi-commodity flow formulation (MILP I) of the AVRPWVS (see Section 4.3).

## 4.1 Notation

General sets and parameters for the problem statement are given in Sections 4.1.1 and 4.1.2. The vertices and arcs required for the time-space network representation are given in Sections 4.1.3 and 4.1.4. In Section 4.1.5, we introduce the decision variables.

### 4.1.1 Jobs and time

We define a set of ground handling jobs $\mathcal{J}$ that can only start when all workers assigned to the job have arrived. Neither can a job be interrupted nor can a worker leave during the processing of the job. Each job $h \in \mathcal{J} = \{1, ..., n\}$ has several start times $\mathcal{T}_h^{\mathrm{S}} = \{ES_h, \ldots, LS_h\}$ with $ES_h$ and $LS_h$ representing the earliest and the latest start of a job, respectively. For security reasons and to avoid long waiting times for workers on the apron, a worker must not arrive earlier than $\tau \in \mathbb{N}_0^+$ periods before the execution of a job starts and must be assigned to a new job or has to be picked up $\tau$ periods after the job was processed. Thus, the earliest arrival and latest departure times for a worker and a vehicle at job $h$ is given by $\mathcal{T}_h^{\mathrm{A}} = \{EA_h, \ldots, LD_h\}$ where $EA_h = ES_h - \tau$, $LD_h = LS_h + p_h^{\max} + \tau$ and $p_h^{\max}$ stands for the longest processing time of job $h$. Each job $h$ is executed exactly in one mode $m \in \mathcal{M}_h$ where $m$ stands for the number of workers assigned to job $h$ and is related to a processing time $p_{h,m}^{\mathrm{job}}$. In addition, we define the start depot as a dummy job 0 and denote the set of ground handling jobs and the start depot as $\mathcal{J}^0$. Likewise, the end depot is modelled as a dummy job $n+1$ and the set of ground handling jobs and the end depot is defined as $\mathcal{J}^{n+1}$. Eventually, the set of all jobs is given as $\mathcal{J}^{0,n+1}$. Precedence relationships exist because certain jobs (e.g. loading baggage) can only be started after other jobs (e.g. unloading baggage) have been completed. We define the set of all precedence relationships as $\mathcal{E} \subset \mathcal{J} \times \mathcal{J}$.

### 4.1.2 Workers and vehicles

We define the set of ground handling workers as $\mathcal{W} = \{1, ..., W\}$ and the set of vehicles as $\mathcal{K} = \{1, ..., K\}$. We assume that the vehicles' capacity, i.e. the number of workers that can be transported at a time, is identical for each vehicle and given as *cap*.

### 4.1.3 Vertices

We propose a set of vertices $\mathcal{V}$, where each vertex $i(h, t, type) \in \mathcal{V}$ is uniquely defined by job $h \in \mathcal{J}$, a discrete time $t \in \mathcal{T}_h^A$ and a *type* indicating if the vertex is an arrival or departure vertex. We define $\mathcal{V}_h^{\mathrm{arr}}$ and $\mathcal{V}_h^{\mathrm{dep}}$ as the sets of arrival or departure vertices of job $h$. The start depot job 0 features several departure vertices and the end depot several arrival vertices. Similar to jobs, we denote the set of all vertices including start and end depot vertices as $\mathcal{V}^{0,n+1}$. Throughout the paper, we use indexes $i$ and $j$ to denote vertices, and indexes $g$ and $h$ for jobs.

#### 4.1.4 Arcs

An arc $(i, j) \in \mathcal{A}$ connects two vertices $i$ and $j$ with $i \neq j$. The subset of all ingoing arcs to vertex $i$ and subset of all outgoing arcs from $i$ are denoted as $\mathcal{A}_i^{\text{in}}$ and $\mathcal{A}_i^{\text{out}}$, respectively. We further cluster the set of all arcs $\mathcal{A}$ into four arc types: processing, transitioning, waiting and travel arcs denoted by the sets $\mathcal{A}^{\text{proc}}$, $\mathcal{A}^{\text{trans}}$, $\mathcal{A}^{\text{wait}}$ and $\mathcal{A}^{\text{travel}}$, respectively. Workers can either work on a job, which means they must be routed across a processing arc, or not work on a job, which means that they are routed across a transition arc to get from an arrival to a departure vertex in the same time and location. Workers may also wait at a parking position using waiting arcs or travel to other job locations using travel arcs. We name waiting arcs between arrival vertices "arrival waiting arcs" and waiting arcs between departure vertices "departure waiting arcs".

The set of travel arcs with a positive travel duration is denoted as $\mathcal{A}^{\text{travel},>}$ and the set of travel arcs with a travel duration of zero as $\mathcal{A}^{\text{travel},0}$. Finally, we define the set of all outgoing travel arcs from the start depot as $\mathcal{A}^{\text{travel,start}}$ and the set of all ingoing travel arcs to the end depot as $\mathcal{A}^{\text{travel,end}}$. Table 1 shows an overview of all sets of arcs used in the multi-commodity formulation I.

Vehicles cannot process jobs and therefore are not routed across processing arcs. To reduce symmetry, vehicles do not use the set of departure waiting arcs but the set of arrival waiting arcs which we denote as $\mathcal{A}^{\text{k,wait}}$. The resulting set of vehicle arcs is denoted as $\mathcal{A}^{\text{k}}$ excluding both processing and departure waiting arcs. Following the definitions for ingoing and outgoing arcs above, the subsets of vehicle ingoing arcs to vertex $i$ and outgoing arcs from vertex $i$ are given by $\mathcal{A}_i^{\text{k,in}}$ and $\mathcal{A}_i^{\text{k,out}}$, respectively.

Table 1: Arcs and subsets of arcs used for the multi-commodity formulation I.

| **Arc set** | **Description** |
|---|---|
| $\mathcal{A}$ | Set of all arcs |
| $\mathcal{A}^{\text{proc}}, \mathcal{A}^{\text{trans}}, \mathcal{A}^{wait}, \mathcal{A}^{travel}$ | Set of processing, transition, wait or travel arcs |
| $\mathcal{A}^{\text{travel},>}(\mathcal{A}^{\text{travel},0})$ | Set of travel arcs with a positive (zero) travel duration |
| $\mathcal{A}^{\text{travel,start}}$ | Set of outgoing travel arcs from the start depot 0 |
| $\mathcal{A}^{\text{travel,end}}$ | Set of ingoing travel arcs to the end depot $n+1$ |
| $\mathcal{A}^{\text{k}}$ | Set of all arcs for vehicles (excluding processing and departure waiting arcs) |
| $\mathcal{A}^{\text{k,wait}}$ | Set of all waiting arcs for vehicles (i.e. arrival waiting arcs) |
| $\mathcal{A}_i^{\text{in}}$, $\mathcal{A}_i^{\text{k,in}}$ | Set of (vehicle) ingoing arcs to vertex $i$ for arc set $\mathcal{A}$ and $\mathcal{A}^{\text{k}}$ |
| $\mathcal{A}_i^{\text{out}}$, $\mathcal{A}_i^{\text{k,out}}$ | Set of (vehicle) outgoing arcs from vertex $i$ for arc set $\mathcal{A}$ and $\mathcal{A}^{\text{k}}$ |

Each arc has a weight representing time. The weight of a processing arc $(i, j)$, $p_{i,j}^{\text{arc}}$, equals the processing time $p_{h,m}^{\text{job}}$ for a job $h$ performed in mode $m$. For a transition arc, the weight is 0 and for a waiting arc, the weight is 1 indicating the length between two successive points in time. Finally for a travel arc, the weight equals the travel duration $d_{i,j}$.

#### 4.1.5 Decision variables

We use the following decision variables:

- Worker routing: $x_{w,i,j}^{\text{w}}$ is 1, if worker $w$ goes from vertex $i$ to $j$
- Vehicle routing: $x_{k,i,j}^{\text{k}}$ is 1, if vehicle $k$ goes from vertex $i$ to $j$

- Job start and mode: $y_{h,m,t}$ is 1, if job $h$ starts with $m$ workers at time $t$

### 4.2 Network representation

We define the worker network $\mathcal{G}^{\mathrm{w}} = (\mathcal{V}, \mathcal{A})$ based on the set of all vertices $\mathcal{V}$ and the set of all arcs $\mathcal{A}$ (see Figure 1). The vehicle network is similar to the worker network but features the set of arcs $\mathcal{A}^k$. The resulting vehicle network $\mathcal{G}^{\mathrm{k}} = (\mathcal{V}, \mathcal{A}^{\mathrm{k}})$ is given in Figure 2. An example for an excluded departure waiting arc in the vehicle network is the lack of a connection between the departure vertices $(g, t_1)$ and $(g, t_2)$, and an example for an excluded processing arc is the lack of a link between arrival vertex $(g, t_1)$ and departure vertex $(g, t_2)$.

