10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A model and fast heuristics for the multiple depot bus rescheduling problem

Balázs Dávid · Miklós Krész

Abstract The daily schedule of a transportation company is often disrupted by unforseen events. As a result, a new schedule has to be produced as soon as possible to restore the order. In this paper, we consider the bus rescheduling problem for solving such a scenario. We present a mathematical model for the problem, and also introduce fast solution methods that give efficient solutions with short running time. These methods are tested on different random and real-life instances, and their results are compared to that of the optimal solution of the mathematical model.

Keywords Disruption management · Vehicle scheduling · Heuristic

1 Introduction

Public transportation companies create their daily schedules in advance for a longer planning period. This planning process is carried out by a complex system, an example for which can be seen in [1]. However, several events (the most common of which are vehicle breakdown and lateness) can render the pre-planned schedules infeasible. In most of these cases, companies want to restore the order as soon as possible, and need a new feasbile schedule where all of their tasks are carried out in a feasible manner once again.

Such an unforseen event is called a disruption, and is defined by Clausen et al. in [7] as "an event or a series of events that renders the planned schedules for aircraft, crew, etc. infeasible". Although the above definition was given in a technical paper about airline disruptions, this could be generalized for any

B. Dávid
University of Szeged, Gyula Juhász Faculty of Education
E-mail: davidb@jgypk.u-szeged.hu

M. Krész
University of Szeged, Gyula Juhász Faculty of Education
E-mail: kresz@jgypk.u-szeged.hu

means of transportation. In our paper, we will be dealing specifically with the rescheduling of disruptions arising in public bus transportation.

To our knowledge, there are only a handful of papers that deal with disruptions in bus transportation [12–14]. In practice, the problem is solved by the operators of the company who use their past experience for constructing the new schedule. However, the problem size is large, and many feasible solutions exist for a disruption. Not even the most skillful operator can see all the good possible solutions.

Addressing such a disruption and restoring the order in public transportation should be done as quickly as possible. The reason for this is the fact that if a disruption remains unresolved, it could result in several other disruptions. This is why we need to introduce fast methods for the problem that also give good quality solutions. Instead of solving the problem to optimality, these algorithms should give a number of good quality feasible solutions to the operator as suggestions. Using these suggestions, the operator can make the final decision on how to reschedule the disrupted trips.

In the following sections, we define the bus rescheduling problem (BRP), and give a mathematical model for it. As the size of this mathematical model is large even for smaller instances, we propose two fast algorithms to solve the BRP: a recursive heuristic and a local search method. We analyze the solutions of these algorithms, and compare their results on random instances to the optimal solution of our mathematical model.

2 Disruption management and rescheduling

The structure of the bus rescheduling problem is similar to that of the vehicle scheduling problem (VSP). We are given a set V of vehicles and set T of service trips. Every trip has a departure and arrival time, a starting and ending location, and a set of vehicles that are able to serve the trip. A (t, t') pair of trips are compatible if a vehicle can service both trips with respect to the running time and distance between the arrival location of t and the departure location of t' (such a journey is called a deadhead trip).

The VSP assigns the trips of the given timetable to the vehicles, satisfying certain conditions:

- Every trip in $t \in T$ must be executed exactly once.
- For every vehicle $v \in V$, the trips assigned to v must be compatible with each other.
- The cost of the assignment must be minimal. The cost of the VSP is usually given by two components: a cost proportional to the distance travelled in the solution, and a cost given by the number of buses used.

Furthermore, vehicles can be classified into depots depending on two characteristics: the type of the vehicle (eg. solo or articulated bus), and its starting location at the beginning of the day. In this case, every trip is also assigned a depot-compatibility vector which corresponds to the depots that can feasibly serve it. Moreover, the arising costs can be different from depot to depot.

If the problem has only 1 depot, it is called a single depot vehicle scheduling problem (SDVSP), and can be solved in polynomial time. A formulation for the SDVSP can be seen in [3]. If the number of depots is at least 2, we get a multiple depot vehicle scheduling problem (MDVSP). The MDVSP was introduced by Bodin et al. in [4], and proven to be NP-hard by Bertossi et al. [2]. An overview of different VSP models can be found in [5].

