Optimizing bus line platform assignment across bus stations in Utrecht

Wouter ten Bosch¹ · J. A. (Han) Hoogeveen¹ · Marcel E. van Kooten Niekerk^{1,2}

Accepted: 28 May 2021 / Published online: 1 July 2021
© The Author(s) 2021

Abstract

Due to its increased popularity, public transport has grown considerably, which has resulted in more lines that are operated with higher frequencies. As a result, the current bus stations that are located in strategic places like city centers to serve as a hub are becoming too small. If there is no space to enlarge the station, then usually the best option is to create a second bus station close-by. This raises the problem of distributing the bus lines over the bus stations, such that good connections are offered to transfer passengers. We have considered this problem in the city of Utrecht, which is a middle-sized city in the center of the Netherlands. The central transit hub, which is located next to the central railway station, consists of several separate bus stations. The goal of the research is to minimize the total travel time for all passengers who want to transfer at the hub. Here we are not allowed to adjust the current timetable, and we have to take the capacity and vehicle limitations into account. To find out which journeys are made daily and by how many people we use data from a digital fare system. This results in passenger groups, and for each group we compute the relevant travel options given the current timetable. Thereto, the routes are split into an inbound itinerary, a transfer within the same bus station, and an outbound itinerary; the validity of a travel option depends on the assignment of the bus lines to the bus stations. We decompose the problem into first finding a distribution of the lines over the stations and then assigning them to a platform at the station of choice. In the first subproblem, we find the best set of transfers using Integer Linear Programming (ILP), resulting in a station assignment. In the second subproblem, for each station, we distribute the bus lines over the platforms. In this subproblem, there can be multiple lines assigned to a single platform, as long as there are never more vehicles at the platform simultaneously than physically fit. The goal is to maximize the comfort of the transfer passengers by assigning tight transfers to adjacent platforms; this problem is solved using ILP as well.

Keywords Public transport · Journey groups · Bus line assignment · Itineraries · ILP · Routing · Decomposition · Transfers · Travel data

1 Introduction

In many cities public transport has been organized according to the hub and spoke paradigm, with a hub in the city center. Usually this hub is located next to the central railway station such that passengers can transfer between lines, change modality between train, bus or tram, and arrive within walking distance of the city center. The down-side of such a location in the city center is that the area is densely built, which makes space scarce, and hence it may not be possible to enlarge the hub to keep up with the growth in public transport. Therefore, the only option is then to build a second bus station close-by. Consequently, this raises the question of how to divide the lines among the bus stations, such that transfer passengers are offered a good connection, preferably without having to walk to the other bus station in between.

The problem sketched above was inspired by the situation that occurs in Utrecht, which is a medieval, middle-sized town in the center of the Netherlands.¹ Located in the center of the city of Utrecht is the railway station, *Utrecht Centraal*, which is the central railway station of the region and the busiest and largest railway station in the Netherlands. For obvious reasons, the hub was built next to the railway station, but it has become too small to serve all buses. Recently, another bus station has been built at the other side of the station. Because of the size of the train station and the crowds in the station hall, walking between both ends of the station takes quite some time and should be avoided as much as possible. Therefore, it is mandatory to find a division of the bus lines that provides good transfer possibilities within the same bus station.

At the moment this bus line assignment is done manually, based on the experience of public transport developers. In this paper, we present a data-driven method. We start with determining the journeys that are made by bus passengers in Utrecht; hereto, we use passenger travel data collected from the digital fare system. We use this as input to obtain a good platform assignment such that the passenger travel time is minimized. Although we base our case study on the situation in Utrecht, our techniques are generally applicable.

The paper is organized as follows. In Sect. 2 we provide a short overview of the literature on related problems. This is followed by a more precise description of the problem in Sect. 3. The problem is decomposed into two subproblems, station assignment and platform assignment, which in turn are discussed and solved in Sects. 4 and 5. Both problems require substantial preprocessing in order to find good itineraries to determine the travel time for usage in a model. The required preprocessing is also discussed in these sections. Finally, Sect. 6 provides an overview of the findings.

¹ See for example <https://en.wikipedia.org/wiki/Utrecht>.

2 Literature review

Adenso-Díaz (2005) has developed a method for a similar problem at a bus station in Oviedo, Spain. In this approach all constraints are mapped as rules, for example, “the buses to Madrid must be close to the buses to Barcelona”. These rules are used to generate the assignment using a heuristic approach: start assigning the lines that have the fewest alternative options and continue until all lines have been assigned to a platform. This can be a feasible approach as long as rules are well defined and their state can be checked fast enough. In our problem, however, there are no predefined rules and a more data-driven method is used.

Daduna and Voß (1995) consider the problem of constructing a good timetable for a given bus line schedule; here the goal is to enable good transfers to allow smooth connections and hence minimum travel times. Although our goal is the same, the two problems are quite different in a number of aspects. First of all, our time table has already been determined, albeit that we can make small changes to model the driving times from the last stop to the station and from the station to the first stop. Second, we do not know yet which itineraries the passengers will choose, as this depends on the assignment of the buses to the stations; hence we neither know the number of passengers nor the transfers that they will make.

At first sight, the problem in Utrecht seems similar to the AIRPORT GATE ASSIGNMENT PROBLEM (AGAP) as defined by Braaksma and Shortreed (1971). This is a widely studied problem where incoming aircraft flights need to be assigned to airport gates subject to certain constraints. Constraints can depend on the type of aircraft, the origin or destination of the aircraft, arrangements with the airline, etc. The primary constraint is that a gate can handle one flight at a time and that the gate is able to handle the type of aircraft. Gates can be assigned for a day period, but robustness has to be included for delayed and early flights. Besides the listed constraints, the assignment can be optimized for specific objectives, for example, by minimizing the walking time for transfer passengers. AGAP is an NP-hard problem, as shown by Obata (1979). In recent years AGAP has been studied using various algorithms; Dorndorf et al. (2007), Cheng et al. (2012) and Aktel et al. (2017) present overviews of the recent developments on this problem.

