Utrecht University

Master Thesis (ICA-3227170)

Synchronizing transportation of people with reduced mobility through airport terminals

Author:
R.P. van Twist

Supervisors:
dr. J.A. Hoogeveen
dr. ir. J.M. van den Akker

April 30, 2015

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 are looking at an airport that assists between 300 and 500 of those passengers daily. We want to find a schedule for the airport's employees to support as many of those passengers as possible while ensuring a smooth journey with little waiting time. In addition we want to find a robust schedule to handle minor disturbances which we test using a simulation at the end. 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 connections while supporting nearly every passenger in the given instances.

Contents

1 Introduction

Imagine a major airport where a lot of passengers go through daily. Some passengers arrive by plane, some depart by plane and some need to catch a connecting flight after arriving by plane. A typical journey of a passenger with a connection flight starts with arriving at one terminal by plane. After arriving the passenger travels to the terminal where the connecting plane departs from, he waits there for boarding to start and leaves the airport by plane. This process is simple for most passengers, but when you are restricted in movement, visually impaired or have some other kind of disablement that restricts your journey through the airport this is not so simple. That is why the airport offers a service to assist those passengers with their journey through the airport and makes sure they are supervised at all times. The largest airports have an average of 500 passengers daily who need such assistance. We are looking for a schedule where we assign a start time and an employee for each part of the journey of those passengers. The aim for the schedule is to provide the best service possible to the passengers with reduced mobility.

However scheduling those passengers is not so straightforward since the passengers need to be supervised at all times. This can be done either by the employee assisting the passenger or bringing him to a special supervised lounge. Furthermore there are points in the journey where the passenger needs to be handed over to a different employee. Because the terminals are not connected with each other, PRMs are required to take a special bus to go to another terminal. Since employees are allowed to work at their own terminal area only, 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 plane the passenger needs to catch is not directly located at the gate and hence the passenger needs transport by platform bus to get there.

In this paper we research this scheduling problem for a graduation thesis done at Utrecht University. We call these passengers who require assistance with their journey: *people with reduced mobility* or *PRM* for short, we use this term for the rest of the paper. This problem is presented by Reinhardt et al (2012)[1]. We use the data of the airport layout and problem instances provided by them. Which airport they used is unknown to us, we only know it is a major airport. We didn't have contact with the airport in question so the context of the airfield came from the paper and e-mail contact with the authors of the paper, who kindly presented us with their instances.[1]

1.1 The airport

In the airport in question the company serving PRMs has about 120 employees, who transport between 300 and 500 PRMs everyday. The airport consists of 11 disjunct terminals where planes arrive and depart from. Each terminal has its own staff, who can only serve the PRMs inside the terminal he is assigned to. Besides the terminals we also have 2 bus sites, each with its own fleet of buses and staff. One bus site is for inter terminal travel and the other for the platform busses who drive between gates and remote plane locations. The inter terminal busses are the only way we can transport a PRM from one terminal to another. Each terminal has a bus stop where busses pick up and deliver their passengers. Just like the employees the busses have to stay within their designated area, platform busses are not used for inter terminal travel and inter terminal busses are not used for traveling between gates and planes. The busses used to transport PRMs are specially reserved for them and are not to be used by passengers who didn't request assistance by the service provider.

Some planes are not directly connected by a gate but stand at a remote location. Passengers are then brought to the remote location by platform busses from the platform gate where they start the boarding proces. Although every passenger on that plane needs to travel from the gate to the plane by platform busses, PRMs are brought using special busses.

All employees in a terminal or a bus at a bus site are considered to be identical in terms of capacity and times it takes to travel between 2 locations. Each employee has a start time for his shift and an end time of the shift, between which they can serve PRMs. Furthermore the staff members and buses also have a location where they start their shift and a location where they end their shift. Although in the provided instance data all employees got identical shifts, the problem definition presented in the paper of Reinhardt et al[1] does not mention that.

Each terminal staff member is able to serve 1 or 2 PRMs at the same time depending on the disability of the PRM. For instance an employee can only transport one wheelchair but can hold 2 PRMs who can walk. Each bus is able to serve a number of PRMs, also depending on the disability of the PRM. A wheel chair again for example takes up more room in the bus than a single person.

Every area, representing a terminal or a bus site, has a number of locations the PRMs and employees can visit. Each terminal has one special supervised lounge, one bus stop for inter-terminal busses, a number of gates, a number of platform gates, a location for the staff to start and end their shift and a number of locations where PRMs can be picked up and delivered to. The inter-terminal bus site has one bus stop at each terminal and a depot for the buses. The platform bus site has a number of platform gates at the terminals, a number of remote plane locations and a depot for the buses. The locations of the bus stops and platform gates are shared between the terminal and bus sites. These locations are used to define the journeys of the PRMswhich consists of a sequence of locations that a PRM must visit and could possibly handed over to other employees to supervise and assist them.

A lounge is a special location in each terminal. In the lounge the PRM is supervised and can wait for the continuation of his journey. Whereas waiting at an unsupervised location causes discomfort to the PRM, waiting in the lounge it does not. In the meanwhile the employee who assisted the PRM could then do another job. For example if a PRM needs to board a plane then the employees can leave the PRM in the lounge of the departing planes terminal where he could wait till the boarding starts. Instead of uncomfortably waiting at the gate with an employee the PRM could wait more comfortably at the lounge.

1.2 The journeys of the PRMs

PRMs, People with Reduced Mobility, request assistance at the service provider of the airport because they have some kind of disability that makes their journey through the airport difficult. We call the assisted journey through the airport with a PRM a *route*. A route follows a sequence of locations that the PRM must visit though the terminals and bus sites. This route is predefined for each PRM and depends only on the arrival location and destination of the journey, which gates are assigned to the planes and whether or not a lounge needs to be visited. A remote stand for planes could be accessed by multiple platform gates but per flight only one of those is assigned for boarding or disembarking. When there is not enough time between the arrival of the PRM at the airport and deadline to visit a lounge, we could skip the lounge visit.

If we see the route 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. We call an edge of the PRM's journey that can be served by an employee a *segment*. In order for an employee to assist the PRM with a segment, both begin and end location of the edge the segment represents must be in the same area, otherwise we violate the constraint that employees may not leave their terminal or buses their buss site. 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). Each segment takes time for the assigned employee to serve, which is equal to the travel time between those 2 locations.

A special segment is the one that represents the boarding process. When a PRM needs to catch a plane, he needs to be at the gate when the boarding begins. The employee assisting the PRM through the boarding process must then stay with the PRM till the gate closes, which is 20 minutes after the boarding has begun. This is represented as a special segment in the journey both starting and ending at the gate, which takes 20 minutes to complete and can't start earlier than the gate opens.

The route a PRM takes is predefined depending on the location at which he arrives and the location where he needs to go to; we divide all those routes into 3 categories: Arrival, Departure and Transfer. PRMs in the category Arrival, arrive at the airport by plane and need to be transported to a certain location in the airport that isn't a plane. PRMs in the category Departure, check in at a location at the terminal of the departing plane and then need to be transported to a gate for embarking. PRMs in the category Transfer, need to be transferred from an arriving airplane to a departing airplane;l if the planes resides in different terminals the PRM needs to be transported between these terminals. Examples of journeys are shown in Figure 1, 1 and 1. In those figures two terminals are presented along with the platform bus site and inter terminal bus site.

You can start an arrival journey either at a gate or a remote plane and end at a deliver location, see Figure 1 for an example. The segments for this journey will be (Remote plane, Gate), (Gate, Deliver). If the plane is located directly at the gate we do not need the first segment from the remote plane to the gate.

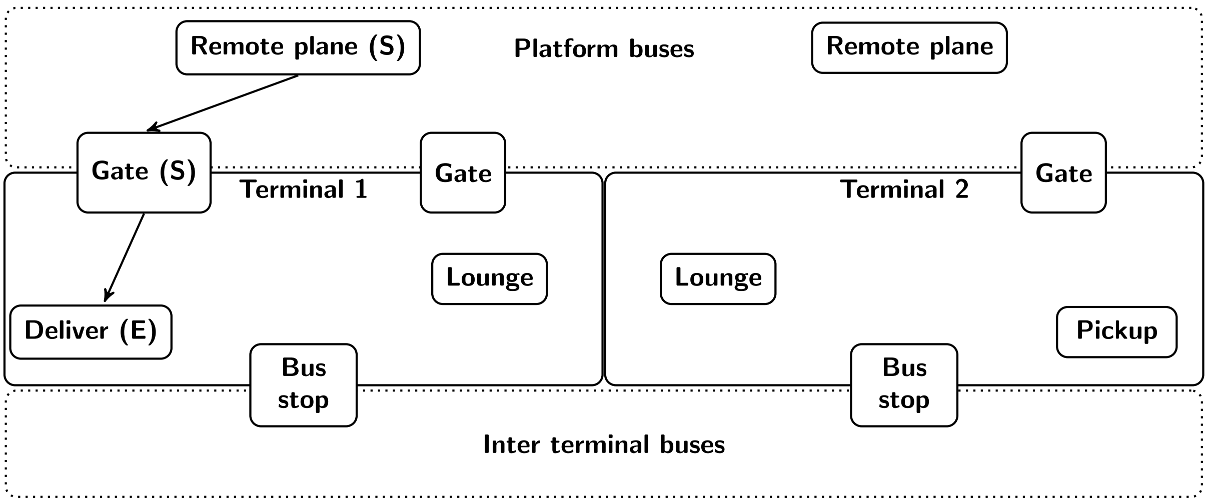

Figure 1: This figure shows an example for the arrival journeys a PRM could take. We show two terminals, the inter terminal bus site and the platform bus site. The rest of the nodes represents locations and the edges represent segments. Locations ending with (S) act as a possible start location and locations ending with (E) act as a possible end location.

You can start a depart journey at a pickup location and end either at a gate or a Remote plane, see Figure 2 for an example. The segments for this journey will be (Pickup, Lounge), (Lounge, Gate), (Boarding), (Gate, Remote plane) or if you skip the lounge visit (Pickup,Gate), (Boarding), (Gate, Remote plane). If the plane is located directly at the gate we do not need the last segment from the gate to the remote plane.

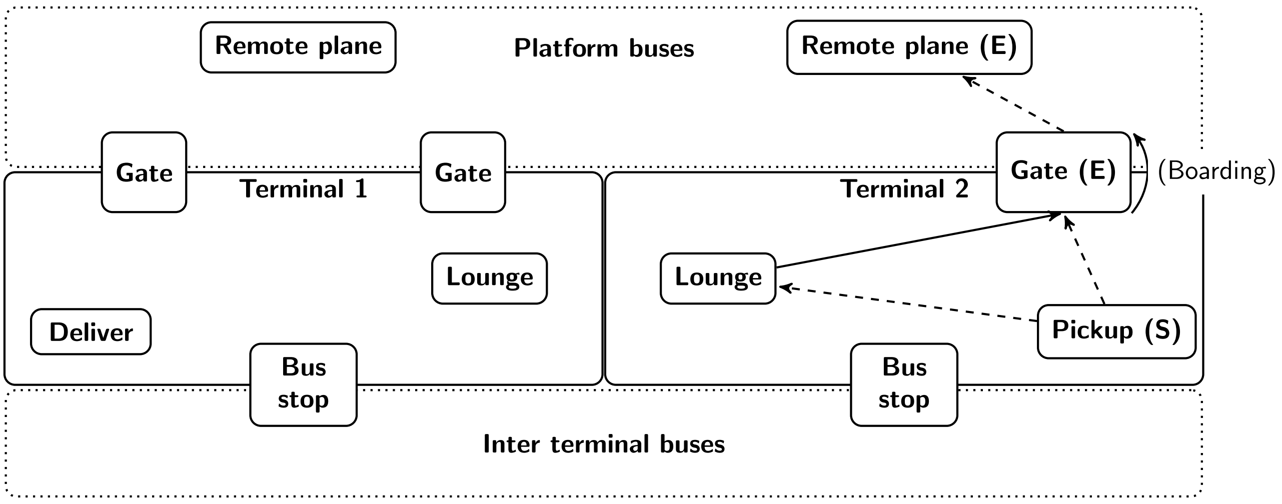

Figure 2: This figure shows an example for the departure journeys a PRM could take. Dashed edges are either optional segments or choices between 2 routes.

A transfer journey is basically an arrival and a departure journey in one, see Figure 3 for examples. This journey got some variants depending on whether or not lounges are visited and on the location of the connecting plane. If the connecting plane resides in the same terminal then after arriving and passing through the gate the PRM is brought to the lounge. If the plane needs to be boarded at another terminal the PRM is brought to the bus stop after traveling through the gate. It is possible that before visiting the bus area the PRM is seated in the lounge. At the bus stop the PRM is handed over to a bus driver who brings the PRM to the terminal he must board the connecting plane in. After the PRM arrived at the terminal he can be seated in the lounge to await the boarding to start.

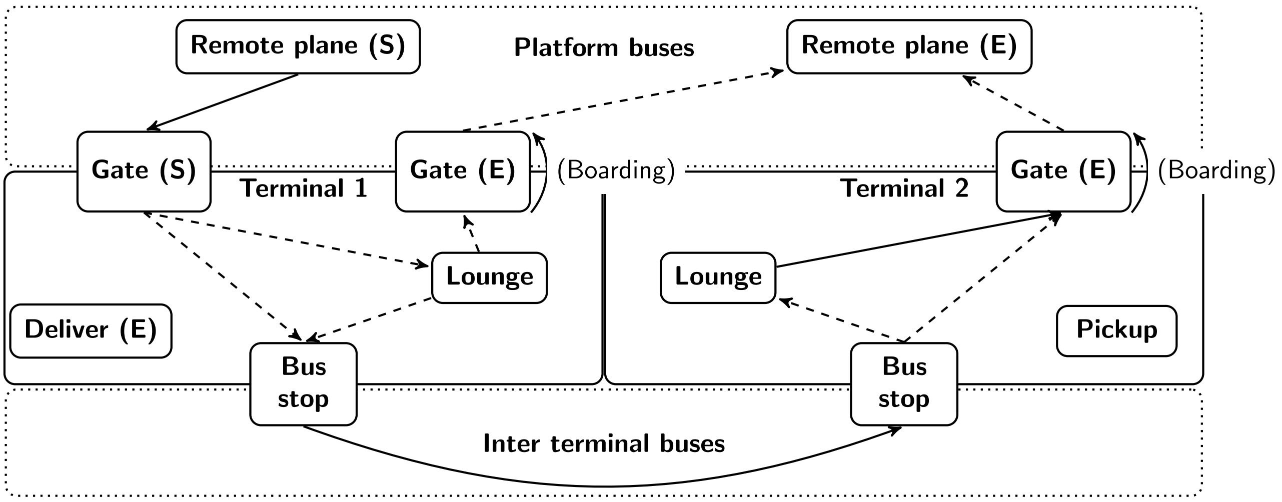

Figure 3: This figure shows examples of all possible transfer journeys a PRM could take. Dashed edges are either optional segments or choices between 2 routes.

PRMs can book assistance in advance and notify the company of their journeys at least a day before their visit at the airport; we call these persons prebooked PRMs. However there are also PRMs who check-in during the day they visit the airport; we call these persons immediate PRMs. When an immediate PRM requests at from a different airport and arrives here by plane later in the day we know for some time in advance that this PRM is going to arrive. Although the PRM is known to us some time in advance before he must be scheduled, if the request is done during the day the PRM visits the airport, the PRM is still considered an immediate PRM. Once an immediate PRM checks in he then needs to be inserted in the schedule, which might require rescheduling the other PRMs. The company wants to prioritize service of booked PRMs over immediate PRMs, when having to decline one of them.

Because of the rule that the employees may not leave their designated area, at some points of the journey the PRM needs to be handed over to another employee. For example when the PRM arrives at one terminal and is brought to an inter-terminal bus to catch a plane in another terminal, the terminal employee then must hand over the PRM to the bus driver. Generally a handover could take place between each subsequent segment, because the next segment could have been assigned to a different employee than the previous one even though the current employee could serve both segments if they are in the same terminal or bus site. Because the PRM needs to be supervised at all times the employee currently serving the PRM needs to wait till the PRM is handed over before he can leave to serve the next PRM. We want these connections to be smooth because if employees with PRMs are waiting for the other employee to take over the PRM we do not only keep the current employee occupied longer than needed, but it also causes some discomfort to the PRM himself.

1.3 Objective

The company's main objective is to give the best service possible to the PRMs. The worst kind of service the company can give is to decline a PRM from service. Unfortunately there could be situations where some PRMs couldn't be scheduled and needs to take a later flight. If that is the case then the company prefers to decline immediate PRMs above prebooked PRMs. Besides that the company wants to improve service further by minimizing unnecessary travel time of the PRMs, which is the extra traveling time it takes to get a PRM towards his destination. For instance you get unnecessary travel time if an employee assisting a PRM takes a detour or is waiting for someone else to pick up the PRM. The time that a PRM is seated in a lounge does not contribute to the unnecessary travel time.

Ideally every PRM is served without any unnecessary waiting time in their journeys; therefore we are minimizing the number of declined PRMs and the sum of unnecessary waiting time. Here we weigh declining of service prebooked PRMs higher than to immediate PRMs. We want to find a schedule for the PRMs where we minimize the amount of unnecessary traveling time over all journeys where we only decline service to PRMs when there is no way to avoid it.

In the schedule we assign start times and employees to the segments of all planned PRMs. If a PRM has at least one segment that is not served by an employee then that PRM is considered declined. Of course some other constraints must be considered. For example an employee and a bus may not exceed their capacity at any given time. Also we must respect the travel times between locations. All other requirements listed in the sections above must also be respected. Below we summarize the most important constraints for our scheduling problem.