2.1 The bus rescheduling problem

The bus rescheduling problem considers a given daily bus schedule. The schedule consits of several vehicle duties, each such duty corresponding to a unique vehicle. A vehicle duty is a sequence of compatible trips, where compatible means that they can legally be executed one after the other. When a disruption happens in the daily schedule of a company at time s, the current schedule becomes infeasible, and as a result, a number of trips can no longer be executed by their original vehicles.

As an input for the problem, we need to consider the disrupted daily schedule DS, which contains all the vehicles and trips that are still executed according to the original schedule. We also have the set DT of disrupted trips, which contains the trips that cannot be served due to the disruption. The set DT can contain timetabled trips that no longer have their assigned vehicle as a result of the disruption, and it can also contain newly introduced trips that were not part of the original daily schedule. Let $T' \subseteq T$ be the subset of trips that start later than s. The aim once again is to give a feasible solution to the problem by executing the trips $T' \cup DT$ and minimizing the arising costs.

The cost of the problem depends on the restrictions that are taken into consideration. We introduced the following cost components:

- **Operational costs**: This cost is proportional to the distance covered by the given vehicle. If a new vehicle is introduced, it also has a fixed daily cost. This cost can be scaled with a penalty parameter for new vehicles if we want to primarily use our current vehicles in service.
- **Deviation from the original schedule:** If a trip is carried out by a different vehicle in the solution of the BRP than in the original schedule, we introduce an extra penalty. If we take the physical needs of the drivers into consideration, the solution should be as close to the original schedule as possible, and as a result, this cost should be high.
- **Lateness of the trips:** It is possible for a trip to be shifted in time, thus introducing lateness to its starting time. Each minute of lateness should be penalized.
- **Trip cancellation:** We can allow trips to be cancelled, but its cost must always be higher than the actual cost of the trip and its possible deadhead trips, and it must also be higher than the cost introduced to the deviation from the original schedule.

Figure 1 models a typical situation that can arise in the daily practice of a transportation company. A schedule is disrupted, and 2 trips (coloured blue)

have to be rescheduled using 3 remaining vehicle duties (part *a)*). We give 2 different solutions: in part *b)*, a task is moved from the first duty to the third one before the insertion of the disrupted trips, while part *c)* gives a solution with trivial insertion into the duties. Note, that different solutions are also possible, for example one where we introduce lateness to one of the trips.

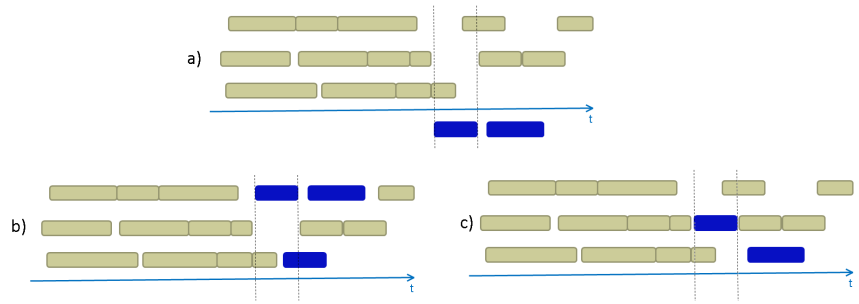

Fig. 1 A typical example for the BRP: there are two disrupted trips in example a) that have to be inserted into the given duties, b) and c) represent possible solutions

2.2 Related work

Literature usually addresses recoveries from disruptions under the field of disruption management. Depending on its effect, there are two main types of disruptions:

- A short term disruption only affects the schedule of the given day, and can be addressed quickly.
- A long term disruption has a more lasting effect, and can affect several days of the companies long term plan.

In this paper, we only deal with short term disruptions, where a few trips of the original daily schedule become infeasible, and must be rescheduled. This is the typical case when a vehicle that is late with regards to its schedule, and would only be able to start some of its service trips with significant lateness, or a vehicle that can not carry out some of its trips due to technical difficulties.

The first research into disruption management was carried out in the airline industry. Clausen et al. give a thorough overview of this field in [6,7]. The underlying network is somewhat similar to the problem of the BRP. However, the methods used for airline disruptions are computationally intensive, and have a long running time on the significantly larger bus transportation problems. This size difference comes from the smaller instances sizes of the airline industry, and the limited deadheading possibilites of the aircraft.

