Synchronizing transportation of people with reduced mobility through airport terminals

René van Twist, Marjan van den Akker, Han Hoogeveen

Department of Information and Computing Sciences

Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands

Rene@vanTwist.net, J.M.vandenAkker@uu.nl, J.A.Hoogeveen@uu.nl

April 11, 2017

Abstract

Navigating through an airport is easy enough for most passengers, but when you are reduced in mobility it is a different story. In this paper we look at an airport at which between 300 and 500 Passengers with Reduced Mobility (PRMs) arrive daily, who need assistance from airport employees in their journey. We want to find a schedule for these employees to support as many PRMs as possible while ensuring a smooth journey with little waiting time. PRMs may not be left unsupervised, except in the lounge. Since the employees are only allowed to work within their own terminal, the tasks of the employees must be synchronized to enable a smooth hand-over of a PRM. Moreover, we want to find a robust schedule to be resistant against minor disturbances. Since we must be able to reschedule in case of a major disturbance, the maximum computation time of our algorithm is restricted to two minutes. We present a decomposition model in which we first determine feasible start times for the tasks describing the journeys of the passengers using Simulated Annealing, after which in each iteration we assign the tasks to the employees in the next phase using a matching algorithm or heuristic. Experimental results show that our algorithm is able to ensure smooth, robust connections, while supporting every passenger in the given instances. Finally, we present a simulation study to test our approach in a dynamic environment. It turns out that it can easily deal with these disturbances in real time and come up with very good solutions.

Keywords. Passengers with Reduced Mobility; airport planning; synchronization; decomposition; local search; simulated annealing; simulation.

1 Introduction

Imagine a major airport where a lot of passengers go through daily. Consider for example a typical journey of a transfer passenger. After arriving at one terminal the passenger travels to the terminal where the connecting flight departs from; there he [1] waits for the boarding process to start and leaves the airport by plane. This process is simple for most passengers, but when you are e.g. restricted in movement or visually impaired, this is not so simple. That is why the airport offers a service to assist such passengers, which are called Passengers with Reduced Mobility (PRMs), with their journey through the airport and makes sure they are supervised at all times. Consequently, there always must be an employee guiding the PRM unless the PRM gets seated in a special, supervised lounge. Since an employee is only allowed to work in his own terminal, the PRM must be handed over as soon as he moves to another terminal. Moreover, since the terminals are not connected with each other, PRMs are required to take a special bus to go to another terminal. Hence, the PRMs are handed over at the bus station, and upon arrival the PRMs are handed over to another employee, who assists them further. It is also possible that the departure flight of the passenger is not directly located at the gate and hence the passenger needs transport by a special platform bus for PRMs to get to the aircraft.

In this paper we consider the problem of scheduling the PRMs and the employees supervising them such that as many PRMs as possible are served, and such that the served PRMs get the best possible service. Due to the increase of mobility of elder people, this problem will become more important during the next decades, and several studies have been conducted to find out how the process of transporting PRMs can be improved (see for example Chang and Chen [4] and Arcidiacono et al. [1]). The problem situation that we sketched above was first considered by Reinhardt et al. [12], who very kindly presented the anonymized data of the airport layout and the problem instances.

The main objective of the company (or department of the airport) responsible for the support of PRMs, is to give the best service possible. Unfortunately there can be situations in which the company has to decline one or more PRMs, who must then take a later flight. In thes situation the company prefers to decline immediate PRMs above prebooked PRMs, where a PRM is considered as a prebooked PRM if he has required assistance by notifying the company of his journey at least a day in advance.

Next to that the company wants to improve service further by providing smooth connections to the PRMs. This is achieved by minimizing unnecessary travel time, which is the extra

[1]Whereever we use ‘he’ to indicate a passenger or employee, it should be read as ‘he/she’

time that the PRM spends on moving through the airport. A PRM gets unnecessary travel time if an employee assisting a PRM takes a detour or has to wait for some other employee to pick up the PRM for the next part of his journey. The time that a PRM is seated in a lounge does not contribute to the unnecessary travel time.

The algorithm of Reinhardt et al. [12] is based on a greedy insertion heuristic. The PRMs are put in two ordered lists: one containing the prebooked and one containing the immediate PRMs. The greedy insertion heuristic adds PRMs one by one into the schedule in the order of the lists, where first all prebooked PRMs are inserted. When the greedy insertion heuristic plans a PRM, it inserts all the PRM's segments beginning with the earliest one. The insertion of a segment is done by investigating the employees and buses that could serve that segment, where the algorithm is only allowed to push already inserted segments forward in the schedule as result of the insertion. The feasible insertion that causes the least increase of the objective function is used for the insertion. If the heuristic fails to find a feasible spot for the segment the PRM is not included in the schedule and all his segments are removed from the schedule.

Simulated Annealing is used to mutate the order of the PRMs in the two lists. Initially, both lists are sorted in ascending order by the time the PRM shows up at the airport. There are two possible mutations. The first one is to take a not yet assigned PRM and move him a random number of places forward in the list. The second one is to swap two random PRMs from the same list. Both mutations only change the order of the PRMs in the list; the lists are not mixed.

Computational experiments show that their algorithm can find good solutions in two minutes and high quality ones in 10 minutes. Very few PRMs are rejected by the greedy insertion heuristic, but there is still some unnecessary waiting time. Due to the low running time of two minutes the algorithm can be run multiple times a day in a dynamic real world situation. In their conclusions the authors suggest to investigate different solution methods; we will present one such method in the remainder of the paper.

Like [12] we minimize the number of declined passengers. We have analyzed the problem to find out what makes the planning problem difficult and what can be done to overcome this. The key feature of our algorithm is *synchronization*. We synchronize subsequent segments of a PRM to remove *all unnecessary waiting time*, where a *segment* is a part of the PRM's journey that can only be served by one employee. The solutions of [12] do contain such unnecessary waiting time. Moreover, we synchronize segments of different PRM's to allow them to share a bus trip and share an employee. This reduces the number of tasks and in this way increases the number of PRMs that can be assisted.

In addition to [12] we also look for a *robust* schedule to achieve that real-world disruptions can be dealt with more easily and connections run smoother. A delay in one segment of the journey of the PRM could delay the remainder of the journey as well. If an employee serves a delayed segment, the following segments he must serve might also be delayed, causing a cascade of delays. Therefore we want to make those connections robust such that delays will affect the schedule as little as possible.

We apply a *decomposition* approach. Since synchronization is crucial, our decomposition is different from the most obvious one. In the first phase we decide on the starting times of segments and on sharing employees and/or buses, i.e., on synchronization, and on declining passengers. In the second phase, try to assign an employee to each segment such that the solution becomes as robust as possible.

Observe that our problem is a three-criteria problem. In our solution approach for the deterministic problem we only consider solutions without unnecessary travel time; in our computational experiments, we show that this does not reduce the quality of the solution in terms of declined passengers. Since the main goal of the company is having as few declined PRMs as possible, the resulting bi-criteria problem is in fact only a lexicographical minimization problem: we look for a solution with an optimum score on the declined passengers, and within this set of solutions, we look for a solution that is as robust as possible.

The paper is organized as follows. In Section 2, we describe the problem and define our robustness function. In Section 3, we study the main feature of our approach: synchronization. In Section 4, we present our decomposition algorithm, and in Section 5 we present our computational experiments. In Section 6 we describe our simulation experiment to test our approach in real-life. Finally, we draw conclusions in Section 7.

2 Problem description

2.1 The airport

At the airport in question there are about 120 employees, who guide between 300 and 500 PRMs per day. The airport consists of 11 disjoint terminals. Each terminal has its own staff, and an employee can only serve the PRMs inside the terminal he is assigned to. Traveling between the terminals takes place by bus. The inter-terminal buses are the only way we can transport a PRM from one terminal to another. Each terminal has one bus stop. The buses used to transport PRMs are specially reserved for them and are not to be used by passengers who do not need assistance. All inter-terminal buses used to transport PRMs have the same capacity and travel times; hence, these are considered to be identical. The capacity depends on the disability of the PRM: transporting a person in a wheelchair requires more capacity than a person who just needs a standard bus seat.