Constraints:

1. Release and deadlines of PRMs must be respected otherwise decline the PRM from service.
2. Embarkment lasts 20 minutes; the gate closes 20 minutes after opening for embarkment.
3. PRMs who arrive need to be scheduled at their release time.
4. To be scheduled the PRM must have each segment served by an employee.

5. A segment may not take more than 30 minutes of unnecessary travel time.
6. A PRM may not be left unsupervised, therefore an employee needs to wait till another employee takes over the PRM before he can serve another PRM unless the PRM is dropped off at a lounge.
7. Employees and buses must respect the travel times towards locations to pickup and deliver PRMs.
8. An employee or bus cannot help more PRMs at the same time than his capacity allows.
9. An employee can't leave his designated area.

In addition the planning algorithm needs to work fast as in a dynamic real world disruptions of the schedule can occur and require rescheduling. For instance a new immediate PRM arrives and must be scheduled or a plane got a major delay and the PRMs on that plane must be boarded much later. That is why a maximum computation time of 2 minutes is given to the algorithm, such that when unexpected events occur, we have a new schedule fast.

In addition to the original problem presented by Reinhardt et al[1] we are also looking for a robust schedule. We hope that by using a robust schedule, real world disruptions could be caught more easily and connections run smoother. A delay of one segment in the journey of the PRM could delay the rest of the journey as well, delaying segments planned after those. 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 could happen without a major effect on the schedule.

We score the robustness depending on the slack that is between two subsequent segments in the schedule of an employee. The more time between 2 subsequent segments an employee must serve the more robust the connection is. There is one exception on that however, when the employee is also serving the previous segment of the given PRM. It would be inefficient if the employee has to wait for an identical employee of the same area to take over the PRM, while he could serve the PRM himself. Therefore it is more efficient and robust if the employee is able to handle both segments after each other. When he handles both the previous and next segment we know that during the exchange, the employee does not have to wait. These kind of connections must get a better robustness score than normal handovers. It is nice that by using robustness we can favor those connections. The robustness score is more effective when PRMs need to change terminal or board a platform bus, where possibly delaying handovers must happen. More on robustness in a later section.

1.4 Reinhard et al.

The algorithm of Reinhardt et al [1] uses a Greedy Insertion heuristic which transforms an abstract representation of a candidate solution to an actual schedule. Simulated Annealing is used to mutate the abstract representation.

The abstract candidate representation consists of 2 lists; A list representing the pre-booked PRMs and a list representing the immediate PRMs. The order of the lists is important since the Greedy Insertion heuristic adds PRMs one by one into the schedule. All PRMs in the pre-booked list are inserted before any PRM in the immediate PRM list. For the initial candidate solution they sort both lists on release time in ascending order. Simulated Annealing is used to mutate those 2 lists by shuffling the order of PRMs. Then the Greedy Insertion heuristic is used to generate an actual schedule.

The Greedy Insertion heuristic inserts the pre-booked PRMs first to make sure they are prioritized like the service provider wishes. 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 other segments forward in the schedule as result of the insertion. Of the feasible insertions, the one who causes the least increase of the objective function is used for the insertion. The calculation of the increase in objective function per insertion is not easy. If you insert one segment on the route of a bus or employee it might delay other segments on that route. Since PRMs could travel with different employees or buses, a simple insertion could have a cascade of effects on the times of segments and possibly increase the objective value. Therefore we couldn't just use the increase of travel time of a single segment to update the objective function. If the heuristic fails to find a feasible spot for the segment the PRM is not included into the schedule and all his segments are removed from the schedule.

The simulated annealing does mutations on the 2 lists representing the abstract representation, after which it calculates the actual schedule and objective score using the greedy insertion heuristic. Simulated annealing is used to decide to keep the new solution or not. The algorithm uses 2 mutations. The first mutation takes a not assigned PRM and moves it a random number of places forward in the PRM list it is in. That way the PRM gets a somewhat higher priority to be planned. The second mutation swaps 2 random PRMs from the same list. With these the mutations it is impossible to end up in a different list such that an immediate PRM could not end up as pre-booked and vice versa.

Computational experiments have shown that the algorithm could find good solutions in 2 minutes and high quality ones in 10 minutes. Very few PRMs have been rejected by the Greedy insertion heuristic, but there is still some unnecessary waiting time. Due to the low running time of 2 minutes the algorithm can be run multiple times a day in a dynamic real world situation. However due to the static formulation of the problem to truly work in a dynamic environment a different formulation will be needed. Their solution work well with the short time constant and tests show that by increasing the solution time could give significant improvements.[1]

2 Analysis and approach

In this section we analyze the problem, trying to find out what makes the planning difficult and what could be done to overcome that. We are looking into sharing bus trips and making robust schedules. Also in the end of this section we present an NP-Hardness proof.

2.1 Sharing bus trips

The busses could transport multiple PRMs at the same time, as long their capacity allows it. Following from the sample data the service provider has between 7 and 13 buses in the inter-terminal bus area, where each bus has a capacity of 12. The capacity expresses how many people the bus could transport. Passengers with a wheelchair require a capacity of 1.5 and passengers without require a capacity of 1. Because the busses have such a large capacity you can transport easily multiple PRMs at the same time.

Although there is a big capacity, during our e-mail contact with Reinhardt et al[1] they hinted there might be a bottleneck in the inter terminal bus area. Hence either there aren't enough busses or the capacity of the busses isn't used efficiently enough. It would be very efficient to carry multiple PRMs per bus trip, reducing the number of bus trips and be able to transport more customers. The problem is however that PRMs don't arrive at the bus stop at the same time and letting them wait increases the unnecessary waiting time and the objective function. Therefore we want to synchronize the arrival of PRMs at the bus stop.

According to the paper of Reinhard et al[1] you can bring PRMs towards another location to pickup another PRM and then drop him off later with the cost of having unnecessary travel time for that PRM. The question is do we want to bring 2 passengers with 2 different locations in the same bus? Suppose 2 passengers with both a different destination but the same source have been picked up by the same buss. Then the bus needs to deliver one of those passengers first, lets say the first one at the closest location. The travel time between terminals of the inter-terminal bus area is almost always 5 minutes. When sharing a bus ride one passenger will have a travel time of 5 minutes and the other has a travel time of 10 minutes, 5 minutes more than necessary. Moreover we might pick up other passengers on that location and stepping in and out could take additional time depending on the handicap or hand over. We think its more efficient to just send passengers directly to their destination instead of having them bringing around locations where they don't need to be.

We could let the buses drive in shifts, since we got plenty of buses and 11 terminals. The downside of this is that buses are likely to make detours to bring PRMs to their destination, which increases unnecessary waiting time. For instance to get from terminal 1 to 5, you don't want to pass through terminals 2, 3 and 4 while there is nobody there to enter the bus. Also the distance matrix between those terminals is a bit odd, the travel time between almost all terminals is 5 minutes, which means that you have to drive like 20 minutes in case there are three terminals between your destination and source terminal for what might been a 5 minute drive. It would be more efficient to bring PRMs directly towards their destination terminal. The distances between terminals may be explained by that loading and unloading of PRMs are included in the distance matrix, which we are not sure of.

Another approach is to put as many PRMs as possible in a bus at the same terminal with the same terminal as destination. To do that we must synchronize their journeys such that they meet each other at the bus stop at the same time. These synchronized journeys must be planned together to ensure the journeys align at the bus stops. If a mutation is made in a journey that shares a bus ride with other PRMs, the other PRMs might require a mutation as well to ensure all PRMs who want to share a bus arrive at the bus stop at the same time.

We can use this synchronization for all segments where a worker could service two or more PRMs instead of solely the bus segments. If two PRMs need to catch the same plane and they got a light disability such that a worker can help both at the same time, they could be picked up together at the lounge before departure and be brought through the boarding process with only 1 employee instead of 2. By planning passengers such that they can be transported together, we 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 location and end location.
- An employee situated in the terminal or a buss in the buss 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 been infeasible.

We hope that synchronizing the segments as much as possible will relieve the stress on the employees and enables us be able to plan PRMs more efficiently. It will at least free up some employee while boarding PRMs on the same plane and will make better use of the bus capacity. We assumed that if multiple PRMs get picked up by the same buss 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.

2.2 Robustness

In the real world many unexpected events could take place such that the schedule could be disrupted. For example a plane could be delayed, gates could be switched or a new PRM appears and requests to be scheduled. Or minor events like an inter-terminal bus gets delayed by traffic, a booked PRM arrives late or it takes longer than expected to take a PRM from one location to another. Those people we are transporting are reduced in mobility after all, and it could be possible that they walk slower than anticipated and arrive later than expected at a location.

A delay can have a big effect on the schedule. If a segment of an PRM is delayed it could delay the segments of both the PRM and the employee, who in turn could delay other employees and PRMs causing a cascade of delays if there is no slack to catch it. We hope by making the schedule more robust 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 in addition to providing customer friendliness. A nice side effect is that if we manage to do this for all segments then there will be no unnecessary waiting time between subsequent segments of a PRMs journey, if everything goes allright and there are no bigger delays than the amount of idle time we build in.

When the employee has to pickup a PRM he travels towards the pickup location, and once the employee is there he might have some slack time before the PRM arrives. We define the robustness score by specifying a function based on the slack time. We want to maximize the robustness, since the more slack the employee has the more robust the connection is. Since we do a minimization on the main problem where we minimize the amount of declined PRMs and unnecessary waiting time, we should also calculate robustness as a minimization problem. Therefore we calculate 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. The function shouldn't be a linear function in terms of slack time because that will allow a long slack time and a very short slack time to have the same weight as 2 average slack times. We would prefer solutions that avoid short slack times over a solution that compensates short slack times with long slack times elsewhere.

Robustness has some connection to unnecessary waiting time since unnecessary waiting time by our definition of robustness signals a situation hat isn't robust. In case the PRM has unnecessary waiting time, because he has to wait for an employee or bus to pick him up, this implies that the employee or bus in question couldn't arrive earlier on the pickup spot because of the last segment he served or the time he starts the shift. Therefore there is no slack between the 2 segments and hence the connection is not robust. In the other case where the employee has to take a detour with a PRM to pick up another PRM to travel along, there is no slack between picking up the first PRM and the second one , and hence the connection of the second PRM isn't robust. Therefore if there is any unnecessary waiting time, there is also an connection that isn't robust. We want to make an exception for the robustness score when a PRM could do 2 subsequent segments with the same employee. It would be silly to call for another employee to serve the PRMs next segment to optimize robustness and that is why that connection should have a robustness penalty of 0. Remember because the employees and buses are identical they could easily swap tasks.

In our project we use the formula $rp(s)$ for calculating the robustness penalty between two subsequent segments where s is the slack time between those.

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

There are various ways of calculating robustness score, but we choose this one. For instance we could have taken the arc tan of van Diepen at al[13], who did research on scheduling platform busses at Schiphol airport. Since the problem instances are rounded on whole minutes and the objective function only contains integer components, the objective is also integer. So let's keep the robustness penalty also integer and not linear by using a square function. Using this function we return a robustness penalty of 0 for slack times of 20 minutes or more, since in our opinion after the 20 minute mark, robustness doesn't matter anymore. The worst robustness penalty is achieved by a slack of 0 with a penalty of 400. This worst case penalty for a segment can be used to set the weight of robustness in the objective function.

Now we have specified the robustness penalty we have to decide how to put it in the objective function. This is now a multi-objective problem. Kevin Ian Smith[6] has researched techniques for simulated annealing for multi-objective problems. Although we didn't use one of his techniques it gave us some insight.

A common way to deal with multi-objective problems is a matter of dominance. Hereby we assume that all objective functions must be minimized, since any function that must be maximized could easily be rewritten to a minimization problem. We denote that solution f dominates g as $f \prec g$. Solution f dominates g if in all values of the objective functions f is no more than the values of the objective of g and at least one objective function is better. A solution is then a Pareto optimum if there exists no other solution that dominates that solution. The set of solutions that are Pareto optimum is then called the Pareto optimum front.[6]

The Pareto optimum solutions are tradeoffs between objective functions, and we need to choose among those solutions which one we prefer. Some solutions will have a better score for the main objective and others will have a better score for the robustness objective. The main objective of this problem is providing good customer service like stated in Reinhardt et al[1], and the robustness penalty is added by us to handle small disturbances in the schedule. Since the main goal is to offer service, offering service should be weighted heavier than having a robust schedule.

But what would be a good solution and what kind of solutions do we prefer? Imagine there exists a perfect solution with 0 penalty on the main objective and 0 penalty on the robustness. That solution will by definition dominate all other solutions since it got the best score of both objectives. But if no such answer exists we will have to make some tradeoffs. In worst case we could end up with a solution with 0 penalty on the robustness but many unplanned PRMs because adding them would increase the robustness penalty and the new solution would not dominate the older. Adding a new PRM in the schedule is a major operation since it introduces new segments which all need to be planned simultaneously. There is a good chance that the PRM could be planned but that this increases the robustness penalty on some segments by the insertion. It could be possible that later after the insertion we could shift some segments to make a more robust solution. The solution where no any more PRMs could be inserted that has a 0 robustness score may be a Pareto optimum solution but not one we would prefer.

A possible way to handle the objective function is by adding a weight to both objective functions and take the sum as score for a solution s. For example we have 2 objective functions, which we denote by $f_1(s)$ and $f_2(s)$, where s is the current solution. We could weigh the objective functions, lets say that w_1 represents the weight of the first objective function $f_1(s)$ and w_2 the weight of the second objective function $f_2(s)$. To decide or a new solution s' is better than s we could compare the values of $w_1 * f_1(s) + w_2 * f_2(s)$ and $w_1 * f_1(s') + w_2 * f_2(s')$ just like we do when using a single objective function. The weights w_1 and w_2 could be chosen such that decreases in the primary objective are much more valuable than decreases of the robustness score. Then when everyone been planned the robustness penalty starts to matter. This method be used to set some preference on what solutions of the Pareto optimum front you prefer to have.

We could also use a variation of dominance called lexicographical optimization[2] to decide dominance of a solution. Lexicographical optimization is used when one objective function is way more important than the other. The idea is that the solution is found in two phases, where in phase one the primary objective function is optimized disregarding the second objective function and in phase 2 the secondary objective function is optimized without worsening the primary objective. Lets say solution f dominates solution g if it either has a better score in the primary objective or when the value for the primary objective is the same and has a better robustness. This way any improvement in the first objective function regardless of our secondary solution is taken, the secondary objective function only matters between solutions with the same value for the first objective functions. We use the lexicographical optimization for our program and experiments. The next subject we are going to look at is whether or not including unnecessary waiting time can improve the quality of the schedule.

2.3 Unnecessary waiting time

The company that transports the PRMs through the airport wants to give good service towards their customers. Besides trying to minimize the declination of service towards the customers they want to minimize the unnecessary traveling time of the customers. When analyzing the situations where unnecessary waiting could occur we see that a lot of those situations could be easily avoided without decreasing the quality of the solution.

First we look at the unnecessary waiting time that occurs when a PRM must be handed over to a different employee or bus. Here there are 2 scenarios. In the first scenario the PRM is handed over to a similar employee in the same terminal. In the second scenario the PRM is handed over to an employee of another terminal or a bus.

In the first scenario where the PRM is handed over to a similar employee in the same terminal any unnecessary waiting could easily be avoided. Since all employees in the same terminal are identical the current employee could do the same segments the employee he is waiting for could do. Therefore to avoid unnecessary waiting the employee could serve the up following segment as well. If we look at the possible journeys in Section 1.2, those handovers in the same terminal only occur when boarding the plane. The journey from the lounge or pickup location and boarding could be easily be done by the same employee, especially if all PRMs boarding the same plane are waiting at the special lounge of the terminal. Bringing and taking someone from the lounge is no handover because the PRM could be seated there safely at all times. Therefore handovers in the same terminal could be easily avoided.

In the second scenario where the PRM is handed over to an employee of another terminal or a bus the waiting time can be avoided by good planning. If we look at the possible journeys in Section 1.2, those handovers are always between a terminal employee and a bus or vice versa. To avoid unnecessary waiting time we must make sure the next method of transportation either a bus or an employee is already at the spot when the PRM arrives. We have unnecessary waiting time if the bus or employee arrives later than the PRM, we can avoid that by shifting the time the PRM visits the bus if possible. Unfortunately we cannot shift the time of platform busses because planes depart and arrive at certain times, therefore PRMs need to be transported by those busses 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 plane 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 plane, 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 till its possible to bring the PRM with a smooth connection. So in this case the unnecessary waiting time could be avoided by allowing flexible pickup times for the busses.

Another observation if you look at the problem is that many times at which a segment must be start are fixed due to the arrival and departure times of planes. For most segments like embarking and disembarking the only choice is who is going to serve it. Allowing a lounge visits before and after inter-terminal bus allows for more choices to be made, because otherwise the PRM must be brought to the bus immediately after arriving by plane. And this allows us to shift the jobs of inter-terminal bus rides to a more favorable time.

Now we look at unnecessary waiting time when taking detours. Again we looked at the possible journeys in sections in Section 1.2. Between boarding and embarking the planes the employees are not allowed to bring additional PRMs, so taking detours is prohibited there anyway. Since the PRM is usually in the lounge before boarding, unless he got to hurry to catch the plane after arriving at the terminal, all PRMs of the same plane start at the same location in most cases. A single employee then could take up to 2 PRMs from the lounge and help them through the boarding process where no detours are allowed.