Disruption management in railway transportation is covered in [10]. These problems have a different structure, which mainly comes from the fixed rail-

way network, and the capacity limit of the tracks. Because of this, railway disruptions are handled in a different manner than in the airline industry.

To our knowledge, bus rescheduling as we defined in the above section was only considered by Li et al. In [13], they propose a quasi-assignment model and an auction algorithm for the problem, while in [14] they introduce a network flow model for the single depot BRP, which they solve using a lagrangean method.

2.3 Mathematical model

In this section we give a multi-commodity network flow model for the BRP described in subsection 2.1, which is similar to the network flow model reported by Li et al. in [14]. While Li et. al presented a model for the single depot BRP that allows trip cancellations, we propose a model for the multiple-depot BRP with trip cancellations and lateness.

Our input is the schedule of the company for a given day, which is disrupted at time point s. Let D be the set of depots, V be the set of vehicles currently in service, and $P = D \cup V$.

Let T' be the set of non-disrupted service trips of the given day that depart after time s, and thus still need to be serviced, and let set DT contain the disrupted trips. Let the set $T = T' \cup DT$ represent all the trips of our problem. Every trip $t \in T$ has a departure time $dt(t)$, arrival time $at(t)$, starting location $sl(t)$ and ending location $el(t)$. The set of depots and vehicles that can execute a trip t is denoted by $g(t)$. Let $T_d \subseteq T$ be the set of trips that can be executed from depot d, and $T_v \subseteq T$ the set of trips that can be carried out by vehicle v.

For every depot $d \in D$, we introduce notations $sl(d)$ and $el(d)$. A depot d is represented by $sl(d)$ when we consider it as the starting location of a vehicle, while we use $el(d)$ when it gives the ending location of the vehicle. Similarly for every vehicle $v \in V$ currently in service we define a starting location $sl(v)$ and ending location $el(v)$. For a vehicle v, $sl(v)$ corresponds to the current geographical location of v at the time of the disruption, and $el(v)$ is the geographical location of its depot. The set of nodes of our network will be the following:

$$N = \{dt(t) \cup at(t) \cup sl(d) \cup el(d) \cup sl(v) \cup el(v) | t \in T, d \in D, v \in V\}.$$

Using the nodes above, we define the different edges of the network. Let

$$J^d = \{(dt(t), at(t)) | t \in T_d\}$$

be the set of trips that can be served by depot d, and let

$$J^v = \{(dt(t), at(t)) | t \in T_v\}$$

be the set of trips that can be executed by vehicle v. Let

$$K^d = \{(at(t), dt(t')) | t, t' \in T_d \textit{ are compatible}\}$$

be the possible deadhead trips of a depot d and

$$K^v = \{(at(t), dt(t'))|t, t' \in T_v \textit{ are compatible}\}$$

be the possible deadhead trips of a vehicle v.
Let

$$L^d = \{(sl(d), dt(t)), (at(t), el(d))|t \in T_d\}$$

be all the pull-in and pull-out edges of depot d, and let

$$L^v = \{(sl(d), dt(t)), (at(t), el(d))|t \in T_v\}$$

be the pull-in and pull-out edges of vehicle v. The above sets together with circulation edges for every depot and vehicle give us the set of edges of our network:

$$E = \{J^d \cup J^v \cup K^d \cup K^v \cup L^d \cup L^v \cup \{(el(d), sl(d))\} \cup \{(el(v), sl(v))\} \textit{ for every } d \in D,\ v \in V\ \}.$$

With the nodes and edges introduced above, a solution of the vehicle rescheduling problem can be determined by using the network (N, E). We define an integer vector x for every edge of the network. Every $p \in P$ defines a component belonging to edge e, and is denoted by x_e^p. We also introduce a variable w_t for every $t \in T$, which allows the cancellation of t.

The difference between the original schedule and the resulting schedule should also be modeled. For every vehicle v, and trip-edge $e \in J^v$, we introduce a constant q_e. The value of q_e is 0, if the trip corresponding to edge e is carried out by the same vehicle v as in the original schedule, and 1 otherwise. The cost of a trip will depend on this constant, because αq_e will be added to the each such edge, where α is the penalty for deviation from the original schedule.

