A data-driven approach for baggage handling operations at airports

Christian Ruf

TUM School of Management
Technical University of Munich
Arcisstr. 21
D-80333 Munich
christian.ruf@tum.de

Sebastian Schiffels

Lancaster University Management School
Campus in Leipzig
Nikolaistr. 10
D-04109 Leipzig
s.schiffels@lancaster.ac.uk

Rainer Kolisch

TUM School of Management
Technical University of Munich
Arcisstr. 21
D-80333 Munich
rainer.kolisch@tum.de

Markus Frey

TUM School of Management
Technical University of Munich
Arcisstr. 21
D-80333 Munich
markus.frey@tum.de

December
2021

A data-driven approach for baggage handling operations at airports

Abstract

Before each flight departs, baggage has to be loaded into containers, which will then be forwarded to the airplane. Planning the loading process consists of setting the start times for the loading process and depletion of the baggage storage, as well as assigning handling facilities and workers. Flight delays and uncertain arrival times of passengers at the check-in counters require plans that are adjusted dynamically every few minutes, and hence an efficient planning procedure. We propose a model formulation and a solution procedure that utilize historical flight data to generate reliable plans in a rolling planning fashion, allowing problem parameters to be updated in each re-optimization. To increase the tractability of the problem, we employ a column-generation-based heuristic, in which new schedules and work profiles are generated in subproblems, which are solved as dynamic programs. In a computational study we demonstrate the robust performance of the proposed procedure based on real-world data from a major European airport. The results show that (i) the procedure outperforms both a constructive heuristic that mimics human decision-making and a meta heuristic (tabu search), and (ii) being able to dynamically (re-)allocate baggage handlers leads to improved solutions with considerably fewer left bags.

Keywords: Data-driven optimization; Rolling horizon; Airport operations; Baggage handling; Integer programming; Column generation

1 Introduction

Baggage handling (BH) is one of the main services at airports (Boysen et al. 2019). If a bag fails to arrive at its destination on time the customer satisfaction is reduced, and the cost of reuniting mishandled bags with their owners is significant, at an average of US$101 with an industry total in the order of US$2.3 billion in 2017 (SITA 2018). Since existing airport infrastructure can only be expanded in the long-term and at high cost, airport resources need to be used as efficiently as possible, and a major airline hub requires well-functioning BH.

Central to BH is the *baggage handling system* (BHS), an automated conveyor system which transports baggage at airports (see Boysen et al. 2019 for a review of related literature). Arriving flights carry inbound baggage that is forwarded to the baggage claim as well as transfer baggage that is directed to departing flights via the BHS. For departing flights, checked-in baggage and transfer baggage is either transferred to the central storage or directly to one of several baggage sorting stations, which are called *carousels*. A carousel is a circular conveyor belt with a limited number of working stations where a worker loads the baggage into *unit load devices* (ULD). A working station handles one flight at a time and consists of a hand-held scanner to read each bag's tag and a screen to display the corresponding ULD. Once loading is complete, trucks transport the ULDs to the airplane whose departure time determines the end of its BH period.

With several hundred flights per day, BH poses a challenging planning task, and it is important to consider interdependencies, e.g. between inbound processes, the conveyor system, and outbound processes (Boysen et al. 2019). By planning intelligently, we seek to ensure that all bags can be loaded into ULDs in time, that is, before the end of the BH process of the corresponding flight. We do not consider the assignment of inbound flights to baggage claims in the arrival hall, studied for example by Frey, Kiermaier, and Kolisch (2017), which is a loosely related problem as remote infeed stations employ the BHS for transport; however, in most cases, baggage claims are fed by directly connected infeed stations, i.e. the BHS is not involved. Figure 1 provides an overview of the baggage flow (gray arrows) and the related planning decisions that we address: Flights have to be assigned to carousels (1), their loading process and storage depletion have to be scheduled (2), and baggage handlers have to be assigned to load the bags into ULDs (3). If a carousel's or the storage's capacity is used up, bags can no longer be forwarded to the carousel or storage, respectively, and

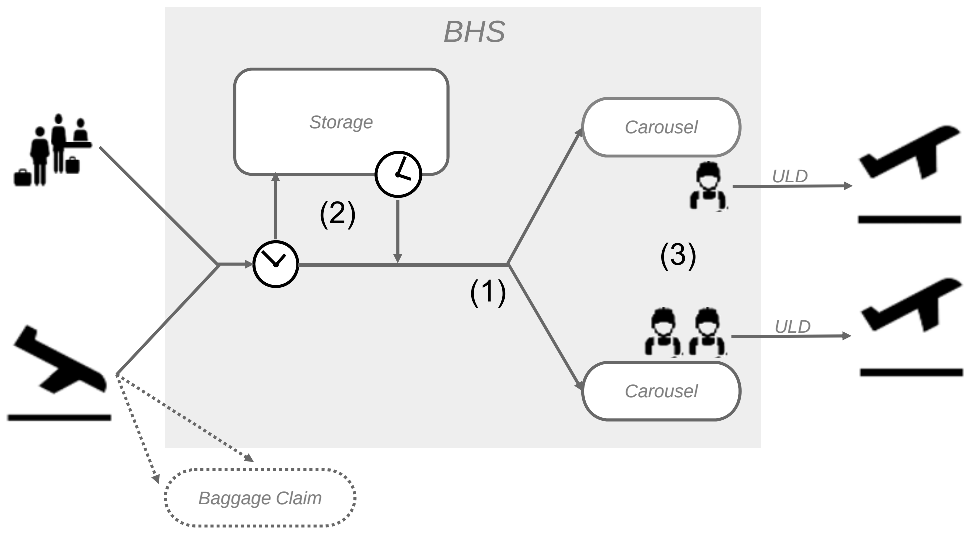


Figure 1: Planning decisions and baggage flow

bags remain within the BHS. Resulting backlogs can hinder the baggage transport for other flights, and have to be avoided. While bags are autonomously transported within the BHS, workers are required for bags to leave the BHS, i.e. to load the bags into ULDs. The loading process is very labor intensive, and requires an efficient assignment of the workforce due to the high cost pressure for ground handling. While the number of ground handling workers assigned to a flight can change during the loading process, workers' shift schedules are fixed ahead of the planning day, thus limited personnel is available on the day of operation.

Complicating things further, BH is carried out in a dynamic environment with uncertainties about future flight and baggage data. For example, in the year 2017, only 80% of all departures and only 79% of arrivals were on time worldwide (Bureau of Transportation Statistics (BTS) 2018). These unexpected changes, as well as unknown arrival times of passengers at the check-in counters, often render carefully conceived plans suboptimal or even infeasible, which can lead to left bags, poor utilization of workers and backlogs of bags in the BHS. Hence, dynamic re-optimization and a robust planning approach are crucial to manage daily operations. The challenge of the planning task is exacerbated by the need to obtain updated solutions within minutes.

To the best of our knowledge, no previous work has been performed on dynamic planning of the baggage handling, though several studies consider static planning of the baggage handling for deterministic flight schedules, which provides valuable information ahead of the operations. However, unless the planning does not introduce some form of robustness, the resulting plans are likely to become infeasible on the day of operations, e.g. due to flight delays or higher than expected quantities of check-in bags.

Abdelghany, Abdelghany, and Narasimhan (2006) propose a greedy based sequential allocation heuristic for assigning prescheduled flights to baggage sorting stations, decision (1). Ascó, Atkin, and Burke (2012) and Ascó, Atkin, and Burke (2014) extend the previous work and investigate several algorithms aiming for objectives, such as minimizing the distance between the assigned baggage sorting station and the aircraft's parking position (called the stand); fairly distributing the workload across sorting stations; and maximizing the buffer times between two consecutive flights on the same baggage sorting station. The latter objective, also considered by Ascó (2016), is an approach to account for uncertainty and increases the applicability of the generated plans.

However, neither of these approaches considers the storage, which is a key element of BHSs at major airports since baggage that arrives before the loading process has started must be stored. For example, at Munich Airport, more than 32% of the bags are stored. The storage allows to influence the loading process by scheduling the depletion of the storage (which might start once the loading process has started or later). Another aspect that is highly interrelated but not considered by the previous approaches is that workers are required in order for bags to leave the BHS. The assignment of a limited workforce, decision (3), is a crucial factor for ground handling to decrease labor costs (Barbot 2012). Frey, Kolisch, and Artigues (2017) present a model that considers both the assignment of flights to carousels and the scheduling of flights' baggage handling based on simple point estimates for the baggage arrivals. They create a plan for the next day which minimizes workload peaks and can be used as a basis for ground handler staffing before the day of operations. While this staffing is an input in the intra-day problem that we consider, their generated plans become infeasible on the day of operations due to the uncertain and dynamic environment. To manage the day of operations, dynamic re-optimization and a robust planning approach, addressed in this paper, are crucial. Furthermore, as we address the intra-day operations, we have to relax the simplifying assumption made by Frey, Kolisch, and Artigues (2017) that the number of workers assigned to a flight must be constant for the complete duration of the loading process. Baggage arrivals fluctuate and loading may take a long time, e.g. up to several hours at Munich Airport.

Consequently, it is common that the number of handlers loading bags changes throughout a flight's loading process, as a flexible assignment allows a better utilization of the workforce.

We provide a model formulation for the intra-day BH problem, which allows flexible work profiles and is designed to create reliable solutions based on past flight data. The main contributions in this paper are: (i) a robust planning approach for dynamic baggage handling, embedded in a rolling horizon solution architecture, that largely mitigates uncertainty employing historical data, (ii) a column-generation-based solution method to quickly obtain high-quality solutions, and (iii) management insights regarding the benefits of flexible work profiles based on real-world data. In practice, the BH process is typically planned and controlled by a dispatcher using up-to-date information and a graphical user interface. However, without any sophisticated decision support, the combinatorial nature and complexity of the planning task make it impossible to derive good solutions manually. Our empirical results, based on real-world data from Munich Airport, show that our procedure outperforms a constructive algorithm, that mimics the logic a human dispatcher applies, as well as a tabu search heuristic in terms of mishandled bags and capacity violations. Furthermore, we show that flexible work profiles significantly improve the system's performance. The highest degree of flexibility allows plans to be generated using about half the resources currently available, and almost entirely without left bags or capacity violations.

The remainder of this paper is structured as follows: in Section 2 we present the dynamic planning approach. The mathematical model is presented in Section 3, and the column-generation-based solution procedure is developed in Section 4. The computational study in Section 5 compares the performance of the proposed procedure to a constructive heuristic as well as a tabu search heuristic, and elaborates on the benefits of worker flexibility. Finally, we provide conclusions in Section 6.

2 Solution Approach

The overall aim of the airport management is to plan the baggage loading process such that, over the course of a day, the total number of left bags, as well as capacity violations of the carousels' conveyor belts and the storage remain as low as possible. In order to reach this goal, we introduce a rolling horizon approach that (i) delivers an optimized plan in each re-optimization, even when resources are insufficient, (ii) exploits information that becomes known as time progresses, and increases the likelihood that plans are feasible by utilizing historical information about baggage arrivals. (iii) We introduce the concept of minimum block length and limit the number of changes to analyze the value of worker flexibility.

Reacting to changing data - Rolling planning. To take into account updated data (including arrival of flights / transfer luggage, arrival of check-in luggage, and departures of flights) the planning is revised over the course of the planning day in a rolling horizon fashion. The quantity of transfer luggage of an inbound flight becomes known once the BH at its outbound airport is completed. The on-block time of inbound flights (arrival time of an airplane at its stand) and the departure time of outbound flights may change depending on factors such as airspace capacity, airport capacity, and weather conditions. For inbound flights, once an aircraft has reached its stand, we assume a constant duration until bags are fed into the BHS and a constant duration for the transport of bags through the BHS. Furthermore, we assume the rate at which the bags are fed into the BHS to be constant, which makes the arrival of transfer baggage deterministic, once the on-block time is known. Arrivals of check-in luggage become known once bags are registered at the check-in. For outbound flights, delays anticipated well in advance also change the end of the BH. In contrast, delays that occur on short notice due to unexpected problems, such as when refueling or cleaning takes too long, mechanical breakdowns, or missing crew do not influence the end of the BH as it ends some time before the flight's estimated departure time.