The problem studied in this project differs from airport gate assignment in many ways. First of all, gate assignment is a problem in a dynamic environment: the incoming flights and their schedule differ on a day-to-day basis, while in our problem the final assignment is static. In the case of gate assignment similar flights can be assigned to different gates during the day, while we want to keep the same platform for buses from a particular line throughout the day.

Similar remarks can be made with respect to the seeming correspondence between our problem and the berth allocation problem (see for example Imai et al. 2001; Cordeau et al. 2005). In our problem we work with fixed lines, and hence each bus used to serve this line is assigned to the same bus station, whereas in the berth allocation problem the best place to moor is determined for each vessel individually depending on the available space of the piers in the port, which changes each day.

3 Problem description

We want to obtain an optimal assignment of bus lines to bus platforms, which are located at several bus stations. The objective is to minimize total travel time. This can be attained by putting bus lines that have many transfer passengers on the same bus station. Of course, this effect is more significant for busier lines, and for passengers who have fewer alternative travel options.

The timetable is considered static and known in this problem. However, small changes will have to be made to the timetable to incorporate different station assignments, for example: when a bus line is assigned to another bus station compared to the current situation, the required driving time may change. We will estimate these driving times, for which we use data from comparable situations at other lines.

At most platforms at the hub in Utrecht multiple buses can stop simultaneously, depending on the length of the platform. Also, there are multiple types of vehicles used. Vehicles may vary in the drivetrain (conventional or electric) and length (and thus turning circle), making some platforms unsuitable for specific vehicle types. In some cases, two or more bus lines have to be assigned to the same station or platform, or in other cases, individual lines have to be assigned to specific platforms. The latter constraints can be modeled as restrictions on which lines can be assigned to which platforms.

Decomposition In this paper, we take a decomposition approach. The final goal is to find a suitable assignment of bus lines to platforms, but, in this approach, first a station assignment is constructed, where bus lines are assigned to one of the stations. The next problem is, to find a suitable platform assignment for each station. This decomposition makes both subproblems easier to solve, and thus the combined problem easier to solve. While this approach loses optimality for the combined problem, we expect that this approach will perform well, because we expect the platform assignment only to have a small effect on the overall score. For example, if two lines will swap platforms on the same station then, in the worst case, the involved groups of passengers have to additionally walk the distance between these platforms. The maximum distance between two platforms on the same station will typically be a lot shorter than the minimum distance between two stations, meaning that the maximum impact of a change in the platform assignment will typically be a lot smaller than the minimum impact of a change in the station assignment.

Historical travel data The objective of this problem is to minimize overall travel time which can be achieved by assigning bus lines with transfer passengers to the same station. To determine which lines have transfer passengers, we use historical data from the digital fare system, which records most journeys made by passengers. These journeys give a good indication of typical routes and transfers taken by passengers, under the condition that the timetable remains unchanged. However, we can not construct a station assignment solely based on the routes and transfers made by passengers now, because potentially good transfers would be left undiscovered. If a transfer is currently not attractive, because for example, two connecting lines

terminate at different stations, it might never be considered an option because it can not be used now.

Besides that, in a high-density public transport network like Utrecht, passengers often have many options to reach their destination. The location and timing of the transfer in their journey determine for a large part which itinerary is optimal for a given passenger. These alternative routes will have to be included in the model because it frequently occurs that two lines share a part of a line route in these high-density networks.

To this end, we do not use the routes or transfers from historical travel data but solely use the pairs of origin and destination locations, referred to as *journeys*. Station assignments are scored against a set of typical journeys made on the network. These scores are based on the total travel time: the time between departure at the origin location and arrival at the destination. This travel time is based on the shortest route on the network, given the station assignment and associated timetable.

Grouping journeys For computational reasons, not all individual journeys are included, but they are combined into *journey groups*. During the day, frequencies and departure times may change for individual lines in the timetable; in other words, the timetable is aperiodic, and therefore journeys taken at different times of the day may result in different optimal routes. This means we can not merely combine journeys with the same origin and destination locations without considering the time aspect. To this extent, journeys are categorized in time windows, where each time window represents a period of the day (rush hour, midday, evenings). To limit the size of the model journey groups with low weights (few daily passengers) are discarded.

We define a journey group as a tuple consisting of the origin and destination locations and a time window with an associated weight (number of daily passengers making the journey). Later we try to find a station assignment that allows the optimal route between the origin stop and destination stop within the time window, where the scores are weighed by the number of daily passengers.

4 Station assignment

4.1 Solution approach

In the station assignment subproblem, bus lines have to be assigned to stations subject to certain constraints. Before solving the assignment problem, we need to generate feasible transfers. The assignment problem can then be reduced to finding the assignment that allows for the best set of feasible transfers that satisfy the practical constraints. These constraints can be expressed well in a linear program, allowing the problem to be solved by Integer Linear Programming.

Finding feasible transfers For each of the journey groups described earlier, we need to find potential station transfers, regardless of the specific station assignment. To do this we split the journey of a passenger in three sections: the section from the origin location to the last stop before the central hub (the *inbound section*), the section where the passenger transfers on another line at the central hub (the *transfer*

section), and finally the section from the first stop after the central hub to the destination (the *outbound section*), as depicted in Fig. 1. The reason to split a journey in this way is that the inbound and outbound section can be precomputed; they do not depend on the station assignment, but solely on the timetable (which is fixed).