Some aircraft are not located at a gate with a passenger boarding bridge, but are at a remote stand. Passengers are then transported from the remote stand at the platform to the corresponding terminal and vice-versa by platform buses; each terminal has one bus site for platform buses. Although every passenger on that aircraft needs to travel from the boarding gate to the aircraft by platform bus, PRMs are brought using special buses. Again, these buses have a given capacity, which is the same for all platform buses, where the capacity depends on the disability of the PRM. Platform buses are not used for inter-terminal travel, and inter-terminal buses are not used for traveling between gates and aircraft.

Within a terminal each staff member is able to serve one or two PRMs at the same time depending on the disability of the PRM; for instance an employee can only transport one wheelchair but can hold two PRMs who can walk. A PRM must be supervised at all times, unless he is in a lounge. A lounge is a special, supervised location in a terminal, where the PRM can wait comfortably for the continuation of the journey; every terminal has one lounge. Since an employee is not allowed to work outside his terminal, the PRM must be transferred to a bus and then to another employee whenever a PRM moves to another terminal.

2.2 The journeys of the PRMs

For each PRM we know the places where he enters and leaves the airport, respectively. Since we remove all unnecessary waiting time, the schedule will not contain any detours and hence the locations at which the PRM has to be transferred follow immediately. The only flexibility left is whether or not the journey will be interrupted in a lounge; when there is not enough time between the arrival of the PRM at the airport and deadline for boarding to visit a lounge, we skip the lounge visit. In Figures 1-3 we have depicted a schematic overview of an arrival, departure, and transfer journey, respectively. Locations ending with (S) act as a possible start location and locations ending with (E) act as a possible end location.

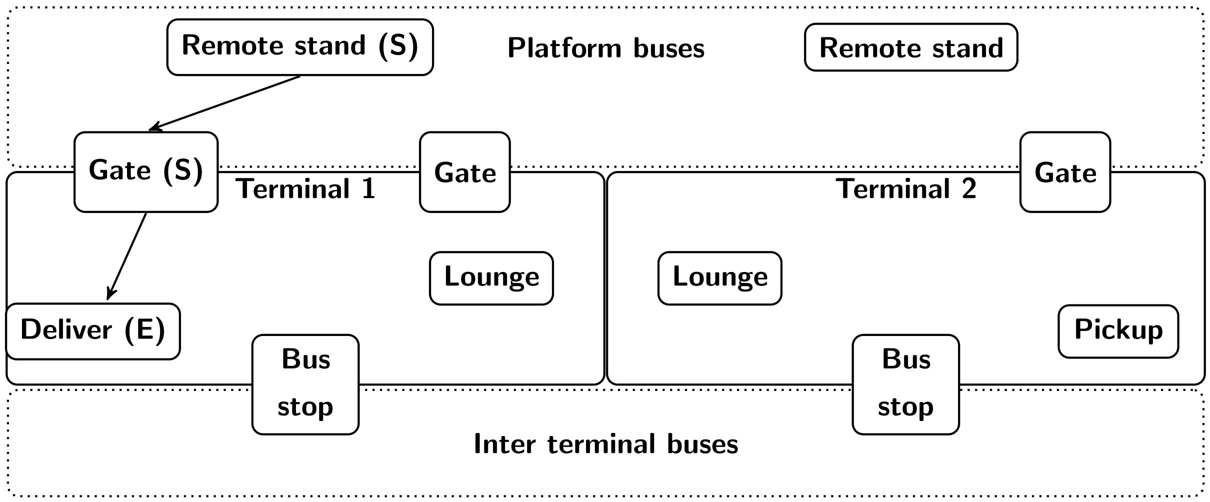

Figure 1: This figure shows an example for the arrival journeys a PRM could take.

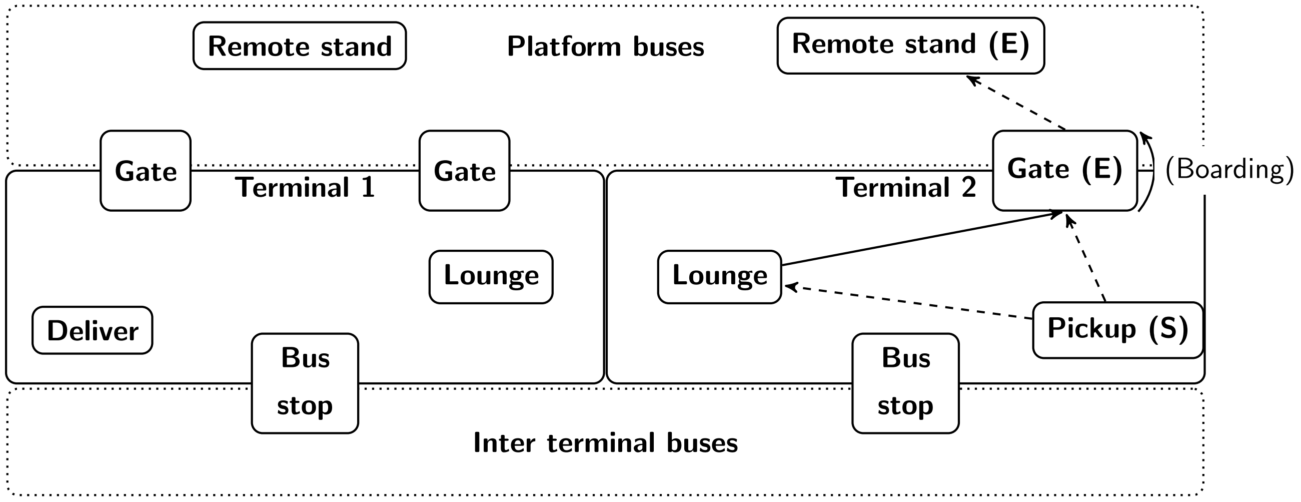

Figure 2: This figure shows an example for the departure journeys a PRM could take. We show two terminals, the inter-terminal bus site and the platform bus site. The other nodes represents locations and the edges represent segments. Dashed edges are either optional segments or choices between two routes.

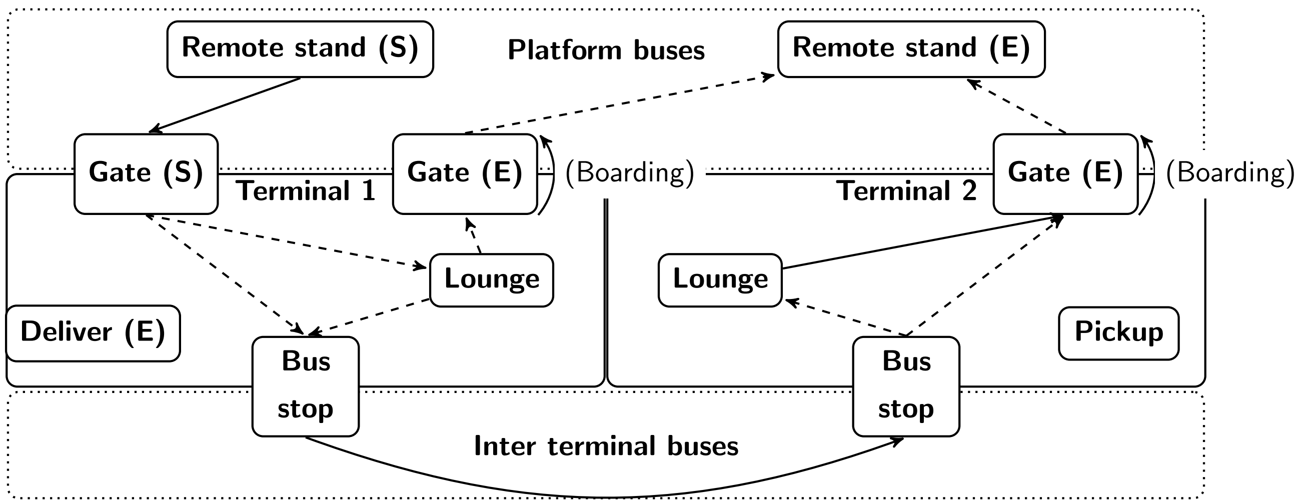

Figure 3: This figure shows examples of all possible transfer journeys a PRM could take, assuming that the arrival and departure terminal are different. Dashed edges are either optional segments or choices between two routes.