Planning is based on a discrete horizon that consists of T periods of equal length which are numbered from 0 to $T-1$. For simplicity, we map each period's start time to its period number, i.e. the discrete planning horizon $\mathcal{T} = \{0, \dots, T\}$ represents $T+1$ evenly spaced points in time. An algorithm requires time to generate a solution, and the available time for the optimization is set to one period. P^t denotes the problem that is solved in period t, which takes into account the available data and the plan in the beginning of period t, while its solution takes effect in period $t+1$. To increase the tractability of each re-optimization problem, the length of the planning horizon of problem P^t is limited by the parameter Δ, i.e. $\mathcal{T}^t := \{t+1, \dots, \min\{t+1+\Delta, T\}\}$.

To solve each optimization problem quickly, as required by the real-time environment, we develop a column-generation-based procedure. To obtain integral solutions, column generation is

Table 1: Time and planning horizon

Notation	Description
Δ	Number of periods to limit the planning horizon
P^t	(Re-)optimization problem that is solved in period t
$\mathcal{T} = \{0, \ldots, T\}$	Planning horizon of the complete day
$\mathcal{T}^t = \{t+1, \ldots, \min\{t+1+\Delta, T\}\}$	Planning horizon of problem P^t

embedded into a branch-and-price search, and a heuristic search strategy called limited discrepancy search (see Harvey and Ginsberg 1995) is used.

Utilizing historical information to create reliable plans - A data-driven approach. Re-optimization of problem P^t based on expected values from historical data is likely to lead to poor solutions since planning decisions, e.g. the start of the loading process or storage depletion, have long-term consequences and deviations from the actual data are common. In general, if the point estimates are set too high, e.g., to the 80%-quantiles, it might not be possible to generate feasible solutions. If the point estimates are set too low, e.g., to the 20%-quantiles, the resulting plan might not be “robust” to deviations from the estimates. Stochastic optimization entailing complete knowledge about the underlying probability distribution of uncertainties is unrealistic in BH, and the classic robust optimization paradigm (Ben-Tal, Ghaoui, and Nemirovski 2009), assuming that constraints have to be satisfied for all realizations of the uncertain parameter, is appealing, but the worst case is often infeasible in BH. Instead, we generate plans that are resource-feasible and as reliable as possible employing historical data. In general, robustness is a measure of the immunity to uncertainties or changes and refers to approaches that create robust solutions (Bertsimas and Sim 2004), and in our approach robustness means that plans are designed so that failures such as left bags and backlogs in the BHS are unlikely to occur in the presence of unknown arrivals of bags. In the following, we distinguish check-in baggage and transfer baggage.

The arrival of check-in baggage of a flight is not known, since passengers’ arrival times at the check-in counters, the amount of baggage they carry, and the processing times at check-in are all unknown. However, arrivals of passengers at the check-in counters show similar patterns for flights with identical flight numbers and destinations that depart on the same day of the week (see Stolletz 2011). For example, the passenger arrivals for flight number LH 2472 from Munich to London on some Tuesday are expected to be similar to the passenger arrivals for flight number LH 2472 from Munich to London on the following Tuesday. We exploit these similarities in the optimization. For each departing flight, we use a set of historical check-in arrival data, i.e. we group flights with identical flight number and destination departing on the same weekday. Thus, we employ a data-driven approach, i.e. uncertainty is dealt with by directly exploiting historical data in the optimization.

The arrival time of transfer baggage is based on the estimated on-block time (which is dynamically updated in our approach and which is quite accurate once the flight has departed), and the quantity of transfer baggage becomes known once BH of the flight is completed at the airport of origin.

In our approach, we classify baggage into *known baggage* and *anticipated baggage*. Anticipated baggage comprises baggage where the exact amount is uncertain and known baggage comprises bags that are either already in the BHS or will arrive with certainty (i.e. transfer baggage on aircraft that have already departed). Using scenarios and lexicographic ordering, our planning approach strictly prioritizes known baggage over anticipated baggage, which leads to the optimization being carried out in two phases (as we will explain in detail in the following sections).

Introducing flexible work profiles. The number of workers assigned to a flight can change during the loading process. For example, planners often allocate a few workers for the entire time the flight is being processed and additional workers during the peak periods of the flight's loading process. However, it is impractical to reassign workers too often, because coordinating the workers will become too difficult. Therefore, we introduce the minimum block length, which defines a timespan for which the number of assigned workers must be constant. The minimum block length concept was introduced for the resource-constrained project scheduling problem with flexible work profiles in Fündeling and Trautmann (2010). Additionally, we introduce the maximum number of changes, which defines the maximum number of times the number of workers is allowed to change during a flight's loading process. Thus, a constant number of workers can be enforced for the entire duration of the loading process. Minimum block length and maximum number of changes will allow us to analyze the value of flexible work profiles.

3 Mathematical Model

In this section, we develop the model for the baggage handling problem.

Baggage arrivals. For each flight, our model will use scenarios of baggage arrivals. Each scenario comprises *known bags* and *anticipated bags.* Accordingly, a scenario is referenced by a pair of indices (k,a), where k refers to the known bags and is from index set $\mathcal{K}^{\mathrm{K}} := \left\{1,\ldots,K^{\mathrm{K}}\right\}$ and a refers to the anticipated bags and is from index set $\mathcal{K}^{\mathrm{A}} := \left\{0,\ldots,K^{\mathrm{A}}\right\}$. The number of known bags of flight i in planning problem P^t arriving in period τ in scenario (k,a) is denoted as $A_{i\tau}^{\mathrm{K},tk}$ and the number of anticipated bags is denoted as $A_{i\tau}^{\mathrm{A},ta}$. The sum $A_{i\tau}^{tka} = A_{i\tau}^{\mathrm{K},tk} + A_{i\tau}^{\mathrm{A},ta}$ denotes the number of *known bags* and *anticipated bags* in *scenario* $(k,a) \in \mathcal{K}^K \times \mathcal{K}^A$ (and t,i,τ as previously).

The number of anticipated bags in scenario (k,a) is based on an empirical distribution of bags for check-ins. The distribution is obtained from observations of historical flights with the same flight number departing on the same weekday, e.g. eight observations of the United flight from Munich to Newark on Tuesday at 12:35 pm. $A_{i\tau}^{\mathrm{A},ta}$ consists of the q_a-quantile of the empirical distribution of period τ, where $q_1 < q_2 < \ldots < q_{K^{\mathrm{A}}} = 100\%$. For $a=0$ we set $A_{i\tau}^{\mathrm{A},ta} = 0$, which will guarantee that a feasible solution exists. For example, if the empirical distribution of the anticipated bags is $(0,1,2,2,2,3,3,5)$, and $q_3 = 75\%$, we obtain $A_{i\tau}^{\mathrm{A},t3} = 3$.

The number of known bags in scenario (k,a) is a fraction of the arriving transfer bags rounded to the next integer, i.e. $A_{i\tau}^{\mathrm{K},tk} = \left\lfloor \alpha_k A_{i\tau}^{\mathrm{K},t} + \frac{1}{2} \right\rfloor$, where $A_{i\tau}^{\mathrm{K},t}$ denotes the number of arriving transfer bags, and $0 \leq \alpha_1 < \cdots < \alpha_{K^{\mathrm{K}}} = 1$. For example, if the number of transfer bags is $A_{i\tau}^{\mathrm{K},t} = 5$, and $(\alpha_1,\alpha_2,\ldots,\alpha_{K^{\mathrm{K}}}) = (0.1,0.2,\ldots,1.0)$, we obtain $A_{i\tau}^{\mathrm{K},t2} = \lfloor 0.2 \cdot 5 + \frac{1}{2} \rfloor = 1$.

S*cenario* (k,a) of flight i in planning problem P^t is defined as the vector $\left(A_{i\tau}^{tka}\right)_{\tau=-1,\ldots,E_i-1}$, where E_i denotes the end of flight i's baggage handling and period $\tau=-1$ is reserved for bags that are already in the central baggage storage at the beginning of the planning horizon. In each optimization problem P^t we aim to consider the largest possible number of known bags and the highest possible quantiles of anticipated bags, thereby increasing robustness (the algorithmic details are given in Section 4.1). The *scenarios* are designed such that they can be ordered, that is, they satisfy both conditions $A_{i\tau}^{t1a} \leq A_{i\tau}^{t2a} \leq \cdots \leq A_{i\tau}^{tK^{\mathrm{K}}a}$ and $A_{i\tau}^{tk1} \leq A_{i\tau}^{tk2} \leq \cdots \leq A_{i\tau}^{tkK^{\mathrm{A}}}$. The scenarios are prioritized in the order $(1,0),(2,0),\ldots,\left(K^{\mathrm{K}},0\right)$, i.e. we increase scenario k step by step and for all flights simultaneously with $a = 0$. If scenario k cannot be further increased for a flight, we continue to increase scenario k for the remaining flights, and once k is fixed for all flights, we increase scenario a in the order $(k,0), (k,1), \dots, (k, K^{\mathrm{A}})$. If all known bags and the highest possible quantile of anticipated bags can be taken into account for all flights, our solution is based on the worst case, similar to the robust optimization paradigm.

(Baggage) departures. The set of departing flights of the complete planning day is denoted as $\mathcal{F}$. In each optimization problem P^t, we have to consider the state of the system at the beginning of period $t+1$, which depends on the state of the system at the beginning of period t and the plan executed in period t. Flights are categorized into the following subsets which are illustrated in Figure (2), of which we formally define only those needed in the model. The *past flights* (gray

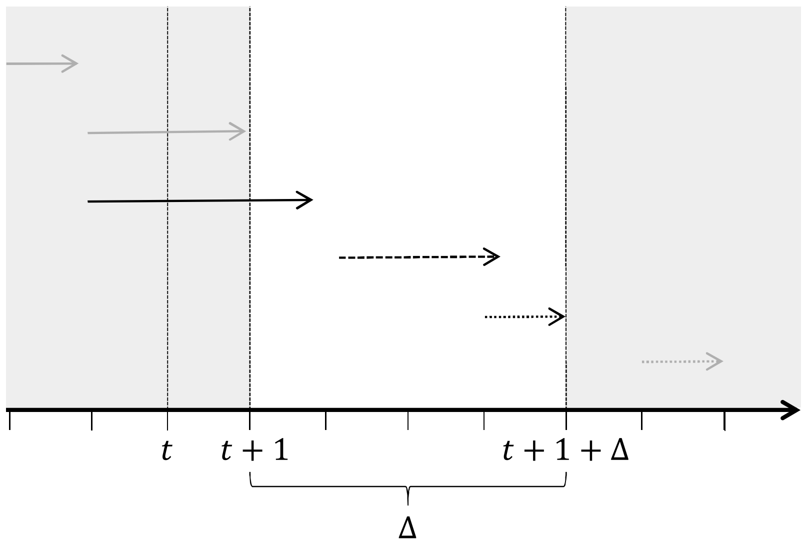


Figure 2: Categorization of flights

solid lines), i.e. flights that are finished by the beginning of period $t+1$ as well as *future flights* (gray dotted lines), *i.e.* flights that can start at earliest in period $t+1+\Delta$ do not need to be considered and are excluded from the decision-making. Let $\mathcal{F}^t$ denote the subset of flights of $\mathcal{F}$ that are included in P^t. They are categorized into *flights in progress* (black solid lines), whose loading process has or will be started by the beginning of period $t+1$ (storage depletion may or may not have started), *planned flights* (black dashed lines), for which a plan is available from the previous optimization, but the loading process has not been started, and *new flights* (black dotted lines), which are considered for the first time in P^t (i.e. they were not considered in the previous optimization).