We process each journey group as follows. First we compute all inbound itineraries to a last stop before the hub, for each of the lines. We also compute all outbound itineraries (from a first stop after the hub). We can now ignore the specific routes in these itineraries. In the example of Fig. 1, for the inbound itineraries only, all we need to know is which travel options are possible between s and $\{l_1, l_2\}$, which are characterized by their departure times and the arrival times at the hub. The route is calculated until the last stop before the station area, because in this stage we have not decided to which of the stations the line is assigned. To compute these itineraries we use an adjusted version of the **RRAPTOR** algorithm developed by Delling et al. (2015); this is described in Sect. 4.2.1. Similarly, all outbound itineraries are all travel options between $\{f_1, f_2\}$ and t .

We can now find transfers as follows: For each combination of an inbound and an outbound line, and for each combination of an inbound station and an outbound station, we compute a score. This score is based on the total travel time and the number of transfers underway; if necessary we can adjust the scores to take additional, station-related characteristics into account. In the example of Fig. 1, the total travel time can be computed as the time it takes from s to $\{l_1, l_2\}$, the driving time from $\{l_1, l_2\}$ to the inbound station, the time it takes to transfer at the station(s), the driving time from the outbound station to $\{f_1, f_2\}$ and finally the travel time from $\{f_1, f_2\}$ to t . If the inbound and outbound line are located on the same station, this transfer time is the time between arrival and departure. If the inbound and outbound line are located on different stations, this transfer time between arrival and departure should be at least equal to the walking time between the stations. How this can be implemented is described in Sect. 4.2.2

4.2 Preprocessing

4.2.1 Public transport network

To find all inbound and outbound itineraries, to and from the central transit hub (i.e. the stations), we use a public transport routing algorithm. Because all itineraries that need to be computed either start or end in one of the stations, we can use a single-source multiple-sink routing algorithm. For a certain stop p we are interested in all optimal (fastest in time, fewest transfers) itineraries to and from the central transit hub for every line l ; in other words, which are the travel options between this stop p and the hub when arriving or departing at the hub with line l . However, it is much faster to compute the answer to the question: for a particular line l , which are the optimal itineraries from the hub to every stop p , and vice versa. Given these values we can then compute options for the inbound (outbound) itineraries including a transfer to another line and/or walking.

Fig. 1 Splitting journeys into multiple parts s is the origin location, t is the destination. The numbers on the arcs represent bus lines. l_1 and l_2 are the last stops before the central hub of lines 1 to 4, f_1 and f_2 are the first stops after the central hub of lines 5 and 6. In this illustration there are three routes to get from s to the hub, by taking line 1, line 2, or line 3 and 4 with a transfer at u . There are two routes to get from the hub to t , lines 5 and 6

These itineraries are obtained using Round-Based Public Transit Routing (Raptor) as proposed by Dellling et al. (2015), which is adjusted to be suitable in the situation described below. This algorithm can compute all *Pareto-optimal* itineraries between an origin stop and a destination stop given a departure time. Pareto-optimality means no improvement of a single aspect exists while not making at least one aspect worse, i.e., for all of the resulting journeys we can not find a journey with lower travel time while also not increasing the number of transfers, and vice versa.

The RAPTOR algorithm By using the structure of public transport networks, this algorithm performs well even on large networks. The key idea behind this algorithm is that if two trips follow the same route and have the same *runtime pattern*, then the earlier trip can never overtake the later trip. In other words, if a trip can be taken, all later trips (with the same route and runtime pattern) do not have to be considered. This assumption is used to process routes instead of trips since for every route there is only one optimal trip, depending on the time and location of the passenger, and assuming that there are no capacity limits.

By default RAPTOR is able to find the earliest arrival time for a certain destination stop, given a departure stop and time. The basic principle behind the algorithm is to work in *rounds*, where in each round we increase the number of transfers that are allowed by one; this is comparable to the idea behind the Bellman-Ford shortest-path algorithm of increasing the number of arcs that can be used in each iteration. After round k every stop that can be reached with at most $k - 1$ transfers is known. This is not necessarily the fastest itinerary to that destination, but it is the fastest itinerary that can be made with at most $k - 1$ transfers. The earliest arrival time at stop p in round k is denoted as $\tau_k(p)$, the earliest arrival time regardless of the number of transfers is denoted as $\tau^*(p)$. During the execution of the algorithm a *marked set* is kept, containing all stops that had an improvement of $\tau^*(p)$ in the previous round.

The algorithm starts with a departure stop p_s , and departure time $\tau_0(p_s)$. The departure stop p_s is also marked since it is updated. In every round $k \leftarrow 1, 2, \dots$ the following is done: each route is processed from the earliest stop that is marked in

that route. At this stop, the earliest trip t from the route that one can catch is retrieved. For each following stop p on the route the variables $\tau_k(p)$ and $\tau^*(p)$ will be updated with the arrival time of that trip at stop p , and p is marked. Unless $\tau^*(p)$ already has an earlier arrival time prior, in that case there is another itinerary that reaches p earlier and with fewer transfers. In that case the algorithm checks whether it is possible to catch an earlier trip on the route, and then continues with each following stop p in the route using that trip.

After all routes have been processed, the footpaths are included as follows: for each marked stop p_1 , for every stop p_2 that can be reached by a footpath from stop p_1 , update $\tau_k(p_2)$ and $\tau^*(p_2)$ accordingly and mark p_2 . At the end of the round $\tau_k(p)$ is correct up to round k , and there is a new marked set for the next round. The algorithm terminates if the marked set is empty, meaning no improvements in arrival times can be made by increasing the number of transfers. For destination stop p_d now $\tau_k(p_d)$ contains the Pareto-optimal set of all itineraries.