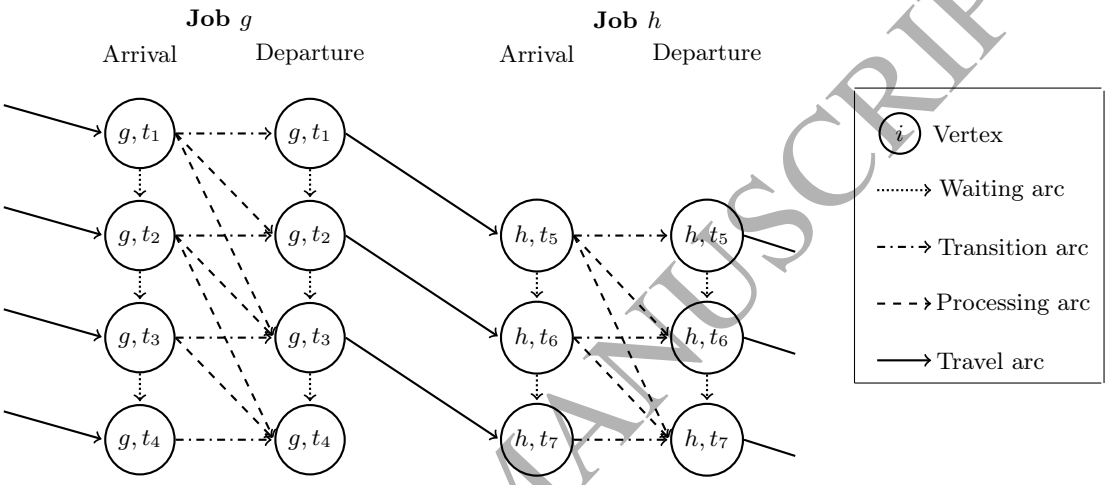


Figure 1: Part of a worker network

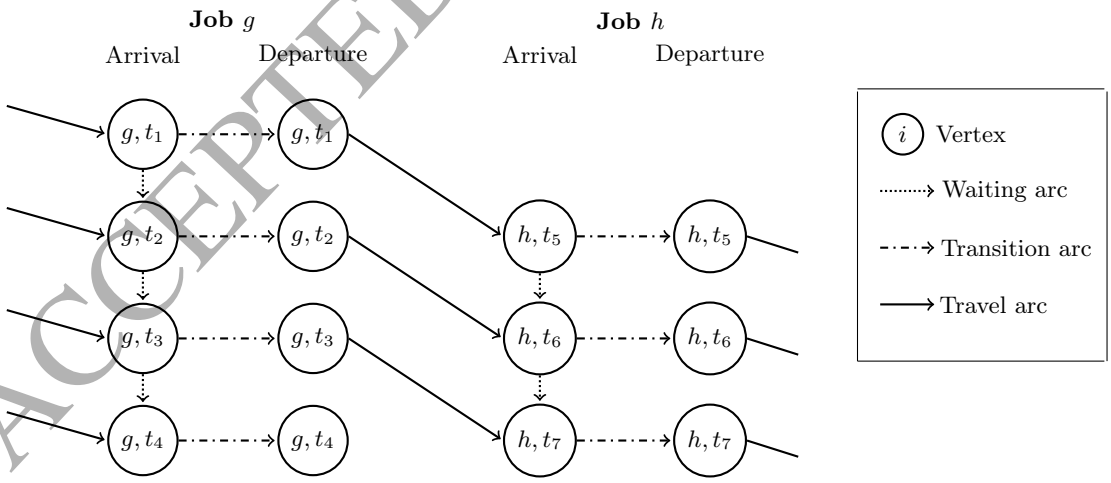


Figure 2: Part of a vehicle network

### 4.3 Mixed integer linear program I

We first derive the objective function in Section 4.3.1, before we state model MILP I in Section 4.3.2.

#### 4.3.1 Objective function

The main goal of operational ground handling planning is to minimize the delay of jobs, but we also want to minimize workers' and vehicles' route costs. Consequently, we define the objective function weight $\alpha$ for the delay of jobs, $\beta$ for worker and $\gamma$ for vehicle routes. We attain the parameter $\sigma_h$ to job $h$. In our case, $\sigma_h$ grows linearly with the job's aircraft size because the larger an aircraft, the more passengers are affected from a job delay. The duration of a tour for worker $w$ is defined as

$$C^{\mathrm{w}}_w = \sum_{(i,j)\in\mathcal{A}^{\mathrm{travel}}} d_{i,j}\cdot x^{\mathrm{w}}_{w,i,j} + \sum_{(i,j)\in\mathcal{A}^{\mathrm{proc}}} p^{\mathrm{arc}}_{i,j}\cdot x^{\mathrm{w}}_{w,i,j} + \sum_{(i,j)\in\mathcal{A}^{\mathrm{wait}}} 1\cdot x^{\mathrm{w}}_{w,i,j} \tag{1}$$

where the first, second and third term yields the travel, the processing and the waiting time, respectively, of worker $w$. The length of a tour for vehicle $k$ is given as

$$C^{\mathrm{k}}_k = \sum_{(i,j)\in\mathcal{A}^{\mathrm{travel}}} d_{i,j}\cdot x^{\mathrm{k}}_{k,i,j}. \tag{2}$$

For vehicles, we explicitly do not minimize the waiting time, as a high vehicle waiting time can result in less vehicle movements which in turn may lead to reduced fuel and maintenance requirements.

#### 4.3.2 Model formulation

The MILP I is stated as follows:

$$\text{Minimize } \alpha\cdot\sum_{h\in\mathcal{J}}\sigma_h\cdot\Big(\sum_{m\in\mathcal{M}_h}\sum_{t\in\mathcal{T}^{\mathrm{S}}_h}(t+p^{\mathrm{job}}_{h,m})\cdot y_{h,m,t} - ES_h\Big) + \beta\cdot\sum_{w\in\mathcal{W}} C^{\mathrm{w}}_w + \gamma\cdot\sum_{k\in\mathcal{K}} C^{\mathrm{k}}_k \tag{3}$$

subject to

*Job constraints:*

$$\sum_{m\in\mathcal{M}_h}\sum_{t\in\mathcal{T}^{\mathrm{S}}_h} y_{h,m,t} = 1 \qquad \forall\, h\in\mathcal{J} \tag{4}$$

$$\sum_{w\in\mathcal{W}} x^{\mathrm{w}}_{w,i(h,t,arr),j(h,t+p^{\mathrm{job}}_{h,m},dep)} = m\cdot y_{h,m,t} \qquad \forall\, h\in\mathcal{J},\ m\in\mathcal{M}_h, t\in\mathcal{T}^{\mathrm{S}}_h \tag{5}$$

$$\sum_{m\in\mathcal{M}_g}\sum_{t\in\mathcal{T}^{\mathrm{S}}_g}(t+p^{\mathrm{job}}_{g,m})\cdot y_{g,m,t} \le \sum_{m\in\mathcal{M}_h}\sum_{t\in\mathcal{T}^{\mathrm{S}}_h} t\cdot y_{h,m,t} \qquad \forall\,(g,h)\in\mathcal{E} \tag{6}$$

*Worker constraints:*

$$\sum_{(i,j)\in\mathcal{A}^{\mathrm{travel,start}}} x^{\mathrm{w}}_{w,i,j} = \sum_{(i,j)\in\mathcal{A}^{\mathrm{travel,end}}} x^{\mathrm{w}}_{w,i,j} = 1 \qquad \forall\, w\in\mathcal{W} \tag{7}$$

$$\sum_{(j,i)\in\mathcal{A}^{\mathrm{in}}_i} x^{\mathrm{w}}_{w,j,i} = \sum_{(i,j)\in\mathcal{A}^{\mathrm{out}}_i} x^{\mathrm{w}}_{w,i,j} \qquad \forall\, w\in\mathcal{W}, i\in\mathcal{V} \tag{8}$$

*Vehicle constraints:*

$$\sum_{(i,j)\in\mathcal{A}^{\mathrm{travel,start}}} x^{\mathrm{k}}_{k,i,j} = \sum_{(i,j)\in\mathcal{A}^{\mathrm{travel,end}}} x^{\mathrm{k}}_{k,i,j} \le 1 \qquad \forall\, k\in\mathcal{K} \tag{9}$$

$$\sum_{(j,i)\in\mathcal{A}^{\mathrm{k,in}}_i} x^{\mathrm{k}}_{k,j,i} = \sum_{(i,j)\in A^{\mathrm{k,out}}_i} x^{\mathrm{k}}_{k,i,j} \qquad \forall\, k\in\mathcal{K}, i\in\mathcal{V} \tag{10}$$

*Movement synchronization constraints:*

$$\sum_{k\in\mathcal{K}} x^{\mathrm{k}}_{k,i,j} \leq \sum_{w\in\mathcal{W}} x^{\mathrm{w}}_{w,i,j} \qquad \forall\ (i,j) \in \mathcal{A}^{\mathrm{travel},>} \tag{11}$$

$$\sum_{w\in\mathcal{W}} x^{\mathrm{w}}_{w,i,j} \leq \sum_{k\in\mathcal{K}} cap \cdot x^{\mathrm{k}}_{k,i,j} \qquad \forall\ (i,j) \in \mathcal{A}^{travel,>} \tag{12}$$

*Variable declarations:*

$$x^{\mathrm{w}}_{w,i,j} \in \{0,1\} \qquad \forall\ w \in \mathcal{W}, (i,j) \in \mathcal{A} \tag{13}$$

$$x^{\mathrm{k}}_{k,i,j} \in \{0,1\} \qquad \forall\ k \in \mathcal{K}, (i,j) \in \mathcal{A}^{\mathrm{k}} \tag{14}$$

$$y_{h,m,t} \in \{0,1\} \qquad \forall\ h \in \mathcal{J}^{n+1}, m \in \mathcal{M}_h, t \in \mathcal{T}^{\mathrm{S}}_h \tag{15}$$

The objective function (3) minimizes the sum of the weighted job delays, the sum of the of the tours of workers' and the sum of the duration of the vehicles' tours.

Constraints can be clustered into job (4) - (6), worker (7) - (8), vehicle (9) - (10) and movement synchronization constraints (11) - (12) as well as variable declarations (13) - (15).

**Job constraints (4) - (6):** Each job is started once in one mode (4). Constraint (5) ensures that all workers use the same processing arc and connects the worker routing with the job variable (5). Constraint (6) represents the precedence relationship between pairs of jobs. To tighten the formulation, we henceforth replace constraint (6) with the precedence constraint proposed by Christofides, Alvarez-Valdés, and Tamarit (1987):

$$y_{g,m,t} + \sum_{u\in\mathcal{T}^{\mathrm{S}}_h : u<t+p^{\mathrm{job}}_{g,m}} y_{h,m,u} \leq 1 \qquad \forall\ (g,h) \in \mathcal{E}, t \in \mathcal{T}^{\mathrm{S}}_g, m \in \mathcal{M}_g \tag{16}$$

**Worker constraints (7) - (8):** Constraint (7) enforces that each worker leaves and returns to the depot. Flow conservation of workers at every vertex other than the depot vertices is expressed by constraint (8).

**Vehicle constraints (9) - (10):** Constraint (9) enables vehicles to either leave or stay at the depot. If vehicles leave the depot, they also need to return. Constraint (10) represents the vehicle flow conservation constraint (see Ahuja, Magnanti, and Orlin (1993)).

**Movement synchronization constraints (11) - (12):** For each travel arc that workers and vehicles share, the number of workers is at least as high as the number of vehicles to guarantee that there is always at least one driver per vehicle (11) and the number of workers is lower or equal to the capacity of all vehicles on the same arc (12). Movement synchronization is only needed when the travel time on a travel arc is greater than zero.

## 5 Multi-commodity flow formulation II

In the following, we present an alternative model for the AVRPWVS denoted as MILP II. It is inspired by Haase, Desaulniers, and Desrosiers (2001) and features worker routes but no explicit vehicle routes. In fact, the flow of vehicles is covered by both an extended worker network including vehicle travel (outlined below) and by additional vehicle flow variables capturing vehicles' waiting and travel of zero distance at the same parking position.