To allow lateness for trips, we introduce variable z_t, which gives a new departure time for every trip t. This value includes the added lateness, if any. For the trips to remain compatible, a constraint has to be added that examines trip compatibilities with respect to z_t:

$$(z_t + length(t) + deadhead_{t,t'} - z'_t) \sum_{p \in P} x^p_{a(t),d(t')} \leq 0, \forall (t, t') \in E, \qquad (1)$$

where $length(t)$ gives the running time of service trip t, and $deadhead_{t,t'}$ represents the running time of the deadhead trip between $al(t)$ and $dt(t')$. Constraint (1) is not a linear equation, but such a constraint can be rewritten with the introduction of a large constant M, as seen in [9].

The IP model of the problem can be formalized in the following way:

$$\sum_{p \in g(t)} x^p_{dt(t),at(t)} + w_t = 1, \forall t \in T \qquad (2)$$

$$\sum_{e:(sl(d),dt(t)) \in K^d} x^d_e \leq k(d), \forall d \in D \qquad (3)$$

$$\sum_{e:(sl(v),dt(t))\in K^v} x_e^v = 1, \forall v \in V \tag{4}$$

$$\sum_{e\in n^+} x_e^p - \sum_{e\in n^-} x_e^p = 0, \forall p \in P, \forall n \in N \tag{5}$$

$$z_t + length(t) + deadhead_{t,t'} - z'_t \leq \sum_{p\in P} 1 - x^p_{a(t),d(t')} M, \forall (t,t') \in E \tag{6}$$

$$z_t \geq start(t), \forall t \in T \tag{7}$$

$$z_t \leq start(t) + L, \forall t \in T \tag{8}$$

$$x_e^p, w_t \in \{0,1\}, \forall e \in E, p \in P, t \in T \tag{9}$$

Due to constraint (2), every trip is either executed exactly once, or cancelled. Constraint (3) gives maximum capacities for the depots of the problem, while vehicles in service are always given duties according to constraint (4). Constraint (5) ensures flow conservation. Constraint (6) is the linear reformulation of constraint (1). Constraints (7) and (8) limit the values of the trip starting times. A trip $t \in T$ will always depart in the $[start(t), start(t) + L]$ time window, where $start(t)$ is the departure time of t and L is the maximum allowed lateness. To solve the problem to optimality, we need to minimize

$$\sum_{e\in E}\sum_{p\in P} c_e^p x_e^p + \sum_{t\in T} \beta(z_t - start(t)) + \sum_{t\in T} \gamma w_t, \tag{10}$$

where β and γ are penalty parameters for lateness and cancelling trips respectively. c_e^p gives the corresponding operational cost of edge x_e^p, along with the possible added penalty of deviation from the original schedule given by α.

2.4 Importance of the model

The size of the above model grows quickly with an increase in the number of compatible trips. As each commodity layer of the model contains the possible connections of the given depot or vehicle, its size will increase significantly with every added commodity. We have to represent every bus currently in service as a new commodity in the model, which results in a significantly high number of vehicle layers. For example, the middle-sized city of Szeged, Hungary has about 2700 trips on a regular workday, which is executed by 108 vehicles (belonging to 4 vehicle types). This would mean a network with over 110 commodities, which is taxing to solve both in memory and running time.

In a real life application, solutions for disruptions are needed in real time, because the resolution of a disruption does not end with the solution of the problem. After solving the BRP, operators still need to communicate with bus drivers about the recent changes, which also takes time.

However, the mathematical model is important because it allows us to compare our results to the optimal solution for the problem. This way it can provide us with a quality control for any heuristic method that we use for the fast solution of the BRP.

3 Applying fast solution heuristics

As we mentioned in subsection 2.4, the size of the model will grow quickly with the increase of the problem size. The number of commodities in the network is given by the number of depots and vehicles, and this will be dominated by vehicle commodities (except for small test-instances).

Exact solution of such a big problem would take a long running time, which is not acceptable for a real life application of the method. Recovering from a disruption needs to be done as quickly as possible, thus fast and efficient solution heuristics are needed.