Schedules / work profiles. A schedule / work profile for flight i defines the start of the loading process, the start of the storage depletion, and the number of workers in each period. The start of the loading process and of the storage depletion of schedule π are denoted as $s^{\mathrm{L}}_{\pi} \in \mathcal{T}$ and $s^{\mathrm{D}}_{\pi} \in \mathcal{T}$ (with $s^{\mathrm{D}}_{\pi} \geq s^{\mathrm{L}}_{\pi}$), respectively. The loading process of flight i can start at the earliest at time $S_i \in \mathcal{T}$, i.e. $s^{\mathrm{L}}_{\pi} \geq S_i$. Let $w_{\pi\tau}$ denote the number of assigned workers in period $\tau = S_i, \ldots, E_i - 1$ (which is equal to the number of working stations employed at the corresponding carousel). Workers must not be assigned before the loading process starts, and it is common that due to organizational regulations, at least one worker must be assigned throughout the entire loading process. The minimum block length M defines a number of consecutive periods that must have a constant number of workers, i.e. if $w_{\pi,\tau-1} \neq w_{\pi\tau}$, then $w_{\pi\tau} = \ldots = w_{\pi,\tau+M-1}$, where it is assumed that $w_{\pi,S_i-1} = 0$ and $w_{\pi E_i} = 0$. The maximum number of changes J limits how often the number of workers assigned to a flight is allowed to change during its loading process. In the following, we use the terms schedule and (work) profile interchangeably. Summarizing, profile π for flight i comprises s^{L}_{π}, s^{D}_{π}, and vector $(w_{\pi\tau})_{\tau=S_i,\ldots,E_i-1}$.

Storage. Before the loading process has started, all incoming bags are sent to the central storage, which has a limited capacity of K^{S} bags. E.g., at Munich Airport, the storage capacity is 6,000 bags, which is about 10% of the daily handled bags. Once the loading process has started, all incoming bags are directly sent to the assigned carousel. The storage depletion can begin, at the earliest, when the loading process starts. Stored bags are transferred to the assigned carousel at a constant rate of R^{S} bags per time period. The number of bags in storage for flight i and scenario (k, a) at time $\tau = 0, \ldots, E_i$, given profile π, is calculated recursively as follows:

$$\phi^{\mathrm{S},tka}_{i\tau\pi} = \begin{cases} A^{tka}_{i,-1} & \text{for } \tau = 0 & (a) \\ \phi^{\mathrm{S},tka}_{i,\tau-1,\pi} + A^{tka}_{i,\tau-1} & \text{for } 0 < \tau \leq s^{\mathrm{L}}_{\pi} & (b) \\ \phi^{\mathrm{S},tka}_{i,\tau-1,\pi} & \text{for } s^{\mathrm{L}}_{\pi} < \tau \leq s^{\mathrm{D}}_{\pi} & (c) \\ \left[\phi^{\mathrm{S},tka}_{i,\tau-1,\pi} - R^{\mathrm{S}}\right]^{+} & \text{for } s^{\mathrm{D}}_{\pi} < \tau, & (d) \end{cases} \tag{1}$$

where $[x]^+$ is used as a short form for $\max\{x, 0\}$. Bags arriving before the loading process starts are directed to the central storage (a) and (b). The bags remain in the storage until the storage depletion starts (c) and are depleted at a constant rate once the storage depletion has started (d).

Carousels. Let $\mathcal{C}$ denote the set of carousels and $\mathcal{C}_i$ the set of carousels flight i can be assigned to. A carousel $c \in \mathcal{C}$ is characterized by the conveyor belt capacity which is measured in the number of bags K_c^{B} and the number of working stations K_c^{N}. For example, at Munich Airport, a carousel has up to six working stations. Once a flight's loading process has started at a carousel, it has to be finished at the same carousel.

Let $\phi_{i\tau\pi}^{\mathrm{S}\to\mathrm{B},tka} = \left[\phi_{i\tau\pi}^{\mathrm{S},tka} - R^{\mathrm{S}}\right]^+ = \phi_{i,\tau+1,\pi}^{\mathrm{S},tka} - \phi_{i\tau\pi}^{\mathrm{S},tka}$ denote the number of bags of flight i that are transferred from the central storage to the carousel in period $\tau = S_i, \ldots, E_i - 1$ given schedule π and scenario (k, a). The number of available workers during period τ is denoted as K_τ^{N}, and the loading rate per worker is R^{W} bags per period, which is assumed to be constant. The number of bags of flight i on the carousel for scenario (k, a) at time $\tau = S_i, \ldots, E_i$, given profile π, can be calculated recursively as follows:

$$\phi_{i\tau\pi}^{\mathrm{B},tka} = \begin{cases} 0 & \text{for } \tau \le s_\pi^{\mathrm{L}} \quad (a) \\ \left[\phi_{i,\tau-1,\pi}^{\mathrm{B},tka} + A_{i,\tau-1}^{tka} - w_{\pi,\tau-1}R^{\mathrm{W}}\right]^+ & \text{for } s_\pi^{\mathrm{L}} < \tau \le s_\pi^{\mathrm{D}} \quad (b) \\ \left[\phi_{i,\tau-1,\pi}^{\mathrm{B},tka} + A_{i,\tau-1}^{tka} - w_{\pi,\tau-1}R^{\mathrm{W}} + \phi_{i,\tau-1,\pi}^{\mathrm{S}\to\mathrm{B},tka}\right]^+ & \text{for } s_\pi^{\mathrm{D}} < \tau. \quad (c) \end{cases} \tag{2}$$

Before the loading process has started, there are no bags on the carousel as they are directed to the central storage (a). From the start of the loading process, arriving bags are directed to the carousel, and the assigned workers load bags from the carousel into containers (b). Once storage depletion has begun, the bags that arrive from the storage are added (c).

Model formulation. In each optimization problem P^t, we find a resource-feasible solution for the planning horizon $\mathcal{T}^t$ with the objective to consider the largest possible number of known bags and the highest possible quantiles of anticipated bags, thereby increasing robustness. At time t, a profile π for flight i can only be feasible at carousel $c \in \mathcal{C}_i$ with respect to scenario (k, a) if the carousel capacity is not exceeded, i.e. if $\phi_{i\tau\pi}^{\mathrm{B},tka} \le K_c^{\mathrm{B}}$ for all $\tau = S_i, \ldots, E_i - 1$, and if there are no left bags at time E_i, i.e. if both $\phi_{iE_i\pi}^{\mathrm{S},tka}$ and $\phi_{iE_i\pi}^{\mathrm{B},tka}$ are zero. Let $\mathcal{S}_{ic}^{tka}$ denote the set of profiles for

$i \in \mathcal{F}^t$, $c \in \mathcal{C}_i$, $k \in \mathcal{K}^{\mathrm{K}}$, $a \in \mathcal{K}^{\mathrm{A}}$ and $t \in \mathcal{T}^t$. Furthermore, let $\mathcal{S}_{ic\tau}^{tka} := \{\pi \in \mathcal{S}_{ic}^{tka} \mid s_\pi^{\mathrm{L}} \leq \tau < E_i\}$ be the subset of $\mathcal{S}_{ic}^{tka}$, where flight i's loading process is in progress during period τ.

Table 2: Notation for the baggage handling problem

Notation	Description
$A_{i\tau}^{tka}$	Number of bags arriving given the information at time t
$\mathcal{C}/\ \mathcal{C}_i$	Set of carousels / Set of carousels for flight i
E_i	End of baggage handling
$\mathcal{F}/\ \mathcal{F}^t$	Set of flights / Set of flights included in problem P^t
J	Number of times the number of workers can change
K_c^{B}	Conveyor belt capacity in number of bags
K_c^{N}	Number of working stations
K^{S}	Storage capacity
K^{W}	Available workers
$\mathcal{K}^{\mathrm{K}} = \{1, \ldots, K^{\mathrm{K}}\}$ / $\mathcal{K}^{\mathrm{A}} = \{0, \ldots, K^{\mathrm{A}}\}$	Index set for known bags / anticipated bags
$\mathcal{K}^{\mathrm{K}} \times \mathcal{K}^{\mathrm{A}}$	Set of scenarios
$(k, a) \in \mathcal{K}^{\mathrm{K}} \times \mathcal{K}^{\mathrm{A}}$	Scenario
M	Minimum block length
$p_k^{\mathrm{K}}/\ p_a^{\mathrm{A}}$	Costs for known bags scenario k / anticipated bags scenario a
R^{S}	Storage depletion rate per period
R^{W}	Loading rate per worker in bags per period
S_i	Earliest start of the loading process
s_π^{L} / s_π^{D}	Start of the loading process / storage depletion in profile π
$\mathcal{S}_{ic}^{tka}$	Set of profiles based on $A_{i\tau}^{tka}$
$\mathcal{S}_{ic\tau}^{tka}$	Subset of $\mathcal{S}_{ic}^{tka}$ where loading process is in progress during period τ
$w_{\pi\tau}$	Number of workers assigned in profile π
$x_{ic\pi}^{ka}$	Binary decision variable
$\phi_{i\tau\pi}^{\mathrm{B},tka}$	Bags on the conveyor belt given the information at time t
$\phi_{i\tau\pi}^{\mathrm{S},tka}$	Bags in storage given the information at time t

The index t indicating the current time is dropped in the model. Table 2 (where the indices i, c, k, a, τ and π are omitted in the description) and Table 1 provide an overview of the notation. Binary variable $x_{ic\pi}^{ka}$ is one if the loading process of flight $i \in \mathcal{F}^t$ is conducted at carousel $c \in \mathcal{C}_i$ with profile $\pi \in \mathcal{S}_{ic}^{tka}$, and zero otherwise. Planning flight i with profile $\pi \in \mathcal{S}_{ic}^{ka}$ and hence with scenario (k, a) incurs costs of $p_k^{\mathrm{K}} + p_a^{\mathrm{A}}$, which are set so as to ensure lexicographic preferences. Formally, $p_{K^{\mathrm{A}}}^{\mathrm{A}} = 1$, $|\mathcal{F}^t|\, p_k^{\mathrm{A}} \leq p_{k-1}^{\mathrm{A}}$ for $k = K^{\mathrm{A}}, \ldots, 1$, $|\mathcal{F}^t|\, p_0^{\mathrm{A}} \leq p_{K^{\mathrm{K}}}^{\mathrm{K}}$ and $|\mathcal{F}^t|\, p_k^{\mathrm{K}} \leq p_{k-1}^{\mathrm{K}}$ for $k = K^{\mathrm{K}}, \ldots, 2$ holds. Problem P^t can now be stated as follows:

$$(P^t) \quad \min \quad \sum_{i\in\mathcal{F}}\sum_{c\in\mathcal{C}_i}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic}^{ka}} \left(p_k^{\mathrm{K}} + p_a^{\mathrm{A}}\right) x_{ic\pi}^{ka} \tag{3}$$

subject to

$$\sum_{c\in\mathcal{C}_i}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic}^{tka}} x_{ic\pi}^{ka} = 1 \qquad \forall i \in \mathcal{F} \tag{4}$$

$$\sum_{i\in\mathcal{F}}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic\tau}^{ka}} w_{\pi\tau} x_{ic\pi}^{ka} \leq K_c^{\mathrm{N}} \qquad \forall c \in \mathcal{C}, \tau \in \mathcal{T} \tag{5}$$

$$\sum_{i\in\mathcal{F}}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic\tau}^{ka}} \phi_{i\tau\pi}^{\mathrm{B},ka} x_{ic\pi}^{ka} \leq K_c^{\mathrm{B}} \qquad \forall c \in \mathcal{C}, \tau \in \mathcal{T} \tag{6}$$

$$\sum_{i\in\mathcal{F}}\sum_{c\in\mathcal{C}_i}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic}^{ka}} \phi_{i\tau\pi}^{\mathrm{S},ka} x_{ic\pi}^{ka} \leq K^{\mathrm{S}} \qquad \forall \tau \in \mathcal{T} \tag{7}$$

$$\sum_{i\in\mathcal{F}}\sum_{c\in\mathcal{C}_i}\sum_{k\in\mathcal{K}^{\mathrm{K}}}\sum_{a\in\mathcal{K}^{\mathrm{A}}}\sum_{\pi\in\mathcal{S}_{ic\tau}^{ka}} w_{\pi\tau} x_{ic\pi}^{ka} \leq K_\tau^{\mathrm{W}} \qquad \forall \tau \in \mathcal{T} \tag{8}$$

$$x_{ic\pi}^{ka} \in \{0,1\} \qquad \forall i \in \mathcal{F}, c \in \mathcal{C}_i, \; k \in \mathcal{K}^{\mathrm{K}}, a \in \mathcal{K}^{\mathrm{A}}, \; \pi \in \mathcal{S}_{ic}^{ka} \tag{9}$$

Objective function (3) penalizes the use of scenarios with low indices. Partitioning constraints (4) require each flight to be assigned. Constraints (5) limit the number of available working stations for each carousel. Carousels' conveyor belt capacities are limited in constraints (6). The central storage's capacity is limited by constraints (7). Constraints (8) make sure that worker capacities are not exceeded. Finally, constraints (9) define the variables.