## 5.1 Notation

We now present additional notation necessary for the MILP II.

### 5.1.1 Arcs

To capture vehicle travel between locations with a travel duration greater than zero, we distinguish between the set of worker travel arcs $\mathcal{A}^{\text{travel, w}}$ and the set of worker and vehicle travel arcs $\mathcal{A}^{\text{travel, w+k}}$. Consequently, each travel between vertices $i$ and $j$ now features both a worker travel arc and a parallel worker and vehicle travel arc. Note that a worker and vehicle travel arc is weighted twice as much as a worker travel arc in the objective function, i.e. for arc $(i, j)$ we have $2 \cdot d_{i,j}$. The set of all travel arcs is given by $\mathcal{A}^{\text{travel}} = \mathcal{A}^{\text{travel, w+k}} \cup \mathcal{A}^{\text{travel, w}}$. The set of all vehicle arcs $\mathcal{A}^{\tilde{k}} = \mathcal{A}^{\text{k,wait}} \cup \mathcal{A}^{\text{travel,0}}$ is the union of the sets of waiting arcs and zero travel arcs at a location. The set of vehicle vertices is given by $\mathcal{V}^{\text{k}}$ and excludes the start and end depot. A vertex $i \in \mathcal{V}^{\text{k}}$ only corresponds to a job $h$ and its time $t$, but not to an arrival or departure type.

Similar to Section 4.1.4, for arcs with positive travel duration only, we use the notation $\mathcal{A}^{\text{travel,>,w}}$ and $\mathcal{A}^{\text{travel,>,w+k}}$. The subsets of ingoing arcs to vertex $i$ and outgoing arcs from vertex $i$ of arc set $\mathcal{A}^{\text{travel,>,w+k}}$ ($\mathcal{A}^{\tilde{k}}$) are given by $\mathcal{A}_i^{\text{travel,>, w+k, in}}$ ($\mathcal{A}_i^{\tilde{k},\text{in}}$) and $\mathcal{A}^{\text{travel,>,w+k, out}_i}$ ($\mathcal{A}_i^{\tilde{k},\text{out}}$), respectively. The set of outgoing arcs from the depot is given by $\mathcal{A}^{\text{travel,w+k, start}}$. All additional arcs are summarized in Table 2.

Table 2: Arcs and subsets of arcs used for the multi-commodity formulation II.

| **Arc set** | **Description** |
|---|---|
| $\mathcal{A}^{\text{travel, w}}$ | Worker travel arcs |
| $\mathcal{A}^{\text{travel, w+k}}$ | Worker and vehicle travel arcs |
| $\mathcal{A}^{\tilde{k}}$ | Union of waiting arcs and zero travel arcs |
| $\mathcal{A}^{\text{travel,>,w}}$ ($\mathcal{A}^{\text{travel,>,w+k}}$) | Set of worker (and vehicle) travel arcs with a positive travel duration |
| $\mathcal{A}_i^{\text{travel,>, w+k, in}}$ ($\mathcal{A}_i^{\tilde{k},\text{in}}$) | Subset of ingoing arcs to vertex $i$ of set $\mathcal{A}^{\text{travel,>, w+k}}$ ($\mathcal{A}^{\tilde{k}}$) |
| $\mathcal{A}_i^{\text{travel,>, w+k, out}}$ ($\mathcal{A}_i^{\tilde{k},\text{out}}$) | Subset of outgoing arcs from vertex $i$ of set $\mathcal{A}^{\text{travel,>, w+k}}$ ($\mathcal{A}^{\tilde{k}}$) |
| $\mathcal{A}_{i,j}^{\text{travel, w+k}}$ | Subset of worker and vehicle arcs that are parallel to worker travel arc $(i, j) \in \mathcal{A}^{\text{travel, w+k}}$ |

Figure 3 illustrates the new set of worker and vehicle arcs in the so-called extended worker network. We denote the subset of worker and vehicle travel arcs that are parallel to the worker travel arc $(i, j)$ as $\mathcal{A}_{i,j}^{\text{travel, w+k}}$.

All additional introduced arc sets are summarized in Table reftab:Arcs2.

### 5.1.2 Decision variables

The travel of vehicles between locations with travel duration greater than zero is captured using the new set of travel arcs as introduced above. The vehicle flow "at the same location" is modelled by the decision variable $v_{i,j} \in [0, K]$ defined for all $(i, j) \in \mathcal{A}^{\tilde{k}}$.

To sum it up, we replace the vehicle network as in Figure 2 using both the extended worker network and the new flow variables $v_{i,j}$. Consequently, we remove the vehicle routing decision variables $x_{k,i,j}^{\text{k}}$ as they are no longer needed.

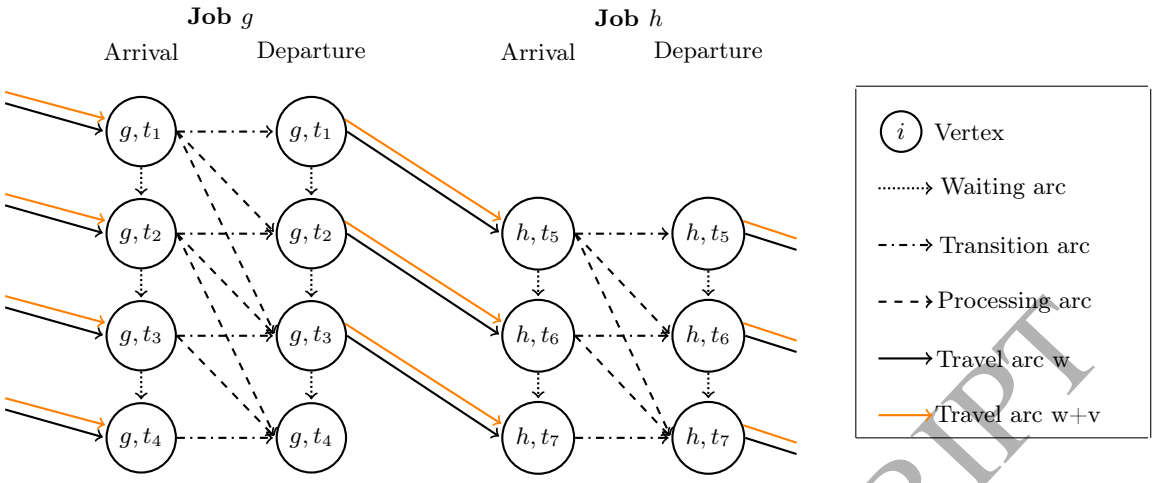


Figure 3: Part of an extended worker network

## 5.2 Mixed integer linear program II

The MILP II is equivalent to the MILP I in the sense that the feasible regions of both models contain the same solutions. Due to the consideration of different arc types, the write up of the objective function is slightly different to that of MILP I (see Section 4.3.1). The complete formulation of the MILP II is stated in Section 5.2.2.

### 5.2.1 Objective function

To account for the additional worker and vehicle travel arcs $\mathcal{A}^{travel,w+k}$, the cost of a route of worker $w$ is now given by

$$C_w^{\mathrm{w+k}} = \sum_{(i,j)\in\mathcal{A}^{travel}} d_{i,j} \cdot x^{\mathrm{w}}_{w,i,j} + \sum_{(i,j)\in\mathcal{A}^{proc}} p^{\mathrm{arc}}_{i,j} \cdot x^{\mathrm{w}}_{w,i,j} + \sum_{(i,j)\in\mathcal{A}^{\mathrm{wait}}} 1 \cdot x^{\mathrm{w}}_{w,i,j} \tag{17}$$

### 5.2.2 Model formulation

The MILP II is stated as follows:

$$\text{Minimize } \alpha \cdot \sum_{h\in\mathcal{J}} \sigma_h \cdot \Big( \sum_{m\in\mathcal{M}_h} \sum_{t\in\mathcal{T}_h^{\mathrm{S}}} (t + p^{\mathrm{job}}_{h,m}) \cdot y_{h,m,t} - ES_h \Big) + \beta \cdot \sum_{w\in\mathcal{W}} C_w^{\mathrm{w+v}} \tag{18}$$

subject to

*Job constraints:*

$$\sum_{m\in\mathcal{M}_h} \sum_{t\in\mathcal{T}_h^{\mathrm{S}}} y_{h,m,t} = 1 \qquad \forall\, h \in \mathcal{J} \tag{19}$$

$$\sum_{w\in\mathcal{W}} x^{\mathrm{w}}_{w,i(h,t,arr),j(h,t+p^{\mathrm{job}}_{h,m},dep)} = m \cdot y_{h,m,t} \qquad \forall\, h \in \mathcal{J},\ m \in \mathcal{M}_h, t \in \mathcal{T}_h^{\mathrm{S}} \tag{20}$$

$$y_{g,m,t} + \sum_{u\in\mathcal{T}_h^{\mathrm{S}}: u<t+p^{\mathrm{job}}_{g,m}} y_{h,m,u} \le 1 \qquad \forall\, (g,h) \in \mathcal{E}, t \in \mathcal{T}_g^{\mathrm{S}}, m \in \mathcal{M}_g \tag{21}$$

*Worker constraints:*

$$\sum_{(i,j)\in\mathcal{A}^{\text{travel,start}}} x^{\text{w}}_{w,i,j} = \sum_{(i,j)\in\mathcal{A}^{\text{travel,end}}} x^{\text{w}}_{w,i,j} = 1 \qquad \forall\ w \in \mathcal{W} \tag{22}$$

$$\sum_{(j,i)\in\mathcal{A}^{\text{in}}_i} x^{\text{w}}_{w,j,i} = \sum_{(i,j)\in\mathcal{A}^{\text{out}}_i} x^{\text{w}}_{w,i,j} \qquad \forall\ w \in \mathcal{W}, i \in \mathcal{V} \tag{23}$$