It might well be that the costwise optimal solution is not be the best one regarding operations planning. However, a solution given by an algorithm can help the operator decide faster about how to resolve the disruption. Integrating fast algorithms into a decision support system that gives multiple suggestions for the solution can speed up the decision of the operators.

In the following sections, we present fast solution methods that provide multiple good quality solutions in a short running time. These methods can easily be integrated into a decision support system for disruption management in public transportation, which recommends possible solutions for the operators depending on their parameter settings. We will be dealing with such a system in one of our future papers.

Li et al. also presented a prototype decision support system in [12] for the single depot BRP. In this system, they were only dealing with the optimal solution for the problem that the operators could change on an interactive interface.

3.1 A recursive search algorithm

One of the algorithms we propose is a recursive search heuristic for the problem. Recursive search seems an ideal method because of the expectations described above. This algorithm is able to find multiple solutions with a short running time. Our method can be seen in Algorithm 1.

The input of the algorithm is the following:

- **vDuties:** The disrupted daily schedule. It includes all available vehicles and the duties assigned to those vehicles that were not disrupted.
- **dTrips:** The set of disrupted trips, which have no assigned vehicle duties.
- **depth:** A non-negative integer parameter that limits the depth of the search tree. Every time trips are removed from a schedule, the value of this parameter is decreased.

Algorithm 1 Recursive search for bus rescheduling.

```
 1: procedure RecSearch(vDuties, dTrips, depth)
 2:     if depth = 0 then
 3:         return 0
 4:     end if
 5:     for i = 1 to Size(dTrips) do
 6:         Trip = dTrips[i]
 7:         for j = 1 to Size(vDuties) do
 8:             nDuties = vDuties
 9:             nTrips = dTrips
10:             Duty = vDuties[j]
11:             if Trip and duty are not compatible then
12:                 continue
13:             end if
14:             if Trip overlaps with trips in Duty then
15:                 if Possible solution with lateness then
16:                     Duty' = Insert Trip into Duty with introducing lateness
17:                     nDuties' = nDuties with Duty' inserted into nDuties[j]
18:                     nTrips' = nTrips without Trip
19:                     if Size(nTrips') = 0 then
20:                         Add(Solutions, nDuties')
21:                     else
22:                         RecSearch(nDuties', nTrips', depth)
23:                     end if
24:                 end if
25:                 tRemoved = overlapping trips from duty
26:             end if
27:             Insert Trip into Duty
28:             Remove Trip from nTrips
29:             nDuties[j] = Duty
30:             if Size(nTrips) = 0 then
31:                 Add(Solutions, nDuties)
32:             else if Size(tRemoved) > 0 then
33:                 Add(nTrips, tRemoved)
34:                 RecSearch(nDuties, nTrips, depth-1)
35:             else
36:                 RecSearch(nDuties, nTrips, depth)
37:             end if
38:         end for
39:     end for
40:     return Best solution in Solutions
41: end procedure
```

The input for the heuristic is the set of feasible vehicle duties, and the set of disrupted trips. Every function call chooses the disrupted trip dt with the earliest departure time, and tries to fit it into every compatible vehicle duty vd. There are three possibilities for every $dt - vd$ pair:

- One or more trips have to be removed from vd. The removed trips are flagged as temporary disrupted trips.
- Trip dt can be inserted into vd, but lateness has to be introduced for some of the trips of vd.
- Trip dt can be inserted into vd without additional modifications.

If the set of disrupted trips is empty after a modification, and there are no temporary disrupted trips, the heuristic has found a feasible solution, which is saved. Otherwise, the recursive function is called with new parameters:

- **vDuties':** The original $vDuties$ is updated with the modified duty.
- **dTrips':** The temporary disrupted trips are inserted into $dTrips$, while dt is removed.
- **depth':** If the size of $dTrips'$ is smaller than the size of $dTrips$, $depth' = depth$. Otherwise, $depth' = depth - 1$.

The algorithm will explore the solution space determined by the trips and the schedules of the problem, examining every possible solution found during its runtime. The depth of this search tree is limited by the parameter $depth$. Further limitations can be introduced into the method to exclude visiting similar configurations multiple times.