4 Solution Methodology

Model (3) — (9) is $\mathcal{NP}$–complete by reduction of the set partitioning problem and suffers from the defect that the variables, of which there are an exponential number, should be enumerated. For each flight $i \in \mathcal{F}^t$, carousel $c \in \mathcal{C}_i$, and scenario $(k,a) \in \mathcal{K}^{\mathrm{K}} \times \mathcal{K}^{\mathrm{A}}$ the number of profiles $\left|\mathcal{S}_{ic}^{tka}\right|$ is in $\mathcal{O}\left(\left(K_c^{\mathrm{N}}\right)^{E_i - S_i}\right)$. Since most of these variables will be zero in an optimal solution, promising variables, i.e. new profiles, are generated as needed by column generation. The master problem corresponds to problem P^t and there are $\mathcal{O}\left(\left|\mathcal{F}^t\right| \cdot |\mathcal{C}| \cdot \left|\mathcal{K}^{\mathrm{K}}\right| \cdot \left|\mathcal{K}^{\mathrm{A}}\right|\right)$ subproblems — one per flight, carousel, and scenario. As we will show below, the subproblems can be solved efficiently with dynamic programming. To obtain integral solutions, column generation is embedded into branch-and-price. Since reaching and proving optimality can take too long, a heuristic search strategy is applied. The lexicographic objective function (3) can be minimized in stages, which avoids numerical problems that otherwise occur due to the large range of objective coefficients $\left[p^{\mathrm{A}}_{K^{\mathrm{A}}}, p^{\mathrm{K}}_1\right]$ and reduces the number of variables that must be considered simultaneously, which increases the tractability of the problem significantly.

4.1 Minimizing the lexicographic objective function

The lexicographic objective function (3) can be minimized in two phases, consisting of K^{K} and K^{A} stages. In the first phase, a solution is found for scenarios $(1,0),\ldots,\left(K^{\mathrm{K}},0\right)$, that is, ignoring the anticipated baggage. In the first stage of the first phase, a feasible solution must be found for scenario $(1,0)$ of all flights. At each subsequent stage $k = 2,\ldots,K^{\mathrm{K}}$, there is a set of *open flights* and a set of *fixed flights*. At the start of stage 2, all flights are added to the set of open flights, and the set of fixed flights is empty. Assume we are in stage $k \geq 2$ of the first phase. For each open flight there are two scenarios available, $(k-1,0)$ and $(k,0)$, and planning based on the former is penalized in the objective function. Each open flight i that is not planned based on scenario $(k,0)$ is moved to the set of fixed flights, and its index of the known bags is fixed to $\hat{k}_i = k-1$. Afterwards, we move on to the next stage $k \leftarrow k+1$. Proceeding this way, only the variables $x^{\hat{k}_i 0}_{ic\pi}$ are needed for the fixed flights, and only the variables $x^{k-1,0}_{ic\pi}$ and $x^{k,0}_{ic\pi}$ are needed for the open flights, where all $x^{k-1,0}_{ic\pi}$ are penalized. Note that the solution of the previous stage is always feasible for the current stage. At the end of the first phase, $\hat{k}_i$ is already set for all fixed flights, and for all remaining open flights $\hat{k}_i$ is set to K^{K}. In the second phase, the same procedure is applied for the anticipated baggage. All flights are added to the set of open flights and the set of fixed flights is now empty. Again, we iteratively try to increase the scenarios of the open flights. Assume we are in stage $a \geq 1$ of the second phase. For each open flight there are two scenarios available, $\left(\hat{k}_i, a-1\right)$ and $\left(\hat{k}_i, a\right)$, and planning based on the former is penalized. Each open flight i that is not planned based on scenario $\left(\hat{k}_i, a\right)$ is moved to the set of fixed flights, and its index of the anticipated bags is fixed to $\hat{a}_i = a - 1$. Therefore, its final scenario is $\left(\hat{k}_i, \hat{a}_i\right)$. All flights that are open at the end of the second phase are planned with scenario $\left(\hat{k}_i, K^{\mathrm{A}}\right)$.

4.2 Column generation

Problem P^t is the *master problem* (MP) of the column generation. Since there is an exponential number of profiles, the sets of profiles are restricted to subsets $\tilde{\mathcal{S}}_{ic}^{tka} \subseteq \mathcal{S}_{ic}^{tka}$. The MP containing only the columns in $\bigcup_{i\in\mathcal{F}^t, c\in C, k\in\mathcal{K}^{\mathrm{K}}, k\in\mathcal{K}^{\mathrm{A}}} \tilde{\mathcal{S}}_{ic}^{tka}$ is called the *restricted master problem* (RMP). As the RMP does not necessarily yield an optimal solution to the MP, the integrality requirements on the variables are relaxed, and the continuous relaxation of RMP is solved to optimality. Then, the optimal dual variables are used to identify new columns with negative reduced costs, which is done in subproblems (*pricing problems)* — one for each flight-carousel-scenario triplet. If a column with negative reduced costs is found, it is added to the RMP and the process repeats until no more column with negative reduced costs can be found. As there is then no candidate column that would improve the objective function value, the solution is optimal for the linear relaxation of the MP.

Pricing refers to any method of computing the reduced costs vector of the non-basic variables. However, it is not necessary to compute the reduced costs of all non-basic variables, and pricing out only a few candidates for entering the basis is known as *partial pricing* (Chvátal 1983). Column generation can be considered a (partial) *pricing scheme* for large-scale linear programs (see Lübbecke and Desrosiers 2005). When there are many subproblems, as is the case here, it is often more efficient to consider only a few of them in each iteration, which is called *partial column generation*. When choosing a pricing scheme, it is important to strike a balance between the computational time spend for solving the subproblems and for solving the RMP. On the one extreme, when all subproblems are solved and only the best column is added to the RMP, a lot of time is spent on solving the subproblems, but the RMP receives relatively few high-quality columns. On the other hand, if more columns are added to the RMP, it takes longer to solve, but fewer iterations may be needed. For our problem, since we can efficiently solve the pricing problems, it is best to solve all pricing problems and add the best column. As every subproblem runs in a single threat, the pricing problems can be solved in parallel, which speeds up each iteration further.

Notations	Descriptions
$s_\tau^{\mathrm{L}} \in \{0,1\}$	1, if loading process is in progress in period τ; 0 otherwise
$s_\tau^{\mathrm{S}} \in \{0,1\}$	1, if storage depletion is in progress in period τ; 0 otherwise
$w_\tau \in \left\{0,\dots,K_c^{\mathrm{W}}\right\}$	Number of workers (and working stations) in period τ
$\delta_\tau \in \{1,\dots,M\}$	Number of periods with constant number of workers at time τ
$\gamma_\tau \in \{1,\dots,J\}$	Number of times the number of workers has changed until time τ
$\phi_\tau^{\mathrm{S}} \in \left\{0,\dots,\sum_\tau A_{i\tau}^{tka}\right\}$	Number of bags in the central storage at time τ
$\phi_\tau^{\mathrm{B}} \in \left\{0,\dots,K_c^{\mathrm{B}}\right\}$	Number of bags on the conveyor belt at time τ

Table 3: Decision variables for pricing problem PP_{ic}^{tka}

4.3 Subproblems

Assume the continuous relaxation of RMP is solved. Let λ_i^{F}, $\lambda_{c\tau}^{\mathrm{N}}$, $\lambda_{c\tau}^{\mathrm{B}}$, $\lambda_\tau^{\mathrm{S}}$, and $\lambda_\tau^{\mathrm{W}}$ denote the dual variables of the assignment constraints (4), and the capacity constraints (5), (6), (7), and (8), respectively. The most promising schedule for flight $i \in \mathcal{F}^t$, carousel $c \in \mathcal{C}_i$, and scenario $(k,a) \in \mathcal{K}^{\mathrm{K}} \times \mathcal{K}^{\mathrm{A}}$ is the solution of the *pricing problem* PP_{ic}^{tka}, which is a shortest path problem from a source node to a destination node (one-to-one) in an acyclic network. Hence, PP_{ic}^{tka} can be solved efficiently with dynamic programming. Let $\bar{S}_i := \max\{S_i, t+1\}$ denote the first period of PP_{ic}^{tka}. The decision variables are defined in Table 3, where the indices t, i, c, k, and a are omitted.

System states. The state at time $\tau = \bar{S}_i,\dots,E_i$ is defined as $z_\tau = \left(s_{\tau-1}^{\mathrm{L}}, w_{\tau-1}, \delta_\tau, \gamma_\tau, s_{\tau-1}^{\mathrm{S}}, \phi_\tau^{\mathrm{S}}, \phi_\tau^{\mathrm{B}}\right)$ and refers to time τ immediately before a decision is made. The initial state $z_{\bar{S}_i}$ at time $\bar{S}_i$ depends on whether the flight's loading process and storage depletion are in progress and how many bags are in the storage and on the carousel's belt at time $\bar{S}_i$.

Transitions. When making a decision $a_\tau := \left(s_\tau^{\mathrm{L}}, s_\tau^{\mathrm{S}}, w_\tau\right)$ at time $\tau = \bar{S}_i,\dots,E_i - 1$ given state z_τ, the following constraints must hold.

$$s^{\mathrm{L}}_{\tau-1} \leq s^{\mathrm{L}}_{\tau} \tag{10}$$

$$s^{\mathrm{S}}_{\tau-1} \leq s^{\mathrm{S}}_{\tau} \tag{11}$$

$$s^{\mathrm{S}}_{\tau} \leq s^{\mathrm{L}}_{\tau} \tag{12}$$

$$s^{\mathrm{L}}_{\tau} \leq w_{\tau} \leq K^{\mathrm{N}}_{c} s^{\mathrm{L}}_{\tau} \tag{13}$$

$$\delta_{\tau} < M \Rightarrow w_{\tau} = w_{\tau-1} \tag{14}$$

$$w_{E_i-M} = \ldots = w_{E_i-1} \tag{15}$$

$$\gamma_{\tau} = J \Rightarrow w_{\tau} = w_{\tau-1} \tag{16}$$

Constraints (10) and (11) state that both the loading process and the storage depletion cannot be interrupted once started. Constraint (12) makes sure that the storage depletion does not start before the loading process. Constraint (13) requires at least one worker to be assigned during the loading process (and at most as many as the carousel has working stations), and workers may not be assigned before the loading process has started. Constraints (14) and (15) enforce the block length restriction, where the latter affects the end of the loading process. Finally, constraint (16) limits the number of times the number of workers can change.