Adjusting RAPTOR Before this algorithm can be used for our problem, some changes have to be made, since instead of calculating optimal journeys between two locations, we try to find optimal itineraries for each stop x from and to the central hub (inbound and outbound itineraries). Which one of the stations in the hub will be used for the final journey is not known in this stage of the approach. The algorithm should start or end in a certain line that must be assigned to a bus station instead of a certain stop. In addition, it is not enough to know which is the optimal itinerary at a particular departure time, because at this stage the departure time after a transfer is made at the hub is unknown. Which transfers can be used at the hub is determined later, and if a faster transfer at the hub can be found, it might be possible to catch an earlier trip, resulting in an earlier arrival. Both these problems can be solved by adjusting the algorithm as described in the remainder of this section. We solve the departure time problem first, next we create two versions of the algorithm: to find inbound itineraries and to find outbound itineraries.

To solve the problem of varying departure times, we use rRAPTOR to find all optimal itineraries in a certain time window. This variant of the RAPTOR algorithm is also described in the paper by Delling et al. (2015). Instead of finding the Pareto-optimal itinerary from a certain location given a certain starting time, it finds all Pareto-optimal itineraries in a certain (departure) time window. We do this because we are interested in the best journeys within the time window. Dominated itineraries are discarded, where one itinerary dominates another one if it arrives later but departs earlier. These itineraries can be calculated efficiently by running the RAPTOR algorithm for each one of the departure times of trips at the departure stop, starting with the latest trip in the time window, and processing each trip by departure time in decreasing order. A further explanation of this method can be found in the paper by Delling et al. (2015). Now, rRAPTOR can be used to find all optimal itineraries in a time window, between two given locations. However, in this phase it is yet unknown at which one of the stations a line stops. To this end, rRAPTOR is adjusted as described next, to find inbound itineraries (rRAPTOR-IN) and to find outbound itineraries (rRAPTOR-OUT).

We start by describing rRAPTOR-OUT. Here, instead of starting the journeys at an origin stop, we want to start while traveling with a certain line. After all, we are

interested in all optimal itineraries to every stop while leaving (or entering) the hub. We consider line l_s as the line used to leave the hub, with the first stop after the hub being p_s . The algorithm is extended as follows:

Execute the RAPTOR algorithm for every trip t of line l_s in the time window, sorted by decreasing departure time, just like RRAPTOR does. Now, start each execution with a special round $k = 0$. This special round works as follows: For every stop p_i in trip t (after the departure stop p_s), set $\tau_0(p_i)$ and $\tau^*(p_i)$ equal to the arrival time of trip t at stop p_i , i.e. we can reach every stop p_i in the trip of the line we use to depart from the hub in round $k = 0$ (without transfers). Next mark each p_i , so the next iteration considers transfers from this stop p_i . After this special round $k = 0$, continue with the normal rounds $k = 1, 2, \dots$, which start iterating on the marked stops. The number of transfers required is now k instead of $k - 1$ (because round 0 now processes all stops that can be reached directly using line l_s). This results in all Pareto-optimal itineraries from the central hub to every stop in a certain time window.

The RRAPTOR-IN algorithm is very similar. We also need all the itineraries from every stop to the hub. For that to work, we need to invert three elements of the RRAPTOR-OUT algorithm: (1) Start with the earliest trip in the time window instead of the latest trip. (2) Start at the destination line instead of the origin line, i.e. instead of finding all journeys to every stop while starting with a certain line, we want to find all journeys from every stop while ending with a certain line. (3) Process every trip backwards. Instead of processing stops in the trip in the order that they arrive, we start with the latest stop in the trip and work backwards until the first stop. Footpaths are symmetric and do not have to be reversed. The RRAPTOR-IN algorithm thus finds all Pareto-optimal (minimum travel time, minimum number of transfers) journeys between every stop in the network to the station area while entering the station area with a certain line.

Preprocessing lines Every line in the network as seen from the central hub is either an inbound line, outbound line, both or neither. An inbound line is a line that has an arrival stop (i.e. not the first stop in the route) at the hub. An outbound line is a line that has a departure stop (i.e., not the last stop in the route) at the hub. Lines that do not stop at one of the stations are ignored in this stage. Lines can be both inbound and outbound; either passing through the station or starting at the station, driving a route and ending at the station.

For every outbound line RRAPTOR-OUT is executed. This results in all optimal journeys from every outbound line at the stations to every stop in the network. Likewise, for every inbound line RRAPTOR-IN is executed. This results in all optimal itineraries from every stop in the network to every inbound line at the stations. All results are stored in memory and are used for computing transfers.

4.2.2 Transfers

Next, we generate feasible *station transfers* and their associated score for each one of the journey groups based on the computed itineraries. A station transfer connects an inbound line and an outbound line, where preferably both lines are located at the

same station. So, a station transfer can be denoted as a combination of inbound line, inbound station, outbound line and outbound station.

First, we need to obtain a *base score* for the journey group. This is the score if no station transfer is used, i.e. passengers in this journey group travel without passing the hub. Station transfers that result in a worse score than this base score will be discarded since it is always faster to travel without passing the hub. The generation of station transfers is done merely by enumerating all possible travel options. Fortunately, many combinations can be discarded before or during the calculation.

Given a certain station transfer connecting an inbound and an outbound line, there are multiple travel options because the frequencies of the inbound line and outbound line do not match and the timetable is not periodic. Based on the travel options possible with the station transfer, a score is assigned to the station transfer. This score can be constructed in different ways, depending on what the assignment should optimize for. At least the score should contain the travel time, but this can be extended by, for example, the number of transfers made in the entire itinerary (an itinerary can include transfers outside the hub), walking distances, waiting times or travel distance. This scoring function must also give a score to the situation where no transfer at the station is used. We use the following function:

$$\begin{aligned} \text{transferScore}(\text{travelOptions}) = & \text{travelOptions} \cdot \min(\text{travelOption}_{\text{travelTime}} \\ & + \text{transferPenalty} \cdot \text{travelOption}_{\# \text{transfers}}) \end{aligned} \quad (1)$$

where *transferPenalty* is a configurable number of seconds to make station transfers that contain more transfers in the rest of the route unpopular. To score a station transfer, the minimum travel time (time between departure at the origin stop and arrival at the destination stop increased with transfer penalties) of each of the travel options is used. The base score of a journey group is in the same way based on the best travel option while not traveling via the hub.