The rest of the journeys in the terminal are relatively short compared to boarding; from a pickup location to the lounge or gate and from the gate to a drop off location, bus stop or lounge. Since an employee could only bring 1 or 2 PRMs what extra would making a detour do. Suppose we have a PRM at the gate which is picked up by an employee first, followed by bringing him to a pick up location to pickup another PRM and then bring both of them to the lounge. It might be a bit more efficient but you have unnecessary waiting time on the first PRM, and the connection is not even robust, whereas it is only a short walk to the lounge, seat the PRM there and then pickup the next PRM or let some other employee do the work. We win a little time to make detours with these small trips.

Making detours with inter-terminal busses makes more sense if the terminals are far away and you could stop at a terminal in between to pick up someone that needs to go the same terminal as the PRM you currently transport or to a further terminal. But when you look at the distance matrix we see that the distances in minutes between bus stops are 5 minutes in almost every case. Maybe the distances in minutes include the time it takes to load and unload passengers which overshadows the time it actually takes to travel between terminals.

Like explained in section 2.1 we can make the time the PRM visits the bus variable by visiting a lounge before and after the bus ride. We also allow multiple PRMs with the same segments to share the same trip, which should free up some bus capacity. We could avoid doing detours for PRMs and accumulate unnecessary waiting time, by either schedule the bus trip on a time with sufficient capacity or to let as many PRMs as possible travel with the same bus, where all PRMs must travel to the same terminal and get in the bus at the same bus stop.

We have seen in this section that unnecessary waiting time could be easily avoided in most cases. It could be that by allowing a PRM to make a detour, some other PRM that otherwise had to be declined could be served although this only happens when employees have a very busy schedule around the passengers time window. To make the problem easier to solve we could make the algorithm such that the algorithm plans the PRMs in a way there is no unnecessary waiting. This is easier because you don't have to worry about cascading effects of delaying an already planned PRM when doing a mutation or insertion of a PRM in the solution.

2.4 Segment groups

If an employee has finished serving a segment and the PRM needs to be handed over to another employee or bus, we prefer to start the next segment as soon as the employee arrives at the location of the handover. If it starts later then we gain unnecessary waiting time. The only time a segment could start later than the previous segment is finished without gaining unnecessary waiting time is when the previous segment ends in a lounge. After the PRM is seated in the lounge he could be picked up at any time as long he is at his desired destination before his deadline. Hence if you know the start time of a segment, unless it ends in a lounge you know when the next segment must start to avoid penalty.

Using that property we could split the journey in smaller journeys of segments that need to start right after the previous segment in that sub journey has ended. Suppose we have a transfer journey with segments r_1(gate, lounge 1), r_2(lounge 1, bus stop 1), r_3(bus stop 1, bus stop 2), r_4(bus stop 2, lounge 2), r_5(lounge 2, gate) and r_6(boarding). Segment r_1 stands alone and brings the PRM to the lounge. Segments r_2, r_3 and r_4 need to start right after each other due to the handovers and r_4 ends in the lounge. Segments r_5 and r_6 must also be scheduled right after each other and after the last segment the journey is finished. We now got the sub journeys $\{r_1\}$, $\{r_2, r_3, r_4\}$ and $\{r_5, r_6\}$; we call such a sub journey a *segment group*. If we set a start time for the first segment r_2 in segment group $\{r_2, r_3, r_4\}$ then we could set the start times of the other segments in the group such that there is no unnecessary waiting. When you set the start time for segment group $\{r_1\}$ and $\{r_5, r_6\}$, the segments of segment group $\{r_2, r_3, r_4\}$ can be given a start time independently from the start times of the other segment groups without gaining unnecessary waiting time. Of course segment r_2 must still start after segment r_1 has finished and segment r_4 must be finished before segment r_5 starts.

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

This makes the problem a bit similar to the no-wait job shop problem[8], where if you know the start time of the first segment you know the start time of the entire job. Although with our problem you got the option to wait, which is penalized, making it more like a blocking job shop problem[8]. We have explained in section 2.3 that most unnecessary waiting can be avoided, so we will be aiming for zero wait time everywhere. We could even enforce zero waiting time between segments in our schedules.

We use these segment groups for our algorithm. If we want to avoid unnecessary waiting like explained in section 2.3, these segment groups are a useful tool for assigning the start times of all segments of a segment group, we only have to decide when we are going to start this sequence of segments. It also helps with synchronizing journeys when 2 or more PRMs share the same employee, such that they both arrive at the start of their common segment at the same time. We don't have to synchronize the whole journey but only have to look at the segment group of the segment we want them to join.

2.5 NP-Hardness

This Problem is NP-Hard in the strong sense. This can be proven by reducing this problem to the 3 Partition problem. The 3-Partition problem has been proven to be NP-Complete in the strong sense by Garey and Johnson [7]

In the 3-Partition problem we are given a set of $3m$ positive integers $S = i_1, i_2, \dots, i_{3m}$ with a sum of $m * B$. The question is whether we can partition set S into disjoint groups of three such that the integers in each group sums up exactly to B. If every integer in set S is strictly between $B/4$ and $B/2$ then each subset is forced to have exactly 3 elements. For our proof we assume the sets are forced to contain exactly 3 elements by this property. Even with this property the problem remains NP-Complete.

We can reduce our PRM scheduling problem into the 3 partition problem by splitting $3m$ PRMs to m employees. We only need one terminal for this problem where all employees are located, the busses are not needed in this case. For each integer i in the set S we make a PRM who wants to depart on a plane and arrives at the airport at time 0. PRMs who only take a departing plane could wait a bit for the appointed time to start their journey, while PRMs arriving per planed needs to be picked up immediately. All PRMs are in wheelchairs to enforce that an employee could only give service to no more then 1 PRM at a time. The route of the PRM consists of the following segments: (Pickup, Lounge), (Lounge, Gate) and (boarding). The pickup locations are chosen such that the time it takes for the employee to walk from the lounge to the pickup location and bringing the PRM back to the lounge is exactly i minutes, representing the integer value of set S. For the trip to the gate and boarding, all PRMs representing set S are identical. We choose the departure time of the plane such that the PRM must be seated in the lounge at or before time B.

We want to have all $3m$ PRMs to be seated at the lounge at time B by m employees. An schedule till time B of an employee will looks like followed:
- The employee starts at the lounge at time 0.
- The employee walks to the pickup point of PRM 1.
- The employee brings PRM 1 to the lounge, we are at time i_1 now.
- The employee walks to the pickup point of PRM 2.
- The employee brings PRM 2 to the lounge, we are at time $i_1 + i_2$ now.
- The employee walks to the pickup point of PRM 3.
- The employee brings PRM 3 to the lounge, we are at time $i_1 + i_2 + i_3$ now.

The total time it takes to do those 3 PRMs is $i_1 + i_2 + i_3$ which should be equal or less than B otherwise the PRMs won't make it to the deadline of the flight. If all $3m$ PRMs could be planned over m employees, every employee serving those PRMs should be occupied till time B.

However because every PRM needs to catch the plane at the same time, we require $3m$ employees to board them all simultaneously while we must have m employees to seat them before or at time B. We assume here like in the data all employees in the same terminal are identical and have the same shifts. Therefore we must occupy the other $2m$ employees with a task such that they are only available after time B. We add $2m$ additional PRMs who arrive at the airport by plane to occupy the $2m$ employees. The locations and arrival times are chosen such that the employee returns at the lounge at time B and couldn't possibly help the PRMs representing the integers from set S. Now we only have m employees remaining to help the $3m$ PRMs representing the set S to get to the lounge until time B, and $3m$ employees to board them in a plane. We know we got a 3 partition if every PRM in the problem is planned, otherwise we couldn't find a 3 partition.

If we got a polynomial time algorithm for our problem we could solve the 3-Partition problem where the sets are forced to be triples in polynomial time too by using this transformation. But since the 3-Partition problem is NP-Complete even with this restriction our problem should be at least as hard. Therefore our problem is NP-Hard.

2.6 Decomposition model Local Search and Match Making

Since this problem is NP-hard and got many constraints that make solving difficult, solving this problem to optimality within 2 minutes isn't likely to be possible. That is why we have chosen to use a local search approach like Reinhard et al[1]. They have used a model where they do local search on the order of the lists of booked and unbooked PRMs, they make a planning out of that by using an insertion heuristic to insert the PRMs one by one by the order of the list.

We want to use a decomposition model. The layout of the airport, distance matrix between locations, employees, PRMs and which segments the PRMs have in their journeys are given as input. We are looking for a feasible start time and employee assignment for each segment unless the PRM is declined from service. The first step in the decomposition model is to mutate the start times of the segments and then use an algorithm to find a feasible assignment of employees to the segments given the start times; we refer the algorithm as a matching algorithm since it matches the segments to workers. If there couldn't be found a planning using the given start times then the solution in the iteration of the decomposition model is considered unfeasible. The matching method could hopefully give us some feedback on what could be good mutations on the model.

There are various ways local search and matching could be done. For local search we are using Simulated Annealing along with mutations. For the matching problem we are going to test a few possible algorithms. Some methods construct an entirely new schedule out of the input start times, others just update the schedule using the changes given. The matching methods are described in the sections of the respective algorithms they are used in.

The first algorithm we discuss is a product of miscommunication and is slightly flawed, but it still worked quickly and gave us some insight and is therefore listed in this paper. The second algorithm is based on the findings of the first one and the programming is more flexible allowing more kinds of matching algorithms. We are going to discuss the first algorithm first and then discuss the second one.

3 Algorithm 1: Fix times and then match.

In this section we describe our first attempt to make an algorithm to find a schedule for our problem. For each PRM we plan in the schedule we must assign feasible start times and employees. We want to solve our problem using a decomposition model where we first assign start times to every segment using local search. Using these start times we then want to construct the schedule by assigning an employee for every segment if there is sufficient capacity on the employees at the given time. If a segment of a PRM fails to be scheduled then the PRM is declined of service. After assigning the segments to employees the algorithm is done and don't try to reschedule start times; this is one of the flaws of this first attempt.

3.1 Local search

The goal of the local search is to assign start times to segments without having to worry about employee assignment. We wanted to ignore some constraints for the local search to make solving more easily and only those constraints are considered in the matching algorithm such that the solution is feasible.

We make from each terminal a resource / machine with a maximum capacity equal to the total capacity of all transports in that terminal/area. In that resource segments could be planned at a certain time where multiple segments may be overlapping each other as long as the total capacity at every given time doesn't exceed the area's maximum capacity. The resources are used in the first phase to assign start times, while making sure there is enough capacity.

Using the resources we get some idea wether or not there is enough capacity at a given time without the need to make an actual schedule. If there isn't enough capacity in this problem with relaxed constraints for capacity and traveling between locations to start segments, then there is certainly not enough capacity in the actual schedule with all constraints.

Because the employees can transport up to 1 or 2 PRMs depending on the PRMs disability, in the schedule an employee could possibly only need to transport one PRM while he have some leftover capacity. Also it takes some time for an employee to walk from the one job to another, which isn't covered by the resource.That is why we aim to distribute the workload by having some excess capacity at every given time. This hopefully adds some robustness to the schedule.

This model will use a penalty for declining a PRM, where the penalty of prebooked PRMs are higher. We wanted to add some robustness in the model so we also gave a penalty of serving many PRMs at a certain time, where the penalty increases non linearly with the number of PRMs at the same time. Using a linear penalty function would result in a constant that is only dependent on the number of PRMs you plan. We used the formula $c * c * t$ where c is the capacity of the time window and t is the time length of the time window. We made sure that the declining penalty would exceed the penalty for robustness such that solutions where all PRMs are scheduled are preferred, followed by a planning where there is excess capacity left in the area's in case a new unbooked PRM arrives.

We used simulated annealing as local search method using the following mutations: plan PRM, decline PRM and plan segment group The mutations are focused onto giving each segment group a start time. Once a segment group got a start time, it sets the start time of each segment in the segment group such that there is no waiting time between each subsequent segment. If we didn't do that we would have to pay the penalty for letting the PRM wait. The mutation is infeasible if it violates the capacity constraint. And since in this phase we only assign start times and not who is going to handle the PRM, we assume that a transport is available at that time, if not that start time would be infeasible and the PRM isn't planned. In the 2nd phase, the matching phase, of the algorithm we assigning a employee for each segment.

Mutations:

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	Attempt to decline a random served PRM and remove it from the planning.
Plan Segment Group	Attempt to plan a random planned segment group with a random start time inside its time window.

Table 1: Mutations

3.2 Matching

The goal of the matching algorithm is to assign employees to segments, given the start times of the segments. There are multiple ways to do the matching; we explain here what methods to be used for this algorithm and some ideas behind the matching. In our model at the moment we assumed that each PRM doesn't have to undergo unnecessary waiting time and can be taken over immediately and doesn't take any detours. The option to make detours and let PRMs wait was a feature to be added in the future, but that this algorithm was flawed and we could better do a matching every iteration instead of one time at the end. This is fixed in the second algorithm which does have a better framework to work with.

3.2.1 Matching heuristic: First Available

This matching method is more like a greedy heuristic than an exact approach. First it sorts all the segments to be planned in order of start time. Then it goes through each segment in that sorted list, with the lowest start time first and finishing by the segment with latest start time. The algorithm checks what area the segment is in and looks for available employees or buses in that area that can service that segment. If multiple transports are available, then the one is picked who has the most slack, which is defined as the time between finishing the previous segment and starting the to be inserted segment minus the travel time to get to the location of the to be inserted segment. The higher the slack the more likely in a dynamic world where time is uncertain he is going to make the segment. In case there are no transports that can serve the segment, the algorithm then declines the PRM of that segment.

There are however two exceptions on the rule of picking the one with the highest slack. In the journey the segment to the gate and boarding at the gate are 2 separate segments in the same terminal that can be easily done by the same transport because that is more robust like explained in section 2.2. The second exception is when segments two or more PRMs could share the same employee or bus, like explained in section 2.1. In this algorithm we don't have mutations that try to synchronize the journeys such that a segment with similar start and end locations or 2 PRM could be joined together. However since some PRMs have to catch the same plane and board on the same time, those segments are forced to synchronize because of the boarding which happens at a specific time. It could also be a coincidence that some other segments with the same start and end location start at the same time and might be able to share one employee if the employee's capacity allows it.

This method has some downsides. First it is not specified how to handle the case if there is not a transport available but within a small time frame there is. (More on that later) Secondly the pre-booked PRMs do not get preferred, however that can be fixed easily to try to delete a segment of an unbooked PRM and plan the booked PRMs segment instead. And third the choice of who is going to serve the segment may not be the best choice in some scenarios.

A case where this goes wrong is one where we need to plan 2 segments r_i and r_j (Figure 4). Both employee k_1 and k_2 can service segment r_i however while employee k_2 haves a slack of 8 the 10 slack of employee k_1 is better and get chosen. Then segment r_j is been looked at, employee k_2 can't make it to the segment in time because he is far away from the location of r_j and k_1 can't service r_j because he is already serving r_i at the same time. So following from the algorithm we can't serve both of those segments and hence it will decline the PRM of r_j. However if k_1 didn't serve r_i he got plenty of slack to serve r_j. A better solution would be that r_i get served by k_2 and r_j get served by k_1, such that both of them can be planned.

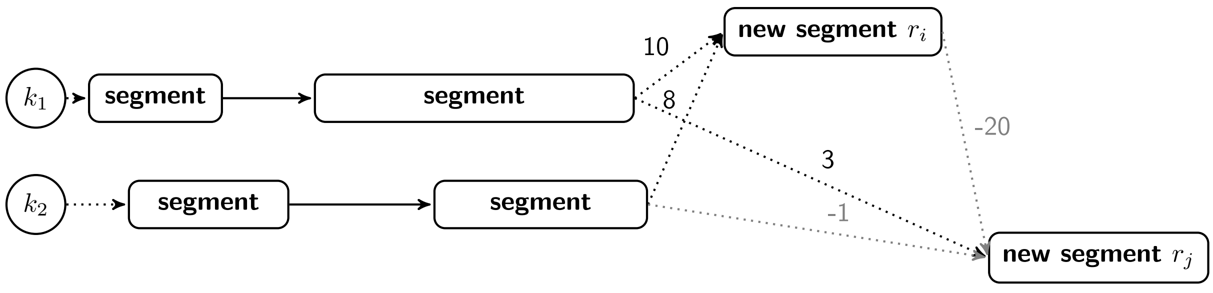

Figure 4: An example of how 2 segments could get in each others way while been planned. The values on the dotted edges represent the slack of that connection where the higher slack is better. A negative slack means that the employee can't be on time for the inserted segment. If segment r_i is planned first it would prefer employee k_1 but then segment r_j could not be planned.

3.2.2 Matching heuristic: Reschedule moving window

This matching method tries to solve the problem that arises when some bad choices are be made when assigning a transports to a to be inserted segment. This method uses the same insertion method as the previous heuristic but when a conflict arises such that a segment couldn't be planned, it uses another heuristic to attempt to plan that segment. The heuristic uses a combination of rescheduling segments which competes for a slot with the to be inserted segment and assignment of multiple PRMs on an employee at the same time.

This method like the previous one first sorts all segments in order of start time and go through them one by one. This method also keeps track of a list R_{dec} of PRMs that has recently been declined because a segment couldn't be scheduled. It first tries a simple insertion like the previous method but when that fails instead of deleting the segment we try to reschedule the last inserted segments for each employee in the terminal or bus site along with the new segment.