The next state $z_{\tau+1}$ results from the current state z_{τ} and the decision a_{τ}. The state variables s^{L}_{τ}, s^{D}_{τ}, and w_{τ} are part of decision a_{τ}, variables $\delta_{\tau+1}$ and $\gamma_{\tau+1}$ are set according to constraints (17) and (18),

$$\delta_{\tau+1} = \begin{cases} \delta_{\tau} + 1 & \text{if } w_{\tau} = w_{\tau-1} \\ 1 & \text{otherwise} \end{cases} \tag{17}$$

$$\gamma_{\tau+1} = \begin{cases} \gamma_{\tau} & \text{if } w_{\tau} = w_{\tau-1} \\ \gamma_{\tau} + 1 & \text{otherwise} \end{cases} \tag{18}$$

and variables $\phi^{\mathrm{S}}_{\tau+1}$ and $\phi^{\mathrm{B}}_{\tau+1}$ are set with Equations (1) and (2).

Reduced costs and optimality equation. The one-period cost for period $\tau = \bar{S}_i, \dots, E_i - 1$ is a function of w_τ, $\phi^{\mathrm{B}}_{\tau+1}$, $\phi^{\mathrm{S}}_{\tau+1}$ and the dual variables, that is

$$c\left(w_\tau, \phi^{\mathrm{S}}_{\tau+1}, \phi^{\mathrm{B}}_{\tau+1}\right) = -\lambda^{\mathrm{N}}_{c\tau} w_\tau - \lambda^{\mathrm{W}}_{\tau} w_\tau - \lambda^{\mathrm{B}}_{c,\tau+1}\phi^{\mathrm{B}}_{\tau+1} - \lambda^{\mathrm{S}}_{\tau+1}\phi^{\mathrm{S}}_{\tau+1}. \tag{19}$$

The cost for being in state z_τ can be defined via a Bellman recursion as

$$v\left(z_\tau\right) = \begin{cases} \min_{z_{\tau-1} \in \mathcal{Z}_{\tau-1}(z_\tau)} \left\{ v\left(z_{\tau-1}\right) + c\left(w_{\tau-1}, \phi^{\mathrm{S}}_{\tau}, \phi^{\mathrm{B}}_{\tau}\right)\right\} & \text{for } \tau > \bar{S}_i \\ p^{\mathrm{K}}_k + p^{\mathrm{A}}_a - \lambda^{\mathrm{F}}_i & \text{for } \tau = \bar{S}_i \end{cases} \tag{20}$$

where $\mathcal{Z}_{\tau-1}\left(z_\tau\right)$ denotes the set of all states for which a feasible decision $a_{\tau-1}$ exists that leads to state z_τ. Consider the set $\mathcal{Z}_{E_i}$ of terminal states where all bags are loaded into ULDs, i.e. z_{E_i} with $\phi^{\mathrm{B}}_{E_i} = \phi^{\mathrm{S}}_{E_i} = 0$. Pricing problem PP^{tka}_{ic} can be solved by finding a state

$$z^* = \arg \min_{z_{E_i} \in \mathcal{Z}_{E_i}} \left\{ v\left(z_{E_i}\right)\right\}. \tag{21}$$

The actual profile can be derived by tracking the sequence of transitions that transforms the initial state $z_{\bar{S}_i}$ into z^*.

State-space reduction. Next, we develop a reduction of the state space by employing a dominance criterion. We say that state z_τ dominates another state z'_τ ($z_\tau \succ z'_\tau$), if the following conditions hold, where the variables annotated with " $'$ " refer to state z'_τ.

$$v\left(z_\tau\right) \leq v\left(z'_\tau\right) \tag{22}$$
$$\phi^{\mathrm{B}}_\tau \leq \left(\phi^{\mathrm{B}}_\tau\right)' \tag{23}$$
$$\phi^{\mathrm{S}}_\tau \leq \left(\phi^{\mathrm{S}}_\tau\right)' \tag{24}$$
$$\neg s^{\mathrm{L}}_\tau \vee \left(s^{\mathrm{L}}_\tau\right)' \tag{25}$$
$$\neg s^{\mathrm{S}}_\tau \vee \left(s^{\mathrm{S}}_\tau\right)' \tag{26}$$
$$\delta_\tau \geq \delta'_\tau \tag{27}$$
$$w_{\tau-1} = w'_{\tau-1} \tag{28}$$
$$\gamma_\tau \leq \gamma'_\tau \tag{29}$$

Proposition. *$z_\tau \succ z'_\tau$ implies that an optimal sequence of actions and states $\left(z'_\tau, a'_\tau, z'_{\tau+1}, a'_{\tau+1}, \ldots, z'_{E_i}\right)$ starting in state z'_τ has an objective value that cannot be better than the objective value of an optimal sequence $(z_\tau, a_\tau, z_{\tau+1}, a_{\tau+1}, \ldots, z_{E_i})$ starting in state z_τ.*

Proof. From (25) — (29) it follows that the set of all possible sequences $\left(z'_\tau, a'_\tau, z'_{\tau+1}, a'_{\tau+1}, \ldots, z'_{E_i}\right)$ is a subset of the set of all possible sequences $(z_\tau, a_\tau, z_{\tau+1}, a_{\tau+1}, \ldots, z_{E_i})$. Because of (22) — (24) we have

$$\begin{aligned} &\min_{(z_\tau, a_\tau, z_{\tau+1}, \ldots, z_{E_i})} \left\{ v\left(z_\tau\right) + \sum_{\tau'=\tau}^{E_i-1} c\left(w_\tau, \phi^{\mathrm{S}}_{\tau'+1}, \phi^{\mathrm{B}}_{\tau'+1}\right) \right\} \\ \leq &\min_{\left(z'_\tau, a'_\tau, z'_{\tau+1}, \ldots, z'_{E_i}\right)} \left\{ v\left(z'_\tau\right) + \sum_{\tau'=\tau}^{E_i-1} c\left(w'_\tau, \left(\phi^{\mathrm{S}}_{\tau'+1}\right)', \left(\phi^{\mathrm{B}}_{\tau'+1}\right)'\right) \right\} \end{aligned} \tag{30}$$

which is the desired result. □

A dominated state z_τ can be removed from further consideration. Hence, we do not consider any transition that starts in z_τ. Whenever we reach a state z_τ, we discard it if it is dominated by any other state that has already been evaluated. Otherwise, we keep it and discard states that are dominated by z_τ.

Complexity. There are $n \approx \mathcal{O}\left(2 \cdot K^{\mathrm{N}}_c \cdot M \cdot J \cdot 2 \cdot \sum_\tau A^{tka}_{i\tau} \cdot K^{\mathrm{B}}_c\right)$ states in each period because of the variable domains (see Table (3)). Since there are only arcs from one point in time to the next, the network has $\mathcal{O}\left(n^2\left(E_i - \bar{S}_i\right)\right)$ arcs. Theoretically, the time complexity is $\mathcal{O}(m)$ with m as the number of arcs of the network, since we have to evaluate each arc once at most. However, due to the dominance criterion, much fewer arcs need to be evaluated in practice.

4.4 Branch-and-price

The column generation procedure solves the linear relaxation of the MP. Hence, the solution does not necessarily satisfy the integrality conditions, and column generation must be embedded into a branch-and-bound procedure to solve the original integer problem. The overall procedure is then called *branch-and-price* (see Barnhart et al. (1998) for an overview and Lübbecke and Desrosiers (2005) for a list of applications).

After branching, a column may exist that would price out favorably, but is not present in the RMP. Therefore, it is necessary to continue column generation after branching (Barnhart et al. 1998). Usually, standard branching on the RMP is not appropriate because the columns that have been excluded in the RMP need to be prevented from being regenerated in the subproblems. The additional constraints may destroy the structure of the subproblems that is exploited or make them too hard to be solved in a short computational time. Furthermore, branching on the RMP leads to an unbalanced branch-and-bound tree (Vanderbeck 2000). Hence, we branch on the decisions of the original problem, for which we have not presented a mathematical model, but its content should be clear from Section 3. Assume that the column generation terminated with an optimal solution for the MP at an arbitrary node of the branch-and-price tree. Branching is necessary if there exists a flight whose plan is fractional. A flight's plan can be fractional with respect to (i) the selected scenario, (ii) the assignment to a carousel, (iii) the number of assigned workers in each period, (iv) the start of storage depletion, (v) the minimum block length, and (vi) the maximum number of changes. For each of these, a branching rule that partitions the solution space and excludes the current fractional solution is required. Therefore, we state the following branching rules.

Assignment to carousel and scenario. If flight i is assigned to more than one carousel, we pick one of the carousels, and branch such that flight i is exclusively assigned to that carousel on the left branch and is not assigned to that carousel on the right branch, which partitions the solution space into two sub-spaces and excludes the current fractional solution. If flight i's planning is based on two scenarios (more than two is impossible as we proceed in stages), one scenario is enforced or excluded on the left and right branch, respectively. The aforementioned two branching rules are enforced by removing variables from the MP and by not solving the subproblems of the excluded carousel or scenario.

Work profile. The number of workers assigned to flight i in period $\tau = S_i, \dots, E_i - 1$ is $w_{i\tau} = \sum_{c \in \mathcal{C}_i} \sum_{k \in \mathcal{K}^{\mathrm{K}}} \sum_{a \in \mathcal{K}^{\mathrm{A}}} \sum_{\pi \in \mathcal{S}_{ic}^{tka}} x_{ic\pi}^{ka} w_{\pi\tau}$. If $w_{i\tau}$ is fractional, we branch and either require $w_{i\tau} \geq \lceil \hat{w}_{i\tau}^{n} \rceil$ or $w_{i\tau} \leq \lfloor \hat{w}_{i\tau}^{n} \rfloor$ on the left and right branch, respectively, where $\hat{w}_{i\tau}^{n}$ is the value of $w_{i\tau}$ in the optimal solution of the current node n.

Time period	1	2	3	4	5
Profile π_1 $\left(\frac{2}{3}\right)$	1	1	7	7	7
Profile π_2 $\left(\frac{1}{3}\right)$	4	4	4	1	1
Profile π^n	2	2	6	5	5
Indicator $\zeta_{i\tau}$	1	0	$\frac{2}{3}$	$\frac{1}{3}$	0

Table 4: Combination of profiles

Storage depletion. An indicator for the storage depletion of flight i in period $\tau = S_i, \ldots, E_i - 1$ is $s^{\mathrm{S}}_{i\tau} = \sum_{c\in\mathcal{C}_i} \sum_{k\in\mathcal{K}^{\mathrm{K}}} \sum_{a\in\mathcal{K}^{\mathrm{A}}} \sum_{\pi\in\mathcal{S}^{tka}_{ic}} x^{ka}_{ic\pi} I\left(s^{\mathrm{S}}_{\pi} \leq \tau\right)$, where $I(\cdot)$ is an indicator function that returns one if the given condition is true and zero otherwise. If $0 < s^{\mathrm{S}}_{i\tau} < 1$ in the current node's optimal solution, we branch and either require $s^{\mathrm{S}}_{i\tau} = 1$ or $s^{\mathrm{S}}_{i\tau} = 0$.