The algorithm starts with calculating a base score for the journey group and then processes every combination of line and station. Processing consists of three steps: it first checks whether the combination is allowed, then enumerates all possible travel options with this transfer, and finally computes the score based on these travel options. All the journey groups can be processed in parallel, because there are no relations between journey groups.

Since this preprocessing only needs to be done once, we let our algorithm naively check all travel options possible in a transfer. This can be done more efficiently. Depending on the scoring function used, not all travel options have to be checked. More precisely, only the Pareto-optimal travel options have to be included, where the included aspects depend on the scoring function. For example, if only the travel time is included, then every inbound itinerary only has to be combined with the outbound itinerary that does not leave earlier than possible by the transfer and arrives the earliest. Every later or earlier itinerary can be discarded.

Some travel options can be discarded early if they have a worse score than the base score. For example, if only the travel time is included in the score, then every inbound itinerary or outbound itinerary that already has a longer travel time than the

base score can already be discarded, removing a larger number of lines that are not useful for the journey group.

4.3 Model

The Station Assignment Problem is solved using Integer Linear Programming. The input for this ILP is the data generated in the preprocessing phase. We use the following notation:

$l \in \{1, 2, \dots, L\}$	denotes the lines to be assigned
$s \in \{1, 2, \dots, S\}$	denotes the stations that the lines can be assigned to
$j \in \{1, 2, \dots, J\}$	denotes the journey groups
$t \in T_j$	denotes the set of feasible station transfers for each journey group j as generated in the previous section

Furthermore, $line_{in}(j, t)$ and $line_{out}(j, t)$ represent, respectively, the inbound and outbound lines associated with transfer t of journey group j and $station_{in}(j, t)$ and $station_{out}(j, t)$ represent the inbound and outbound station of the transfer.

As described earlier, each transfer t for a given journey group j consists of two line-to-station assignments (one inbound and one outbound) and a calculated score $s_{j,t}$. In short, transfer t of journey group j has a score of $s_{j,t}$ if $line_{in}(j, t)$ is assigned to $station_{in}(j, t)$ and $line_{out}(j, t)$ is assigned to $station_{out}(j, t)$; $s_{j,t}$ is defined as the improvement of the score of journey group j if transfer t is used compared to the situation where no transfer at the hub is used. Hence we find that

$$s_{j,t} = weight_j \cdot (baseScore_j - score_{j,t}) \quad (2)$$

where $weight_j$ is the average number of daily passengers in journey group j .

A solution is only valid if we obey the capacity constraints. Station capacity can be defined in multiple ways, but a reasonable method is to set the required capacity of a line to the daily number of trips. The available capacity of a station is then set to the maximum number of daily trips the station can handle. This number is computed by simply counting the total space available at the stops measured in number of buses times 60 and divided by the nominal time for one bus visit at the stop. This is a quite primitive method, but at least it makes sure that the model does not allow to assign all lines to the same station while at the same time prevents the problem from turning into mainly a knapsack problem. Besides, most trips are evenly spread out over the hour. We use that c_l denotes the required capacity of line l , while C_s denotes the available capacity at station s .

Some lines must be assigned to a specific station (for example, if the line is run by electric vehicles and charging equipment is only available at a particular station). In this case, it makes little sense to include the line in the final ILP formulation handed to the solver. To maintain a feasible outcome, all transfers requiring a line-station assignment that is contrary to the fixed assignment of that line are removed from the input. Subsequently, the capacity of the station C_s is lowered by the required capacity of the fixed line c_l .

Another constraint is that certain sets of lines have to be assigned to the same station, but it does not matter to which station. This constraint might occur in a situation where two lines run in more or less the same direction, but with, e.g. a half-hourly frequency. Together they will form a quarter-hourly frequency for the shared section of the route of both lines. In this case, it is useful for passengers that both lines are assigned to the same station. Another situation where this constraint occurs is when a line continues after stopping at the station as another line. This constraint is also implemented by preprocessing the input; all lines that form a combination are removed from the input and a combined new line l is added to the input. The required capacity of l is set to the total capacity of all lines in the combination. All transfers that contradict this constraint (i.e. require two lines in the combination to be assigned to different stations) are removed from the input data.

4.4 ILP formulation

The ideas described in the previous subsections let us obtain the following ILP:

Decision variables There are two sets of (binary) decision variables in this model. $a_{l,s} \in \{0, 1\}$ indicates whether line l is assigned to station s , and $x_{j,t} \in \{0, 1\}$ indicates whether journey group j uses station transfer t .

Objective The objective in this problem is to select the transfers with the highest score in each of the journey groups such that the constraints below are met.

$$\text{maximize } \sum_{j=1}^J \sum_{t \in T_j} s_{j,t} \cdot x_{j,t} \quad (3)$$

Constraints Every line should be assigned to a station, in other words, no line may be left unassigned. A line cannot be assigned to multiple stations.

$$\sum_{s=1}^S a_{l,s} = 1 \quad \forall l = 1, \dots, L \quad (4)$$

Every journey group can only have at most one active transfer. If multiple transfers are possible with the current assignment, then the model will use the transfer with the highest score. It is possible that no transfer is active; in that case the shortest route for passengers in the journey group stays outside the hub.

$$\sum_{t \in T_j} x_{j,t} \leq 1 \quad \forall j = 1, \dots, J \quad (5)$$

In order for a transfer to be active, the line-to-station assignments required by that transfer should be active as well. In other words, if a transfer is active ($x_{j,t} = 1$), then both $a_{\text{line}_{in}(j,t), \text{station}_{in}(j,t)}$ and $a_{\text{line}_{out}(j,t), \text{station}_{out}(j,t)}$ must have a value of 1.