When the to be inserted segment r_i couldn't be planned, the algorithm first remove all last scheduled segments from the transports of the area of r_i and stores them in a set A. Then we look at recently declined segments in that area or they could share a trip with the new segment. Its better to decline one PRM who was already in the schedule, then two PRMs who could share a trip. After that we go through the list of R_{dec} and look for a segment r_j with the same start time, start location, end location and end time and that the combined capacity doesn't exceeds that of a transport in that area. If there exists such a segment r_j then the 2 of them could be planned together and we merge those 2 segments into one and store it in A while removing r_j from A. If the merged segment then gets assigned to an employee, all segments of the same merged segment are planned on that employee. If none such segment exists then segment r_i is added to A.

The segments got a priority of been planned, we want to have prebooked PRMs to be prioritized above unbooked. Also we want to prioritize merged segments that serve multiple PRMs, and extra priority is given if one of them is prebooked. We define the function that gives out priority as $prio(r)$ where r is an segment and returns an value representing its priority.

What is left is a matching problem where we try to match each (merged) segment to a transport. This is a know problem named Maximum bipartite matching[3]. In the Maximum matching problem there is an edge between a segment and a transport if the transport could service the segment at his in the local search set start time. If we give each edge towards a transport the weight representing the priority of the segment we get a minimum costs maximum matching problem. Also since the graph is bipartite (the segments are matched to transports), there is a polynomial algorithm to solve it.

Lemma 3.1. *If a to be inserted segment r_i could be merged with a declined segment r_j in R_{dec}, its better to decline another segment and merge segments r_i and r_j such that they could be executed on the same employee.*

Proof.

1. We know that r_i couldn't be planned in the schedule using the matching algorithm: First Available.
2. We know that r_j in R_{dec} couldn't be planned using the matching algorithm: Reschedule moving window.
3. We know that r_i and r_j could be merged into a segment r_m.
4. Following from rule 2 and the definition of the matching algorithm we know that all planned segments in the current window W have a priority equal or higher than r_j.
   $\forall r_l \in W :: prio(r_l) \geq prio(r_j)$
5. Because segment r_m is a merge between r_i and r_j, $prio(r_m) > prio(r_i)$ and $prio(r_m) > prio(r_j)$.
6. Following from rule 5 it could be possible that there exists a planned segment $r_l \in W$ such that $prio(r_l < prio(r_m)$.

□

We translate our problem into a minimum costs matching problem where we match employees with to be scheduled segments in set A, which contains all segments who are competing to be scheduled at the time. There is an edge between the employee and segment if the employee could serve the segment next. The costs of each edge between an employee and segment represents the priority of the segment, higher priority are given to segments that are shared with multiple PRMs and when the employee served was serving the previous segment of the PRM.

After solving the resulting problem we translate the found matching back into the planning. Each pair of matched segment and transport is put back in the schedule while the unmatched segments are returned to the set R_{dec}. Note that for each PRM only one segment is in the set R_{dec} namely the latest that is attempted to assign, if a segment of a PRM needs to be scheduled and the previous one is declined then the algorithm stops trying to insert that PRM into the schedule and declines service. After all all segments with a lower start time are already scheduled or declined and the previous one still couldn't be planned or merged with another.

This is an improvement on the insertion Heuristic because it could plan segments that would otherwise be declined due a poor choice of employee assignment of a previous segment like the example in Figure 4 in Section 3.2.1. The Aproach Reinhardt et al [1] are using is also a Greedy Insert Heuristic that tries to insert a segment somewhere in the schedule within the time window with as less unnecessary waiting time as possible, but as far as we know it doesn't reschedule previous planned segments to other employees. We hope this leads to an improvement over the original algorithm.

3.3 Experiments

We have run some experiments with this algorithm using a Windows 7 Ultimate Computer with a Intel(R) Core(TM) i7 CPU 2.80 GHz quad core and 4 GB of RAM. The used instances are provided from Line Blander Reinhardt one of the authors of the paper this thesis is based from. It looks like they are the same instances they used for their paper but adjusted to not reveal sensitive data. We view a certain airfield with 11 instances of PRMs requesting services and available workers of that day. Unfortunately its not listed whether or not the PRMs have prebooked so we have to handle them all the same. It doesn't matter if they are prebooked or not if we manage to schedule them all which is our goal.

We test both variants of the matching strategy while the local search stays the same. First we show the results of executing the program with deleting the conflicting PRMs then we show the results of the program with optimizing conflicting PRMs. Each instance is run 20 times and the averages, minimum and maximum values for computation time, PRMs declined in the local search and PRMs declined in the matching. Each run has 1000000 iterations and begins with an empty solution where no PRM or segment is planned.

The results of the experimentation runs are displayed below. We tracked 3 statistics: the computing time of the scheduling algorithm, the number of declined PRMs in the local search before applying the matching heuristic and the total number of declined PRMs. Of each statistic we listed the minimum value, maximum value and the average value of the statistic over 20 runs. Since we did not know which PRMs are prebooked we couldn't make distinction between declined prebooked PRMs and those who reserved during the day like Reinhardt et al [1] did.

Results: Local search + Matching: First Available

	time(ms)			**declined before matching**			**total declined**		
	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
Instance 01	2855	3192	2947	0	0	0	0	1	0
Instance 02	3058	3338	3135	1	1	1	1	3	2
Instance 03	3083	3198	3198	0	0	0	1	3	1
Instance 04	3010	3089	3036	0	0	0	4	10	6
Instance 05	3151	3229	3187	0	0	0	8	10	9
Instance 06	3152	3276	3198	0	0	0	2	5	3
Instance 07	2995	3058	3020	0	0	0	0	2	0
Instance 08	2995	3074	3037	0	0	0	1	4	2
Instance 09	2995	3058	3027	0	0	0	1	3	1
Instance 10	3120	3214	3157	0	0	0	3	7	4
Instance 11	3213	3292	3245	0	0	0	2	8	5

Table 2:

Results: Local search + Matching: Reschedule moving window

instance	**time(ms)**			**declined ls**			**total declined**		
	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
Instance 01	2823	3042	2886	0	0	0	0	1	0
Instance 02	3042	3276	3095	1	1	1	2	3	2
Instance 03	3042	3167	3096	0	0	0	1	3	1
Instance 04	2979	3198	3024	0	0	0	2	8	4
Instance 05	3151	3229	3190	0	0	0	8	10	9
Instance 06	3135	3214	3166	0	0	0	2	5	3
Instance 07	2948	3026	2988	0	0	0	0	1	0
Instance 08	2964	3057	3001	0	0	0	1	4	2
Instance 09	2949	3027	2993	0	0	0	0	2	1
Instance 10	3057	3214	3108	0	0	0	4	7	4
Instance 11	3167	3261	3212	0	0	0	3	6	4

Table 3:

3.4 Discussion

When looking at the total declined we see a minor improvement of total declined in the *Local search + Mathcing : Reschedule moving window* algorithm over *Local search + Matcihng : First Available* algorithm. There are sill some declines, with only 2 instances where we could schedule every PRM. However these results doesn't beat the results of Reinhardt et al [1] where often everyone was planned after 2 minutes. It beats the initial solution of them though, but that is not good enough. Our algorithm is better in the sense that PRMs have no unnneccesary waiting time, which Reinhardt does. But as long we got these declines, its not better.

Although this algorithm is fast and only takes around 3 seconds to compute such that there is a lot of time we could improve the solution, the algorithm gets stuck at a local optimum. The problem is the local search method finds a good solution for the constraints given to them where with all runs except with one instance every PRM gets planned. But then using the planning heuristic, some PRMs get declined in progress. The planning heuristic couldn't give feedback to the local search to find a better solution, therefore this algorithm is flawed.

We feel like we are in the right direction with sharing employees, avoiding unnecessary waiting time and rescheduling already planned segments in the planning heuristic. We used that as base for our next algorithm that constructs a schedule every iteration instead of at the end of the local search.

4 Algorithm 2: Local Search and match

Our first algorithm had some flaws, it didn't perform better than Reinhardt et al [1]. The biggest issue was that the actual schedule got constructed using a heuristic after fixing start times and then didn't provide feedback to the local search to find a better schedule. While the local search had a good solution for its subproblem, after using the matching heuristic it declined some PRMs and then stops. After we used the matching heuristic we should give feedback to the local search to find better solutions. The algorithm described in this section does that. After each iteration in the local search we now apply a matching heuristic and score the mutation based on the result of the generated schedule.

4.1 Local Search

Like the previous algorithm this one uses local search as basic, in the local search we try to set the start times for each PRM by mutation. The difference from the previous algorithm is that after each mutation, an actual schedule get constructed using an matching algorithm. The matching algorithm match up segments with available employees. Also the matching algorithm tries to assign employees such that the schedule is as robust as possible. In this framework both mutations and matching algorithm are changeable so some more experimenting with those can be done.

We also wanted to add the option that there could be some time between the end time of the segment and the time the PRM could get picked up by the next employee, hence resulting in unnecessary waiting time. But since that is inconvenient for the PRM and employee at the same time we decided to put that in as a later option if it has proven to be necessary. Our first mutations and matching algorithm are assuming that the PRM could be planned such that each subsequent segment if not starting in a lounge could be started immediately without having to wait for an employee. Meaning that if there isn't directly an employee available at the time of handover, the segment couldn't be planned. In section 2.3 we have seen how we can avoid unnecessary waiting time. The mutations and matching algorithm assume we can avoid unnecessary waiting time completely, so the planning returned by this algorithm won't have unnecessary waiting. Not allowing to take detours or waiting for an employee to pick up the PRM makes matching more easily since a mutation then does not have to delay some already planned segments resulting in a cascade of changes through the planning.

The framework supports restoring to a previous state. After accepting a mutation all affected objects store their new values in their restoring space. Then when we decline a change we restore all affected objects to their old stored values. Because the mutations could possibly create a complex shift in the order of segments and assigned employees over a few areas it is really hard to predict the change in objective score before executing the physical changes. For example when inserting a PRM you insert several segments in the planning spread over possibly several area's and affect a number of employees in the area to change their route so the segment could be inserted robustly.

4.2 Sharing trips

Like our first algorithm we split the route of each PRM into segment groups like explained in section 2.4 and in section 2.1 we saw that synchronizing journeys to allow PRMs to share an employee or bus in their journey is good for freeing up employees for other tasks. In this section we explain how we generate pairs of segment groups (g_1, g_2) of different PRMs that could share an employee or bus on at least one segment of g_1 with a segment of g_2. We then use the properties of the segment group 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.

For our algorithm we construct a list M of possible pairs of segments that could share an employee together. That list is used by mutations of the local search to synchronize the journeys of the PRMs such the segments of the pair could share the same employee or bus. We say that 2 PRM's p_1 and p_2 could travel together if there is at least one segment pair (r_1, r_2), segment r_1 from p_1 and segment r_2 from p_2, where the segments r_1 and r_2 have the same locations as source and destination and could be planned simultaneously without violating the release and deadline of neither PRM. Also the sum of the capacity requirements of PRM p_1 and PRM p_2 must be no more than the capacity of an employee or bus that must serve the combined segment. The segment pair (r_1, r_2) is then a member of list M.

We see segment-groups as individual journeys from a start location to an end location with a separate release times and deadlines where all segments should be planned right after the previous segment in the segment group has finished. That makes segment groups a great tool to synchronize journeys of PRMs such that segments could share the same employee. We can synchronize the segment groups of a pair (r_1, r_2) in M if we choose to use that combination in the schedule to share a trip. All segments following and before segments r_1 and r_2 in the same same segment group as segment r_1 or r_2 are bound to the feasible start times of the combined segment r_1 and r_2 if you wish to avoid unnecessary waiting time. Therefore if we plan 2 segments (r_1, r_2) simultaneously on the same employee we must also plan the rest of the segments in the segment groups relative to the merged segment of the pair (r_1, r_2). We could say that the 2 segments groups when synchronizing their journeys are merged in one entity with his own release and deadlines, a merged segment group. After assigning a start time on one of the segments of such merged segment group all segments are dependent on those start times if you wish to avoid unnecessary waiting time.

Because we are matching segment groups it could be possible that there are multiple pairs of segments (r_1, r_2) that could share a bus or employee between the same 2 segment groups of different PRMs. For example when 2 PRMs go from the lounge to the same plane, they could share the trip to gate, boarding and possible the platform bus together. If we merge the segment groups such that a matching pair (r_1, r_2) share an employee or bus we also look wether the other segments could share an employee or bus at the same time. In our example if we then want to use one of the pairs (r_i, r_j), the boarding segments for example we also let the other segments share the same employee or bus when possible. Therefor we only need one of those segments as combination in M, the others are redundant since they result in the same merged group and shared trips.

4.3 Mutations

For this algorithm we have come up with several mutations for our local search. Some of them you have already seen in the first Algorithm and some of them are new and focus on matching PRMs to travel together with the same employee or bus.

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	Attempt to 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 as described in section 4.2 and attempt to merge them. If either PRM 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 4: Mutations Algorithm 2

4.4 Generating the Schedule

After each mutation a schedule is generated for all currently accepted PRMs. For this subproblem the start times of all planned segments are set so we only have to assign which segment is served by which employee. 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 schedule the mutation is considered infeasible and the changes are reversed. The objective is to find a schedule with all segments planned and optimizing the robustness score and give that as feedback to the local search who decides whether or not to accept the solution.

Since we avoid having unnecessary waiting times by disallowing detours and enforcing availability of employees for subsequent segments when they end, we don't have to worry about that. This problem result to be a matching algorithm where we assign segments to employees and minimize edge costs for robustness. The details of the matching algorithms are explained in their respective sections. We tried both tried optimizing the entire schedule and insertion heuristics who updates the schedule more locally but are faster.

Because 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 we design our algorithms around that. However in the problem description its not mentioned that all employees should have the same start and end of the shift. In an airport planes are arriving and leaving early in the morning and late night, so its likely there are shifts employees takes. That is why in this section we also will consider extensions where its possible to let employees have different shifts we call that the shifts variant.

Unfortunately while our subproblem with identical employees and busses represents a single depot vehicle scheduling problem[9] which can be solved in polynomial time, the problem where employees and busses could have different shifts represents a multi depot vehicle scheduling problem[10], which is NP-Hard. Therefor the problem gets much harder. We first focus on the matching methods for the variant without shifts, show some ILP models that can solve the different shift variant and then propose a greedy heuristic to overcome this obstacle.

4.4.1 Matching Algorithm without shifts

For our subproblem we know the start times of every segment in our planning and got to find an schedule such that every segment of a planned PRM is assigned to an employee. In this section we talk about the variant of our subproblem where the shifts of every employee are the same. One of the methods we used in our approaches is transforming the sub problem into a weighted maximum matching problem in a bipartite graph. This problem can be solved using the Hungarian Algorithm in $O(V^3)$ time[11], where V is the number of vertices and E is the number of edges. If it wasn't for the robustness we could have transformed it into the maximum matching problem which is solved easily in $O(VE)$ time [3]. This section explains how to transform our sub problem into a bipartite graph and get an route from it.

We make a directed graph $G = (V, E)$ representing our solution, the nodes represents the tasks and the edges represents what tasks follow can be planned after each other on the same employee or bus. Edge (n_1, n_2) says that the segment of node n_1 can be planned before node n_2 on the same employee. A path in the graph could then represent the route an employee takes during his shift. Remember in this subproblem we know which segments are planned and the time they should be served by the employee. Therefore there is a strict order in which nodes can be processed by the same employee or buss.

A shift of an employee begins and ends at the depot and in between he serves the segments assigned to him. This creates a path from s to t where every node n_i represents a location that the employee must visit to serve the segments. In the path every node n_i should have a successor and a predecessor, except for s and t representing the start and end location. We also know the exact time for each node n_i should be executed in the path, such that each node have a disjoint set of possible predecessors and successors. This is also known as a single depot vehicle scheduling problem[9].

For each employee e we add segment nodes s_i representing the start of the shift and t_i representing the end of the shift. Although all shifts of all employees in the same terminal are the same, they still can't serve PRMs that might fall outside their shifts. Those PRMs who got segments that must be executed outside of the employees shifts are presumed to be errors in the data set. Edge (n_i, n_j) says that segment i could be planned directly before segment j without violating constraints such that j is a possible successor of i. The weight of the edge equals the robustness score which says how efficient the connection is, where a smaller score is better than a higher.

To make this problem bipartite we have to split each node n_i into 2 nodes n_i^s representing the arriving edge and n_i^t representing the departing edge. We transform the edges such that n_j^t gets all arriving edges of n_j transforming each edge $(*, n_j)$ into $(*, n_j^t)$ and n_i^s gets all departing edges of n_i transforming each edge $(n_i, *)$ into $(n_i^s, *)$. The dummy nodes doesn't need to be split up because nodes representing the start of the shifts shouldn't have predecessors and nodes representing the end of shifts shouldn't have successors. See Figure 5 for an example how a graph could be transformed in a bipartite graph.