Minimum block length and maximum number of changes. Finally, we need to branch to enforce the constraints for minimum block length and maximum number of changes in the original problem. For example, the profile π^n in Table 4 results from adding 2/3 of profile π_1 and 1/3 of profile π_2. Both π_1 and π_2 satisfy a minimum block length of $M = 2$ and a maximal number of changes of $J = 1$, but π^n violates both. Let variable $\zeta_{i\tau}$ indicate a change in the number of workers at time τ. If $\zeta_{i\tau}$ is fractional, we branch by requiring that either $w_{i\tau} = w_{i,\tau-1}$ or $w_{i\tau} < w_{i,\tau-1}$ or $w_{i\tau} > w_{i,\tau-1}$ holds, which requires variable $\zeta_{i\tau}$ to be integral. For the example, requiring $w_{i,3} = w_{i,2}$ excludes profile π_1 and requiring $w_{i,3} > w_{2,\pi}$ excludes profile π_2. Requiring $w_{i,3} < w_{i,2}$ excludes both profiles. Hence, in all cases the current fractional solution is excluded, and the three branches partition the solution space into three sub-spaces. To enforce the three branching rules, all variables $x^{ka}_{ic\pi}$ violating the corresponding branching rule are removed from the RMP, and decisions and states violating the rules are discarded in the subproblems.

4.5 Search strategy

The branching rules presented are sufficient to obtain an optimal integral solution to our problem. However, the search tree becomes very large for difficult problems. Obtaining an optimal solution and proving optimality can take more than 24 hours. As the real-time environment requires a solution to be available within a few minutes, the goal is to generate the best possible feasible solution within the given time.

For that purpose, (i) column generation is warm started so as to improve the dual information in the first iterations and to have useful columns ready for obtaining incumbents; (ii) the branch-and-price search is guided with a *limited discrepancy search* as proposed in Harvey and Ginsberg (1995); and (iii) incumbents are created repeatedly during the search to obtain primal bounds and to have a solution ready in case the time limit for optimization is reached.

Generating schedules. Intelligent initialization of the RMP offers speedup potential (see Vanderbeck 2005), but is a balancing act, since RMP becomes harder to solve as more columns are added. We generate a set of profiles in a preprocessing phase at the beginning of each stage using a recursive algorithm (see Appendix A), which takes into account the following restrictions allowing us to control the number of generated profiles. Parameters $minDur^{\mathrm{L}} \geq M$ and $minDur^{\mathrm{S}} \geq M$ define the minimum duration of the loading process and the storage depletion, respectively. Parameter $LMod$ ("L" for start loading process and "mod" for modulo) restricts the options for the start of the loading process such that it may start at time

$$\tau \in \left\{\tau = S_i, \dots, E_i - minDur^{\mathrm{L}} \mid LMod \text{ divides } (E_i - \tau)\right\}.$$

In the same way, parameter $SMod$ ("S" for start storage depletion) restricts the options for the start of the storage depletion. Parameter $M' \geq M$ sets the minimum block length for the profiles generated to a value at least as large as the actual minimum block length M. Similarly, parameter $J' \leq J$ sets the maximum number of changes to a value at most as large as the actual limit J, and parameter $\bar{G}$ defines the maximum allowed difference in numbers of workers between consecutive periods ("$\bar{G}$" for gap), i.e. $|w_{\pi\tau} - w_{\pi,\tau-1}| \leq \bar{G}$. Finally, we make sure that only non-dominated profiles are added to the RMP. A schedule π dominates a schedule π' ($\pi \succ \pi'$) if

$$w_{\pi\tau} \leq w_{\pi'\tau} \qquad \forall \tau = \bar{S}_i, \dots, E_i - 1, \tag{31}$$

$$\phi^{\mathrm{S},tka}_{i\tau\pi} \leq \phi^{\mathrm{S},tka}_{i\tau\pi'} \qquad \forall \tau = \bar{S}_i, \dots, E_i, \tag{32}$$

$$\phi^{\mathrm{B},tka}_{i\tau\pi} \leq \phi^{\mathrm{B},tka}_{i\tau\pi'} \qquad \forall \tau = 0, \dots, E_i. \tag{33}$$

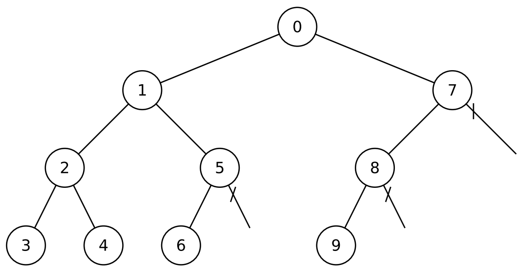


Figure 3: Limited discrepancy search

Limited discrepancy search. Assume that the search strategy is a *depth first search* (DFS) in the branch-and-price tree. DFS sometimes leads directly to an optimal solution. If not, the tree is pruned, and the algorithm backtracks. However, if the search tree is deep and a node without a new incumbent in its subtree was chosen early on, a DFS searches the complete subtree, which may be large, until it moves to the other branch. A limited discrepancy search aims to overcome "wrong turns" at any depth in the search tree by deviating only a few times from the initial search path (Harvey and Ginsberg 1995). This principle is illustrated in Figure 3, where the number of *allowed discrepancies* is one, and the nodes are numbered in the order they are visited. After the search returns from Node 6 to Node 5, it does not visit the branch to its right, as no further right turns are allowed because the number of right turns on the path from the root to Node 5 is equal to the number of allowed discrepancies. Therefore, the search gets back to the root node more quickly, at the cost of possibly missing a solution at the right branch of Node 5.

Solving the RMP as an integer program. Often it is possible to obtain incumbents improving the current primal bound by solving the RMP as an integer program. Especially in the first stages when $A_{i\tau}^{tka}$ are comparably small, we often directly receive an optimal solution for the current stage. Since solving the RMP as an IP may take a considerable amount of time, the attempt is only made at the root and, from there, every 10 nodes of the search tree, if at least 20 new columns have been generated since the previous attempt. This prevents many similar problems from being solved when long cascades of branching decisions without many new columns occur. Additionally, a time limit of 100 seconds is set for the computations in each attempt, and if the time limit is reached before an optimal solution is found, an incumbent, if available, may still improve the current primal bound.

5 Computational Study

In this section, we report the experimental results of our solution procedure. In order to evaluate and compare the performance of the proposed procedure, we developed an event-based simulation, which tracks the state of the relevant entities (flights, carousels, central storage, and baggage handlers), updates the flight information, executes the planning decisions, and simulates the baggage flows. However, since modeling the BHS with its complex conveyor belt network would be too cumbersome for our purposes, the BHS is treated as a black box, and we assume in the simulation that backlogs of bags in the BHS do not negatively influence baggage transport. This must be kept in mind for solutions in which the carousel capacities are exceeded. In these cases, the performance of the BHS could be noticeably reduced, and as a consequence, a higher number of left bags (which we also refer to as unloaded bags in the following) would be expected.

All instances are solved with a period length of five minutes and the limit on the computational time is set to one period. A sufficiently large planning horizon must be chosen because loading may take up to several hours and decisions that are too shortsighted may cause backlogs in the BHS and unloaded bags. We consider a planning horizon of three hours. The loading process can last up to 3 hours, i.e. the earliest start of the loading process is 180 minutes before the end of the BH. In accordance to the average duration at Munich Airport, we set the time between the on-block time and the start of the feed-in of bags into the BHS to 20 minutes, and the time to transport bags through the BHS to 5 minutes. For simplicity, we assume that the on-block time is not re-scheduled within the last 30 minutes and that the end of the BH is not re-scheduled within the last 15 minutes before it ends. Furthermore, we assume that BH ends 15 minutes before the flight's estimated departure time. Thus, the quantity of transfer baggage becomes known 15 minutes before the departure at the airport of origin. The storage capacity is $K^{\mathrm{S}} = 6,000$ bags, and we set the storage depletion rate to $R^{\mathrm{S}} = 19$ bags per period. Furthermore, we consider a loading rate of $R^{\mathrm{L}} = 10$ bags per period.

All experiments were executed on Haswell-based nodes of the Linux cluster of the Supercomputing Centre [blinded for review]. The storage was limited to 24 GiB, and the number of cores was limited to 14. Each core had a nominal frequency of 2.6 GHz. All algorithms were implemented in Java, and all linear and mixed integer programs were solved using Gurobi version 8.1 and its Java

API (Gurobi Optimization 2018).

5.1 Instances and benchmark algorithms

For the baseline setup, the parameters determining the worker flexibility are set to values reflecting current practice at Munich Airport. The minimum block length is set to 30 minutes ($M = 6$), which also means that the duration of the loading process must be at least 30 minutes. The number of changes is restricted to $J = 2$, i.e. apart from the start and end of a flight's loading process, the number of assigned workers can be changed twice. The tests are based on 20 real-world problems from Munich Airport.

To compare the proposed solution procedure with the practice prevalent at most airports, all instances are solved with a constructive algorithm that aims to minimize the left bags and uses a greedy logic mimicking the way a human dispatcher approaches the task. The algorithm is given in Appendix B. We observed that dispatchers take into account the bags to be loaded and the remaining time until the departure in order to prioritize flights. The constructive algorithm (i) employs a metric that tackles various key aspects of the problem, i.e. the urgency in terms of remaining bags (both currents bags in the baggage handling system and the expected bags arriving in the future) as well as the urgency in terms of remaining time until baggage handling ends, (ii) it dynamically updates the metric after each assignment, thereby taking into account previous assignments and remaining capacity, and (iii) it is employed in a rolling horizon fashion, i.e. all assignments are re-evaluated in each period. The computational times of the constructive algorithm is less than a second. However, its performance is mediocre with many resource violations and mishandled bags, while our procedure generates a robust performance without any unloaded bags or resource violations within seconds.

In order to study challenging instances, we increase the resource scarcity. For Munich Airport the resource scarcity is expected to increase since the number of passengers is continuously increasing (the number of annual passengers increased from 12 million in 1992 to more than 48 million in 2019 (Flughafen München GmbH 2019), and since extending the infrastructure is very costly. As our data-driven approach requires historical flight data, we do not increase the passenger volume but rather create resource scarcity by reducing the carousel layouts as well as worker capacities. At Munich Airport there are three types of carousels. There are seven carousels of type

ID	Flights			Bags			Carousels			Worker
	min	avg	max	min	avg	max	type 1	type 2	type 3	(K_τ^{W})
Mon	86	89	92	5,715	6,270	7,012	6	7	1	70
Tue	85	97	105	4,264	5,380	6,929	5	7	1	42
Wed	91	95	98	4,464	5,048	5,527	5	6	1	38
Thu	98	102	107	5,160	5,706	6,525	5	5	1	36
Fri	414	425	436	20,509	21,324	22,138	5	5	1	40

Table 5: Instances

1 (with $K_c^{\mathrm{N}} = 4$ and $K_c^{\mathrm{B}} = 20$), 14 of type 2 (with $K_c^{\mathrm{N}} = 4$ and $K_c^{\mathrm{B}} = 25$), and one of type 3 (with $K_c^{\mathrm{N}} = 6$ and $K_c^{\mathrm{B}} = 40$). For the computational study, we create instances with different degrees of resource scarcity. The carousel configuration for each set of instances is defined by a triplet, where the first, second, and third integers give the numbers of carousels of type 1, 2, and 3, respectively. Furthermore, since identical weekdays have similar air traffic patterns, we group the 20 instances by weekday giving five sets of instances (Monday - Friday). The instance sets are listed in Table 5. The "Flights" and "Bags" columns show the minimum, mean, and maximum number of flights and bags, respectively. The "Carousels" column shows the number of carousels of each type, and the "Worker" column shows the number of workers. In contrast to all other instances, the Friday instances cover not only one peak period but the complete planning day, which generally has three peaks during the day at Munich Airport.

We compare the proposed solution procedure with the constructive algorithm (see Appendix B) and a more sophisticated tabu search algorithm, which is capable to overcome local optima based on a short-term memory (see Appendix C). A tabu search algorithm is quite suitable for our real time environment, since we can run the algorithm until the time limit is reached, and then return the best solution found.