$$a_{line_{in}(j,t),station_{in}(j,t)} + a_{line_{out}(j,t),station_{out}(j,t)} - 2 \cdot x_{j,t} \geq 0 \quad \forall j = 1, \dots, J \quad \forall t = 1, \dots, T_j \quad (6)$$

The total required capacity by the lines assigned to a station cannot exceed the available capacity at that station.

$$\sum_{l=1}^L a_{l,s} \cdot c_l \leq C_s \quad \forall s = 1, \dots, S \quad (7)$$

All decision variables are binary.

$$a_{l,s} \in \{0, 1\} \quad \forall l = 1, \dots, L \quad \forall s = 1, \dots, S \quad (8)$$

$$x_{j,t} \in \{0, 1\} \quad \forall j = 1, \dots, J \quad \forall t \in T_j \quad (9)$$

5 Platform assignment

5.1 Solution approach

Based on the resulting station assignment, a platform assignment has to be made for each one of the stations. With this platform assignment we assign lines to platforms subject to the constraints in Sect. 3. This platform assignment will be static, and will be used throughout the day. It is known which transfers are being used because this can be derived from the station assignment. This information can be used to find a good platform assignment. This subproblem can be solved using an ILP as well. We first try to find a feasible assignment subject to the constraints, before adding an objective function later this section.

Some platforms are unsuitable for specific vehicle types. In some cases, two or more bus lines have to be assigned to the same platform, or in other cases, lines have to be assigned to specific platforms. These constraints can be modeled as restrictions on which lines can be assigned to which platforms.

At most of the platforms multiple buses can stop simultaneously, depending on the length of the platform. Of course, there should never be more vehicles at a platform than fit if the timetable is followed. We refer to this as the *platform capacity constraint*. We discuss how to model it below.

5.2 Preprocessing

Platform capacity constraint Every line needs to be assigned to exactly one platform, but a platform can have multiple lines assigned to it. Whether a combination of lines can be assigned to the same platform depends on their timetable; there should never be more vehicles simultaneously at the platform than physically fit. Some platforms might only have space for one vehicle, in which case lines can only be assigned to this platform if their timetables do not overlap.

This constraint is hard to solve in the ILP formulation because it depends on the timetable of the entire day, which can be irregular. However, whether two or more lines can be assigned to the same platform does not depend on which specific platform they are assigned to, except for the length of the platform, which may vary between platforms. This makes this constraint suitable for preprocessing; feasible combinations of lines can be generated beforehand. The ILP now assigns combinations to platforms, where every platform can only have one combination assigned to it, and every line should be in precisely one combination.

The problem of different lengths at the platforms within a station is solved using the following approach: V_{max} is set to the largest platform length (in terms of lengths of buses, where we assume that all buses have equal length) occurring at the station. Now we generate feasible line combinations, i.e., combinations of lines, where combinations are discarded if at any point in time more than V_{max} vehicles are at the platform (when assuming all vehicles have equal lengths). For platforms with less capacity than V_{max} a constraint can be added to the ILP, where the combinations that can be assigned to that specific platform are limited to the subset of combinations that fit that lower capacity. To make sure that there are combinations suitable for platforms smaller than V_{max} as well, every line combination with at most V_{max} vehicles is considered, including combinations of one line only.

Combining lines Whether a combination of lines never has more than V_{max} vehicles at the platform can be determined by the timetables of the individual lines. This is done by listing every *trip* of these lines, with their associated arrival and departure time at the station. To make sure small deviations from the timetable do not cause problems with full platforms, the arrival time is decreased with some time, and the departure time is increased with some time.

One thing to note is that the timetable does not state the arrival and departure time at the station, because this depends on the assignment determined in the previous stage. Just like in Sect. 4.2.2 these arrival and departure times are calculated based on the stops before and after the station and the estimated driving times between these stops and the station where the line is assigned to.

For a given set of lines, with a list of arrival and departure times, the maximum number of simultaneous vehicles at a platform can be calculated by processing the list in chronological order and keeping a counter. For each of the arrivals, increase the counter by one, and with each departure, decrease the counter by one. The maximum value of the counter during the day is the maximum number of simultaneous vehicles at the platform. This can be calculated for a single line or a combination of lines by combining the list of events while keeping the chronological order. An example of this is shown in Table 1.

Generating combinations efficiently We need to find the set of feasible combinations of lines that satisfy V_{max} . However, this number of combinations of lines grows quite quickly. We denote L_{max} as the maximum number of lines that can be assigned to a single platform, which is an input parameter. The number of combinations of lines is thus $\sum_{c=1}^{L_{max}} \binom{L}{c}$. Most of these combinations do not satisfy V_{max} . The observation can be made that for a combination to satisfy V_{max} , all proper

Table 1 Example: Combining lines under V_{max}

line 1		line 2		merged into	lines 1 and 2	
t	$V_{max} = 1$	t	$V_{max} = 1$		t	$V_{max} = 2$
11:10	Arrive	11:08	Arrive	merged into	11:08	Arrive
11:13	Depart	11:11	Depart		11:10	Arrive
11:25	Arrive	11:23	Arrive		11:11	Depart
11:28	Depart	11:26	Depart		11:13	Depart
...		...			11:23	Arrive
					11:25	Arrive
					11:26	Depart
					11:28	Depart
					...	

subsets must satisfy V_{max} as well. This can be used to compute all allowed combinations efficiently.

This method works as follows: combinations are checked against V_{max} in rounds. In the first round individual lines are checked, in the second round every combination of 2 lines will be checked, and so forth, until round L_{max} checks combinations of L_{max} lines. However, every round only generates combinations of lines by extending the combinations that were feasible in the previous round. In other words, if a combination of lines is not feasible, all supersets are not checked, saving a lot of computation time. This approach was inspired by the *Apriori algorithm* as used for mining association rules in the data mining field, as described by Agrawal and Srikant (1994). The working of the algorithm is depicted in Fig. 2.