If we see the journey of the PRM as a path in a graph, then each edge in the journey of a PRM must be covered by one employee for a PRM to be fully assisted with his journey through the airport. Observet that an edge of the PRM's journey that can be served by an employee corresponds to a *segment*. For example a route visiting locations (l_1, l_2, l_3, l_4) then consists of segments (l_1, l_2), (l_2, l_3) and (l_3, l_4). The time needed to serve a segment is equal to the travel time between the start and end location, which is known. A special segment is the one that represents the boarding process. To catch a departing flight, the PRM needs to be at the gate when the boarding begins. We assume that the employee assisting the PRM through the boarding process stays with the PRM until the gate closes, which is 20 minutes after the boarding has begun. We assume that the employee who brings the PRM from the lounge to the gate will also assist the PRM with boarding. The boarding is represented as a special segment in the journey both starting and ending at the gate, which takes 20 minutes to complete and cannot start earlier than the gate opens. If the aircraft is at a remote stand, the boarding consists of tranferring the PRM to the platform bus. Upon arrival at the aircraft, the bus driver will assist the PRM in boarding the aircraft.

2.3 Robustness

In a dynamic environment many unexpected events can take place, which will disrupt the original schedule. For example a flight can be delayed, gates can be switched, or a new PRM appears and requests to be scheduled. Moreover, since we are working with PRMs, something simple like a delay due to walking slower than expected or having difficulty getting into a bus will occur frequently. A delay can have a big effect on the schedule. If a segment of a PRM is delayed it can delay the subsequent segments of both the PRM and the employee, who in turn can delay other employees and PRMs, thereby causing a cascade of delays if no slack has been built into the schedule. We hope that by including slack we can make the schedule more robust, such that small delays in the real execution of the planning will cause fewer passengers to be delayed. To encourage robust solutions we add a robustness score as an extra objective.

We define the robustness score by specifying a function based on the slack time. Since we minimize the number of declined PRMs and unnecessary waiting time, we redefine the problem of maximizing the robustness as a minimization problem. Thereto we introduce a robustness penalty, where a robustness penalty of 0 is the best achievable score of a connection, and the higher the robustness penalty the worse the connection.

There are several possible choices for the penalty function. Bolat [2] uses a square function to minimize the variance, whereas Diepen et al. [7] use a penalty function based on the arc tangent, since this heavily penalizes small slack time values and then flattens. In any case, the function should not be linear, since we do not want a long slack time and a very short slack time to yield the same penalty as two average slack times. We assume that 20 minutes is enough to be robust; a slack of size of 20 minutes or more will yield a penalty of size 0. Furthermore, as mentioned before we favor that two consecutive segments of one PRM are served by the same employee, and therefore, we use a penalty of 0 as well in this case. We have decided to use the simple square function $rp(s)$ below for calculating the robustness penalty between two subsequent segments where s is the slack time between those:

$$rp(s) = \begin{cases} 0 & \text{if efficient next} \\ (20 - \min\{s, 20\})^2 & \text{otherwise} \end{cases}$$

A small advantage of using this function is that the objective value will be integer, since the data in the problem instances are rounded to minutes.

3 Synchronization

3.1 Synchronizing segments to avoid unnecessary waiting time

The company that transports the PRMs through the airport wants to make it as comfortable as possible for the PRMs. To avoid unnecessary waiting time we do not allow any detours. Important part of this is that the transfers go as swiftly as possible; hence, we want to minimize the total waiting time and the total unnecessary travel time of the PRMs. Since waiting in a lounge is considered to be comfortable for the PRM, and since the employee who brings the PRM from the lounge to the gate will also assist the PRM with boarding, the only handovers that may result in unnecessary waiting concern the transferal of a PRM from a (interterminal or platform) bus to an employee and vice versa. If we synchronize the corresponding segments, then we can eliminate all unnecessary waiting time for the deterministic case, in which all arrivals, departures, driving times, and walking times are known. The only thing we have to do is to ensure that the employee serving the next segment is already on the spot when the PRM arrives; if the PRM would arrive before the next employee, then we can avoid that by making the first segment start later. Unfortunately we cannot shift the time of platform buses because aircraft depart and arrive at certain times, and therefore PRMs need to be transported by those buses at fixed times. But we can shift the inter-terminal bus rides a bit if we allow a lounge visit before and after the bus ride. For example the PRM arrives by aircraft in terminal 1 and is seated in the lounge; when the bus is about to arrive the PRM is picked up from the lounge and brought to the bus stop where the bus awaits the PRM. The bus then brings the PRM to the terminal of his departing flight, where an employee awaits him to bring the PRM to the lounge. If there isn't an employee or bus available this journey can be shifted a bit in time to provide the PRM with a smooth connection. Obviously, this is not possible for the first segment, which must start when the PRM arrives at the airport, and for the last segment of a PRM departing by plane, which must start such that he arrives at the right time for boarding.

From now on, we will split the route of a PRM into *segment groups* by grouping the segments that must be executed contiguously. For a typical transit passenger, for example, the route is split in three segment groups. The first segment group concerns traveling from the arriving aircraft to the lounge in the arrival terminal. The second segment group corresponds to the travel from the lounge of the arrival terminal to the lounge in the departure terminal, and the third segment group concerns the travel from the lounge in the departure terminal to boarding the aircraft. Since the start time of the first and last segment group follow from the arrival and departure times of the flights, we can only shift the start time of the second segment group, but within the bounds dictated by the first and last segment group; we will denote this as the *window* of the segment group. Furthermore, we have to assign an employee to each segment.

We can make segment groups for the departure journey and other variants of the transfer journey in a similar way. An arrival journey does not have a lounge visit and therefore consists of only one segment group.

We use these segment groups in our algorithm to avoid unnecessary waiting time; we see to it these segments are planned contiguously. Moreover, working with these segment groups also helps with synchronizing journeys when two or more PRMs share the same employee or bus, such that they both arrive at the start of their common segment at the same time. We do not have to synchronize the whole journey but only have to look at the segment group containing the segment we want them to join.

3.2 Synchronization by sharing bus trips or employees

The buses can transport multiple PRMs at the same time, as long as their capacity allows it. According to the problem instances that we got, the service provider uses between 7 and 13 buses in the inter-terminal bus area, and each bus has a capacity of 12. When computing the capacity necessary for transporting the PRMs, a PRM in a wheelchair counts for 1.5; the other PRMs count for 1. Hence, we can transport at least 8 PRMs at the same time. Despite this big capacity, personal communication with Reinhardt et al. revealed that there might be a bottleneck in the inter-terminal bus area. Therefore, it seems essential to plan the buses efficiently, such that in each bus trip multiple PRMs are transported. Since we want to avoid that PRMs have to wait at the bus station, we must synchronize the arrival times of the PRMs. Moreover, even though it is possible to have PRMs in the bus that need to go to different destinations, this implies that some of the PRMs have to wait because of the detour, which is not allowed since we assume the unnecessary waiting time to be 0. Hence, we aim at a solution in which all PRMs in the bus always have the same destination.

In a similar way, we can synchronize segments of different PRMs such that these are served by one employee. If two PRMs need to catch the same flight and they have a light disability such that an employee can help both of them at the same time, then they can be picked up together at the lounge before departure and be brought through the boarding process with only one employee instead of two. In this way, we can save employees for helping other PRMs.