For a given weekday, we randomly select one day to provide the actual data used in the simulation, while the remaining days provide the data for the optimization (we have on average 12 check-in arrival processes for each departing flight, that is, 12 times the same flight number and the same destination on the same weekday departing from Munich Airport). Table 6 shows the results, where each row corresponds to a set of instances for a weekday.

The columns contain performance measures for the constructive algorithm, the tabu search, and

	Constructive algorithm				Tabu search				Main algorithm				
Instances	Unloaded bags		Belt vios		Unloaded bags		Belt vios		Comp. time	Unloaded bags		Belt vios	
	avg	max	avg	max	avg	max	avg	max	avg mins	avg	max	avg	max
Mon	816	989	117	132	27	48	1	3	0.5	0	1	0	1
Tue	426	786	73	103	159	321	190	25	0.4	0	2	0	0
Wed	432	586	52	58	186	258	5	9	0.5	1	10	0	1
Thu	687	927	76	101	332	498	30	61	0.8	75	201	11	36
Fri	1,805	1,992	183	203	1,281	1,465	161	170	1.9	4	5	6	7

Table 6: Results for baseline setup grouped by weekday

our algorithm, and the computation times (average minutes per re-optimization) for our algorithm. The "Unloaded bags" columns report the number of left bags on this day. The extent to which the conveyor belt capacity is violated is measured as follows: A *belt capacity violation* of a carousel occurs if at least one bag remains in the BHS because the carousel's belt capacity is completely used up. The *total belt capacity violations* are the sum of the belt capacity violations over all periods and carousels. This is reported in the "Belt vios" columns. We did not observe any violations of the storage capacity in the simulation. Table 6 shows that the constructive algorithm is outperformed by the tabu search algorithm and the tabu search algorithm is outperformed by our algorithm. The results of the constructive algorithm and the tabu search algorithm are the worst for Friday, which covers the complete planning day with three peaks instead of a single peak which is covered in the other instances. The results of our algorithm are the worst for the Thursday instances, which have the lowest capacity (with only 11 carousels in total and 36 workers). Overall, our approach achieves an optimal performance (i.e. zero belt and capacity violations) in 55% of the 20 instances.

5.2 Evaluation of flexibility

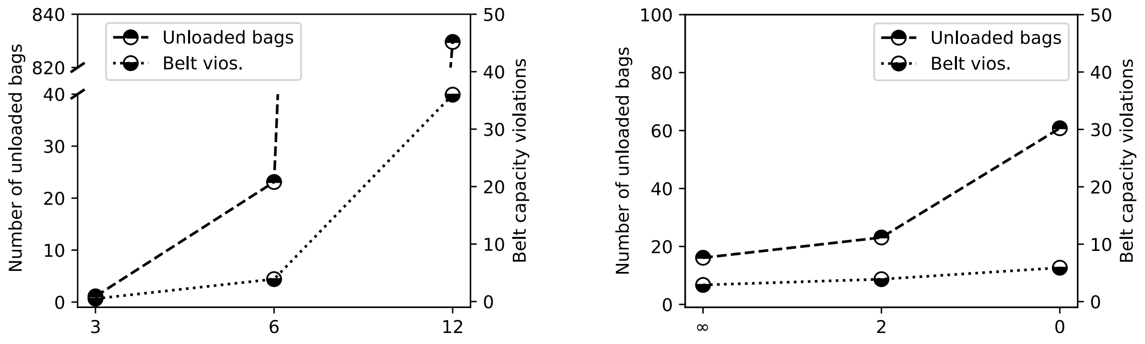
In the following, we elaborate on the value of worker flexibility based on the proposed solution procedure. We consider different degrees of worker flexibility by varying the minimum block length M and the number of times the number of workers is allowed to change during a flight's loading process J. We run experiments with $(M, J) = (M, \infty), (M, 2)$ and $(M, 0)$ for $M = 3, 6, 12$ periods corresponding to $15, 30$ and 60 minutes, respectively. Table 7 shows the aggregated results of all 20 instances, where each row corresponds to different values for the parameters that determine the flexibility, and where the first row is the most flexible with a minimum block length of 15 minutes and an arbitrary number of changes $(M, J) = (3, \infty)$, and the last row is the least flexible with a minimum block length of one hour and a constant number of workers throughout a flight's loading process $(M, J) = (12, 0)$.

Flexibility		Main algorithm				
Minimum Block	Max. no. of	Comp. time	Unloaded bags		Belt vios	
length (M)	changes (J)	avg secs	avg	max	avg	max
3	∞	41	0.2	2	0.4	4
3	2	39	1.1	10	0.5	6
3	0	33	0.6	4	1.3	10
6	∞	39	16.1	163	2.9	24
6	2	42	23.1	201	3.9	36
6	0	33	60.8	405	5.8	42
12	∞	31	818.1	2,177	37.5	97
12	2	34	829.6	1,975	36.0	83
12	0	35	850.8	2,176	45.7	104

Table 7: Computational results for different degrees of flexibility

Flexibility tends to positively reduce both the number of mishandled bags and the capacity violations, and with the highest flexibility (minimum block length of three periods, i.e. 15 minutes, and an arbitrary number of changes) there is less than one mishandled bag and less than one capacity violation on average. Hence, for functioning baggage handling under scarce resources, flexibility is key. Furthermore, we visualize the effect of varying M and J compared to the baseline setup $(6, 2)$. Figure 4a illustrates that the minimum block length has a strong effect on the number of unloaded bags, as well as the belt capacity violations. Having a closer look, we observe that a long minimum block length is particularly problematic for flights with very little baggage, i.e. where a single worker could load all bags within a short time. If the minimum block length is longer than this time, a worker needs to be assigned for at least that long, thereby wasting capacity that otherwise could be used for other flights which explains the strongly reduced performance when the minimum block length is increased from 15 minutes ($M = 3$) to 30 minutes ($M = 6$) or to one hour ($M = 12$). In particular for the latter, the capacity that is left for the remaining flights is insufficient, and as a result our procedure is forced to employ low baggage scenarios.

The effect of the number of allowed changes J is not as extreme as illustrated in Figure 4b. The relatively good performance for $J = 0$ seems surprising at first, but taking a closer look at the solutions, we find that there are two common types of profiles: (i) a long loading process with early storage depletion and one (or few) workers, and (ii) a short loading process with late storage



(a) Varying M given $J = 2$ (b) Varying J given $M = 6$

Figure 4: Performance of the algorithm for different levels of flexibility

depletion and many workers. In the case of (i), bags are sent directly to the carousel over a long period. In the case of (ii), bags are buffered in the storage and then sent to the carousel to be loaded into containers by multiple workers in a short period of time. Hence, changes of the number of workers throughout a flight's loading process do not necessarily occur that often. Still, being able to change the number of workers allows the loading processes to be matched to fluctuating baggage streams more efficiently. For example, the average number of unloaded bags decreases from 61, when only a constant number of workers is allowed for each flight ($J = 0$), to 16, when the number of changes is unrestricted ($J = \infty$), which corresponds to a decrease of almost 75%.

6 Conclusion

In this paper, we presented a new model for dynamic baggage handling with flexible work profiles. A flight's work profile determines how many workers should work over the course of the baggage handling period. Solutions to the model provide the assignments of flights to baggage carousels, the work profile and the start of the baggage loading process as well as the start of the storage depletion for each flight. The objective is to generate plans which allow that bags can be loaded in time, i.e. before the end of a flight's baggage handling process, and avoid backlogs in the baggage handling system. To obtain solutions to the model in as close to real-time as possible, we developed a column-generation-based heuristic. The hierarchic objective allows the problem to be solved in stages. At the beginning of each stage, the restricted master problem is fed a set of columns generated in a preprocessing step. To obtain integral solutions, column generation is embedded into a branch-and-price procedure. Incumbents are obtained by repeatedly solving the master problem as an integer program during the search for an integral solution. Using this combined methodology, we are able to solve real-world instances and generate high-quality solutions even for problems with very scarce resources.

Our approach allows the degree of worker flexibility to be adjusted, and several managerial implications can be taken from our computational study. First, our procedure outperforms a tabu search and a constructive algorithm that mimics the logic human dispatchers apply. Second, the inflow of bags to the carousel can be controlled, to some extent, by scheduling the start of the loading process and storage depletion. However, to match the total loading rate to the flow of incoming bags more effectively, flexible work profiles must be employed. Especially when capacity is scarce, it is crucial to maintain sufficient flexibility by allowing workers to be assigned for short timespans and by allowing frequent changes to the number of assigned workers.

Our approach might be employed to support the dispatcher in making more informed decisions. With respect to future research, several opportunities exist for expanding the scope of the problem, for example by incorporating worker capacity taking into account the shifts, breaks, and movements of individual workers. Furthermore, future research could relax the assumption that the rate at which the bags are fed into the BHS and into ULDs is constant. Finally, with regard to alternative solution methodologies, heuristics with very short computational times may be worthwhile from a practical point of view.

Appendix A Profile Generation

The schedules complying with the rules introduced in Section 3 are generated as follows. For each duration of the loading process d that is allowed according to $LMod$, a set of work profiles is generated. For a given duration of the loading process d, the work profiles are built recursively as shown in Algorithm 1. The recursion progresses through the time periods until the numbers of workers have been set for all d periods. Buffer $(w_0, \dots, w_{d-1})$ stores an incomplete work profile. Each time a work profile is completed, it is added to the pool of results $\mathcal{P}$. Parameter c tracks the

number of changes of workers. At the start of each recursion, if values are set for all d periods, the profile in the buffer is added to $\mathcal{P}$, and the recursion ends. Otherwise, if no more changes are allowed because of parameter J' or further changes would violate the block length M', the remaining values are set to the value of the previous period, and the profile is added to the pool of results $\mathcal{P}$. Otherwise, the algorithm loops over the values allowed according to the maximum gap $\bar{G}$. In each iteration, if the new value is equal to the previous value, the new value is stored in the buffer, and the recursion continues with the next period without incrementing the number of changes c. If the value differs from the previous one, the value is set for the next M' periods and the recursion continues M' periods later with the number of changes incremented by one.
Next, each work profile is combined with each allowed start of the storage depletion, completing the schedule. The storage and load on the carousel are determined with Equations (1) and (2). If both storage and belt are empty at the end of the loading process, the schedule π is a candidate to be added to $\tilde{\mathcal{S}}_{ic}^{tka}$.

Algorithm 1: Profile generation

```
Function generateProfiles
    Input: current period $t$,
           profile buffer $(0, w_0, \ldots, w_{d-1})$,
           changes $c$,
           set of profiles $\mathcal{P}$
    if $t = d$ then $\mathcal{P} \leftarrow \mathcal{P} \cup \{(w_0, \ldots, w_{d-1})\}$, return
    if $c \geq J' \;\vee\; d - t < M'$ then
        while $t < d$ do $w_t \leftarrow w_{t-1}$, $t \leftarrow t + 1$
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{(w_0, \ldots, w_{d-1})\}$, return
    for $w = \max\left\{1, w_{t-1} - \bar{G}\right\}, \ldots, \min\left\{\max_{c \in \mathcal{C}_i} K_c^N, w_{t-1} + \bar{G}\right\}$ do
        if $w = w_{t-1}$ then
            $w_t \leftarrow w$, generateProfiles$(t + 1, c, (w_0, \ldots, w_{d-1}))$
        else
            for $t' = t, \ldots, \min\{t + M', d\} - 1$ do $w_{t'} \leftarrow w_{t-1}$
            generateProfiles$(t' + 1, c + 1, (w_0, \ldots, w_{d-1}))$
```

Appendix B Constructive Algorithm

The constructive algorithm mimics the approach of a human dispatcher that aims to minimize the left bags. At Munich Airport, we observed that human dispatchers allocate the available resources to flights in a way that ensures that “urgent flights” with many bags and limited time receive the resources first. Therefore, we introduce (i) a metric that captures the urgency and (ii) a way to allocate resources depending on that metric, so that both (i) and (ii) reflect the logic of a dispatcher. Note that the algorithm mimics a dispatcher’s logic but not the human precision, i.e. humans do not have the cognitive ability to calculate complex metrics for all flights, update the metrics after each assignment, check feasibility for each assignment, and repeat this procedure every few minutes.