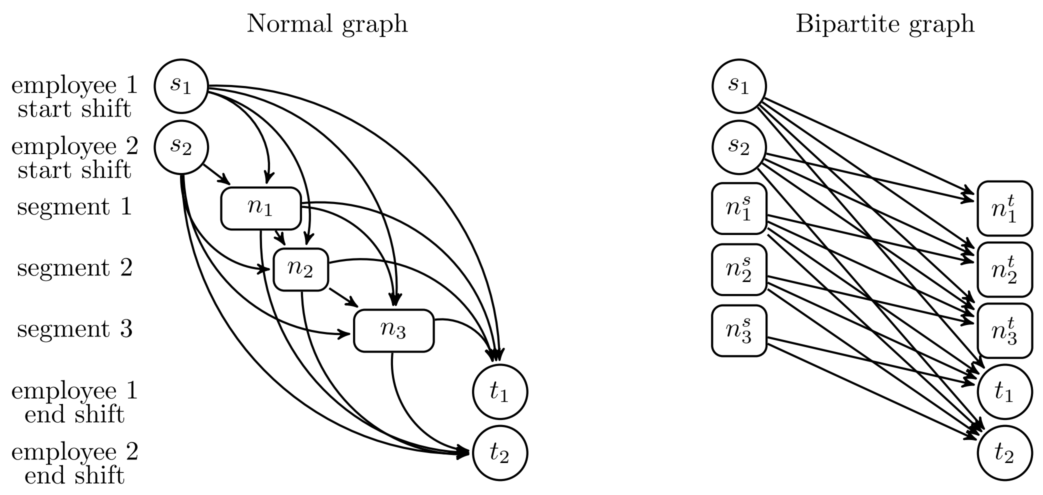

Figure 5: An example of how a normal graph can be transformed into a bipartite graph by splitting up the nodes. The segments are ordered in order of start time. An edge (n_i^s, n_j^t) presents that n_i^s could be planned before n_j^t by the same employee without violating any constraints.

The method only works because there are no cycles in the graph such that we could guarantee paths starting and ending at the employees depot. Otherwise it would be very similar to the Hamiltonian path problem which is NP-Hard[12]. Because the arcs has to preserve feasibility and cannot connect to a node that is planned earlier in time the chain of segments can only progress forward in time and never backwards, hence there are no cycles. Its another story if we have segments with length 0, which is very unlikely. If we have those segments that can planned on the same moment and location it creates a cycle in the graph because it doesn't matter which segment gets served first. That case could be caught by giving those segments a virtual order and don't allow a successor of a same segment with length of 0 and a distance of 0 with a lower ordering.

After solving this weighted maximum matching problem we get a set of Edges used for the matching. We know there is a feasible matching for this subproblem if we find a perfect matching: every node is member of exactly one matched edge. Otherwise there would be a segment who doesn't have a successor or predecessor and wouldn't be a member of a planning. If we got a perfect matching we could translate the edges back to the original graph and get a solution where each path starts with a start shift segment of an employee and ends with an end shift segment of an employee, those start and end segment doesn't necessary have to be from the same employee. Since there are no cycles in the graph and every nodes excepts the start shift have a predecessor and every node except the end shift have a successor the routes must start and stop using the start shift and end shift nodes.

This method is not fit for the problem where employees could have different shift starts and ends. We can't force that each route starts and ends with a segment from the same employee because every employee's end shift node that is a feasible connection from a segment could be matched even though if the employee start segment earlier in the route should be matched with an shift end that is scheduled at an earlier time than the selected end shift edge. Which could cause that employees get segments matched that is outside of their shift time. This makes this method not fit for problems with shifts to solve it to optimality. After all while the variant with identical shifts represents a single depot vehicle scheduling problem[9], the variant with different shifts represents a multi depot vehicle scheduling problem[10] which is NP-Hard.

If we wish to use this method for the different shift variant we propose a greedy heuristic to fix routes that violate shift constraints which is explained later on. First we are going to explain which Heuristics we used for our experiments, then we talk about why the shift variant is hard and show the greedy heuristic.

4.4.2 Insert Heuristic: Free Spot

This is one of the most simple Insert heuristics. It might be simple but it got its uses, we can compare this strategy with those of our more advanced heuristics or use this method if we may not change the schedule too much. Since its an heuristic it doesn't solve this subproblem to optimality and could lead to false infeasible but have fast results. From the previous generated schedule it only plans the segments who are added with the current mutation that calls this matching method. The algorithm doesn't touch the previous assignments of employees to segments.

The heuristic plans every segment that are added with the current mutation one by one. It just goes through the list of available employees and check whether or not they could serve the to be inserted segment. Among the available employees the one is taken with the least lost in robustness in the schedule. The robustness value is calculated using the predefined formula for calculating robustness score between the previous segment and the current one. Then the segment got planned with the employee it gets the best robustness score from.

This method is even suitable for problems including shifts. Since it only inserts a segment in the route it doesn't change the employees end shift segment and the shift constraint is not violated.

The downside of this method is that it might be too simple. For example when solving the traveling salesman problem, using insertion heuristics alone doesn't necessary give good solutions in local search because to swap an entire section which might be better than takes many mutation of deleting and reinserting the node. A method to fix that is adding 2-opt or 2.5 opt to overcome this. We could either add that as an additional mutation or integrate it in the scheduling heuristic which is done in the next Insert Heuristic using the bipartite matching problem.

4.4.3 Insert Heuristic: Reschedule overlapping segments

This method is an improvement over the Insert Heuristic: Free Spot and allows more possible connections to be made while only mutating the schedule partially. Like the previous heuristic the base solution is the previous schedule and only the newly added segments are processed and included in the schedule. It also inserts the to be inserted segments one by one and doesn't necessary provide an optimal answer.

This method is based on the assumption that of the current solution all subpaths of all employees in a certain timeframe are locally optimal, just like every subpath of a shortest path is a shortest path as well. We can assume that a path of segments till a certain node in the journey of a single employee must serve also have these properties, otherwise we would be able to find a better path. When we look for adding a new segment r to the schedule we can speed up the algorithm to only optimize a subset of all nodes instead of the entire schedule. This might not result in an optimal solution, but not a bad one either.

We are only going to reschedule the segments that can't be planned either before or after segment r without violating the constraints. We call such segment an overlapping segment. The problem is then transformed to a weighted maximum matching problem like in section 4.4.1. This method will make 3 sets of segments R_{source}, R_{sink} and $R_{overlap}$. Instead of the shift start of an employee as source node for an employee we use the last node that is already planned on the employee that can be planned before segment r as source and to represent the employee and is put in set R_{source}. Due the shortest path properties, the schedule before that node should be already a good one. Instead of the shift end as sink node for an employee we use the first node that is already planned on the employee that can be planned after segment r as sink and is put in set R_{sink}. Again the path after the sink node should be already a good one due the shortest path properties. The rest of the nodes between the source and sink of an employee are rescheduled along with segment r and is put in set $R_{overlap}$. Last we also put segment r in set $R_{overlap}$ such it represents all segments that needs a new assignment.

We use the 3 sets R_{source}, R_{sink} and $R_{overlap}$ as input for the weighted maximum matching algorithm to generate a schedule like in section 4.4.1. The conversion to a bipartite graph $G = (V, E)$ is the same as explained in that section. The segments represents the nodes, the edges possible connections and the robustness score is used as a weight for the edge. The only differences are that only a subset of all nodes are been scheduled to optimality and different start and end nodes are used for the employees.

We are basically cutting the routes of the segments on the spot where the to be inserted segment r could be planned in 2 and disconnects every overlapping segment like illustrated in Figure 6. Then we want to schedule all segments in $R_{overlap}$ including r such we got routes for all employees again. This allows us to rearrange the segments that are competing to be scheduled as well as allowing a n-opt like operation. The head of the route of an employee doesn't necessary have to be matched to its tail. For example now it could decide to take a tail of employee e_1 plan segment r and then continue in the tail of e_2 if that is more efficient.

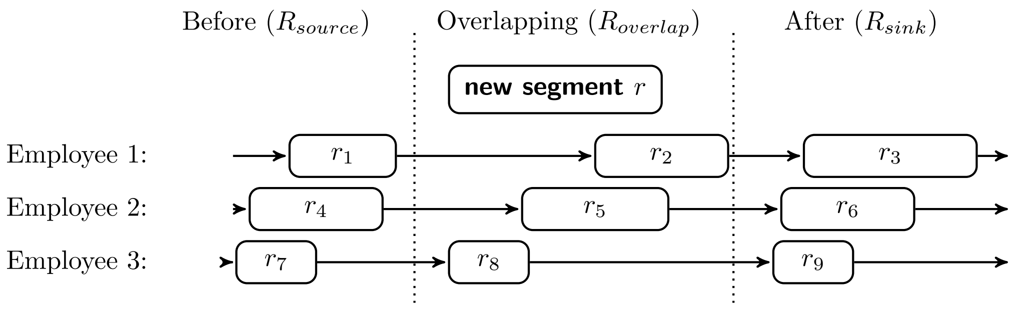

Figure 6: An example of how the reschedule overlapping segments heuristic can find a better robust answer using matching. We show the schedule of 3 employees and the path they take. The edges represents the current path of the employee. Segment r is to be inserted, and below the paths of employees 1, 2 and 3 are given, where segments s_1, s_2 and s_3 are currently scheduled on employee 1, segments s_4, s_5 and s_6 on employee 2 and segments s_7, s_5 and s_6 on employee 3.

Now we got a matching problem where we want to find which segments could be planned after each other just like in section 4.4.1. We could use the same algorithm to solve this problem only we try to match the segments in $R_{overlap}$ while using the segments in R_{source} replace the function of the employee shift start segments and segment nodes in R_{sink} replaces the function of the shift end segment nodes. We are basically cutting down the graph to make a more smaller problem and therefore winning valuable computing time.

Figure 7 shows the edges considered in this subproblem for the example in Figure 6. We show the schedule of 3 employees and the path they take. The dazed edges represents possible connections while the closed arrows represents the current path. In this example the new segment r couldn't be planned using the insert heuristic. But if you plan $r_7 \rightarrow r_8 \rightarrow r_2 \rightarrow r_3$ you get room to plan $r_1 \rightarrow r \rightarrow r_9$ and leave $r_4 \rightarrow r_5 \rightarrow r_6$ alone you get a feasible solution. It might be not the optimal solution but we haven't given the edge costs either in this example. We have done an n-opt and a insert at the same time.

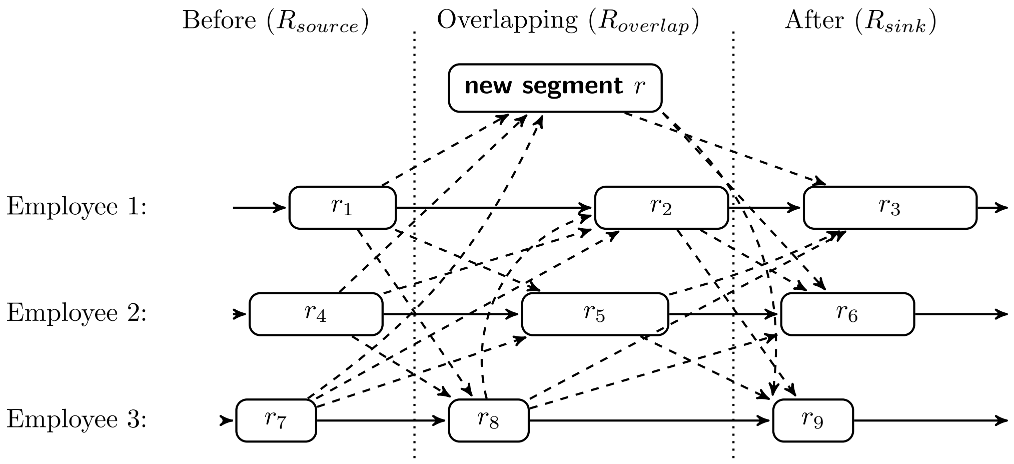

Figure 7: An example of how the reschedule overlapping segments heuristic can find a better robust answer using matching. Showing the exact same situation as in Figure 6 but then showing the possible edges for the matching problem as dazed edges.

We can solve this subproblem using the hungarian algorithm like in section 4.4.1, such the subset of segments are planned to optimality. The solution might not be an optimal matching considering all segments in the schedule, even though the path before the source segments and sink segments are optimal in the previous solution. For example to make room for the new segment r, 2 nodes considered to be sources in this problem might be more efficiently planned to a single employee to make room for a segment r_i not r in set $R_{overlap}$. That segment might not be able to be planed simultaneously with a source node. While this heuristic might not be optimal it should result in good solutions with little computation time allowing more iterations to be done.

4.4.4 Reschedule All: ILP

With this method we reschedule all segments in the problem for each mutation step. After the mutation the changed segments are collected and this method makes the schedule. We could just use the method described in 4.4.1 for that purpose but we decided to try ILP in addition. Exact algorithms could outperform an ILP, but using ILP grands us some more flexibility. For example we could add constraints such that employees could have different shifts.

For this problem we again use the same graph $G = (V, E)$ as described in section 4.4.1 where every node represents a segment and an edge $(i, j) \in E$. For the ILP don't have to transform the graph to a bi-partite graph as long every segment except for the segment representing the start of a shift get a predecessor and every segment except for the segment representing the end of a shift gets a successor.

Again the objective function we want to optimize for robustness. The objectives of planning most PRM's and penalties let them possibly wait we have discarded, because otherwise we could have just used the ILP to solve the whole problem taken care of by the local search. All segments are forcefully planned using the constraints, so the objective function only have to focus on optimizing the robustness. Meaning that if we find a feasible solution for the ILP we know for sure that all segments are planned properly, otherwise the mutation is unfeasible. let $w(e)$ be the function that calculates the robustness of edge e.

We use one ILP model for the problem variant where all shifts of employees are identical. We can add additional constraints to the ILP such that the problem variant where shifts are not identical could be solved. The 3 ILP formulations we list below all have the same base but differ how they implement the constraint for the variant where shifts are not identical.

ILP1:
The first ILP notation got 2 kinds of binary variables. Variable $x_{i,j} \in \{0, 1\}$ is used to represent the edges of graph G and defines wether or not they are taken into the schedule. We make a variable $x_{i,j}$ for each edge in $(i, j) \in E$. If the variable $x_{i,j}$ is set to 1 then segment 76i is planned directly before segment j on the same employee, the variable is set to 0 otherwise.r

The second variable $v_i^t \in \{0, 1\}$ and represents whether or not segment i is served by employee or bus t. We make a variable v_i^t for every segment i and every employee or bus who can serve segment i. If the variable is set to 1 then the segment i is served by the employee or bus t, the variable is set to zero otherwise. The segment representing the start of the shift s and the segment representing the end of the shift e of an employee or bus t, could only be served by t himself. In that case v_s^t and v_e^t are given a fixed value of 1, making it a parameter. Each segment may only be assigned to a single employee at the time.

The ILP will look as followed where R is the set of segments in the problem, R_{start} is the set of segments representing the shift start of an employee or bus, R_{end} is the set of segments representing the shift end of an employee or bus, E_i^{in} is the set of all incoming edges of node i, E_i^{out} is the set of all outgoing edges of node i and T_i is the set of employees or busses that can serve segment i.

min: $\sum_{(i,j)\in E} w((i,j)) \cdot x_{i,j}$

1	$\left(\sum_{(i,r)\in E_r^{in}} x_{i,r}\right) = 1$	$\forall r \in R \cup R_{end}$
2	$\left(\sum_{(r,j)\in E_r^{out}} x_{r,j}\right) = 1$	$\forall r \in R \cup R_{start}$
3	$x_{i,j} + v_i^t - v_j^t \leq 1$	$\forall (i,j) \in E, \forall t \in T_i \cap T_j$
4	$x_{i,j} + v_i^t \leq 1$	$\forall (i,j) \in E, \forall t \in T_i / T_j$
5	$x_{i,j} + v_j^t \leq 1$	$\forall (i,j) \in E, \forall t \in T_j / T_i$
6	$x_{i,j} \in \{0,1\}$	$\forall (i,j) \in E$
7	$v_i^t \in \{0,1\}$	$\forall i \in R,\ t \in T_i$
8	$v_i^t = 1$	$\forall i \in R_{start} \cup R_{end},\ \{t\} = T_i$

The first 2 constraints makes sure that every segment get a predecessor and a follow up. The first 2 constraints are enough for the variant where all employees shifts are the same. We are only using constraints 3 to 5 when shifts aren't homogenous. The third constraint makes sure that the employee assigned to the predecessor is equal to the employee assigned for the current segment if the current segment could be handled by the employee. Constraint 4 and 5 makes sure that if there exists an employee that couldn't support the previous or next segment that then no such match in the bi-partite graph may exists. Constraint 8 tells that the shift start and end segments can only be served by the employee they represents. This works because the start segment of each employee can only be served by the employee itself and the end segment of each employee could only be served by the employee itself. For the no shift version we could only use the first and second constraint. Because the segments already have a fixed start time and no cycles are in the graph finding a predecessor and successor for each node is sufficient, like explained in section 4.4.1.

When implementing this ILP we struggled with an hardware limit of not having enough memory to construct the ILP from java for the shifts variant. The variant without shifts wasn't an issue, the additional constraints and variables had greatly increased the matrix size. We could have used an alternative ILP assignment that is actually more commonly used for this kind of problems where multiple 'vehicles' exists. we made the previous notation because we used the matching Algorithm as base, and added employee assignment in them. We will list down below the alternative ILP formula to add those extra vehicles. The ILP formulation is based on a multiple vehicle planning problem.