We allow segments r_1 and r_2 of two different PRMs to be synchronized when:

- Both segments r_1 and r_2 have the same start and end location.
- An employee situated in the terminal or a bus in the bus area of r_1 is able to service both PRMs of segments r_1 and r_2 at the same time without violating the capacity constraint.
- There exists a start time that can be set for both segments r_1 and r_2 such that neither journey ends up being infeasible.

We hope that synchronizing the segments as much as possible will relieve the stress on the employees and will enable us to be able to plan PRMs more efficiently. It will at least free up some employee while boarding PRMs on the same flight and will make better use of the bus capacity. We assume that if multiple PRMs get picked up by the same bus then the destination terminal has enough staff available to bring them towards their next stop in their journey. If there are not enough employees in a terminal to support all PRMs at the same time, the synchronization will lead to an infeasible solution, and we reverse it.

4 Decomposition approach: local search and matching

Since we must produce a solution in two minutes, we apply a decomposition approach. In the first stage, we determine for each one of the accepted PRMs a start time for each segment group and decide on sharing buses and employees. In the second stage, we try to assign an employee to each segment, such that the robustness penalty is minimized. Then we apply local search to change the solution of the first stage, etc.

After having conducted some initial experiments, it turned out that it was essential to actually solve the matching problem of the second stage *each* time that we made an adjustment in the first stage solution. In this way, we can use the knowledge gained when solving the second stage to find adjustments to the solution of the first stage, especially when we cannot find a feasible solution for the start times set in the first stage.

We first describe the local search approach for the first stage, and after that we discuss the matching algorithm that we use to find a solution for the subproblem of the second stage.

4.1 Local Search

In the first stage, we try to set the start times for each PRM by mutation. We work with the segment groups as described above. Furthermore, we try to free employee or bus capacity by synchronizing segment groups (g_1, g_2) of different PRMs who can share an employee or bus on at least one of the segments in g_1 and g_2. This implies that we have to synchronize their journeys such that both PRMs arrive at the location at the same time to continue their journey together with the same employee or bus.

Before starting the local search, we construct a set M of possible pairs of segments that could share an employee together. If r_1 and r_2 satisfy the properties of Section 3.2 , then we put the segment pair (r_1, r_2) in the set M. We further check if other segments within these merged segment groups can share an employee or bus, too. If in our algorithm we choose to use the segment pair (r_1, r_2), then we synchronize all segments in the corresponding segment groups relative to the merged segment of the pair (r_1, r_2), and we update the windows of these merged segment groups accordingly.

As local search we apply Simulated Annealing and use the following mutations. More details on the implementation will be presented in Section 5.1

Name	**Description**
Plan PRM	Attempt to plan a random, declined PRM into the planning at random start times inside its time window for each segment group.
Decline PRM	Decline a random, served PRM and remove it from the planning.
Move Segment Group	Attempt to move a random, planned segment group with a random start time inside its time window. In case the segment group is merged move the merged group.
Move Segment Group + Split Merged Group	Attempt to move a random planned segment group with a random start time inside its time window. In case the target segment group is merged with another segment group, remove the target segment group from the merged group before moving.
Merge Segment Group	Take a random possible match from M and attempt to merge them. If one of the involved PRMs is unplanned it tries to add the PRM into the schedule; otherwise it tries to schedule the merged segment group on a feasible time.

Table 1: Mutations applied in the local search

4.2 Generating the Schedule

After each mutation a schedule is generated for all currently accepted PRMs. In this subproblem the start times of all planned segments are already set such that we only have to assign which segment is served by which employee. Since we work with segment groups, unnecessary waiting time does not play a role in the objective. The goal of the algorithm is to find a robust schedule with all planned segments assigned to an employee. If the algorithm fails to find such a schedule, then the mutation is considered infeasible and the changes are reversed.

In our sample data all employees and vehicles are homogeneous in terms of capacity, start time of their shift and end time of their shift; hence, we initiallly designed our algorithms around that. In the problem description, however, it is not mentioned that all employees should have the same start and end of the shift. At an airport flights arrive and leave early in the morning and late at night, and so it is likely that employees work in shifts. Therefore, we also will consider extensions where it is possible to let employees have different shifts; we call this extension the *shift variant*.

The subproblem with identical employees and buses is equivalent to the single depot vehicle scheduling problem [9], which can be solved in polynomial time as an assignment problem. The problem where employees work in different shifts, however, represents a multi-depot vehicle scheduling problem [6], which is $\mathcal{NP}$-hard. We first focus on the matching methods for the variant without shifts, and then consider the shift variant.

4.2.1 Solving the problem without shifts as an assignment problem

Since we know for each segment whether it is served by a bus or an employee, we split the subproblem into two parts: one for planning the buses and one for planning the employees. Since these problems can be solved in the same way, we only discuss the employee scheduling problem below. Remark that in case of merged segments, we only include one of these segments in our approach.

Our approach is to transform the subproblem into a weighted maximum matching problem in a bipartite graph, just like happens in [9].

We construct a directed graph $G = (V, A)$ as follows. For each segment i, we introduce a vertex n_i. We further add two dummy vertices s and t. We include an arc (n_i, n_j) if an employee can serve segment j directly after segment i. Finally, we add arcs (s, n_i) and (n_i, t) for each segment i. A path from s to t in the graph then corresponds to the set of segments that must be served by one employee or bus. If we look at a feasible schedule, then we see that each vertex other than s and t must have exactly one predecessor and one successor; hence, if we assign a predecessor and successor to each vertex n_i, then we can construct the solution. We can construct the bipartite graph from the current graph by replacing the vertex

n_i with two vertices n_i^s and n_i^t together with an arc (n_i^s, n_i^t). Each arc (n_i, n_j) is then replaced by an arc (n_i^t, n_j^s), and the arcs (s, n_i) and (n_i, t) are replaced by the arcs (s, n_i^s) and (n_i^t, t) for each segment i. We thus find a bipartite graph, where s and all vertices n_i^t form one color class and t and all vertices n_i^s the other one. We remove the arcs (n_i^s, n_i^t) for all segments i; we give all arcs (n_i^t, n_j^s) a weight equal to the value of the robustness penalty for the slack between segments i and j.

Each perfect matching in the bipartite graph corresponds to a feasible solution for the subproblem in which each segment is covered by an employee; the perfect matching with minimum weight corresponds to the most robust feasible solution. This problem can hence be solved as an assignment problem in a bipartite graph [9], where we bound the number of times that s can be used as predecessor by the number of employees. It can be solved using the Hungarian Algorithm in $O(|V|^3)$ time [3], where $|V|$ is the number of vertices. If we were only interested in feasibility instead of maximizing robustness, we could have transformed it into the maximum matching problem which is solved in $O(|V||A|)$ time [5], where $|A|$ is the number of arcs.

Remark here that this method is not fit for the problem where employees could have different shift starts and ends. We can split s into a source vertex for each separate employee, and we can split t in a similar way, but we cannot guarantee that the path starting in the source node for employee i will end in the sink node for employee i. After all, the problem is $\mathcal{NP}$-hard in the strong sense. We discuss the problem with shifts in a Section 4.2.3.

The disadvantage of solving the subproblem to optimality in each iteration is its relatively large running time. Since we only have two minutes, we cannot execute enough iterations to find a very good solution. Therefore, we will now discuss two heuristics to update the solution of the matching algorithm.

4.2.2 Two heuristics for updating the matching

Our start point is that we have found some solution. We then apply a mutation to the start times of the segments in the local search part; in the mutation also some new segments may arise by merging or splitting of groups or adding PRM's. From now on, we will refer to these modified and new segments as new segments.

In our first heuristic, we simply try to insert the new segments one by one in the current schedule of the employees such that the increase in robustness penalty is minimum; we call this heuristic the *Free Spot Heuristic*. This did not lead to good solutions for the deterministic problem instances, and therefore we do not report this in the computational experiments. Since it leaves the current schedules intact, this heuristic is useful for computing small updates of the schedule, and therefore we use it in the simulation when we want a simple reschedule; more on this in Section 6.2. As the Free Spot Heuristic is not able to plan all segments, we now consider replanning a part of the schedule; we call this the *Reschedule Overlapping Segments Heuristic*. It inserts the new segments one by one, but allows making changes in the current assignment.