*Vehicle flow constraints:*

$$\sum_{w\in\mathcal{W}} \sum_{(i,j)\in\mathcal{A}^{\text{travel,w+k,start}}} x^{\text{w}}_{w,i,j} \leq K \tag{24}$$

$$\sum_{w\in\mathcal{W}} \sum_{(j,i)\in\mathcal{A}^{\text{travel,}>\text{, w+k, in}}_i} x^{\text{w}}_{w,j,i} + \sum_{(j,i)\in\mathcal{A}^{\tilde{k},in}_i} v_{j,i}$$
$$= \sum_{w\in\mathcal{W}} \sum_{(i,j)\in\mathcal{A}^{\text{travel,}>\text{,w+k, out}}_i} x^{\text{w}}_{w,i,j} + \sum_{(i,j)\in\mathcal{A}^{\tilde{k}\text{, out}}_i} v_{i,j} \qquad \forall\ i \in \mathcal{V}^{\text{k}} \tag{25}$$

*Movement synchronization constraints:*

$$\sum_{w\in\mathcal{W}} x^{\text{w}}_{w,i,j} \leq \sum_{w\in\mathcal{W}} \sum_{(i',j')\in A^{\text{travel, }>\text{, w+k}}_{i,j}} x^{\text{w}}_{w,i,j} \cdot (cap - 1) \qquad \forall (i,j) \in \mathcal{A}^{\text{travel, }>\text{ ,w}} \tag{26}$$

*Variable declarations:*

$$x^{\text{w}}_{w,i,j} \in \{0,1\} \qquad \forall\ w \in \mathcal{W}, (i,j) \in \mathcal{A} \tag{27}$$

$$v_{i,j} \in [0,K] \qquad \forall\ (i,j) \in \mathcal{A}^{\tilde{k}} \tag{28}$$

$$y_{h,m,t} \in \{0,1\} \qquad \forall\ h \in \mathcal{J}^{n+1}, m \in \mathcal{M}_h, t \in \mathcal{T}^{\text{S}}_h \tag{29}$$

The objective function of the model (18) is similar to the objective function of the MILP I (3) except that we remove the vehicle routes and use extended worker routes. Note that the objective function value of an optimal solution of the MILP II is equal to that of the MILP I because a worker route features vehicle travels in its cost.

**Job and worker constraints (19) - (23):** Job constraints remain unchanged and worker constraints as well except that the set of all arcs and the set of travel arcs now encompass also worker and vehicle arcs.

**Vehicle flow constraints (24) - (25):** The number of vehicles that leave the depot must be less or equal to $K$ as defined in constraint (24). Constraint (25) ensures that the vehicle flow coming into each vehicle vertex equals the flow leaving the vehicle vertex.

**Movement synchronization constraint (26):** To synchronize workers and vehicles, we use constraint (26) to ensure there is enough vehicle capacity. Note that we can only offer a capacity of $cap - 1$ on a worker and vehicle travel arc as one capacity unit is already reserved for the worker driving the vehicle. As a consequence, we need no constraint enforcing at least one worker per vehicle, as this is given by the worker and vehicle travel arc.

Constraints (27) - (29) define the decision variables. In comparison to the MILP I, the MILP II encompasses approximately half the number of constraints, slightly fewer binary decision variables but considerably more continuous variables.

# 6 Branch-and-price

We propose a branch-and-price heuristic (BAPH) for the AVRPWVS as column generation is renowned as an efficient approach for solving VRPs (see Desrosiers and Lübbecke (2005), Grønhaug et al. (2010) or Desaulniers (2010)). To perform column generation, we reformulate the MILP II as a set-partitioning master problem (MP) in Section 6.1 and define the pricing subproblem in Section 6.2. Column generation (CG) solves a linear relaxation and is an iterative approach that in each iteration considers only a subset of all feasible columns or routes in the MP, which is why it is called restricted master problem (RMP). While the RMP guides the generation of new columns, the creation of new columns is achieved by the subproblem (SP), also called the pricing problem. In each iteration, the RMP is solved with a given set of columns as a linear program (LP). Then, the dual values of the RMP are used in the SP to generate new routes. The SP either finds new columns with negative reduced costs that are thereafter added to the RMP or proofs that there are none and the procedure terminates with a valid lower bound on the optimal value of the minimization problem. To solve the SP, we use a dynamic program labelling algorithm given in Section 6.3. Section 6.4 describes several acceleration strategies we implemented to speed up the procedure. Finally, Section 6.5 outlines how we obtain integral solutions.

## 6.1 Set-partitioning master problem formulation

We reformulate the MILP II as a set-partitioning model using Dantzig-Wolfe decomposition (see e.g. Lübbecke and Desrosiers (2005)). Let $r$ denote a worker route or column and $\mathcal{R}$ the set of all columns. For each route $r$ we define the following notation:

- $f_r$: binary variable equal to 1, if the route $r$ is selected
- $\delta_{i,j,r}$: binary parameter equal to 1, if the arc from $i$ to $j$ is used in route $r$
- $\delta^{\mathrm{w}}_{i,j,r}$: binary parameter equal to 1, if the worker travel arc from $i$ to $j$ is used in route $r$
- $\delta^{\mathrm{w+k}}_{i,j,r}$: binary parameter equal to 1, if the worker and vehicle travel arc from $i$ to $j$ is used in route $r$
- $C^{\mathrm{r}}_r$: cost of route $r$ as given in (17)

We can now define the MP as follows:

$$\text{Minimize } \alpha \cdot \sum_{h \in \mathcal{J}} \sigma_h \cdot \Big( \sum_{m \in \mathcal{M}_h} \sum_{t \in \mathcal{T}^{\mathrm{S}}_h} (t + p^{\mathrm{job}}_{h,m}) \cdot y_{h,m,t} - ES_h \Big) + \beta \cdot \sum_{r \in R} C^{\mathrm{r}}_r \cdot f_r \tag{30}$$

subject to

$$\sum_{m \in \mathcal{M}_h} \sum_{t \in \mathcal{T}^{\mathrm{S}}_h} y_{h,m,t} = 1 \qquad \forall\, h \in \mathcal{J} \tag{31}$$

$$\sum_{r \in \mathcal{R}} \delta_{i(h,t,arr),j(h,t+p^{\mathrm{job}}_{h,m},dep),r} \cdot f_r = m \cdot y_{h,m,t} \qquad \forall\, h \in \mathcal{J},\ m \in \mathcal{M}_h, t \in \mathcal{T}^{\mathrm{S}}_h \tag{32}$$

$$y_{g,m,t} + \sum_{u \in \mathcal{T}^{\mathrm{S}}_h : u < t + p^{\mathrm{job}}_{g,m}} y_{h,m,u} \leq 1 \qquad \forall\, (g,h) \in \mathcal{E}, t \in \mathcal{T}^{\mathrm{S}}_g, m \in \mathcal{M}_g \tag{33}$$

$$\sum_{r \in \mathcal{R}} f_r = W \tag{34}$$

$$\sum_{r\in\mathcal{R}}\sum_{(i,j)\in\mathcal{A}^{\text{travel},>,\text{w+k,start}}}\delta^{\text{w+k}}_{i,j,r}\cdot f_r \leq K \tag{35}$$

$$\sum_{r\in\mathcal{R}}\sum_{(j,i)\in\mathcal{A}_i^{\text{travel, }>\text{, w+k,in}}}\delta^{\text{w+k}}_{j,i,r}\cdot f_r + \sum_{\mathcal{A}_i^{\tilde{k},\text{in}}} v_{j,i}$$
$$= \sum_{r\in\mathcal{R}}\sum_{(i,j)\in\mathcal{A}_i^{\text{travel},>,\text{w+k,out}}}\delta^{\text{w+k}}_{i,j,r}\cdot f_r + \sum_{\mathcal{A}_i^{\tilde{k},\text{out}}} v_{i,j} \qquad \forall\, i\in\mathcal{V}^{\text{k}} \tag{36}$$

$$\sum_{r\in\mathcal{R}}\delta^{\text{w}}_{i,j,r}\cdot f_r \leq \sum_{r\in\mathcal{R}}\delta^{\text{w+k}}_{i,j,r}\cdot(cap-1)\cdot f_r \qquad \forall\,(i,j)\in\mathcal{A}^{\text{travel},>,\text{w}} \tag{37}$$

$$f_r\in\{0,1\} \qquad \forall\, r\in\mathcal{R} \tag{38}$$

$$v_{i,j}\in[0,K] \qquad \forall\,(i,j)\in\mathcal{A}^{\tilde{k}} \tag{39}$$

$$y_{h,m,t}\in\{0,1\} \qquad \forall\, h\in\mathcal{J}^{n+1}, m\in\mathcal{M}_h, t\in\mathcal{T}_h^{\text{S}} \tag{40}$$

The objective function (30) corresponds to that of MILP II (18) as it includes only extended worker routes.

Constraints (31) - (33) correspond to constraints (19) - (21) in the MILP II. Constraints (34) and (35) limit the number of workers and vehicles in the problem. The vehicle flow constraint (25) from MILP II is modelled to constraint (36). Constraint (37) corresponds to constraint (26) in MILP II and constraints (38) to (40) define the decision variables.

## 6.2 Pricing problem

To generate new columns, we solve a worker subproblem or pricing problem which corresponds to a shortest path problem with resource constraints (SPPRC) on the extended worker network. For a general introduction to SPPRCs, see Irnich and Desaulniers (2005).

The SPPRC aims at finding the route $r$ with the lowest reduced cost, where the reduced cost of a route is defined as the sum of the reduced costs of its arcs. The dual multiplier for constraint (32) is given as $\pi^{32}_{h,m,t}$. Likewise, we set $\pi^{34}$, $\pi^{35}$, $\pi^{36}_i$ and $\pi^{37}_{i,j}$. Table 3 summarizes the reduced cost per arc type.

Table 3: Reduced cost per arc type

| **Arc** | **Reduced cost** |
|---|---|
| $(i(h,t,arr),j(h,t+p_{h,m},dep))\in\mathcal{A}^{\text{proc}}$ | $\beta\cdot p_{i,j}$ - $\pi^{32}_{h,m,t}$ |
| $(i,j)\in\mathcal{A}^{\text{travel},>,\text{ w+v}}$ | $\beta\cdot 2\cdot d_{i,j} + \pi^{36}_i - \pi^{36}_j + \pi^{37}_{i,j}$ |
| $(i,j)\in\mathcal{A}^{\text{travel},>,\text{ w}}$ | $\beta\cdot d_{i,j}$ - $\pi^{37}_{i,j}$ |

## 6.3 Subproblem labelling algorithm

We solve the pricing problem with a dynamic programming labelling algorithm. Such an algorithm iteratively builds new paths starting from a source to a sink. All relevant information of a path is stored in a label (see Section 6.3.1) for each partial path. The whole algorithm is described in Section 6.3.3.