ILP 2: Instead of making variables for every segment which employee they are served with, we bind that decision on the edges. Each variable $x_{i,j}$ in ILP 1 gets a copy for each employee or bus $t \in T_i \cap T_j$, creating the variable $x_{i,j}^t \in \{0,1\}$ representing the edge supported by employee or bus t. If the variable $x_{i,j}^t$ is set to 1 then segment i is planned directly before segment j on employee t. Where otherwise the employee is deducted by tracing back the path to the source or keeping track of an variable, the variable states which employee is used. When a segment j is assigned to employee or bus t by variable $x_{i,j}^t = 1$ then the edge selected for the successor k must also be of employee t, and only edges like $x_{j,k}^t$ may be selected. Again the only viable employee or bus of a start and end segment of an employee or bus, is that employee or bus himself. With this we greatly increase the number of variables but greatly decrease the number of constraints.

min: $\sum_{(i,j)\in E} \sum_{t\in T_i \wedge t\in T_j} w((i,j)) \cdot x_{i,j}^t$

1	$\left(\sum_{t\in T_r} \sum_{(i,r)\in E_r^{in}, t\in T_i} x_{i,r}^t\right) = 1$	$\forall r \in R \cup R_{end}$
2	$\left(\sum_{t\in T_r} \sum_{(r,j)\in E_r^{out}, t\in T_j} x_{r,j}^t\right) = 1$	$\forall r \in R \cup R_{start}$
3	$\left(\sum_{(i,r)\in E_r^{in}, t\in T_i} x_{i,r}^t\right) = \left(\sum_{(r,j)\in E_r^{out}, t\in T_j} x_{r,j}^t\right)$	$\forall r \in R, \forall t \in T_r$
4	$x_{i,j}^t \in \{0,1\}$	$\forall i \in R \cup R_{start} \cup R_{end}, \forall t \in T_i \cap T_j$

Again constraints 1 and 2 make sure every regular segment got a predecessor and a follow up and the start and end shift segments are been planned. Its slightly altered to include all variants of variable $x^t_{i,j}$. Constraint 3 says that every segment and employee combination must have as much incoming edges as out going edges belonging to the same employee. For example if segment r_1 is followed by segment r_2 with employee t_1 by variable $x^{t_1}_{r_1,r_2}$ then the successor segment k must also be set through a variable $x^{t_1}_{r_2,k}$ enforcing the segment to be planned on the same employee.

Unfortunately this ILP after implementing also exceeds our memory limit using the interface from java. Therefore we needed either more memory, skip this algorithm or design a new ILP. Fortunately we found an alternative ILP notation.

ILP 3:
This notation is also based on the matching problem. The problem that arrises when we use the formulation of the variant with identical shifts for the variant with diverse shifts is that segments representing the start of a shift don't meet up with a viable segment representing the end of a shift. For example if we got 2 employees t_1 and t_2. Employee t_1's shift start at 3:00 represented by segment $i^{t_1}_{start}$ and ends at 11:00 represented by segment $i^{t_1}_{end}$. Employee t_2's shift start at 7:00 represented by segment $i^{t_2}_{start}$ and ends at 15:00 represented by segment $i^{t_2}_{end}$. When we use the graph and weighted maximum matching algorithm as discussed in section 4.4.1, we can't guarantee that the path starting with segment $i^{t_1}_{start}$ ends with segment $i^{t_1}_{end}$. Employee 1's path might end up starting with segment $i^{t_1}_{start}$ at 3:00, assist some PRMs and end with segment $i^{t_2}_{end}$ at 15:00. While employee 2's path might end up with starting with segment $i^{t_2}_{start}$ at 7:00, assist some PRMs and end with segment $i^{t_1}_{end}$ at 11:00. All these routes are viable connections in the graph, to prevent this we must enforce the the first and last segment of a path of an employee are compatible with their shift times. Since we we linked an employee or bus to the shift start segment, the rest of the path must be served by that employee, we therefore only need to make sure that the last segment is compatible with the shift.

Instead of tracking which employee or vehicle serves each edge or segment we could use a single continues variable per segment instead of a few binary variables. That variable y_i represents the time the employee serving segment i should end their shift. The values of y_i from the start and end segment of each employee have a static value that equals to the shift end. This makes sure that the value of y_j of all segments j in the path starting from y_i representing the start segment on an employee, must have a shift end time equal to y_i, which is a set value.

Since every segment must have a successor except for the shift end segments, the last segment in the path must be a shift end segment. And since all segments i in the path for an employee or bus must have the same value for y_i including the last segment and the value of y_i for shift end segments are fixed, the last segment must either belong to the employee or bus himself or an employee or bus that has the same shift end time. In the later option we could easily rearrange the end nodes such that each path starts and ends with a segment belonging to that employee. We could even relax the path such that every segment j of a path must have an y_j value equal or less than the previous segment in the path. It doesn't hurt when an employee is finished early but if you assign an earlier shift end segment j like employee 2 in the earlier example of the 2 employees with different shifts, you must give one shift end segment to an employee who should end earlier, which is a violation of the constraints.

Either way the values y_j of all segments j in a path of an employee will have the same value.

min: $\sum_{(i,j)\in E} w((i,j)) \cdot x_{i,j}$

1	$\left(\sum_{(i,r)\in E_r^{in}} x_{i,r}\right) = 1$	$\forall r \in R \cup R_{end}$
2	$\left(\sum_{(r,j)\in E_r^{out}} x_{r,j}\right) = 1$	$\forall r \in R \cup R_{start}$
3	$y_j - (1 - x_{i,j})M \leq y_i$	$\forall (i,j) \in E$
4	$x_{i,j} \in 0, 1$	$\forall (i,j) \in E$
5	$y_i \geq 0$	$\forall i \in R$
6	$y_i = shift\_end(i)$	$\forall i \in R_{start} \cup R_{end}$

Constraints 1 and 2 are basically the same constraints from ILP1 again. Constraint 3 basically tells that if edge (i, j) is selected by setting $x_{i,j}$ to one then the shift_end variable y_j of segment j should be equal or smaller than the shift_end variable y_i of segment i like explained above, it enforces that the path of an employee ends with a compatible segment to represent the end of the shift. Rule 6 ensures that y_i of the shift start segments and shift end segments of an employee equals the shift end time of that employee. This approach uses a smaller ILP and memory usage than the other 2 approaches and was able to run.

We could also have numbered the employees using y_j as employee $1, 2, 3, 4, 5, ...$ and used y_j as an element variable. Then we could have forced the end segments to be assigned belonging to the same employee. But this was the original idea and fixing the shift end segments is really easy just ignore the planning and add them at the end.

In the Experiments section we compare the performance of those 3 ILP's. Because the first 2 didn't load we will use a subset of the PRM's to compare these 3. But since ILP 3 only did load using the sample problems, its the only one viable for computers with 4 GB of RAM.

4.4.5 Greedy shift fix

The subproblem for the variant where shifts are not identical is NP-Hard since it represents a multi depot vehicle scheduling problem. By using the ILP we could solve instances but that will probably take much computation time. That is why we propose a greedy fix for this problem to use on strategies that doesn't take care of different shifts and correct them afterwards to enforce the shift constraints. The input for this algorithm are routes for employees that doesn't necessary obey their shifts. We want this greedy fix to be performed fast so we just focus on finding a feasible solution with connections that aren't too bad.

First we assume that all employees start shift segments belongs to their employee and the sequence of segments behind that the route the employee takes. Then the last segment of the path the employee takes which should be a shift end segment, since that is the only segment who may not have a successor. If we use our heuristics for the variant with identical employees the path of the employee might end with a shift end segment belonging to an employee who's shift end earlier or later then the employee who is assigned that path to. We can not have segments before the shift start segment and therefore we do not have any issues with assigning segment earlier then the shift ends, but we do have a problem at the shift end segments.

We can fix that by finding a second route with an acceptable shift end such that the shift constraints are satisfied. We cut those 2 routes in 2 and reconnects them such that the first route is satisfying the shift constraints. Because we are searching for an robust solution the algorithm checks all viable employees for the second route and all spots in the route for the cut with the best change in robustness score. The cut is only feasible if the head of the first route could continue with the tail of the second route and the head of the second route could be continue with the tail of the first route such that no segments are lost. We then score the cut by adding the robustness costs of the new formed edges minus the robustness cost of the old edges.

To avoid chaos of breaking good routes again by changing the second route which might be already fixed we don't allow to change fixed routes. 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 list as a second route to cut. This way we avoid possibly breaking an route that is already declared feasible or fixed. The order for small to large is because the employees with a smaller shift end are more constrained than the higher shift end. After all an employee with a higher shift end could possibly do a tail route from a lower shift end employee and be done early while the lower is less likely to be able to serve the tail of a higher shift end employee.

With this method we could quickly fix the schedules of the heuristics to satisfy the shift constraints for methods that couldn't originally, it might be dirty but we need to find a solution quickly. We could always do an exact algorithm like ILP at the end to optimize the schedule given the current start times. Its possible that this algorithm doesn't find a fix and then the mutation should be called infeasible although when done with an exact algorithm there might be a feasible answer. In the experiments section we try this approach to compare with the ILP version for the modified sample data with shift variants.

5 Computational experiments

We have run some experiments with the second algorithm using a Windows 7 Ultimate Computer with a Intel(R) Core(TM) i7 CPU 2.80 GHz quad core and 4 GB of RAM. Again the used instances are provided from Line Blander Reinhardt one of the authors of the paper this thesis is based from and we don't know or PRMs are prebooked or not. We first experiment with the current situation where the shifts of all employees are identical, then we show the results of comparing ILP formulations and finally show experiments for the harder problem with generated shift.

5.1 no-shifts

For the no-shift experimentations we only add lounge visits to the instances and possibly fix some minor inconsistencies like travel time from A to A be greater then 0. For the rest we leave the workers and PRMs as if. For this problem we will only use one ILP variant because if all employees have identical shifts all ILPs use the same constraints. We have used the following solver settings for this problem:

Because for generating the schedule after each iteration the exact methods seems to take much longer to calculate then the heuristic methods, we give them an other amount of max iterations such that they are near done by the 2 minute maximum computing time. Otherwise for the exact methods with an high number of iterations when the 2 minute limit has passed, the Temperature of the local search is still very high at that point and need to cool down faster then while using a fast heuristic approach.

We list the decline penalty for unbooked and booked, but since we don't know who is booked or not we treat all PRMs as booked. The same temperature value is used for the second objective value for robustness, but when evaluating the chance to accept an solution or not, we multiply the robustness score by 3 for the input of the formula. The formula to calculate the chance to accept a solution or not looks then like: $e^{3*(f-f')/T}$, where f is the old robustness score and f' the new one. 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 Matching method. The result table contains averages of the computing time and actual iterations, best and worst solution with the number of declined customers and robustness score. The Average column list the averages of the number of declined customers and robustness with a double precision of 1.

solver 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	(The degree T is multiplied every I/Q iterations.)
Decline penalty unbooked:	400	
Decline penalty booked:	1200	
		before the PRM arrive if he is at the location.)
Mutations:	Plan_PRM, Decline_PRM, Merge_SegmentGroup,	
	Move_SegmentGroup, Move_Split_SegmentGroup	

5.1.1 Results: Simple Matching

The table below shows the results of the experiment using the Insert Heuristic: Simple Matching as described in 4.4.2. This is the simplest matching heuristic we did and also the one with the least flexibility in answers. It just tries to insert each segment in an existing workers schedule.

instance	time(ms)	iterations	Best		Average		Worst	
			declined	robust	declined	robust	declined	robust
Instance 1	5119	485267	0	0	0,0	249,5	0	700
Instance 2	7423	500000	1	41	1,6	415,7	3	1383
Instance 3	7481	500000	0	608	1,2	652,9	2	1584
Instance 4	6043	500000	0	215	0,0	837,3	0	1313
Instance 5	6954	500000	0	32	0,1	722,3	1	627
Instance 6	7412	500000	0	262	0,0	829,4	0	1344
Instance 7	5778	500000	0	16	0,0	493,9	0	1069
Instance 8	6505	500000	0	800	0,1	1224,9	1	1306
Instance 9	5934	500000	0	218	0,0	775,0	0	1313
Instance 10	9337	500000	1	622	1,6	1051,0	2	1701
Instance 11	9294	500000	0	634	1,4	767,5	4	643

Table 5:

Although using a very simple insert heuristic it managed to find some pretty good answers really fast. The average computation time for 500.000 iterations is below 10 seconds. Looking at instance 1 the best solution of the 30 runs got a score of 0 for robustness and no declines which is using our objectives an optimum answer! Also except for Instance 2 and 10, the best solutions found have 0 declines, averages looking good too. However the worst solutions got up to 4 declines, for all instances except for 2 the worst solution have improved in comparison to our first Algorithm, where at instance 5 there where 10 declined in the worst solution versus 1 using this algorithm.

Because we make a solution every iteration we come up with no unexpected surprises at the end where PRM's couldn't be scheduled like our first algorithm. Also because this algorithm focuses on letting PRM's take tours together where possible, the buss area should be less of a bottleneck then in Algorithm 1 and have more available employees. Looks like we are in the right direction even with a simple heuristic for generating schedules.

5.1.2 Results: Reschedule overlapping segments

The table below shows the results of the experiment using the Insert Heuristic: Reschedule overlapping segments as described in 4.4.3. This matching heuristic is more advanced then the previous one. It reschedules segments that are competing with the to be inserted segment for a spot on an employee, allowing 2-opt like operations on schedules.

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 6:

These results couldn't almost be better using the described objective functions. All instances except for instance 2 and instance 5, got 0 declined and 0 robustness penalty as their worst solutions out of the 30, which means an optimum score. And the worst solution of instance 5 got a robustness penalty of 1 which means that that solution got one connection where the employee got 19 minute slack time instead of the preferred at least 20.

The only odd one out is instance 2 with a best declined 1 and worst declined 3, this instance seems to be much harder then the others to plan everyone. The average is 1.4, so by the looks of it he finds more answers with 1 and 2 declined. However if you look at table 3 in section 3.3 where we did computational experiments with our first attempt for an algorithm, we see that only in instance 2, it couldn't schedule 1 PRM in the local search phase before assigning employees. In all other instances in the local search phase all PRMs are given start times for their segments. Which means there is not enough capacity in a terminal to serve one PRM, which the algorithm do check in that phase. After more closely examining the instance and our best result we saw that one PRM had a segment in terminal 11 who doesn't got any assigned workers to it, actually its the only person traveling through that terminal, so its pretty impossible. Since the experiments are already done, we let this person be.

Another observation could be made when looking at instance 2, the worst solution had 0 robustness penalty, but 3 declined persons of which we know 1 is impossible to schedule. Maybe because we are also optimizing robustness and prefer a lot of slack we push the solution in a certain direction. When examining the solution it had an odd favor to have multiple employees start a job at the same time in the buss area and employees have peak hours when many PRMs need to travel. It looks like the downside of merging rides is that a lot of PRMs have to be brought to the bus stop at the same time or PRMs coming from 2 different locations to a same bus stop are been walked together to the lounge. We could discourage this behavior by removing those matches from been selected by the mutation, but then we might need more trips from buses, which is inefficient as well.

5.1.3 Results: Optimise matching ILP

The table below shows the results of the experiment using a ILP as described in 4.4.4. Since we have identical shifts we can use ILP1, without the variable and constraints for the variant with different shifts. This method optimize the entire matching after every mutation.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	120698	270	165	444	187,6	170,0	210	73
Instance 2	120770	232	226	1353	240,7	1616,5	253	1500
Instance 3	120830	193	276	2401	290,7	1767,7	304	2905
Instance 4	120872	218	202	338	215,1	452,5	233	64
Instance 5	120911	169	295	1555	303,3	501,2	315	235
Instance 6	120930	169	301	122	311,1	544,5	321	980
Instance 7	120840	227	203	161	224,4	628,9	239	755
Instance 8	120745	213	232	210	247,2	207,8	264	375
Instance 9	120814	220	213	9	226,3	435,1	241	9
Instance 10	121092	177	275	45	287,3	684,5	299	164
Instance 11	121202	145	367	245	378,9	287,8	392	144

Table 7:

The results are disappointing, the number of iterations that could be done in 2 minutes are way to few. Instance 1 has the highest number of average iterations of 270 and still got at best 165 declined passengers, which isn't odd considering the number of iterations. This algorithm just performs too poorly, one ILP solve just takes too long to solve to be used every iteration. Next up is the Hungarian algorithm for matching that also matches the full matching problem, hopefully it performs faster.

5.1.4 Results: Optimise matching Hungarian

The table below shows the results of the experiment using the Insert Heuristic: Reschedule overlapping segments as described in 4.4.1. This method optimizes the whole matching after each iteration like the ILP but then using the hungarian algorithm.

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 8:

Although the results are better then the ILP variant, its still kinda disappointing. The most runs didn't even reach the 3000 iterations limit, which also means that too much time is used on optimizing each iteration. Results are way better though then ILP, but still worse then the perfect solutions the heuristic found. There is a lot of everyone been planned, although not very robust and at worst 7 declines. Again the algorithm could do too few iterations because of the optimizing step after each iteration.

We can the processing time of the heuristic by using the dynamic Hungarian algorithm[14]. We currently generate the input and execute the whole Hungarian algorithm every mutation, but this could be done smarter. 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. The paper doesn't explain how to remove nodes (segments in our case) from the matching but that should also be possible. The dynamic Hungarian algorithm described by Mills-Tettey et al[14] require both the dual variables and the previous matching as input, when we do backtracking we should restore those values as well. We discovered this too late and experiments are already run, but might be worth to look at this in futhure.

5.2 Looking closer to the solution

We made a program that could visualize the generated results as a gantt chart for each worker and by doing so we found out something quite interesting.

First of all, employees at terminals surprisingly have loads of time, many slack times are way over the 20 minutes. And then suddenly everyone has to work because several gates are boarding roughly at the time. The problem is that there is such a high need of personal because of these peak hours. If planes do depart with people needing assistance more spread over the day, you will properly need less personal. Scheduling a lunch break can be done with ease, they have loads of time. After all if we managed to have a 20 minutes slack everywhere, there should be some room for a break.

The inter-terminal traveling by buss however got busier schedules but nevertheless got some time for a break. Its workload is better spread over the day but that might also because we park PRMs at a lounge before a buss transfer and after a buss transfer for that reason. Also there are a lot of merged rides with 2 or more PRMs joining a ride and then still the bus drivers have enough work.