Suppose that we have to insert a new segment r. In our replanning we only consider rescheduling the *overlapping* segments, which are the segments that cannot be planned either before or after segment r without violating the constraints. The problem is then transformed to a weighted maximum matching problem like in Section 4.2.1. This method will make three sets of segments: R_{source}, R_{sink}, and $R_{overlap}$. See Figure 4 for an illustration.

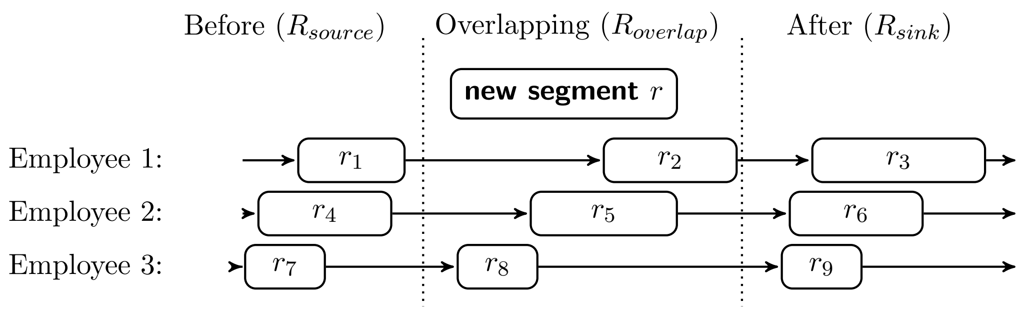

Figure 4: The creation of the sets for the Reschedule Overlapping Segments Heuristic. Each segment is drawn such that the endpoints correspond to the start and end time of the segment. We show the schedule of 3 employees; the edges represent the current paths of the employees: Employee 1 serves segments r_1, r_2, and r_3, etc. Segment r has to be inserted; it overlaps in time with segments r_2, r_5, and r_8.

$R_{overlap}$ is the set of overlapping segments. Given the current schedule, we determine the last segments served by the employees in the set R_{source}; these segments will take the role of the vertex s in the bipartite graph. Similarly, we find the first segments in the set R_{sink}, which will be used instead of the dummy vertex t in the bipartite graph. We then solve the assignment problem where we want to assign predecessors and successors for all segments in the set $R_{overlap}$; see Figure 5 for an illustration. The conversion to a bipartite graph $G = (V, E)$ is the same as explained before.

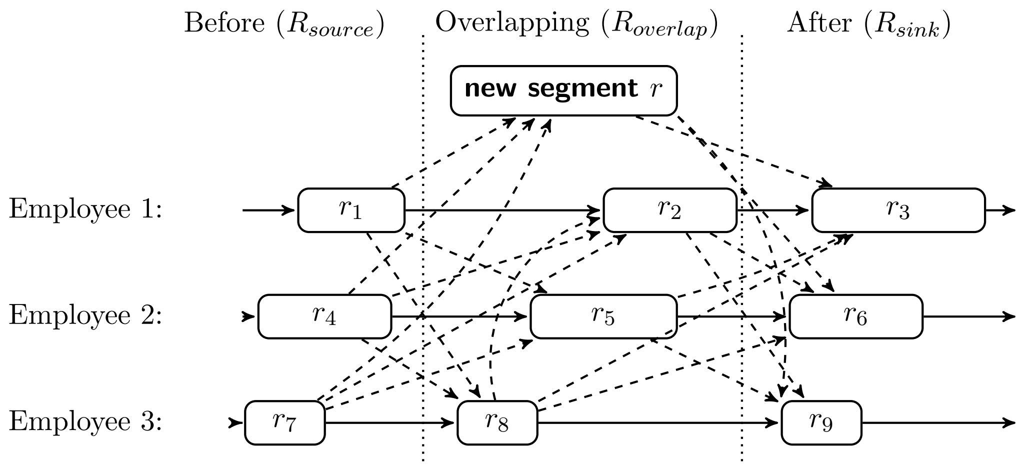

Figure 5: The restricted matching of the Reschedule Overlapping Segments Heuristic. We have added to the instance of Figure 4 the possible arcs for the matching problem as dazed arcs. The solid arcs correspond to the incumbent solution.

We can solve the corresponding matching problem just like in Section 4.2.1. The only difference with the previous subsection is that we now solve only a small part of the problem: we basically cut down the graph to make the problem smaller and thereby winning valuable computation time. Remark that the head of the route of an employee does not necessarily have to be matched to its tail again. Hence, this method cannot be applied directly in case the employees have different shifts. We discuss this part in the next subsection.

4.2.3 Adjusting the solution in case of different shifts

The subproblem for the variant where shifts are not identical is $\mathcal{NP}$-hard, since it corresponds to a multi-depot vehicle scheduling problem. We can solve this problem by using Integer Linear Programming (ILP), but as might be expected this takes too much time to be applicable in our decomposition approach. Therefore, we propose a greedy heuristic. At first, we make a planning in which we include the times at which the shifts start and end, but we do not enforce the connection between the two. We then adjust the solution to correctly match the start and end times of the shifts by applying a greedy fix. We want this greedy fix to be performed fast, since we have to do it in each iteration, and since we want to focus on finding a feasible solution with connections that are not too bad.

We start by adjusting the bipartite graph that we use to solve the matching problem. Just like in the Reschedule Overlapping Segments Heuristic, we split the source node s and the sink node t: for each employee j we include his own source node s_j and sink node t_j, and we add the arcs (s_j, n_i) and (n_i, t_j) according to the start and end time of the shift of employee j, for each j. When we then solve the matching problem, we assume that each employee j carries out the day plan corresponding to the path starting with s_j. If this path ends with the node t_j, then we have a feasible day task for employee j. If this path is connected to a shift end t_k with a different shift end time than t_j, then we must fix it. Thereto, we find a second route with an acceptable shift end. We cut these two routes somewhere and reconnect them such that the route for employee j satisfies the shift constraints. Because we are searching a robust solution the algorithm checks all day plans that are eligible to fix the route of employee j and all spots in the route for the cut with the best change in robustness score.

To avoid the possibility of cutting good routes again we don't allow to change routes that were already fixed. For that reason we sort the employees by shift end time, starting with the smallest ones first. We try to fix the routes in increasing order and only use routes that are higher on the ordered list as a second route to cut. In this way we avoid possibly cutting a route that has just been declared feasible or fixed. We work from small to large, because the employees with a smaller shift end are more constrained than the ones with a higher shift end.

With this method we can quickly fix the schedules to satisfy the shift constraints, which is important to limit the running time of the algorithm. It is possible that this algorithm cannot find a fix and then the mutation should be called infeasible although when done with an exact algorithm there might be a feasible answer. If necessary, we can always apply an exact algorithm like ILP at the end to optimize the schedule given the current start times.

5 Computational experiments

We have run our experiments on a Windows 7 Ultimate Computer with a Intel(R) Core(TM) i7 CPU 2.80 GHz quad core and 4 GB of RAM. The 11 instances were provided by Reinhardt et al. In these instances there is no distinction between prebooked and immediate PRMs, and hence we consider all PRMs as prebooked. Moreover, no shifts have been specified. We first present our experiments on the problem with identical shifts, and finally we present our results on the shift variant, where we have defined the shifts ourselves according to the arrival and departure times of the flights.