### 6.3.1 Label definition

A labelling algorithm generates labels for each partial or complete path. Each label $l_i$ for a vertex $i$ corresponds to a tuple with five elements: (i) the current vertex, (ii) the parent

label id, (iii) the cumulative reduced cost, (iv) the cumulative travel cost of the partial or full path, and (v) the last time a transition arc was used on the path (called "last idle time"). The reason for storing the last idle time in a label is to avoid cycling. The extended worker network in Figure 3 would in theory allow for starting a job several times given large enough time windows. As it will be stated in Section 7.1, our instances do not permit starting a job twice for one route due to tight time windows. However, it is possible that a cycle of zero length occurs because workers may use a transition arc in combination with a zero travel arc enabling a return to the same vertex at zero cost. To avoid this, we save the last idle time and include it as a dominance criterion in our dominance rule.

#### 6.3.2 Dominance rule

We store all non-dominated labels of one vertex $i$ in a container of labels called bucket $F_i$. A label $l \in \mathcal{F}_i$ dominates a label $l' \in \mathcal{F}_i$, $l \neq l'$, if the reduced cost of label $l$ is equal or less than that of $l'$ and if the last idle time of $l$ is equal or less than that of $l'$, with at least one of the two inequalities being strict. When they are both satisfied at equality, one of the two labels must be kept, the other can be discarded. To test the dominance of a new label for vertex $i$, we compare it to each label already stored in $F_i$.

#### 6.3.3 Labelling algorithm

Having outlined both the label definition and the dominance rule, we describe the label correcting algorithm in the following. The overall procedure is introduced in Algorithm 1 and the label extension in Algorithm 2 being used to search for new labels.

In the initialization phase of Algorithm 1 (see Steps 1 - 5), method `order`($\mathcal{V}$) sorts the set of vertices $\mathcal{V}$ by ascending time, i.e. $i(h_i, t_i, type_i) < j(h_j, t_j, type_j)$ iff $t_i < t_j$. If two vertices have the same time, we use two tie breakers: first, arrival vertices have precedence over departure vertices and second, the vertex with the smaller job id comes first. In Step 2 the start label $l_0$ and the bucket $\mathcal{F}_0$ for start vertex 0 is set. Step 3 creates an empty list of vertices with the same associated time that we explain in more detail below. The initialization finally sets vertex 0 as the previously visited vertex *prev* in Step 5.

In the main loop of Algorithm 1 (see Steps 6 - 15), we iterate through each vertex $i \in \mathcal{V}$ and distinguish between two cases. First, if vertex $i$ does not have the same time as vertex *prev*, we update *prev* with $i$ (Step 13) and create new labels for all vertices $j \in \mathcal{V}$ adjacent to $i$ (see Step 7) using the label extension method `extendVertex`($i$) (see Algorithm 2, explained in the paragraph below). Second, if vertex $i$ has the same time as *prev*, then all vertices in $\mathcal{V}^{\text{sameTime}}$ are examined again. List $\mathcal{V}^{\text{sameTime}}$ contains only vertices that have already been examined but where a repeated check is necessary to further reduce the solution space as some arcs $(i, j)$ have a time consumption of zero and the vertices are examined in the order of their associated times. For example let vertex $v_1$ be examined before vertex $v_2$ where both vertices are associated with the same time and both are connected by an arc. Then it may be possible that a label $l_2$ in $v_2$ can be feasibly extended to vertex $v_1$ to create a label $l_1$ which is not dominated by the other labels in $v_1$. In this case, label $l_1$ also needs to be extended along the arcs going out of $v_1$. Note that the extension along an arc with a zero time consumption does not change the reduced cost nor the last idle time. Therefore, such extensions cannot be performed indefinitely because of the dominance rule stated in Section 6.3.1. Once all vertices are examined, the shortest paths are constructed recursively in Step 16.

Algorithm 2 describes the procedure `extendVertex`($i$) in detail. For a given vertex $i$, we

iterate over all vertices $j \in \mathcal{V}$ adjacent to $i$, $i \neq j$. A new label $l_j$ is generated and the dominance rule is checked in Step 4: we add $l_j$ to bucket $F_j$ iff $l_j$ is not dominated by any other label contained in $F_j$. We also remove all labels from $F_j$ that are dominated by the generated label $l_j$. If $l_j$ is added to the bucket $F_j$ and if the vertex $j$ is associated with the same time as vertex $i$, we add vertex $j$ to the list of vertices we need to examine again $\mathcal{V}^{\text{sameTime}}$ (see Steps 5 and 6).

**Algorithm 1** Labelling correcting algorithm

1: `order`$(\mathcal{V})$ //*order vertices* $\mathcal{V}$ *by ascending time*
2: $l_0 \leftarrow (0,0,0,0,0)$ //*initialize label* $l_0$
3: $F_0 \leftarrow l_0$ //*initialize* $F_0$ *with* $l_0$
4: $\mathcal{V}^{\text{sameTime}} \leftarrow \emptyset$ //*initialize set of same time vertices as empty*
5: $prev \leftarrow 0$ // Initialize the previously extended vertex with vertex 0
6: **for** $i \in \mathcal{V}$ **do**
7:  **if** ! ( `sameTime`$(prev, i)$ ) **then**
8:  **while** $j \in \mathcal{V}^{\text{sameTime}} \neq \emptyset$ **do**
9:  `removeVertex`$(j)$ //*Remove* $j$ *from the set of vertices with the same time*
10:  `extendVertex`$(j)$ // *Extend labels by searching from vertex* $j$*, see Algorithm 2*
11:  **end while**
12:  **end if**
13:  $prev \leftarrow i$ // Mark the previous visited vertex by $i$
14:  `extendVertex`$(i)$ // *Extend labels by searching from vertex* $i$*, see Algorithm 2*
15: **end for**
16: `findShortestPaths`$(\mathcal{F})$ //*use set of buckets* $\mathcal{F}$ *to recursively find shortest paths*

**Algorithm 2** extendVertex$(i)$ procedure

1: **for** $l_i \in F_i$ **do**
2:  **for** $j \in SUCCESSORS_i$ **do**
3:  $l_j \leftarrow (j, l_i, reducedCost, travelCost, lastTimeIdle)$ // *Create label* $l_j$ *for vertex* $j$
4:  `checkDominance`$(l_j, F_j)$ //*Check dominance and add* $l_j$ *to* $F_j$ *iff* $l_j$ *is not dominated by any other label*
5:  **if** $l_j \in F_j$ & `sameTime`$(prev, i)$ **then**
6:  $\mathcal{V}^{sameTime} \leftarrow j$
7:  **end if**
8:  **end for**
9: **end for**

## 6.4 Acceleration Strategies

We apply several acceleration strategies to reduce the runtime of the CG both in the SP and the RMP.

### 6.4.1 Stabilization of dual vector

We use dual variable stabilization as introduced by Wentges (1997) to reduce the well-known oscillation of the dual variables passed from the master to the subproblem. It combines the latest dual vector $\pi^{\text{last}}$ with the so far best dual vector $\pi^{\text{best}}$ so that the duals employed in the

current iteration are $\pi^{\text{now}} = \theta \cdot \pi^{\text{best}} + (1 - \theta) \cdot \pi^{\text{last}}$ where $\theta$ denotes the weight assigned to the best dual vector.

#### 6.4.2 Dynamic constraints

In the RMP, the capacity constraint (37) is constructed for each worker and vehicle travel arc with a positive distance resulting in a very large number of constraints. However, only few travel arcs are likely to be used and therefore only few capacity constraints are binding. Thus, we use dynamic constraints as proposed by Desaulniers, Desrosiers, and Solomon (2002). We start with an RMP excluding constraints (37) and add only those constraints to the RMP that are needed when appending new columns to the RMP.

#### 6.4.3 Adding multiple columns

To speed up column generation, adding multiple columns per iteration can help. However, one needs to choose the type and number of columns wisely, as many non-useful columns in the RMP may increase the runtime of the procedure both because more columns mean more variables in the RMP and by leading the duals astray. Therefore, we add several task-disjoint columns to the RMP in each iteration (see Desaulniers, Desrosiers, and Solomon (2002)). Two columns are task-disjoint if they cover different jobs. For example, let column $r_1$ process the jobs $1, 2, 4$, column $r_2$ process the jobs $3, 5, 6$, and column $r_3$ process the jobs $1, 3, 5$. Then columns $r_1$ and $r_2$ are disjoint, but $r_3$ is not disjoint from $r_1$ and $r_2$. We add to the RMP at most $m$ sets of task-disjoint columns.

### 6.5 Integer solutions

To obtain integer solutions, we propose a branch-and-price heuristic (BAPH) that combines branching on the number of vehicles and on jobs with fixing of routes. The novel aspect of the BAPH is that we do not fix single columns, which has often led to successful heuristic procedures (see for instance Joncour et al. (2010), Frey, Kolisch, and Artigues (2016)), but rather a sum of specially modified routes.

Figure 4 shows an overview of the BAPH. Until there is no unexplored node any more or until a given time limit is reached, the BAPH procedure iteratively runs column generation (CG) that is adjusted based on the currently selected node. In a node, we store information of a (partial) solution such as the lower and upper bounds on the number of vehicles, forbidden processing arcs or fixed sums of routes. We employ a depth-first search. During the course of the BAPH, we first aim at an integral number of vehicles and integral job variables, which we achieve through branching (see Section 6.5.1) and which corresponds rather to a traditional branch-and-price approach. When the variables for the number of vehicles and for jobs are integral, we have the option to either branch on travel arcs which would result in a fully-fledged branch-and-price procedure or to fix columns. We found that fixing sums of specially modified routes (see Section 6.5.2) and then handing the solution to a MIP solver (see Section 6.5.3) and thus solving the problem heuristically yields good results at a comparability low runtime. Naturally, extending the BAPH to an exact branch-and-price procedure is straightforward as only branching on travel arcs is required. In the following, we describe the three major components of the BAPH in more detail.