Another interesting observation is that our algorithm apparently have a preference to put every employee at work roughly at the same time, especially early in the schedule. This was to be expected when planes arrive and depart around the same time, but even the inter-terminal buses travel around the same time. Even though the time windows allows the segment for traveling by bus to be planned earlier or later. This is properly due the merging of segments, such that 2 or more PRMs are served by the same employee on a segment. The bus area is used efficiently by letting multiple PRM joining the rides, but they also need to be brought to the bus stop and from the bus stop to the lounge. The segment towards the lounge is also sharable even though buses are coming from 2 different locations such that buses arrive at the same time. This was discovered after the experiments are ran. A way to discourage these kind of solutions is to filter and disable matches that are purely based on walking from and to the bus stops. But currently these algorithm gives good solutions in score so there is currently no need for alteration.

5.3 Conclusion no shifts

It looks like we where on the right track to join PRMs together on segments to increase the efficiency of the bus area. This algorithm provides better answers than our previous one in section 3 like expected. The heuristics for the matching sub-problem at the end of nearly every iteration performs way better than solving the matching problem to optimality.

Of the optimizing matching strategies the ILP performed very poorly only capable of performing a few hundred iterations within the 2 minute limit. The exact method using the Hungarian Algorithm performed better but also only managed a couple of thousand iterations. The Hungarian method managed to plan nearly everyone within those iterations but have a high robustness penalty. It looks like that the optimizing methods are way too time consuming to be used efficient in this problem, after all its a subproblem with a big dense graph.

Of the heuristic strategies the rescheduling of overlapping segments performed better than the simple inserting heuristic. The simple inserting wins in computation speed, most instances are solved within 10 seconds doing 500.000 iterations against 10 till 71 seconds of the rescheduling overlapping segments algorithm. The score of Simple inserting is pretty decent most of the time everyone is planned or have a few unplanned, but have some robustness penalty. The score of rescheduling overlapping segments is near perfect for those instances, nearly always everyone planned and 0 robustness penalty, which was unexpected. The odd one out proved to have a PRM who could be impossibly planned since there are no employees in an area he must travel through for some reason.

If we compare our results to those of Reinhardt et al [1] see appendix A, where this research is based on, we have found an improvement. For half of the instances Reinhardt et all always found a solution where every PRM is planned, but for those who doesn't we have a better average in our best algorithm. Also they always had some unnecessary waiting time, while our algorithm guarantees smooth connections. Therefore our best algorithm is an improvement to the previous work.

Rescheduling overlapping segments seems to be a good strategy to use to plan in new PRMs without resolving the whole matching problem that generates the planning. ILP seems way too slow, but nevertheless we go continue testing those in the shift variant, where it is the only exact optimization method we tried. We have to result to heuristics if we want to make the 2 minute maximum computation time for this problem. By using simplifications, making some soft constraints like allowing waiting and taking detours hard constraints we made this problem easier and could provide good answers well within the 2 minute time constraint.

5.4 shift variant

For the shift experimentations in addition of the previous settings we generate shifts. Exactly how we generate those shifts we are going to explain in the next part. The only exact algorithm we got that solves the assignment problem after each iteration is using the ILP. Considering only one ILP variant is feasible and it didn't run so well for the variant where all shifts are identical we don't expect much from it. Nevertheless we are going to compare those ILP notations with each other for completeness. After that we show some results of the better algorithms for the shift variants. Since the shift variant is harder we don't expect better solutions or been able to do more iterations within the 2 minute limit.

Unfortunately our example data experiences some problems to convert it to a shift variant. We don't know for sure how in the real world situation they distributed the shifts and when they do change over. We don't want the change of shifts to occur when many people are trying to board a plane. Another problem that was discovered too late is that the instances that had been said to represent a day could have a PRM that has to be served outside the days window. If you look at the data most PRM's are within the day's 24 hour time window but a few defy that for some odd reason. Maybe its a PRM that couldn't get a plane that day and is postponed to the next day or something.

5.4.1 Generating shifts

Before we go jump to results we have to explain how the shifts are inserted in the problem. Our program are added 2 ways to generate shifts. The first one generates some shifts with some overlap, the second one generate more realists 8 hour shifts.

The first way takes the employees as input data and slice their shifts inside 3 parts. Then new employees are been generated each serving 1 or 2 neighboring parts such employee shifts are overlapping each other. The first kind of employees only serve part 1, the second kind serves part 1 and 2, the third kind serves part 2 and 3, and the last kind serves part 3. During generation its made sure that during every moment of the day the same number of employees are active as in the original problem. The downside of this method is that its not realistic in hours, but since we don't have the actual shifts all we can do is guessing.

The second one is based on 8 hour shifts starting at 3:45, which is seemly the start of the shifts of the current employees. There are generated 3 sets of employees starting at 3:45, 11:45 and 19:45. But these generation actually causes some problems namely during the shift changes its possible that planes need to be boarded. Its better if the flight boarding times and shift changes are coordinated such that isn't a problem. To give some room for serving PRMs we allow each worker to work 30 minutes overtime. Since we aim to minimize slack even for going towards the workers end location, working overtime more than 10 minutes is not very likely. 30 minutes is chosen to allow boarding someone because that already takes about 20 minutes, higher overtime times risks more violations. The problem with this approach are the faults inside the example data, namely the PRM's that needs to be served outside the 24 hour day time window.

5.4.2 Results: Comparing ILP for shift variant

We have 3 ILP notations that solves the assignment problem that needs to be solved after each iteration. In this section we show the results of experimenting with those 3 notations. Unfortunately only one of the instances will load on the computer and is feasible for a computer with 4 GB RAM. Therefor we reduces the instances such that the computer could load them. We only do this experiment to check whether or not the only ILP that loads is the best ILP formulation we have for this problem.

To make sure the ILPs load we are trowing away PRMs at random such that the instance is roughly 40% of its original size. To compare those ILPs we test who can do 1000 iterations the fastest, or runs out of the 2 minute computation time. Each Algorithm is run 30 times using a set of 30 seeds, such that the algorithms makes the same decisions each iteration and only calculated the ILP differently. We generate the shifts by splitting the day in 3 parts like explained in section 5.4.1.

Algorithm	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
ILP1	126028	33	134	0	141,0	0,0	149	0
ILP 2	123706	37	130	0	138,0	0,0	145	0
ILP 3	127514	55	121	0	126,2	10,4	138	0

Table 9:

Judging from the results all ILP's do poor even on this reduced instance. ILP 2 performs slightly better than ILP 1 doing 37 iterations versus 33. ILP 3 managed to have the most iterations during the 2 minutes: 55. Well it was be expected since the ILP also performed poor on the no-shifts variant, besides that this variant turns out to be harder to solve to optimality since its NP hard.

The ILP performed not so good, lets try how well the Hungarian method does with the shift variant. Because this algorithm could not guarantee that shifts are been planned correctly we use a greedy algorithm that fixes violations of the constraint that segments must be planned within the employees shift. The results of this quick experiment is listed below:

Algorithm	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Hungarian + Shift Fix	18476	1000	47	0	75,7	2,8	117	4

Table 10:

This variant did manage to do the full 1000 iterations for this experiment, where the ILP did only about 55 Iterations on the fastest ILP. The algorithm couldn't plan everyone even though its a reduced instance. However 1000 iterations is too few to explore the search space. We could have allowed the algorithm to do more iterations but since this one already takes about 18.4 seconds to do 1000 iterations we couldn't do more than around 8000 iterations in 2 minutes on the decreased instance. Because the instance is decreased to roughly 40% of the instance size, the matching algorithm with an $O(n^3)$ running time should take longer to solve using the full instance. So we can't expect that when doing to full iterations on the full instance, the algorithm would give outstanding results.

Also since the ILPs are our only really exact matching algorithms that respect the shift constraints and the Hungarian algorithm is too slow for the no-shift variant, we better use the heuristics again on this variant swell. We are going to take a look at the heuristic algorithms next.

5.4.3 Results: Simple matching

For this set of test runs we used the same settings for the Local search as with the no-shift variant, where shifts are identical. We run 2 sets of experiments using the Simple matching algorithm, each with a different method of generating shifts. The first shift generations uses the method that split the day into 3 parts. The second shift generation method is run using 8 hour shifts with 30 minutes overtime Both methods are explained in section 5.4.1.

Results Simple matching using shift generation method 1: splitting day in 3 parts.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	6009	486201	0	0	0,1	375,5	2	149
Instance 2	9034	500000	1	151	2,6	473,7	5	365
Instance 3	8652	500000	0	1213	1,3	726,9	2	1662
Instance 4	6839	500000	0	473	0,0	960,0	0	1421
Instance 5	9184	500000	0	847	0,6	1274,4	3	871
Instance 6	8998	500000	0	258	0,0	919,9	0	1736
Instance 7	6641	500000	0	85	0,0	475,8	0	997
Instance 8	7580	500000	0	617	0,2	1282,3	1	1225
Instance 9	7334	500000	0	784	0,1	1462,5	1	1938
Instance 10	11028	500000	1	769	1,7	1052,9	3	1239
Instance 11	10810	500000	0	859	1,1	1526,8	3	2292

Table 11:

Results Simple matching using shift generation method 2: 8 hour shifts + 30 minutes overtime.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	9949	500000	3	1126	3,0	1398,0	3	1996
Instance 2	11159	500000	7	1674	7,6	2290,9	9	2228
Instance 3	11665	500000	6	1125	7,4	1203,9	8	1584
Instance 4	12201	500000	2	1777	2,0	2240,0	2	2806
Instance 5	15516	500000	6	1166	6,1	1939,2	7	2472
Instance 6	13037	500000	4	2228	4,0	2912,0	4	3718
Instance 7	11851	500000	6	1241	6,0	1566,2	6	1924
Instance 8	11462	500000	9	826	9,3	1645,9	11	1323
Instance 9	13065	500000	2	1416	2,0	2128,4	2	2763
Instance 10	14900	500000	6	1618	6,9	1966,6	8	2374
Instance 11	15505	500000	4	1427	5,0	1817,6	7	2051

Table 12:

The results of the first shift generating method actually doesn't differ that much with the original problem. Some worst answers are slightly worse or better but in overall it looks similar. That is probably because using this simple insertion problem you don't need any modifications to enforce the shift constraint. The amount of overlapping between shifts is enough to allow almost everyone to be planned.

That is unlike using shift generation method 2, where quite a few PRMs are declined of service. That is properly mainly because of the shift changes and departing airplanes are colliding. Its also possible that this algorithm just have trouble finding a good solution if the shifts doesn't overlap much. Maybe our best algorithm for the no-shift variant could do a better job.

5.4.4 Results: Reschedule overlapping segments + fixing shifts

Like the previous experiment for this set of test runs we used the same settings for the Local search as with the no-shift variant, where shifts are identical. We run 2 sets of experiments using the Simple matching algorithm, each with a different method of generating shifts. The first set is run using shift that overlap each other. The second set is run using 8 hour shifts with 30 minutes overtime. Both methods are explained in section 5.4.1. Because the Reschedule overlapping segments algorithm doesn't ensure shift constraints we use the greedy fix algorithm to correct that

Results Results: Reschedule overlapping segments using shift generation method 1: splitting day in 3 parts.

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 13:

Results Results: Reschedule overlapping segments using shift generation method 2: 8 hour shifts + 30 minutes overtime.

instance	**time(ms)**	**iterations**	**Best**		**Average**		**Worst**	
			declined	robust	declined	robust	declined	robust
Instance 1	120029	259265	3	964	3,0	1045,1	3	1061
Instance 2	120027	172974	7	1311	7,3	1595,6	9	1504
Instance 3	120027	90492	6	534	6,0	868,2	6	1184
Instance 4	120027	224307	2	1187	2,0	1238,1	2	1405
Instance 5	120027	75785	6	1207	6,0	1781,7	6	2482
Instance 6	120031	76089	4	1535	4,0	1909,7	4	2436
Instance 7	120027	128135	6	883	6,0	993,4	6	1207
Instance 8	120028	212875	9	513	9,0	566,2	9	633
Instance 9	120027	184501	2	949	2,0	962,0	2	999
Instance 10	120027	111807	5	784	5,0	995,3	5	1393
Instance 11	120030	104679	4	796	4,0	987,6	4	1191

Table 14:

Again like with the simple matching, shift generation 1 doesn't differ that much with the no shift variant. The algorithm mange to plan everyone within the time limit. The algorithm is a bit slower though because of the shift fix, but still manage to find perfect score solutions within the time limit.

Again like the simple matching algorithm the solutions of the second shift generation are worse, with quite a few declines. The results are consistent though in the number of PRM's that are declined, apparently they can't be planned using this shift variant. It would have been better if we had the actual shifts of the personal so this could be avoided, but the sample data didn't deliver that. And maybe the problem is actually better solvable when not considering shifts.

5.4.5 Conclusion: shift variant

The problem gets harder to solve when introducing the shift variant, not only you have more employees with smaller shifts, you also need to make sure that the shift constraint is enforced. Our only optimization algorithm for the no-shift variant using ILPs is way too slow to solve the problem within the 2 minutes limit. The heuristics using the greedy shift fix when needed performed way better, similar even compared with the normal variant but then a bit slower.

Unlike the no-shift variant, when we generated 8 hour shifts with an half an hour overlap we where unable to plan in all PRMs. This is most likely due planes are leaving around that time and we have some fixed segment groups. We could have better tested the 8 hour shifts by analyzing the departures but we needed to look at all instances. Also it doesn't help that some PRMs must be planned outside the 24 hours of an instance representing a day.

Maybe the example data only consisted out of shift that long because that makes the problem much easier and then if we got a schedule we decide when shifts end. We actually we don't have a clue how airports divide these shifts or what are the rules of a shift. Remember this problem is based on a paper, which is based on an actual world problem. But unfortunately we don't have any contact with the airport in question to check how the shifts are regulated. If we cooperated with an airfield we might have been able to test this variant better.

But by the looks of this especially when there is some overlap such that the fixed flight departments aren't much of an issue, our algorithm also could find good solutions for the shift variant. Greedy heuristics also in this case makes the problem more easily to solve, the more exact methods just takes too long to solve.

6 Simulation

We now have an algorithm that makes robust schedules but will the schedule perform well in the real world where events could disturb the schedule? Would it be easier to add additional PRMs into the schedule without having to disrupt the schedule too much? Or is robustness not so effective because most pickup and delivery times are fixed by the time the planes arrives and departs. To answer these questions we made a simulation, comparing the solution of our algorithm that uses the robustness score against the same algorithm but with the robustness score disabled. We couldn't use the algorithm used by Reinhardt et al[1] because we only got the pseudo code in their paper. Also they stated that to make their algorithm truly work in a dynamic environment some changes had to be made to the problem formulation. First we talk about the possible disturbances in the schedule, followed by the rescheduling strategy and finally the experiments themselves.

6.1 Disturbances

For our simulation we want to simulate disturbances in our schedule, which our scheduling algorithm must solve. Disturbances involve the arrival of unbooked PRMs asking for assistance, plane delays and trips between 2 locations in a journey taking longer than expected. We explain these below.

6.1.1 Unbooked PRMs

Our problem description states that there could be unbooked PRMs who arrive during the day. It is impossible to know beforehand who is requesting additional aid during the day, so it would be pointless to include them in the initial schedule. They must be added during the simulation, when they contact the service provider.

The unbooked PRMs are excluded from the instance till an event occurs where they check in and request for aid. Otherwise we are planning them in advance using knowledge that we are not supposed to possess. In case the PRM arrives by plane at the airport and requires assistance to get out of the plane, he could have better made the request in advance if he wants an employee to be available at the gate when the plane arrives. For this situation we assume the request was sent before the plane lands on our simulated airport, for instance before the unbooked PRM boards the plane at the airport the plane came from. After all if they need assistance with disembarking they likely need assistance with embarking as well. For the other cases where an unbooked PRM arrives and requests assistance at the airport we assume they want service immediately, so we make the release time of those PRMs the time of request.

Since we do not know which PRMs in the instance data are booked and who are not, we generate 100 additional 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 where we presume to contain both prebooked and unbooked PRMs where between 300 and 500 PRMs request aid.

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 plane schedules from the journeys of the PRMs in those instances. Flights are not directly listed in the instance data, but from the journey of the PRMs we could see by which plane they arrive or depart in if any. In addition to simulating the plane schedule on a given day using the extracted plane schedule we generate some additional planes.

Unbooked PRMs are generated by first deciding whether he is an arriving passenger, departing passenger or a transfer passenger. For each generated PRM a start and end location is chosen at random. If the PRM is arriving by plane we choose a random arriving plane as start location, otherwise a random pickup point in the terminal at which he departs from. If the PRM is departing by plane we choose a random departing plane as end location, otherwise a random drop off point in the terminal at which he arrived. Then between the start and end location the required route through the airport for the given PRM is generated as stated by the problem definition and added to the simulation. We generate these PRMs before executing the simulation but only notify the simulation of their existence at the time that the PRMs make themselves known.

6.1.2 Plane delays

For the plane delays we used the work of Diepen et al[13] for the data. Diepen et al[13], did research on robust scheduling of platform buses, which transport passengers between gates and airplanes of Schiphol airport. The compute a schedule using a column generation algorithm and ran a discreet event simulation to study whether their robust algorithm performed better than the currently used algorithm when disturbances in departure and arrival times occurred. To find realistic values for the disturbance they analyzed the delay statistics of both arrival and departure times of Schiphol airport. As the same disturbances affect our schedule we plan on doing the same.

In their simulation they simulated plane delays, each time a plane sent a last update of their schedule they update the schedule of their platform busses. When such an update happens they update the times for the corresponding platform busses. In case the update causes jobs of the updated bus to overlap or change order of execution they have a conflict and create a new schedule for the day. They measure the amount of conflicts in the execution of the simulation where the less conflicts are better.