5.1 Identical shifts

In the master thesis on which this paper is based we compare four methods to solve the matching problem, but here we only report on our experiments with solving the subproblem to optimality using the Hungarian method (full matching) and solving it heuristically using the Reschedule Overlapping Segments Heuristic. In the first stage we use the following settings:

Max Iterations I:	500.000 for heuristics.	
	3.000 for full matching.	
Max time :	2 minutes	
Temperature start T:	400.0	(Temperature of the simulated annealing at start.)
Number of Q steps:	100	(T is multiplied by a every I/Q iterations.)
Alpha a :	0.95	
Decline penalty booked:	1200	
Mutations:	Plan_PRM, Decline_PRM, Merge_SegmentGroup,	
	Move_SegmentGroup, Move_Split_SegmentGroup	

Since it takes much more time to apply the Hungarian method than to apply the heuristic in each iteration, we use different numbers of the maximum number of iterations I. As a result, the temperature T will reduce faster then, since T is multiplied with α after each set of $I/100$ iterations; otherwise, the temperature would still have been high after the two minutes of running time.

We use the same temperature value for minimizing the cost of the declined PRMs and for minimizing the robustness penalty, but when evaluating the probability to accept a worse solution, we multiply the robustness score by 3 for the input of the formula for the probabilty to accept a worse solution. We decrease the temperature T with every I/Q iterations by a, such that multiplying T with a is done Q times no matter how many iterations we do.

The experiments are run 30 times using the above settings for each instance and method to solve the matching problem. The result table contains averages of the running times and the number of iterations, the best and worst solution with the number of declined PRMs and robustness score. The Average column lists the averages of the number of declined PRMs and robustness with a double precision of 1.

In Table 2 we show the results that were obtained by the decomposition approach when the subproblem was solved from scratch in each iteration by the Hungarian method.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	62852	2938	0	0	0,0	333,7	0	968
Instance 2	114121	2993	1	2844	4,3	3672,6	7	3334
Instance 3	120186	1936	0	6018	1,2	7817,9	5	8070
Instance 4	86868	3000	0	606	0,0	1375,5	0	2316
Instance 5	120184	1769	0	1559	0,0	4066,7	0	7369
Instance 6	120230	1816	0	1346	0,0	2928,7	0	6218
Instance 7	80059	3000	0	219	0,0	1087,8	0	3004
Instance 8	115554	2794	0	265	0,6	1080,1	2	1382
Instance 9	87762	3000	0	263	0,0	1088,5	0	1852
Instance 10	120161	1951	0	3446	0,0	5772,3	1	8123
Instance 11	120231	859	76	3735	96,8	3756,9	116	3576

Table 2: Results when solving the subproblem from scratch using the Hungarian method

The results are reasonable: almost all PRMs are served, except for Instance 11. The number of iterations is rather disappointing, since usually not all of the at most 3000 iterations could be run in two minutes. It is possible to speed up the Hungarian algorithm by working incrementally instead of starting from scratch in each iteration; this is known as the *dynamic Hungarian algorithm* [11]. With the dynamic Hungarian algorithm we can dynamically add segments in the matching problem or change the costs of the matchings when segments are given different start times. We did not implement this, since the Reschedule Overlapping Segments Heuristic yields very good results, as Table 3 shows.

instance	time(ms)	iterations	Best		Average		Worst	
			declined	robust	declined	robust	declined	robust
Instance 1	1831	11143	0	0	0,0	0,0	0	0
Instance 2	70752	500000	1	0	1,4	183,2	3	0
Instance 3	43403	289371	0	0	0,0	0,0	0	0
Instance 4	17698	115761	0	0	0,0	0,0	0	0
Instance 5	51850	243457	0	0	0,0	0,0	0	1
Instance 6	47073	160797	0	0	0,0	0,0	0	0
Instance 7	10415	69227	0	0	0,0	0,0	0	0
Instance 8	14503	82530	0	0	0,0	0,0	0	0
Instance 9	9546	58167	0	0	0,0	0,0	0	0
Instance 10	52418	257654	0	0	0,0	0,0	0	0
Instance 11	51225	174627	0	0	0,0	0,0	0	0

Table 3: Results when solving the subproblem incrementally using the Reschedule Overlapping Segments Heuristic

These results clearly are optimal for our objective function. After inspecting the solution of Instance 2, we found out that the single, declined PRM simply cannot be avoided.

When we take a closer look at the generated solutions, then we see that in general the employees at the terminals have lots of time; usually the slacks between two consecutive tasks for the same employee are much larger than 20 minutes, which implies that we do not have to worry about lunch-breaks, etc. There are some peaks, however, for which we need the employees to serve the PRMs. The inter-terminal buses are busier, but nevertheless the drivers have some time for a break. The workload of the buses is better spread over the day, which may be due to our policy of parking PRMs at a lounge before and after a bus transfer. There are many bus trips in which two or more PRMs are served at the same time, but still the bus drivers have enough work to do. In a similar fashion, we find that many PRMs travel together towards the lounge, even though they come from two different terminals, which implies that the bus segments get synchronized by our algorithm.

5.2 Shift variant

Unfortunately, in the original data there were no shifts involved. To test our algorithms in case of shifts, we therefore generated the shifts ourselves. To make the instances interesting, we introduce overlapping shifts, since the employees that are available at a given time would be identical, otherwise. We slice the shifts from all employees in the input data into three parts. Then new employees are generated, who each serve one or two neighboring parts, such that the shifts overlap each other. The first kind of employees only serve part 1, the second kind serves parts 1 and 2, the third kind serves parts 2 and 3, and the last kind serves part 3. We make sure that at every moment of the day the same number of employees is active as in the original problem.

We use the decomposition approach, and we have tested all our algorithms to solve the matching problem. After each iteration, we apply our greedy heuristic such that the solution obeys the shifts. The computational results show the same pattern as in case of the variant with identical shifts. We only report the results for the Reschedule Overlapping Segments Heuristic; see the table below. Again, the heuristics perform very well.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	2762	7996	0	0	0,0	0,0	0	0
Instance 2	120026	418887	1	0	1,5	348,0	3	307
Instance 3	80280	270502	0	0	0,0	0,0	0	0
Instance 4	29871	95774	0	0	0,0	0,0	0	0
Instance 5	106371	235641	0	0	0,0	0,6	0	10
Instance 6	80400	151739	0	0	0,0	0,5	0	14
Instance 7	18528	63811	0	0	0,0	0,1	0	4
Instance 8	34726	92349	0	0	0,0	0,0	0	1
Instance 9	32381	109791	0	0	0,0	0,0	0	0
Instance 10	94603	223557	0	0	0,0	0,0	0	1
Instance 11	95809	157717	0	0	0,0	2,3	0	16

Table 4: Results for the shift variant with overlapping shifts: Reschedule Overlapping Segments Heuristic in combination with the greedy shift fix

6 Simulation

Our algorithm may be capable of finding optimal solutions for the deterministic case, but how will it behave in a volatile, dynamic environment like an airport? Another question concerns robustness. In our solution approach we have added the concept of robustness: will the inclusion of this robustness pay off in practice? To answer these questions, we will apply discrete-event simulation. We start by discussing possible sources of uncertainty.

6.1 Disturbances

In our simulation we want to include the possible disturbances that occur in practice and that may deteriorate the quality of our schedule. We will consider the following sources of uncertainty:

- Arrival of immediate PRMs;
- Deviations from the scheduled arrival and departure times of flights;
- Deviations from the planned travel times (lengths of segments).

Below we discuss these uncertainties and indicate how we included these in our simulation.

6.1.1 Immediate PRMs

Since we do not know which PRMs in the instance data are prebooked PRMs, we consider all of them to be prebooked. We further generate 100 additional immediate PRMs, who check in during the day. This makes it harder for the algorithm and employees to handle all PRMs than in the original problem instances. We used the instance data to make a template for generating new PRMs, where we extracted the possible pickup points, gates, drop off points and flight schedules from the journeys of the PRMs in these instances. Flights are not directly listed in the instance data, but from the journey of the PRMs we can see which flight is used for arrival and/or departure. Furthermore, we generate some additional flights, which together with the flights from the instance data are used as our complete flight schedule in the simulation. The flights for the immediate PRM's can then be selected from this complete set of flights.