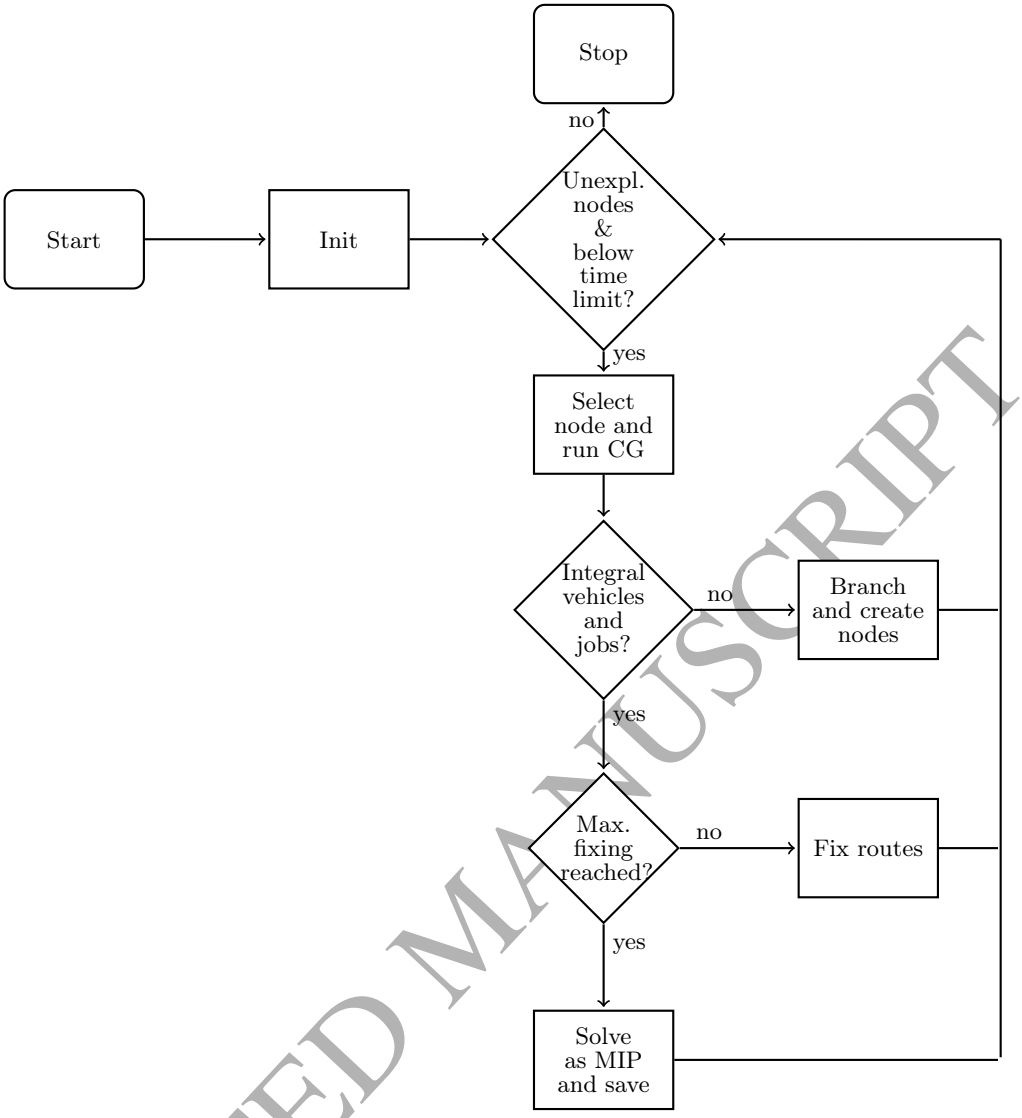


Figure 4: BAPH illustration

### 6.5.1 Branching

We employ a branching hierarchy that first ensures an integral number of vehicles and second integral job variables. The number of employed vehicles $k$ is an auxiliary variable not explicitly shown in the RMP, it is defined as $v = \sum_{r \in \mathcal{R}} \sum_{(i,j) \in \mathcal{A}^{\text{travel,w+k, start}}} \delta^{\text{w+k}}_{i,j,r} \cdot f_r$. Let $\tilde{v}$ equal the number of vehicles and $\tilde{y}_{h,m,t}$ the value of the variable for job $h$ at mode $m$ starting in time $t$ in the current RMP solution. When $\tilde{v}$ is fractional, we branch on this variable by imposing $\sum_{r \in \mathcal{R}} \sum_{(i,j) \in \mathcal{A}^{\text{travel, w+k, start}}} \delta^{\text{w+k}}_{i,j,r} \cdot f_r \leq \lfloor \tilde{k} \rfloor$ on one branch and $\sum_{r \in \mathcal{R}} \sum_{(i,j) \in \mathcal{A}^{\text{travel,w+k,start}}} \delta^{w+v}_{i,j,r} \cdot f_r \geq \lceil \tilde{k} \rceil$ on the other. We enforce the lower or the upper bound by adding an additional constraint to the RMP. When $\tilde{y}_{h,m,t}$ is fractional, we impose $\sum_{n \in \mathcal{M}_h : n \leq m} \sum_{u \in \mathcal{T}^{\text{S}}_h : u \leq t} y_{h,m,t} = 0$ in one branch and $\sum_{n \in \mathcal{M}_h : n > m} \sum_{u \in \mathcal{T}^{\text{S}}_h : u > t} y_{h,m,t} = 0$ in the other. We enforce the branching by removing all columns violating the latter equations from the RMP. In addition, we eliminate the respective processing arcs from the subproblem. The job variable selected for branching is the one with a value closest to 0.5 in the current RMP solution.

### 6.5.2 Fixing of Routes

When both the number of vehicles and the number of jobs are integral, we start to iteratively fix routes using the following three-step process: First, we select a route using some selection strategy. Second, we create all modifications of the selected route. Third, we set the sum of all modified routes plus the selected route greater or equal to one in the RMP. We now illustrate each step in more detail: i) We select the route with a variable value closest to 0.5. ii) Given a selected route $\bar{r}$, we create its set of modified routes $\mathcal{R}^{\bar{r}}$. $\mathcal{R}^{\bar{r}}$ not only includes $\bar{r}$ itself but also all possible route modifications of $\bar{r}$. A route modification is defined as creating a copy of $\bar{r}$ denoted $\bar{r}^{'}$, selecting a travel arc of $\bar{r}^{'}$ and replacing the travel arc with its parallel travel arc. I.e. if the selected travel arc $(i,j)$ is a worker travel arc $(i,j) \in \mathcal{A}^{\text{travel,w}}$, then we replace $(i,j)$ with its parallel worker and vehicle travel arc $(i,j) \in \mathcal{A}^{\text{travel,w+k}}$. Creating modifications of the selected tour $\bar{r}$ has the advantage that for each worker travel arc, there exists at least one route with a parallel worker and vehicle travel arc. In other words, creating modified routes ensures movement synchronization. iii) Finally, we add a new constraint to the RMP such that $\sum_{r \in \mathcal{R}^{\bar{r}}} f^r \geq 1$. We stop fixing when all routes in the RMP that are part of the basis belong to a set of fixed routes.

### 6.5.3 Solve as MIP

Given an integral number of vehicles and jobs as well as a maximum number of fixed sums of routes, we solve the RMP with all its columns as a MIP using CPLEX and save the solution in the pool of integral solutions. When the procedure terminates, the best feasible integral solution of all generated integral solutions is returned.

## 7 Computational study

In this section, we report results of our computational experiments. We investigate the performance of the two MILPs in terms of runtime, integrality gap and number of branch-and-bound nodes. Then, we compare the BAPH with the better performing MILP II solving it both for 1% and a zero optimality gap. Finally, when testing the BAPH, we examine the various implemented acceleration strategies.

All experiments are run on a Windows 7 platform with a 3.4 GHz CPU and 12 GB RAM and on a single thread. The MILP as well as the LP of the BAPH are solved with the off-the-shelf solver IBM CPLEX 12.6. The BAPH and the MILPs are implemented in the programming language Java.

### 7.1 Data instances

During the whole computational study, we employ data from Munich International Airport from 2012 for a regular day of operation entailing 745 aircraft that arrive and leave during the course of the day. It is the same dataset as used in Fink et al. (2016). The data comprises more than 100 parking positions for aircraft, 80 vehicles with a capacity of 5 workers and a workforce of 100 workers working in parallel during peak hours. Depending on the aircraft type, the number of workers per job ranges between 2 and 5 and the processing time of the job between 15 and 35 minutes.

Instances are generated by randomly selecting a point in time and then choosing a given number of aircraft in a given time interval, e.g. selecting 10 aircraft in a time interval between

8 and 9 a.m. We associate each airplane with two jobs, namely unloading and loading baggage because the given workforce data is tailored to these two jobs. Other jobs could easily be added to the model. The job's time windows are relative to the airplane's on- and off-block times and range in between 6 to 10 minutes depending on the aircraft size. This reflects the tight operations and the commonly short aircraft turnaround times at large airports. Each instance set contains 8 individual instances. Table 4 shows the instance sets with the number of workers and vehicles. It should be noted that we solve instances with up to 32 jobs due to runtime limitations and while not removing any arcs from the network. Larger problem instances can be solved with the BAPH when removing arcs at the cost of potentially achieving worse solutions. However, this is not part of this paper but can be a subject of future research.

Table 4: Overview of the instance sets

| **Instance set** | **Jobs** | **Workers** | **Vehicles** |
|---|---|---|---|
| 1 | 4 | 3 | 2 |
| 2 | 6 | 4 | 3 |
| 3 | 8 | 6 | 4 |
| 4 | 10 | 8 | 5 |
| 5 | 12 | 10 | 6 |
| 6 | 16 | 12 | 8 |
| 7 | 20 | 14 | 10 |
| 8 | 24 | 16 | 12 |
| 9 | 28 | 18 | 14 |
| 10 | 32 | 20 | 16 |

### 7.2 Algorithm settings

We set the objective function weights $\alpha = 0.8$, $\beta = 0.1$, $\gamma = 0.1$ as we foremost want to minimize the job delay. The delay weight per job is set according to the associated aircraft type. For small aircraft such as an Airbus A320 we set $\sigma_h = 1.0$, for medium aircraft such as an Airbus A330 we set $\sigma_h = 1.1$, for a large aircraft such as a Boeing $B777$ we set $\sigma_h = 1.2$ and for very large aircraft such as an Airbus A380 we set $\sigma_h = 1.3$. The runtime limit of the MILP models and of the BAPH is $3,600$ seconds. Moreover, we aim for a maximum of three integral solutions in the BAPH because during pretests we found that after three feasible integral solutions, the BAPH did not achieve significantly better solutions. Therefore, when we report the BAPH overall runtime, it is either determined by the time when the BAPH finds the third feasible integral solution, by the runtime limit or by the procedure stopping because there exists no unexplored node any more. We set the weight for the dual stabilization as $\theta = 0.5$.