Fig. 2 Example: generating combinations efficiently. In this example combinations of lines 1 through 4 are checked. A gray box denotes a combination that is feasible. The combination 1, 2 and 3, for example, does not have to be checked because the combination of 1 and 3 is not feasible

5.3 Model

At this stage, all bus lines have been assigned to one of the stations in the area. The remaining problem is to assign bus lines to individual platforms within a station. The platform assignment subproblem is solved using an ILP.

The most important parameters are:

- $c \in \{1, 2, \dots, C\}$: collection of line combinations generated as described above, C denotes the number of line combinations
- $p \in \{1, 2, \dots, P\}$: collection of platforms that line combinations can be assigned to, P denotes the number of platforms
- $m_{l,c}$: a binary parameter that indicates whether line l is included in line combination c ($m_{l,c} = 1$) or not ($m_{l,c} = 0$).
- $a_{c,p}$: a binary parameter that indicates whether line combination c is allowed to be assigned to platform p ($a_{c,p} = 1$) or not ($a_{c,p} = 0$).
- $line_{in}(j)$: The line used as inbound line for the transfer of journey group j .
- $line_{out}(j)$: The line used as outbound line for the transfer of journey group j .

The problem contains two groups of decision variables: $x_{c,p} \in \{0, 1\}$ and $y_{c,p,c',p'} \in \{0, 1\}$. Here $x_{c,p}$ denotes whether line combination c is assigned to platform p , and $y_{c,p,c',p'}$ denotes whether line combination c is assigned to platform p and line combination c' is assigned to platform p' . Decision variable y is used for objectives that allow scoring of transfers. All decision variables are binary. The following constraint makes sure that $y_{c,p,c',p'}$ can only be 1 if $x_{c,p}$ and $x_{c',p'}$ are both equal to 1.

$$x_{c,p} + x_{c',p'} - 2 \cdot y_{c,p,c',p'} \geq 0 \quad \forall c, c' = 1, \dots, C \quad \forall p, p' = 1, \dots, P \quad (10)$$

Every line has to be assigned and thus has to be in a combination that is assigned to a platform. A line cannot be in multiple assigned combinations.

$$\sum_{c=1}^C \sum_{p=1}^P m_{l,c} \cdot x_{c,p} = 1 \quad \forall l = 1, \dots, L \quad (11)$$

Every platform has at most one line combination assigned to it.

$$\sum_{c=1}^C x_{c,p} \leq 1 \quad \forall p = 1, \dots, P \quad (12)$$

In some cases, some lines are excluded from specific platforms. Line combinations containing this line are excluded from these platforms by setting $a_{c,p} = 0$. An example where this might be useful is when a line is operated with electric vehicles, and only some of the platforms have charging equipment installed. In addition, some platforms are shorter than other platforms. If line combination c has any moment during the day in which there are more vehicles simultaneously at platform

p than fit, $a_{c,p}$ is set to 0. The constraints below are handled during the generation of the model.

$$x_{c,p} \leq a_{c,p} \quad \forall c = 1, \dots, C \quad \forall p = 1, \dots, P \quad (13)$$

Finally all variables should be binary.

$$x_{c,p} \in \{0, 1\} \quad \forall c = 1, \dots, C \quad \forall p = 1, \dots, P \quad (14)$$

$$y_{c,p,c',p'} \in \{0, 1\} \quad \forall c, c' = 1, \dots, C \quad \forall p, p' = 1, \dots, P \quad (15)$$

Objectives Multiple objectives are possible when lines have to be assigned to platforms, depending on what is considered important. One of the obvious objectives could be to minimize the walking distance for transfers within a station. This can be done, because it is known which lines have transfer passengers. This turned out to be hard to solve, probably because of the large number of combinations and hence a large number of additional constraints.

To make the problem easier to solve while maintaining a good assignment objective, only journey groups with *tight* transfers are optimized. A transfer is tight if there is little time between the arrival and departure times of the connected lines. The threshold for whether a transfer is tight is configurable. The goal is to assign these lines to an adjacent (or the same) platform.

$$\text{maximize} \sum_{j \in J'} w(j) \sum_{p=1}^P \sum_{p'=1}^P \text{adj}(p, p') \sum_{c=1}^C m_{\text{line}_{in}(j),c} \sum_{c'=1}^C m_{\text{line}_{out}(j),c'} \cdot y_{c,p,c',p'} \quad (16)$$

Equation 16 shows the objective. It contains the following parameters in addition to the parameters described earlier:

- $j \in J'$: collection of journey groups that contain a *tight* transfer in this station.
- $w(j)$: the weight of the transfer made in journey group j . Some transfers can be more important to give a good assignment. Currently this weight is set to the average number of daily passengers. Another option would be to include the time until the next departure, giving priority to transfers where the consequences of a missed transfer are worse.
- $\text{adj}(p, p')$: a binary parameter that has value 1 if platform p and p' are adjacent and 0 otherwise.

The equation in the objective sums over every journey group with a tight transfer, and gives a score equal to the weight of the journey group based on whether the two platforms p and p' are adjacent, and both lines of the journey are assigned to these platforms.

Other objectives Although not tested, other objectives might be possible as well, like:

- *Maximize robustness for line combinations* Using the current constraints, line combinations are assigned to platforms in such a way that, according to the timetable, no more than $V_{\max}$ buses are at a platform at any time. However, some

line combinations are better than other line combinations, because there is more time between the departure of a vehicle and the arrival of another vehicle. A score could be computed for every line combination denoting the available slack in the timetable. Using this objective function, line combinations with more slack can be preferred.