For the statistics of arriving planes they found a normal distribution with a mean of 4 minutes and a variance of 30 minutes, which is a standard deviation of 5 minutes and 28.6 seconds. Although seldomly planes depart early, this happens to 2% of the cases and the amount of time that the plane leaves early follows an exponential distribution with a mean of 2 minutes. In the other 98% of the cases the plane departs later and this time follows an exponential distribution with a mean of 15 minutes. In our simulation we leave out the possibility that planes depart early. We use the distributions to assign new gate opening and closing times; a gate opening a few minutes earlier should have a minor impact on the schedule because the gate is opened for 20 minutes in which the PRMs are helped through.

In reality planes give status updates to the airport from time to time, which might adjust the expected arrival or departure time. Diepen et al[13] assumed that rescheduling is only required when the plane has reported its last update and the time it arrives or departs is known. Otherwise they must do serval reschedules for the same flight while only the last update matters. They have also studied that behavior and found that the last updates are given following an exponential distribution with a mean of 15 minutes before the actual flight time of the plane.

We are also going to use events for the last update for rescheduling but with an addition. In our simulation besides the planes we are also simulating the employees and PRMs and not just the schedule for platform busses like in Diepen et al[13] . If in our case a plane delay is major the last update might come too late.

Consider a departing plane where boarding is scheduled to start at 15:00 and the plane departs at 15:20. PRM p_1 needs to catch that flight where employee e_1 helps him through boarding. Following the schedule employee e_1 brings PRM p_1 to the gate when boarding is supposed to start at 15:00 like schedule. However the plane gets a major delay and departs at 16:20 instead of 15:20 and send its last status update at 16:05. We have known some time in advance that the plane is going to depart late but decided to wait for the last update to change our schedule. At the time we get the last status update we reschedule the employees. Between 15:00 and 16:05 PRM p_1 was uncomfortably waiting at the gate while occupying employee e_1 who could not serve his next segment on time because he could not hand over the PRM he is currently assisting.

In the example above, we let both PRM and employee unnecessary wait till a reschedule is done. Therefore before boarding starts it should be known that the plane is going to be late. We should then update the schedule to make sure the PRM waits for the delayed plane comfortably at the lounge. At that time we could postpone the boarding progress till we know when we actually have to board the plane using the last update.

If an arriving plane arrives early there is no problem, the employees who should pick up the PRMs in the plane are not waiting at the pickup point jet. The last update should then be early enough to schedule the employees to pick up the PRMs when the plane arrives. But if there is a major delay we got the same problem as with the departing plane, although we are not gaining unnecessary waiting time in this case. The employees who must pick up the PRMs from the delayed plane patiently wait till the plane arrives or their schedule tells them to help other PRMs. If the delay takes too long the employees could be late for the PRMs they need to assist next. To prevent that when we know that a plane is going to arrive late we could remove the segments of the PRMs that are in the delayed plane from the schedule. For minor delays this isn't necessary because if the schedules of the employees are robust they could handle small delays without affecting the rest of the schedule.

We assume that at least 10 minutes before the actual arrival time it is known whether or not the plane 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. We also do not want the first update to occur after the last update, making the formula for the first notify for delay: $\min(last\_update\_time, expected\_arrive - 10 - exp(1/15))$.

For departing planes we assume that the plane is already at the airport in advance due the in between flights maintenance like cleaning, refueling and unloading the luggage. When the plane arrives from its previous flight we should know how long the maintenance is going to take and when boarding could start. We assume that we know at least 45 minutes in advance how long that is going to take, we also assume that the plane notifies the airport in advance when they arrive using the same last update distribution as arriving planes making the formula for the last update: $actual\_depart - 45 - exp(1/15)$. We should know at least 45 minutes in advance of the expected depart time whether or not the plane will be late.

We only do a reschedule at the last update because then we know for sure in our simulation when the plane is going to arrive or depart. But at the first update we first check whether or not its a long delay. For arriving planes if its a long delay, we remove the PRM from the schedule till we know when he is going to arrive and then reintroduce him at the last update event of that plane. We first tried to postpone only the first few segments, removing them from the schedule to be reintroduced later, but that caused some trouble when the segments that follow are scheduled before the actual arrival, while we know the actual arrival only at the last update. For the departing planes we do postpone the segments in the last segment group which represents the boarding process. We remove those segments from the schedule since we don't know when they should be planned till the last update. When the last update event of the plane occurs the postponed segments get reintroduced in the schedule.

6.1.3 Traveling times

Traveling times between locations are not always exact, there is some randomness in it. The PRMs from the probability distribution we are traveling with might delay the employees on their journeys. For example if the PRM isn't in a wheelchair and walks a bit slower than expected or because of the crowd at the airport it could take more time to travel between locations. We don't have data for these kind of delays so we just took uniform distributions to tackle them. For wheelchair PRMs we take a uniform distribution of $[0.9, 1.1]$ times the expected traveling time and for other PRMs a uniform distribution of $[0.9, 1.3]$ times the expected traveling time.

If the employee is carrying multiple PRMs then we generate the traveling time of all PRMs the employee is assisting and take the largest as traveling time. The slowest PRM then sets the pace of the journey. We don't draw a delay for boarding since we feel that the 20 minutes the employee needs to spend there is already generous. Delays for busses are the same, although the most delay will properly be caused by either traffic or the time it takes to load and unload the PRM in the bus.

6.2 Rescheduling

We have discussed the possible disturbances, now in this section we are going to talk about how those disturbances are death with. In general we want to try to avoid full rescheduling as much as possible just like Diepen at al[13] in their simulation for the platform busses. When a disturbance is minor then we do not generate a new schedule but deal with that disturbance on the spot. We only want to reschedule for major disturbances like adding a new PRM to the schedule or a major plane delay. We also want to avoid making major changes to the schedule by a full reschedule, such that the jobs the employees have to do do not change dramatically every time something happens.

We consider a delay for a plane major if the delay differs more than 10 minutes from the expected arrival or departure time. For departure it could be easily be taken care of by the boarding time which takes 20 minutes, so 10 minutes later wouldn't matter much. Also for the arriving planes, we assume there is some slack in the schedule of the employee that must take the PRM from the plane to catch up with early arrivals and the employee waits till the plane arrives for late arrivals.

For the simulation we have 3 kinds of adjustments we have to consider when rescheduling: introducing a PRM, moving invalid segments and reintroducing postponed segments. When rescheduling we try to first fix those by applying minor changes in an update before doing a full reschedule. If there is still at least one unplanned new PRM, invalid segment or unplanned postponed segment after applying the update 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 we require to reschedule we first try to add new PRMs to the schedule if any. This is first done by executing the add PRM mutation and Merge segment groups for a number of times using the Insert heuristic: Free spot explained in section 4.4.2. Move segment groups tries to merge a segment group of the to be inserted PRM with another for a more efficient schedule. We use that insert heuristic because that heuristic doesn't dramatically change the schedule of the employees. Here we accept any change that introduces an PRM to the schedule, which is an improvement in the objective function anyways. If solely executing the insert heuristic isn't enough, a full reschedule is needed that hopefully frees up the room for the new PRM.

When making changes to the schedule some already planned segments might become invalid due to disturbances. For example when a delay occurs that pushes some segments of the journey forwards, the later segments then could be planned on an invalid time that starts before the previous segment has been finished. These segments need to be fixed in the schedule during rescheduling. The simulation automatically detects such segments and puts them in a set to be solved while rescheduling. Like the previous disruption of adding a new PRM, we first try to fix this by executing a mutation that moves those invalid segments in the schedule by moving the segment groups they are part of. In case the segment is merged, planned along with a segment of another PRM to share a single employee or bus, we separate those 2 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 insert heuristic: Free spot and if at the end there are invalid segments left, we need to do a full reschedule.

During plane delays segments could be postponed and removed from the schedule till its clear when they are needed to be planned. Again while rescheduling we first try to fix those by using mutations that reintroduce these segments using the insert heuristic: Free spot matching. Both planning in the schedule and merging the segment groups are attempted again. Just like the previous changes if at the end there are unplanned postponed, a full reschedule is required.

When in the simulation we have to do a reschedule, the simulation is already at a certain point in time. PRMs are already at certain locations or traveling towards them with employees and employees have finished a certain part of their schedule and are on their way for their next task. Everything that has happened before that time, should remain fixed because the past is unchangeable. Which implies that the planning algorithm may not make changes in what is already happened or what is

current happening. We just can't ask an employee to drop the current PRM to help another. Also 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.
— Besides we don't allow changes in the past we make a threshold in time where major disruptions may not occur to allow employee time to adjust and finish with what they are doing. We only allow insertions between the current time and the threshold and don't allow segments who started before that threshold to move to a different time or employee in the schedule. We set that threshold on 20 minutes after the current time at which the reschedule happens. We also don't allow to move future or new segments before the current time. For the Insert heuristic we use the less disrupting Free spot on the segments that start before the threshold and the better Reschedule overlapping segments on those who start after the threshold. Using this strategy we allow major changes after the threshold but try to conserve the schedule near the current time as much as possible.

For rescheduling we do not want to decline PRMs who has already planned. They are PRMs who have either prebooked or came earlier this day which we accepted. Once we give a yes to a customer, it would be bad to say no later in order to schedule someone else. Also removing a PRM increases the penalty with no guarantee of being able to reinsert the PRM. Worse would be if the PRM is assisted by an employee at the time of rescheduling, if the PRM gets temporary declined in the scheduling algorithm and must be put back in the schedule it should be scheduled on the exact same employee to continue his journey. We could decline PRMs during rescheduling as long the get reintroduced, but it saves a lot of trouble if we just do not allow that.

We also change the stop condition a bit. Because we fix the past in the schedule, any unnecessary waiting time accumulated there will never improve. We sum up the fixed unnecessary waiting time in the past and use that as lower bound for the robustness function. When we reached that lower bound we know we are robust in the future and could stop solving.

6.3 Simulation experiment

We ran some simulation experiments to test whether adding robustness would result in a better schedule than if we neglect robustness. We simulate the instance using both algorithms, one where robustness is considered like explained in this paper and one where robustness score is neglected by giving everything a robustness score of 0. We then compare the performance of both algorithms using a students t-test, to see who performs better in the simulation.

For each instance we generate 10 seeds which we use as input for both the version with robustness and the version without. The seed mainly determines the immediate PRMs and by how much the planes are delayed. Because we use the same seeds for both sets, the major events of both runs will be the same, while the execution and rescheduling of the schedule will be different. Then we could compare the results of the instances in a paired t-test. For the start solution we use a generated schedule for all 10 runs of the same instance. Since we have found optimum schedules for our scores earlier in the computation experiments of our second algorithm we select one optimum solution at random for each instance. For the variant where robustness is disabled we generated new solutions, using the best algorithm but with robustness disabled.

For rescheduling we use the method described as above, and allow 50.000 iterations to be done by the algorithm, which is 10 times less then in our computational experiments of our best algorithm. Most of the PRMs are already planned when we need to reschedule, and even if something big like a plane delay disrupts the schedule it only affects those PRMs who travel by that plane. Most of the time the rescheduling algorithm gets stuck in a local optimum anyway. This is done such that the simulation runs faster, and it is also its nice if the staff could get a reschedule fast.

In the simulation we are going to track a few statistics. We are tracking how many PRMs are not included in the schedule, where we separate the statistics for prebooked PRMs and immediate

PRMs. Every PRM who is scheduled in the start solution at the beginning of the day is considered prebooked and the rest who arrive later in the simulation immediate. We are also tracking the cumulated unnecessary waiting time in minutes of the PRMs in the actual execution of the schedule. Every minute a PRM has to wait for a connection is considered unnecessary waiting time. Because we do not allow taking detours in our schedule we don't have to worry about tracking that. We are also tacking how many times the simulation executes a full reschedule and how many times it executes just an update while calling the rescheduling protocol like explained in the previous section. We hope that by making a robust schedule, less full reschedules would be required.

After we executed the simulation experiments we did a two tailed paired t-test on each statistic named above to compare the two algorithms. The t-test assumes that the values our simulation output for the statistics follow a normal distribution. The null hypothesis states that the distributions of the statistic of both algorithms we are comparing are equal. Using the t-test we calculate a value p which indicates the chance the distributions of the statistic are equal, where a value of 1 is a 100% chance. We reject the null hypothesis if the p is less then the acceptance threshold. We use a commonly used value of 0.05 as acceptance threshold, if the probability that the 2 distributions of the tracked statistic is below that value we will reject the null hypotheses.

For example if we look at the statistic declined booked PRMs, we take the data set of that statistic of the simulations using the algorithm with robustness, and the data set of the simulations using the algorithm that ignores robustness. Because we used the same instances and seeds for both algorithms we got paired results. If we look at the mean the variant without robustness scored better, to be sure whether or not the variant without robustness is better we preform the test and get a value p. If p is less then 0.05 then we reject that the distributions of both data sets are equal and prefer the algorithm who ignores the robustness for that statistic. If the null hypothesis hold, there is a possibility that the results are identical and the mean of one algorithm is slightly better due randomness.

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 change the 2 distributions are considered identical by the two tailed paired t-test. The annotation '(robust)' indicates the results for the variant where we consider robustness and the annotation '(no robust)' indicates the results for the variant where robustness is ignored. The t-test is calculated by loading the output in numbers (similar to excel but for Mac users) which got a function to calculate a two tailed paired t-test from 2 data sets.

statistic	**mean (robust)**	**variance (robust)**	**mean (no robust)**	**variance (no robust)**	**p score**
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	$11,4$	$20,50$	$11,15$	$21,12$	$0,37$

At first sight it looks like the variant where robustness is ignored is able to plan slightly more prebooked PRMs then the robust variant, with an average of 1,08 of declined prebooked for the no-robust and an average of 1,15 for the robust variant. However after doing the student t-test the p score is 0,52 which is way higher then the threshold and we can't reject the null hypothesis that both distributions for that statistic are equal. Apparently the slightly higher average isn't something to worry about.

The wait time however shows a major improvement in the case we consider robustness over the variant where we neglect the robustness. An average of 87,35 minutes of additional wait time over all PRMs versus an average of 170,6 minutes. The p value for that statistic is way under the threshold, therefore we reject the null hypotheses and conclude that in this statistic the robust variant scores better. This isn't a big surprise because the whole goal of the robustness was to catch up with minor delays.

The number of updates done differs between the robust variant and the variant where robustness is neglected. This is surprising because both algorithms had the same new PRMs and plane delays. While the full reschedules doesn't differ too much to conclude that those 2 distributions are not equal, the p value for reschedule updates falls under the acceptance threshold, probably due the difference in variance. This could be explained by the algorithms might decline different PRMs, or the one algorithm perform better at one instance but worse for another resulting in a different number of reschedules but a similar number of declined PRMs. Also for the robust variant the mean is slightly higher but got a smaller variance. Although the distributions for that statistic are apparently not equal it is also not clear which we would prefer. We had hoped to see a difference in the full reschedule, but there seems to be a very minor difference between the 2 variants, not enough to say one is better. Therefore the difference in updates doesn't say much.

Its seems like the 2 variants provide quite similar results on the statistics except for the accumulated wait time over the PRMs where the robust variant is clearly better. There is no surprise that the robust variant would increase robustness, but it seems like that it solely increases robustness of the schedule. Because our goal is to provide the best service as possible to the PRMs we would prefer using the robustness as an additional objective, since it reduces the chances a PRM has to wait for a connection while being able to serve a similar amount of PRMs. There are still some declines, but since we have added many new PRMs we made the instances harder then the instances given to us by Reinhardt et al[1], which are based on a day at a real world airport containing both prebooked and immediate PRMs. Even with more PRMs our algorithm is able to schedule most of them, which is a nice result.

References

[1] Reinhardt, L.B. Clausen, T. Pisinger, D. Synchronized dial-a-ride transportation of disabled passengers at airports, *European Journal of Operational Research (2012), doi: http://dx.doi.org/10.1016/ j.ejor.2012.09.008*

[2] Han Hoogeveen. Multicriteria scheduling. *European Journal Of Operation Research 167, 2005,* Pages 592-623.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms, second edition. *The MIT Press.* Cambridge, Massachusetts London, England.

[4] James Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics , Vol. 5, No. 1 (Mar., 1957),* pp. 32-38. Published by: Society for Industrial and Applied Mathematics. Article Stable URL: http://www.jstor.org/stable/2098689.

[5] C. H. Papadimitriou, P. C. Kanellakis. Flowshop scheduling with limited temporary storage. *Journal of the ACM 27(3) (1980),* pp. 533549.

[6] Kevin Ian Smith. A Study of Simulated Annealing Techniques for Multi-Objective Optimisation University of Exeter, October 2006.

[7] Garey, Michael R. and David S. Johnson (1979), Computers and Intractability; A Guide to the Theory of NP-Completeness.

[8] Mascis, A. and D. Pacciarelli (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research 143 (3),* Pages 498-517.

[9] Richard Freling, Albert P.M. Wagelmans and José M. Pinto Paixão Models and Algorithms for Single-Depot Vehicle Scheduling. Econometric Institute, Erasmus University, Rotterdam, The Netherlands, DEIO Universidade de Lisboa, Lisbon, Portugal.

[10] Guy Desaulniers and June Lavigne and François Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research 111 (3),* Pages 479494.

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

[12] Richard M. Karp (1972). Reducibility Among Combinatorial Problems. *Complexity of Computer Computations.* New York: Plenum. pp. 85103.

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

[14] G. Ayorkor Mills-Tettey, Anthony Stentz, M. Bernardine Dias (2007). The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213.