At each decision, the flights that cannot be changed due to the block length restriction or due to the limit on the number of changes receive the same number of workers and thus working stations as previously. The remaining flights are added to a candidate list and sorted in decreasing order by the expected number of workers required per period. We introduce

$$WS_i := \frac{1}{(E_i - t)\, R^{\mathrm{W}}} \left(\left(\phi_t^{\mathrm{S}} + \phi_t^{\mathrm{B}} \right) + \sum_{\tau=t}^{E_i - 1} \bar{A}_{i\tau} \right),$$

where $\phi_t^{\mathrm{S}} + \phi_t^{\mathrm{B}}$ are the bags already in the system (on the carousel and in the storage) at time t, and $\bar{A}_{i\tau}$ denotes the expected number of bags arriving for flight i in period $\tau = t, \ldots, E_i - 1$ (derived from past observations). Note that we do not round WS_i. Next, we perform the following steps iteratively, where WS_i will be decremented each time flight i receives a worker and working station.

Step 0 If the candidate list is empty or if there are no resources left, stop and return the solution.

Step 1 Take the first flight from the list. Let this flight be i^*.

Step 2 If flight i^* is in progress, check whether the number of workers at the assigned carousel can be increased by one. If that is impossible because there are no workers left or because the carousel has no additional working stations, remove the flight from the candidate list, and go to Step 0. Otherwise, increase the number of workers of flight i^* by one and decrease WS_{i^*} by one because the flight has just received a worker. Sort the candidate list in decreasing order by WS_i, and go to Step 0.

Step 3 If flight i^* is not in progress, check whether resources are available to start the loading process with one worker. If not, remove the flight from the candidate list and go to Step 0. Otherwise, start the loading process with one worker at the carousel with the most free working stations and start the storage depletion, decrease WS_{i^*} by one, set the flight to be in progress, sort the candidate list in decreasing order by WS_i, and go to Step 0.

Appendix C Tabu Search

As a more sophisticated benchmark to our approach, we developed a tabu search heuristic (see, e.g., Glover 1997, Kirkpatrick, Gelatt, and Vecchi 1983), which is described in the following. The pseudo code in Algorithm 2 provides an overview of our tabu search. The objective function that we use in our main procedure does not provide enough information to effectively drive the search into promising regions of the search space because solutions that are similar enough to be in the same neighborhood are usually based on the same baggage scenario. Therefore, the search is based on a fixed baggage scenario where all known bags and fixed quantiles of the anticipated bags are included. Objective function (34) penalizes left bags as well as capacity violations of carousel conveyor belts and the storage. The penalty is set to 100 for each left bag, and to 1 for each capacity violation, and we employ the 67%-quantiles for the anticipated baggage. The objective can be stated as

$$f(S) = \sum_{i \in \mathcal{F}^t} 100 \cdot \left(\phi^{\mathrm{S},t}_{iE_i} + \phi^{\mathrm{B},t}_{iE_i} \right) + \sum_{\tau = \bar{S}_i}^{E_i - 1} \left[\phi^{\mathrm{B},t}_{i\tau} - K^{\mathrm{B}}_{c_i} \right]^+ \tag{34}$$

where S denotes the current solution, $\phi^{\mathrm{S},t}_{i\tau}$ / $\phi^{\mathrm{B},t}_{i\tau}$ denote the number of bags of flight i in the storage / on the assigned carousel at time τ, and c_i denotes the carousel flight i is assigned to.

The combinatorial nature of our problem makes designing effective local search moves challenging. For example, it is usually infeasible to swap flights between carousels and keep their current work profiles because the available resources at the respective carousels (working stations and belt capacity) do not match up with the respective profiles. When resources are scarce, swap-type moves are more efficient than shifts-type moves since a shift requires free capacity to work. For our problem, the only effective swap-type move is exchanging working stations between two flights that are assigned to the same carousel. Constraint relaxation is an attractive strategy in such cases. By allowing the search to visit infeasible solutions, it can explore the search space while using simpler neighborhood moves. However, we obtained the best results with a two-phase approach where the search space is restricted to the feasible region in both phases. In the first phase the focus is on finding good carousel assignments, and in the second phase the work profiles are fine-tuned with given carousel assignments. The total time available for computing a solution is set to five minutes, i.e. we set the identical time limit as for our main procedure. Furthermore, it is split half between the two phases.

Phase 1. The search space in Phase 1 is restricted to the assignments of flights to carousels. In order to calculate the objective function values for these partial solutions, they need to be completed. For that purpose, we developed a variant of the constructive procedure in Appendix B that builds the work profiles and sets the storage depletion times for given flight-carousel-assignments. The initial solution for the tabu search is created by randomly assigning all flights to carousels where the loading process has not yet started. For the remaining flights the carousel assignments are already determined since their loading processes are in progress. The tabu search then iteratively moves from solution to solution using two different neighborhood structures:

1. a *carousel-shift* moves a flight from its current to another carousel, and
2. a *carousel-swap* moves flight i_1 from its current carousel c_1 to another carousel c_2, while at the same time moving flight i_2 from its current carousel c_2 to carousel c_1.

Only feasible moves are included in both neighborhood structures. A coin-flip determines the move type in each iteration. For the selected move type, we evaluate the complete neighborhood and choose the best non-tabu move. However, tabus are ignored when a move improves the best known solution. At the end of each iteration, the best known solution is updated if the current solution is better, and the tabu list is updated. The tabu list stores each move for a fixed tabu tenure of 25 iterations, and the search cannot reverse any of the transformations present in the tabu list, except when the move would lead to a new best known solution.

Phase 2. The search space in Phase 2 is restricted to changing the work profiles while the carousel assignments are fixed. The two different neighborhood structures are:

1. a *storage-shift* shifts a flight's start of storage depletion by one period forward or backwards in time, and
2. a *working-station-swap* swaps a block of working stations between pairs of flights that are assigned to the same carousel.

Again, the move type is determined with a coin-flip, the tabu tenure is 25 periods, and the tabu list prevents reversing the recent transformations.

Algorithm 2: Tabu search

```
Function tabuSearch
    Input: problem data and initial conditions $D$,
           time limit $timelimit$
    Output: best solution found, $S^*$
    $start \leftarrow now\,()$
    $T \leftarrow \emptyset$                                   // initialize tabu list
    $c \leftarrow generateInitialAssignments\,(D)$        // random, if LP is not in progress
    $S \leftarrow greedy\,(c)$                                // complete initial solution
    $S^* \leftarrow S$
    /* Phase 1                                                                    */
    while $now\,() - start < \frac{timelimit}{2}$ do
        if $random\,() < 0.5$ then
            $S' \leftarrow \arg\min_{s \in N^{\text{shift carousel}}(S)} f\,(s)$  // select best non-tabu carousel-shift
        else
            $S' \leftarrow \arg\min_{s \in N^{\text{swap carousels}}(S)} f\,(s)$  // select best non-tabu carousel-swap
        $T \leftarrow updateTabuList\,(S, S')$; $S \leftarrow S'$
        if $f\,(S) \leq f\,(S^*)$ then $S^* \leftarrow S$
    /* Phase 2                                                                    */
    while $now\,() - start < timelimit$ do
        if $random\,() < 0.5$ then
            $S' \leftarrow \arg\min_{s \in N^{\text{shift storage depletion}}(S)} f\,(s)$   // select best non-tabu SD-shift
        else
            $S' \leftarrow \arg\min_{s \in N^{\text{swap working stations}}(S)} f\,(s)$    // select best non-tabu WS-swap
        $T \leftarrow updateTabuList\,(S, S')$; $S \leftarrow S'$
        if $f\,(S) \leq f\,(S^*)$ then $S^* \leftarrow S$
    return $S^*$
```

References

Abdelghany A, Abdelghany K, Narasimhan R (2006) *Scheduling baggage-handling facilities in congested airports. Journal of Air Transport Management* 12(2):76 – 81.

Ascó A (2016) *An analysis of robustness approaches for the airport baggage sorting station assignment problem. Journal of Optimization* 2016:1–19.

Ascó A, Atkin JAD, Burke EK (2012) *An evolutionary algorithm for the over-constrained airport baggage sorting station assignment problem.* Bui LT, Ong YS, Hoai NX, Ishibuchi H, Suganthan PN, eds., *Simulated Evolution and Learning*, 32–41 (Berlin: Springer), ISBN 978-3-642-34859-4.

Ascó A, Atkin JAD, Burke EK (2014) *An analysis of constructive algorithms for the airport baggage sorting station assignment problem. Journal of Scheduling* 17(6):601–619.

Barbot C (2012) *Opening ground handling markets to competition: Effects on welfare. Transportation Science* 46(4):536–546.

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.

Ben-Tal A, Ghaoui L, Nemirovski A (2009) *Robust Optimization.* Princeton Series in Applied Mathematics (Princeton University Press), ISBN 9781400831050.

Bertsimas D, Sim M (2004) *The price of robustness. Operations Research* 52(1):35 – 53.

Boysen N, Briskorn D, Fedtke S, Schmickerath M (2019) *Automated sortation conveyors: A survey from an operational research perspective. European Journal of Operational Research* 276(3):796–815.

Bureau of Transportation Statistics (BTS) (2018) *On-Time Performance - Flight Delays at a Glance.* URL `https://www.transtats.bts.gov/HomeDrillChart.asp`.

Chvátal V (1983) *Linear Programming* (W. H. Freeman and Company, New York), ISBN 9780716715870.

Flughafen München GmbH (2019) *Statistischer jahresbericht 2019.* URL `https://www.munich-airport.de/verkehrszahlen-88506`.

Frey M, Kiermaier F, Kolisch R (2017) *Optimizing inbound baggage handling at airports. Transportation Science* 51(4):1210–1225.

Frey M, Kolisch R, Artigues C (2017) *Column generation for outbound baggage handling at airports. Transportation Science* 51(4):1226–1241.

Fündeling CU, Trautmann N (2010) *A priority-rule method for project scheduling with work-content constraints. European Journal of Operational Research* 203(3):568 – 574.

Glover F (1997) *Tabu search and adaptive memory programming — advances, applications and challenges.*

Barr R, Helgason R, Kennington J, eds., *Interfaces in Computer Science and Operations Research*, volume 7 of *Operations Research/Computer Science Interfaces Series*, 1–75 (Springer US), ISBN 978-1-4613-6837-3.

Gurobi Optimization L (2018) *Gurobi optimizer reference manual.* URL `http://www.gurobi.com`.

Harvey WD, Ginsberg ML (1995) *Limited discrepancy search. Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, 607–613, IJCAI'95 (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.), ISBN 1-55860-363-8, 978-1-558-60363-9.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) *Optimization by simulated annealing. Science* 220:671–680.

Lübbecke ME, Desrosiers J (2005) *Selected topics in column generation. Operations Research* 53(6):1007–1023.

SITA (2018) *Baggage report 2018.*

Stolletz R (2011) *Analysis of passenger queues at airport terminals. Research in Transportation Business & Management* 1(1):144 – 149.

Vanderbeck F (2000) *On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research* 48(1):111–128.

Vanderbeck F (2005) *Implementing Mixed Integer Column Generation*, 331–358 (Boston, MA: Springer US), ISBN 978-0-387-25486-9.