### 7.3 MILP testing

In the following MILP testing we compare both flow formulations. Table 5 shows for both flow formulations MILP I and MILP II the average runtime in seconds to solve the MIP (Time) as well as the average number of branch-and-bound nodes used by CPLEX (No. nodes). Moreover, we present the share of instances for which the MILP II was faster (II faster) and the average integrality gap (INT) of MILP II which is the same as that of MILP I if sufficient runtime is allowed to find the optimal solution.

The MILP II shows lower average runtimes than the MILP I for all instance sets except 4 jobs and is overall faster for 71.88% of all instances. The integrality gap is on average 2.30%

Table 5: Results for MILP comparison

| **Jobs** | **MILP I** | | **MILP II** | | **Comparison** | **INT** |
|---|---|---|---|---|---|---|
| | Time(s) | No. nodes | Time(s) | No. nodes | II faster(%) | GAP(%) |
| 4 | 1.07 | 6.63 | 1.20 | 5.25 | 37.50 | 2.83 |
| 6 | 20.19 | 135.25 | 11.20 | 156.00 | 100.00 | 2.56 |
| 8 | 231.16 | 527.25 | 88.65 | 385.63 | 75.00 | 1.78 |
| 10 | 2,720.05 | 1,126.75 | 1,499.74 | 1,950.63 | 75.00 | 2.40 |
| *Average* | 743.12 | 448.97 | 400.20 | 624.38 | 71.88 | 2.30 |

which means that in particular for the larger 10 jobs instances, closing the gap via branch-and-bound requires a growing number of branch-and-bound nodes. Note that the MILP I reaches the runtime limit for 5 of all 8 instances with 10 jobs, the MILP II only for 2.

## 7.4 MILP and BAPH comparison

In this section, we compare the MILP II with the BAPH. Table 6 shows runtime (Time) and solution value (Obj) of the MILP II given a zero optimality gap (0% GAP), a 1% optimality gap (1% GAP), and the BAPH. In addition, we show the solution value difference of the BAPH compared to the MILP II 0% (vs 0%) and the 1% (vs 1%). We indicate instance sets for which we could not find at least one optimal solution with a "-".

Table 6: Results for BAPH vs MILP II testing

| **Jobs** | 0% **GAP** | | 1% **GAP** | | **BAPH** | | **Difference** | |
|---|---|---|---|---|---|---|---|---|
| | Time(s) | Obj | Time(s) | Obj | Time(s) | Obj | vs 0%(%) | vs 1%(%) |
| 4 | 1.20 | 71.81 | 1.16 | 71.87 | 2.07 | 71.96 | 0.21 | 0.12 |
| 6 | 11.20 | 115.51 | 9.10 | 115.79 | 3.94 | 115.65 | 0.12 | -0.12 |
| 8 | 88.65 | 165.56 | 60.45 | 165.88 | 9.66 | 165.81 | 0.15 | -0.04 |
| 10 | 1,499.74 | 201.15 | 1,149.58 | 201.69 | 83.05 | 203.59 | 1.20 | 0.93 |
| 12 | – | – | 2,968.89 | 242.87 | 623.66 | 241.75 | – | -0.47 |

We can observe that the solution value of the MILP II 1% is slightly worse than that of the MILP II 0%, but that the runtime of the MILP II 1% is between 3% and 32% lower. Contrasting the BAPH and the MILP II, we can remark that the BAPH runs only for a fraction of the time of the MILP II (both the optimal and 1% version) except for the smallest instances with 4 jobs while achieving an average solution value that is very close to the optimal value: the average difference between the MILP II at 0% and the BAPH ranges between 0.12% and 1.20%. Comparing the MILP at 1% and the BAPH we can see that the BAPH finds better solutions for the instance sets with 6, 8 and 12 jobs on average, and for 58% of all tested instances. Note that the MILP II 0% again reaches the runtime limit for 2 10-job instances and the MILP II 1% does likewise for 6 12-job instances and for 2 10-job instances.

## 7.5 BAPH testing

The goal of the BAPH testing is to demonstrate that the BAPH finds good solutions at a comparably short runtime also for larger problem instances and that the implemented acceleration strategies are efficient. In addition, we provide test results on instances that allow for cycling as well as results on instances with a longer time horizon.

### 7.5.1 BAPH testing: runtime and solution quality

In Table 7, we present the LP objective function value (LP obj) obtained by running the CG without early termination. For the BAPH, we report the first found integral solution's solution value (Int obj f.) and runtime (Time f.), as well as the best integral solution value (Int obj) and runtime (Time). Moreover, we report the number of feasible integral solutions found by the BAPH (Sol) and the integrality gap (GAP) between the LP and the integral solution value.

Table 7: Results for BAPH testing: runtime and solution quality

| **Jobs** | **LP obj** | **Time f.(s)** | **Int obj f.** | **Time(s)** | **Int obj** | **Sol** | **GAP(%)** |
|---|---|---|---|---|---|---|---|
| 4 | 69.78 | 0.64 | 72.02 | 2.07 | 71.96 | 2.38 | 3.03 |
| 6 | 112.55 | 3.05 | 115.76 | 3.94 | 115.65 | 3.00 | 2.67 |
| 8 | 162.61 | 5.83 | 165.88 | 9.66 | 165.81 | 3.00 | 1.92 |
| 10 | 196.33 | 52.76 | 203.70 | 83.05 | 203.59 | 3.00 | 3.57 |
| 12 | 234.35 | 52.82 | 241.85 | 623.66 | 241.75 | 3.00 | 3.06 |
| 16 | 299.92 | 101.20 | 310.56 | 280.97 | 310.07 | 3.00 | 3.27 |
| 20 | 372.28 | 436.89 | 387.58 | 806.69 | 387.19 | 3.00 | 3.85 |
| 24 | 439.61 | 928.11 | 460.14 | 2,037.29 | 460.14 | 2.75 | 4.46 |
| 28 | 506.69 | 1,664.94 | 529.24 | 3,268.64 | 528.77 | 2.13 | 4.17 |
| 32 | 580.48 | 2,139.50 | 605.09 | 3,600.00 | 605.09 | 1.25 | 4.07 |

We can see that the runtime of the BAPH increases with the problem size until it reaches the runtime limit of 3,600 seconds for each 32-job instance. Moreover, for the largest instance sets with 24, 28 and 32 jobs, the average number of found solutions decreases due to the runtime limit. For all other instance sets except for 4 jobs, the procedure stops at three found integral solutions. Also note that when comparing Int obj with Int obj f., we can observe that the average solution value is better than the value of the first found solution, which indicates that continuing the search for an integral solution after having found a first solution makes sense. However, the improvement in solution value comes at the cost of a large runtime increase: the runtime until three solutions are found is often more than double that of the time to find the first feasible solution. Therefore, if one wants to find good solutions quickly, running the BAPH until the first solution is a viable option. The gap between the lowest found LP solution and the best MIP solution stays in an average range of 1.92% to 4.46%. Given that the average integrality gap in Table 5 equals 2.30%, we can conclude that the BAPH finds solutions with values close to the optimum also for larger instances.

### 7.5.2 BAPH testing: acceleration strategies

Table 8 shows runtime (Time) and solution value (Obj) of the best found integral solution by the BAPH and distinguishes between several configurations: the original configuration (Original), the original configuration without stabilization (W/O stab, see Section 6.4.1), the original configuration without dynamic constraints (W/O dynamic, see Section 6.4.2) and the original configuration without adding several disjoint columns (W/O disjCols, see Section 6.4.3). In the latter case, we instead add in each iteration the 10 columns with the lowest reduced cost. Note that for the largest three instance sets with 24, 28 and 32 jobs, the comparison of configurations is futile because not all instances could be solved by each configuration (indicated by a "-"). Instead, Table 9 presents the share of solved instances by the various configurations for the largest three instance sets.

Overall, Table 8 demonstrates that the original configuration performs best in terms of both average runtime and solution value for instances up to 20 jobs where all four configurations can solve all instances. Furthermore, the configuration without stabilization finds only slightly

Table 8: Results for BAPH testing: acceleration strategies

| **Jobs** | **Original** | | **W/O stab** | | **W/O dynamic** | | **W/O disjCol** | |
|---|---|---|---|---|---|---|---|---|
| | Time(s) | Obj | Time(s) | Obj | Time(s) | Obj | Time(s) | Obj |
| 4 | 2.07 | 71.96 | 0.93 | 71.84 | 1.09 | 71.84 | 1.94 | 72.37 |
| 6 | 3.94 | 115.65 | 3.02 | 115.80 | 2.59 | 116.02 | 2.67 | 116.32 |
| 8 | 9.66 | 165.81 | 17.04 | 165.63 | 8.73 | 167.43 | 17.04 | 169.01 |
| 10 | 83.05 | 203.59 | 66.63 | 205.81 | 62.76 | 208.61 | 235.83 | 204.85 |
| 12 | 623.66 | 241.75 | 574.79 | 247.96 | 509.85 | 249.01 | 978.24 | 252.02 |
| 16 | 280.97 | 310.07 | 182.79 | 312.88 | 536.01 | 312.51 | 464.32 | 313.90 |
| 20 | 806.69 | 387.19 | 1,469.86 | 389.62 | 1,784.99 | 389.40 | 2,523.01 | 397.66 |
| 24 | 2,037.29 | 460.14 | 2,563.15 | 459.75 | – | – | – | – |
| 28 | 3,268.64 | 528.77 | – | – | – | – | – | – |
| 32 | 3,600.00 | 605.09 | – | – | – | – | – | – |
| *Average* 4-20 | 258.58 | 213.72 | 330.72 | 215.65 | 415.14 | 216.40 | 603.29 | 218.02 |

worse results in slightly greater runtime and the configuration without adding several disjoint columns performs worst. Considering the solution value for single instance sets, the original configuration performs best for $6, 10, 12, 16$ and 20-job instances and finds the solution with the best value for 54% of all tested instances up to 20 jobs. In the remaining instance sets, it is outperformed only by a small margin by the configuration without stabilization. Moreover, the two configurations without dynamic constraints and without adding several disjoint columns perform worst. Looking at the runtime for single instance sets, the original configuration is fastest for the instance sets with 20 and 24 jobs. For small and medium-sized instances except for 8-job instances, the configuration without stabilization and partly the configuration without dynamic constraints find solutions faster.