Immediate PRMs are generated by first deciding whether it concerns an arriving, departing, or transfer passenger. For each generated PRM we just choose a random start and end location matching with its type of journey; the route of the PRM through the airport follows then immediately. If the PRM is assumed to arrive by aircraft, then we choose a random arriving flight (including the ones we added) as start location; for the remaining PRMs we choose a random pickup point in the terminal for the PRM to show up in our simulation. Similarly, if the PRM departs by aircraft, then we choose a random departing flight as end location; for the remaining PRMs we choose a random drop off point in the terminal at which they arrived.

We generate the additional PRMs before executing the simulation, but to avoid using knowledge that we are not supposed to have, we only notify the simulation of their existence at the time that the PRMs make themselves known. For PRM's arriving by plane, we assume that we are notified some time beforehand; for PRMs arriving by train, bus, or car, we assume that we do not know of their existence until they require assistance somewhere at the airport.

6.1.2 Flight delays

We generated the flight delays by the method used by Diepen et al. [8], who conducted research at Amsterdam Airport Schiphol on robust scheduling of platform buses. They performed a simulation study to test the effect of including robustness in the objective function. To find realistic values for the disturbance they analyzed the delay statistics of both arrival and departure times of Schiphol airport.

For the deviations of the arrival times they found a normal distribution with a mean of 4 minutes and a variance of 30 minutes. For the deviation of the departure times they found that in 2% of the cases a flight left early, where the amount of time that the flight leaves early follows an exponential distribution with a mean of 2 minutes. In the remaining 98% of the cases the flight departs later and this time follows an exponential distribution with a mean of 15 minutes. In our simulation we leave out the possibility that flights depart early; opening the gate a few minutes earlier will have a minor impact on the schedule, since there is enough time remaining to help the PRMs through the boarding process.

Flights give status updates to the airport from time to time, which might adjust the expected arrival or departure time. Diepen et al. [8] report that the time at which the last updates are given follows an exponential distribution with a mean of 15 minutes before the actual arrival or departure time of the flight. We use a similar distribution in our simulation. We assume that at least 10 minutes before the actual arrival time it is known whether or not the flight has a delay or is early, but to add some randomness, we also use the same distribution for the last update to make the warning a bit earlier and more random.

For departing flights we assume that the aircraft is already at the airport in advance to enable the maintenance between the flights, like cleaning, refueling and unloading the luggage. When the aircraft arrives from its previous flight we will know approximately when boarding for departure can start. We assume that we have this information at least 45 minutes in advance.

Just like for the incoming flights, we add some randomness, and therefore, we expect to get the last update at time: $actual\_depart - 45 - exp(1/15)$.

6.1.3 Travel times

In the given deterministic instances the time that it takes to travel a certain segment is considered to be fixed. We assume that these are based on averages, and therefore we include the randomness in these travel times in our simulation. Since we do not have any data on this, we have made the assumption that these travel times are uniformly distributed. Here we distinguish between PRMs in a wheelchair and walking PRMs who we expect to be a little slower in general. For wheelchair PRMs we take a uniform distribution of $[0.9, 1.1]$ times the expected travel time; for the walking PRMs we use a uniform distribution of $[0.9, 1.3]$ times the expected travel time. If the employee assists multiple PRMs, then we generate the travel time of all PRMs that the employee assists and take the largest as the group's travel time. We generate the times needed by the buses in the same way, although the most delay will probably be caused by either traffic or the time it takes to load and unload the PRM in the bus. For the boarding segment we stick to the deterministic boarding time of 20 minutes, since we feel that 20 minutes is more than sufficient for this.

6.2 Rescheduling

In this section we will discuss how our scheduling algorithms will be applied to deal with the described disturbances. In general we want to avoid full rescheduling as much as possible just like Diepen at al. [8] in their simulation for the platform buses; we expect that the solution is robust enough to be resistant against a minor disturbance like an increased travel time of a PRM. We further want to change as little as possible to the solution in case of a full reschedule to avoid that the employees's schedules change dramatically every time something happens. We see to it that a PRM who was accepted in a previous solution will not be declined due to an update, unless it is technically impossible for him to catch the connecting flight.

We only want to reschedule at the important timepoints. Therefore, we distinguish between minor and major disturbances. There are two possible causes of a *major disturbance*: adding a new PRM to the schedule, or having a major difference (more than 10 minutes) between the published and the actual arrival or departure time of a flight. Flights give several status updates to the airport from time to time, and we do not want to update our solution at each update. Diepen et al. apply the strategy of rescheduling (if necessary) only when the aircraft has reported its last update, such that the time at which it arrives or departs is known with certainty. In our case, however, this is not a viable approach. For example, in case of a large delay of a departing flight, adjusting the solution only at the last status update would imply that the PRM has to wait at the gate, instead of in the lounge, which is much more comfortable. Furthermore, the employee accompanying the PRM would be late for his next task. Therefore, we will adjust the schedule, if necessary, both at the first and at the last status update. At the first update, we remove the segments of the PRM from the schedule, and at the last update we have to reintroduce them with their postponed time-stamps.

Causes of *minor disturbances* are changes of less than 10 minutes in flight arrival or departure times or deviations in travelling times of PRMs (by bus or moving through a terminal). We only reschedule in case of major disturbances. In case of a minor disturbance, we just make a small shift in the schedule and assume that there is enough slack in the schedule of the employee to compensate this. Still, any disturbance in flight arrival of departure times or may push the start of a segment group forward in time, which in turn may make the current planning of a consecutive planned segment group invalid. Similarly, a deviation in the travelling time of a PRM, may delay the completion of a segment group and also make the current planning of a consecutive planned segment group invalid. We call segments that are invalid because of a disturbance in another segment *invalid segments*.

Consequently, for the simulation we have three kinds of adjustments that we must consider when rescheduling: introducing a PRM, moving invalid segments, and reintroducing postponed segments. When rescheduling we always first try to solve the problem by applying minor changes in an update before doing a full reschedule.

In our update algorithm we first try to add new PRMs to the schedule if any. Thereto, we apply the 'add PRM' and 'Merge segment group' mutations (see Subsection 4.1) for a number of times in combination with the *Free Spot* insertion heuristic to generate the solution. The Free Spot heuristic simply tries to insert a segment in the schedule of an employee without changing the assignment of any other segment; we use this insertion heuristic, because it leaves most of the employees's schedules intact.

Next, we try to replan the invalid segments. Again, we first try to fix this by executing a mutation that moves the segment groups containing the invalid segments. In case the segment was merged to a segment of another PRM to share a single employee or bus, then we separate those two journeys to allow the segment to be planned individually. We also execute a mutation that tries to merge the segment group with another segment group. Again we use the Free Spot heuristic to solve the subproblem.

Finally, we plan the segments that due to flight delay were postponed and removed from the schedule until their start time became clear. Again, we use the ‘Merge segment group’ mutation in combination with the the Free Spot heuristic to solve the subproblem.

If after the application of our update algorithm, there is still at least one unplanned new PRM, invalid segment or unplanned postponed segment, then we will do a full reschedule. We keep track of how many times we need to do an update and how many times we need a full reschedule.

When in the simulation we have to do a reschedule, the simulation is already at a certain point in time, and we cannot change anything that has already happened and we cannot change tasks that have already started. Moreover, when the employee receives a new task he needs time to adjust, finish what he is currently doing and walk towards the location of the next task. To that end, we set a threshold in time before which major changes in the schedules may not occur; in our simulation experiments, we put the threshold equal to the current time plus 20 minutes. We apply the Reschedule Overlapping Segments heuristic for assigning segments that start after the threshold to empoyees. Finally, we also change the stopping criterium a bit in case of a full reschedule. Since the schedule is fixed until the current time of the simulation, we use the robustness score of this part as a lower bound.

6.3 Simulation experiment

The goal of our simulation is two-fold. The major goal is to find out whether the algorithm can be used in a dynamic setting in practice, and the second goal is to study the effect of including robustness. To answer the second question, we have ran our simulation with two variants of our algorithm: once with the robust penalty set as described in Subsection 2.3, and once with the penalty equal to zero in all cases. We then compare the performance of both versions using Student’s t-test, to see which one performs better.