- *Assign similar destinations to the same platform* It might be preferable to have line combinations with similar destinations to be assigned to the same platform. This might make it easier to find the correct platform for passengers, and it is also useful if a passenger misses a trip. The passenger might directly take another trip (of another line) that runs in the same direction. Whether two lines have similar destinations is to be decided beforehand and used as an input parameter. Using this objective function, line combinations with similar destinations can be preferred.

6 Results

The data to test the method and the implementation was provided by Qbuzz, the bus company that serves Utrecht and the region surrounding it. However, the method was developed in a flexible way, such that its use is not limited to this scenario, but can be used for other possible scenarios as well. We have used the data of all 12 million journeys made on weekdays in the period September–November 2017. These data were grouped into 152 journey groups. There are 42 lines in total, where a line that runs both ways is counted for two; each line is usually operated several times per hour. Combining all these led to a total of 34,776 transfers, which we used as input to our model.

The method described in this paper was implemented in C#. The data import and the subsequent preprocessing steps using the adjusted variants of RAPTOR take about 15 minutes to complete, the actual station, and platform assignment take a few seconds to compute. This shows that this method is computationally viable for an instance of the size of Utrecht. The computations were done on an Intel® Core™ i7-2600K Processor with 8 GB of RAM. The ILP problems were solved using Gurobi 7.5.2.

The resulting plans were shown to the planners, who were positive about the quality of the plans. Therefore, our approach will be implemented in practice.

7 Conclusions

In this paper we have shown a method to assign bus lines to platforms across several bus stations. We have described how historical travel data can be used to obtain a model for this problem. The approach includes a decomposition: first a station assignment is constructed, which serves as input for the platform assignment. This made both subproblems easier to solve and thus the combined problem easier to solve. While this approach loses theoretical optimality for the combined problem, our approach performs well in practice and will be implemented in practice.

Section 4 described the assignment of lines to stations. The preprocessing of the network was done by adjusting an algorithm as proposed by Delling et al. (2015), which gives feasible results quickly. After the computation of these itineraries, they can be combined to form potential transfers at each station. Next, these preprocessing results were used to divide bus lines amongst stations.

Finally, the assignment on station level can be used to find an assignment on the platform level, and how this can be done was described in Sect. 5. The approach uses preprocessing to generate feasible line combinations beforehand, which are assigned to individual platforms under certain constraints.

Future work The method described in this paper solves a real-life problem in public transport in the area of Utrecht, and will be implemented in practice. Although designed with the situation in Utrecht in mind, it can be used for similar areas. While this method delivers good results, further research could focus on these elements:

- One assumption made in this paper is that passengers will choose to travel with the itinerary as suggested by route planners. This is usually the fastest route; however, in some cases, there are alternative routes possible where a slightly longer route is chosen if it has fewer transfers. Currently, the chosen route is the fastest where a time penalty is given for each transfer. This can be extended to include other factors like walking distance or waiting time (for transfers). Research could be done on which factors are possible to include in this algorithm, and which factors are important for passengers (by comparing results to historical travel data).
- The currently used objective in platform assignment is to optimize tight transfers, i.e. transfers with a short time between arrival and departure of connecting lines. This objective could be extended to optimize all transfers, including transfers that concern multiple stations. With the current representation of the problem, and an instance of the size of Utrecht, this was not easy to compute. It might be possible to use another representation of the problem, which might allow for other objectives as well.

Finally, we want to remark that the problem of dividing lines over the two parts of a single hub occurs in other areas of the public transport domain as well, for example when there are two airports close-by that serve the same city or region.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Adenso-Díaz B (2005) Rule-based system for platform assignment in bus stations. In: Multidisciplinary scheduling: theory and applications. Springer, pp 369–379
- Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB Conference, pp 487–499
- Aktek A, Yagmahan B, Azcan T, Yenisey MM, Sansarçı E (2017) The comparison of the metaheuristic algorithms performances on airport gate assignment problem. *Transp Res Procedia* 22:469–478
- Braaksma J, Shortreed J (1971) Improving airport gate usage with critical path. *Transp Eng J ASCE* 97(2):187–203
- Cheng CH, Ho SC, Kwan CL (2012) The use of meta-heuristics for airport gate assignment. *Expert Syst Appl* 39(16):12430–12437
- Cordeau J-F, Laporte G, Legato P, Moccia L (2005) Models and tabu search heuristics for the berth allocation problem. *Transp Sci* 39(4):526–538
- Daduna JR, Voß S (1995) Practical experiences in schedule synchronization. In: Daduna JR, Branco I, Paixão JMP (eds) *Computer-aided transit scheduling. Lecture Notes in Economics and Mathematical Systems*, vol 430. Springer, Berlin, pp 39–55
- Delling D, Pajor T, Werneck RF (2015) Round-based public transit routing. *Transp Sci* 49:591–604
- Dorndorf U, Drexel A, Nikulin Y, Pesch E (2007) Flight gate scheduling: state-of-the-art and recent developments. *Omega* 35(3):326–334
- Imai O, Nishimura E, Papadimitriou S (2001) The dynamic berth allocation problem for a container port. *Transp Res Part B* 35:401–417
- Obata T (1979) Quadratic assignment problem: Evaluation of exact and heuristic algorithms. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Wouter ten Bosch¹ · J. A. (Han) Hoogeveen¹ · Marcel E. van Kooten Niekerk^{1,2}

✉ J. A. (Han) Hoogeveen
J.A.Hoogeveen@uu.nl

Wouter ten Bosch
wouter@tenbos.ch

Marcel E. van Kooten Niekerk
M.E.vanKootenNiekerk@uu.nl

¹ Department of Information and Computing Sciences, Utrecht University, P. O. Box 80.089, 3508 TB Utrecht, The Netherlands

² Qbuzz BV, Piet Mondriaanplein 31, 3812 GZ Amersfoort, The Netherlands