Table 9 shows that for 24 jobs, the original and the configuration without stabilization can find integral solutions for all instances, and that the two configurations can still solve 50% of all 32-job instances. The other two configurations drop to 62.50% for 24 jobs and cannot find a single feasible integral solution for any 32-job instance.

Table 9: Results for BAPH testing: share of largest instances solved

| **Jobs** | **Share of instances with feasible solution in runtime limit (%)** | | | |
|---|---|---|---|---|
| | Original | W/O stab | W/O dynamic | W/O disjCols |
| 24 | 100.00 | 100.00 | 62.50 | 62.50 |
| 28 | 100.00 | 62.50 | 0.00 | 37.50 |
| 32 | 50.00 | 50.00 | 0.00 | 0.00 |

### 7.5.3 BAPH testing: non-elementary paths

We test the BAPH on instances from 4 to 20 jobs where we widen the instances' job time windows by 25%, 50% and 75% to allow for cycling. In case of no changes to the time windows (0% TW incr.), we use the original BAPH. In case of instances with widened time windows, we adapt the labeling algorithm in the column generation algorithm to be able to cope with cycling using the approach presented in Feillet et al. (2004). More precisely, an additional component for each job is added to each label to indicate if the corresponding job can be realized in a feasible extension of this label. In Table 10, we report for the respective time window increases (TW incr.) both the runtime of the algorithm (Time) in seconds as well as the gap of the integral solution compared to the LP solution (GAP) in percent. In the last column, we report for how many of the 8 tested instances per job size the BAPH found a solution for each of the three time window adjustments (IwS). Thus, the reported averages are

computed over the instances that could be solved for all settings.

We can observe that on average, the runtime for solving instances with the largest tested time windows (75%) increases roughly by factor 7 and the gap approximately doubles. This clearly shows that instances with larger time windows are harder to solve. Interestingly, 25% wider time windows already increase the runtime by factor 5 and almost double the gap. Widening the time windows even further has less of an impact on runtime and gap. We can also see that increasing the instance size makes it harder for the BAPH to find solutions in the given runtime of 1 hour.

Table 10: Results for additional BAPH testing: runtime and solution quality given larger time windows

| **Jobs** | **0% TW incr.** | | **25% TW incr.** | | **50% TW incr.** | | **75% TW incr.** | | **IwS** |
|---|---|---|---|---|---|---|---|---|---|
| | Time(s) | GAP(%) | Time(s) | GAP(%) | Time(s) | GAP(%) | Time(s) | GAP(%) | |
| 4 | 1.83 | 3.08 | 2.40 | 3.89 | 3.70 | 3.54 | 2.48 | 8.68 | 7 |
| 6 | 3.94 | 2.64 | 468.90 | 7.17 | 40.63 | 5.59 | 33.81 | 6.14 | 8 |
| 8 | 9.66 | 1.92 | 25.74 | 4.34 | 81.35 | 4.53 | 94.27 | 5.76 | 8 |
| 10 | 101.14 | 3.51 | 591.72 | 5.57 | 362.83 | 8.75 | 1,351.31 | 5.17 | 6 |
| 12 | 165.18 | 3.38 | 1,427.74 | 9.31 | 2,271.78 | 6.74 | 1,415.58 | 11.84 | 4 |
| 16 | 303.56 | 3.57 | 950.42 | 5.50 | 2,441.75 | 7.70 | 1,548.90 | 6.27 | 7 |
| 20 | 1,051.91 | 1.75 | 3,408.23 | 2.34 | 3,600.00 | 5.07 | 3,600.00 | 2.50 | 1 |
| Average | 111.03 | 2.91 | 568.20 | 5.63 | 804.43 | 5.96 | 717.07 | 6.85 | 5.86 |

### 7.5.4 BAPH testing: longer time horizon

We conduct additional experiments for 20-job instances with longer time horizons to allow for longer routes. The results of these experiments are reported in Table 11. We created 5 different instance sets (Job set), each consisting of 8 instances with different number of workers (from 14 to 9), vehicles (from 10 to 5) and a different time horizon (Hor., from 1 to 3 hours). Again, we report the runtime (Time), gap (GAP) and the number of instances for which a solution was found within the time limit (IwS). Also, the reported averages are computed over the instances that could be solved.

Table 11: Results for additional BAPH testing: runtime and solution quality for longer time horizons and changing numbers of workers and vehicles

| **Job set** | **Workers** | **Vehicles** | **Hor.** | **Time(s)** | **GAP(%)** | **IwS** |
|---|---|---|---|---|---|---|
| $20-1$ | 14 | 10 | 1.0 | 806.69 | 3.85 | 8 |
| $20-2$ | 12 | 8 | 1.5 | 2.749.96 | 2.52 | 8 |
| $20-3$ | 11 | 7 | 2.0 | 3.002.91 | 3.02 | 6 |
| $20-4$ | 10 | 6 | 2.5 | 1.598.92 | 1.82 | 3 |
| $20-5$ | 9 | 5 | 3.0 | 3,600.00 | 4.46 | 1 |

Increasing the time horizon and decreasing the number of vehicles and workers makes it harder for the BAPH to find solutions: the runtime increases and we are able to solve less instances overall. However, the BAPH achieves even better gaps on average, except for job set 5, which seems particularly hard to solve as only one instance can be solved in the runtime limit.

# 8 Conclusion

In this paper, we investigate the AVRPWVS as an example and archetype of the class of VRPMSs. The AVRPWVS deals with routing workers to jobs at different locations, determining jobs' start times and modes while synchronizing the worker with vehicle routes. This work focuses on one application of airport ground handling, but the AVRPWVS can also be applied to other industries such as logistics or healthcare.

We present two multi-commodity flow formulations based on a time-space network, the MILP I and MILP II. Both MILPs achieve the same integrality gap of 2.30% on average, but the MILP II performs better in terms of runtime. In addition, we develop a branch-and-price heuristic (BAPH) that, besides conventional variable branching, features a novel concept of fixing sums of modified routes. We demonstrate that the BAPH finds solutions close to the optimal solution at a comparably short runtime.

There are several possibilities for future research. First, one could investigate an extension of the BAPH to an optimal branch-and-price procedure by branching on travel arcs. Furthermore, the novel fixing approach could be applied to other VRPMSs with movement synchronization and one could apply the BAPH to the original VRPWVS by incorporating several subproblems for workers and vehicles. Moreover, an investigation of runtime and solution value of the BAPH when removing arcs could be a possible path for future research.

# References


Ahuja RK, Magnanti TL, Orlin JB, 1993 *Network Flows: Theory, Algorithms, and Applications* (Prentice Hall), 1 edition.

Balakrishnan H, Chandran B, 2010 *Algorithm for scheduling runway operations under constrained position shifting. Operations Research* 58(6):1650–1665.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH, 1998 *Branch-and-Price: Column Generation for Solving Huge Integer Programs. Operations Research* 46(3):316–329.

Christofides N, Alvarez-Valdés R, Tamarit JM, 1987 *Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research* 29(3):262–273.

Desaulniers G, 2010 *Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. Operations Research* 58(1):179–192.

Desaulniers G, Desrosiers J, Solomon MM, 2002 *Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems.* Ribeiro CC, Hansen P, eds., *Essays and Surveys in Metaheuristics*, volume 15, 309–324 (Boston, MA: Springer).

Desrosiers J, Lübbecke ME, 2005 *A primer in column generation.* Desaulniers G, Desrosiers J, Solomon MM, eds., *Column Generation*, volume 1, 1–32 (New York: Springer).

Drexl M, 2007 *On some generalized routing problems.* Doctoral dissertation, RWTH Aachen University.

Drexl M, 2012 *Synchronization in Vehicle Routing A Survey of VRPs with Multiple Synchronization Constraints. Transportation Science* 46(3):297–316.

Drexl M, 2014 *Branch-and-cut algorithms for the vehicle routing problem with trailers and transshipments. Networks* 63(1):119–133.

Feillet D, Dejax P, Gendreau M, Gueguen C, 2004 *An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks* 44(3):216 – 229.

Fink M, Frey M, Kiermaier F, Kolisch R, 2016 *Synchronized worker and vehicle routing for ground handling at airports*, Working paper, TUM School of Management, Technische Universität München.

Frey M, Kolisch R, Artigues C, 2016 *Column generation for outbound baggage handling at airports*, Working paper, TUM School of Management, Technische Universität München.

Grønhaug R, Christiansen M, Desaulniers G, Desrosiers J, 2010 *A branch-and-price method for a liquefied natural gas inventory routing problem. Transportation Science* 44(3):400–415.

Haase K, Desaulniers G, Desrosiers J, 2001 *Simultaneous vehicle and crew scheduling in urban mass transit systems. Transportation Science* 35(3):286–303.

Hollis B, Forbes M, Douglas B, 2006 *Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post. European Journal of Operational Research* 173(1):133–150.

Irnich S, Desaulniers G, 2005 *Shortest path problems with resource constraints.* Desaulniers G, Desrosiers J, Solomon MM, eds., *Column Generation*, volume 1, 33–65 (New York: Springer).

Joncour C, Michel S, Sadykov R, Sverdlov D, Vanderbeck F, 2010 *Column generation based primal heuristics. Electronic Notes in Discrete Mathematics* 36:695–702.

Lahyani R, Khemakhem M, Semet F, 2015 *Rich vehicle routing problems: From a taxonomy to a definition. European Journal of Operational Research* 241(1):1–14.

Lenstra JK, Kan AHGR, 1981 *Complexity of vehicle routing and scheduling problems. Networks* 11(2):221–227.

Lübbecke ME, Desrosiers J, 2005 *Selected Topics in Column Generation. Operations Research* 53(6):1007–1023.

Meisel F, Kopfer H, 2014 *Synchronized routing of active and passive means of transport. OR Spectrum* 36(2):297–322.

Tilk C, Bianchessi N, Drexl M, Irnich S, Meisel F, 2015 *Branch-and-price for the active-passive vehicle-routing problem*, Working paper, Johannes Gutenberg-Universität Mainz.

Visser TR, 2015 *Synchronization in Simultaneous Vehicle and Crew Routing and Scheduling Problems with Breaks.* Master's thesis, Universiteit Utrecht.

Wentges P, 1997 *Weighted Dantzig–Wolfe Decomposition for Linear Mixed-integer Programming. International Transactions in Operational Research* 4(2):151–162.

Xiang Z, Chu C, Chen H, 2006 *A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. European Journal of Operational Research* 174(2):1117–1139.