We use the instances obtained from Reinhardt et al. with the disturbances defined before. To compare the performance of our algorithm with robustness to that without robustness, we use common random numbers. For each instance we generate 10 realizations the immediate PRMs and flight delays, which we use as input for both versions. Consequently, the major disturbances of both runs will be the same, while the execution and rescheduling of the schedule will be different. We compare the results of the instances in a paired t-test. As start schedule for the simulation, we use the results from our earlier computational experiments. Since we have found optimal schedules (see Table 3), for each instance we randomly select one optimal solution as starting solution. For the variant where robustness is disabled we generated new solutions, using the best variant of our algorithm but with robustness disabled.

For rescheduling we use the methods described above. To make the simulation run faster, such that the staff gets the results of a reschedule quickly, we allow 50.000 iterations to be done by the algorithm instead of the 500.000 in our computational experiments of our best algorithm.

In the simulation we keep track of the following statistics:

- The number of rejected PRMs, where we distinguish between prebooked and immediate PRMs. Every PRM who is scheduled in the start solution at the beginning of the day is considered prebooked; the ones who arrive during the simulation are considered immediate.
- Total unnecessary waiting time in minutes of the PRMs in the actual execution of the schedule; the total time a PRM has to wait for a connection is considered unnecessary waiting time. Since we do not allow detours, we can ignore these.
- The number of times that the simulation executes a full reschedule and the number of times that it executes just an update while calling the rescheduling protocol described in the previous subsection.

After having executed the simulation experiments we did a two-tailed paired t-test on each of the statistics mentioned above to compare the two algorithms, with a null hypothesis that the corresponding values are equal for both algorithms. We reject the null hypothesis for a statistic if the corresponding p-value is less than the commonly used acceptance threshold of 0.05.

Below we show the results of the experiments. For each statistic we show the mean and variance of both instances and the p value denoting the probability that the two distributions are considered identical by the two tailed paired t-test. The annotation '(robust)' indicates the results for the variant where we consider robustness, whereas the annotation '(no robust)' indicates the results for the variant where robustness is ignored.

statistic	**mean (robust)**	**variance (robust)**	**mean (no robust)**	**variance (no robust)**	**p-value**
declined booked	$1,15$	$1,49$	$1,08$	$1,49$	$0,52$
declined immediate	$2,4$	$3,89$	$2,37$	$3,88$	$0,56$
wait time	$87,35$	$2024,87$	$170,6$	$8212,02$	$1,10 \times 10^{-15}$
reschedule updates	$145,05$	$49,48$	$144,31$	$53,67$	$0,03$
full reschedules	$11,4$	$20,50$	$11,15$	$21,12$	$0,37$

Although at first sight it looks like the variant where robustness is ignored is able to plan slightly more prebooked PRMs than the robust variant, the corresponding p-value of 0,52 indicates that the null hypothesis that both algorithms perform equally well on this aspect must be accepted. The same conclusion is obtained for the number of rejected immediate PRMs.

With respect to the total unnecessary waiting time the conclusion is simple: including robustness is beneficial. This is not a big surprise of course, since we introduced robustness to be resistant against minor delays.

The number of updates done differs between the robust variant and the variant where robustness is neglected in the sense that the variant without robustness needs slightly fewer updates. This may seem surprising. However, in a robust schedule the slack is spread out quite evenly, whereas in a non-robust schedule there are parts with very little slack and parts with a lot of slack. If there are disturbances in the latter part, it is high likely that an update is not required. Moreover, flight delays for flights of PRMs who become unassigned at some point during the day, may introduce a slight difference in the number of updates or reschedules. Observe that the number of full reschedules does not differ enough to reject the null hypothesis of equality. Although, contrary to what we expected, we found that including robustness increases the number of reschedule updates, this is not as important as the number of full reschedules.

7 Conclusion

In this paper we have studied the problem of assisting PRMs in their journey through the airport. This problem was first presented by Reinhardt et al. [12] for a major, international airport. They present a local search approach based on an insertion heuristic, and raised the question if other approaches would be able to perform better. We present a decomposition approach for this problem. In Phase 1, we set the start times of the segments of the PRM's journeys, and in Phase 2 we assign the work to the employees. By synchronizing the segments of each individual PRM, we can eliminate all unnecessary waiting time for the deterministic case. Because of the limit of two minutes on the computation time, there is no time to solve the subproblem of Phase 2 to optimality at each iteration, but using the Reschedule Overlapping Segments Heuristic, which resolves the assignment problem locally, we can obtain excellent results for the deterministic case. We initially developed the algorithm for the case where all employees are available all day, and further show that with a small adjustment this approach can also be used for the problem in which the employees have different shifts.

Furthermore, we have tested our approach in a simulation experiment in which we let an additional 100 immediate PRMs enter the airport, and in which we introduce deviations from the published arrival and departure times of the flights together with stochastic travel times of PRMs. From the results we may conclude that our approach is very well suited to solve the PRM problem in real-life. Even though we added 100 immediate PRMs, only a few had to be declined. We further studied the use of including robustness as an additional objective by comparing our original algorithm to the variant in which the robustness penalty was set to zero. From this experiment, we draw the conclusion that both algorithms provide quite similar results, but that including robustness reduces the total unnecessary waiting time very significantly. Much to our surprise, including robustness does not reduce the number of reschedule operations.

Acknowledgment

This research was conducted when the first author was a student at Utrecht University. Again, the authors want to thank Line Blander Reinhardt, Tommy Clausen, and David Pisinger for presenting the anonymized data and their willingness to answer questions concerning these.

References

[1] G. Arcidiacono, A. Giorgetti, and M. Pugliese (2015). Axiomatic design to improve PRM airport assistance. In M.K. Thompson, A. Giorgetti, P. Citti, D. Matt and N.P. Suh, editors. *Procedia CIRP Volume 34, 9th International Conference on Axiomatic Design (ICAD 2015)*, pp. 106–111.

[2] A. Bolat (2000). Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research 120*, pp. 63-80.

[3] R.E. Burkard, M. Dell'Amico, S. Martello. (2012). Assignment Problems (Revised reprint). SIAM, Philadelphia (PA.), 2012.

[4] Y.-C. Chang and C.-F. Chen (2012). Meeting the needs of disabled air passengers: Factors that facilitate help from airlines and airports. *Tourism Management, 33* pp. 529–536.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein (2009). Introduction to Algorithms, third edition. *The MIT Press.* Cambridge, Massachusetts London, England.

[6] Guy Desaulniers and June Lavigne and François Soumis (1998). Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research 111*, pp. 479–494.

[7] G. Diepen, J.M. van den Akker, J.A. Hoogeveen, and J.W. Smeltink (2012). Finding robust assignment of flights to gates at Amsterdam Airport Schiphol. *Journal of Scheduling 15*, pp. 703–715.

[8] G. Diepen, B.F.I. Pieters, J.M. van den Akker, J.A. Hoogeveen (2013). Robust planning of airport platform buses. *Computers & Operations Research 40*, pp. 747–757.

[9] Richard Freling, Albert P.M. Wagelmans and José M. Pinto Paixão (2001). Models and Algorithms for Single-Depot Vehicle Scheduling. *Transportation Science 35*, pp. 165–180.

[10] R.M. Karp (1972). Reducibility Among Combinatorial Problems. *Complexity of Computer Computations.* New York: Plenum. pp. 85–103.

[11] G.A. Mills-Tettey, A. Stentz, and M.B. Dias (2007). The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213.
https://scholar.google.nl/scholar?oe=utf-8&gws_rd=cr&um=1&ie=UTF-8&lr&cites=3117902560827097317

[12] L.B. Reinhardt, T. Clausen, and D. Pisinger (2013). Synchronized dial-a-ride transportation of disabled passengers at airports, *European Journal of Operational Research 225*, 106–117.