These limitations (especially the parameter for the depth) help to keep the running time of the algorithm from exploding. Introduction of this parameter was also based on a practical observation: each level of the recursive search tree corresponds to a vehicle whose original duty is modified. Companies want to keep the number of modified vehicle duties low. Because of the way $depth$ is decreased, its initial value also defines the maximum number of vehicle duties from which the algorithm can remove trips. As we mentioned in Subsection 2.1, altering the original schedule of a driver should have a high cost, so it is unlikely that the optimal schedule will be cut from the search tree by this parameter.

The algorithm terminates after it has traversed the above defined search tree. If it has found at least 1 solution, then the one with the lowest cost is returned as a result.

3.2 A local search algorithm

The other proposed algorithm is a local search method for finding a feasible solution for the BRP. A brief outline of the algorithm can be seen in Algorithm 2.

The input of the algorithm is the following:

- **vDuties:** The disrupted daily schedule. It includes all available vehicles and the duties assigned to those vehicles that were not disrupted.
- **dTrips:** The set of disrupted trips. These are currently not executed by vehicles, and have to be assigned to vehicle duties.
- **tRange:** Gives a time window in which the events are considered. The time window begins at the start time of the disruption, and ends after the ending time of the last disrupted trip.

The initial candidate solution of the algorithm is constructed from the original vehicle schedule. A new vehicle duty is added to the schedule, that contains all the disrupted trips, increasingly ordered by their departure time.

Algorithm 2 Local search for bus rescheduling.

```
 1: procedure LocSearch(vDuties, dTrips, tRange)
 2:     Build infeasible duty dt from dTrips
 3:     Label dt temporary
 4:     Add(dt to vDuties)
 5:     tabuList = list of forbidden transformations
 6:     while notEmpty(dt) & !(terminatingConditions) do
 7:         tmpSchedules = empty container for vehicle schedules
 8:         for i = 1 to Size(vDuties) do
 9:             for j = i+1 to Size(vDuties) do
10:                 for each neighborhood transformation t do
11:                     if t(vDuties[i], vDuties[j]) is not forbidden then
12:                         newSchedule = apply t of vDuties
13:                         Add(newSchedule, tmpSchedules)
14:                     end if
15:                 end for
16:             end for
17:         end for
18:         bSchedule = best schedule from tmpSchedules
19:         tS = transformation that can reverse bSchedule
20:         vDuties = bSchedule
21:         Add(tabuList, tS)
22:     end while
23:     return vDuties
24: end procedure
```

If there is more than one disrupted trip, this new duty is more than likely infeasible, which will make our initial solution also infeasible. This new duty is labelled as a *temporary duty*.

In each iteration the algorithm will examine all (i, j) pairs of the duties. It checks the trips of the duties that are in the given *tRange* time window, and examines the following two neighborhood transformations:

- **1-move:** Moves a trip from duty i to duty j. This transformation is not carried out, if j is a temporary schedule.
- **1-change:** Exchanges a trip from duty i with the corresponding trip(s) from duty j. This transformation is not carried out, if any of the duties is temporary.

All feasible neighbors given by the above transformations are assigned a cost. This cost is computed from the operational cost of the duties and the penalties introduced in subsection 2.1. If the transformation moves a trip from a temporary schedule to another schedule, a high negative penalty is added to decrease the cost, which will make it more likely to be chosen in an early iteration of the local search.

The local search algorithm chooses the neighbor solution with the lowest cost as its new candidate. If any trip t was removed from a duty D in the process, the (t, D) pair is saved on a tabu list. For every (t, D) pair on the tabu list, trip t cannot be moved to duty D with any of the transformations.

The algorithm terminates when at least one of the terminating conditions is met. We use the following terminating conditions:

– Limit for the running time.
– If the difference in quality of the consecutive candidates is always below a given gap for a fixed amount of iterations.

If the algorithm has found at least 1 feasible solution, then the one with the lowest cost is returned by default. The number of solutions returned can be set higher using a parameter, if the user wants to ask for more suggestions.

4 Test results

In this section we provide the test results of the heuristics presented in section 3. To analyze the quality of our results, we will compare them to the optimal solution of the BRP model given in subsection 2.3.

As it was mentioned earlier, the size of the mathematical model can grow quickly with the increase of the instance size. The model we presented in 2.3 represents all possible connection between pairs of trips with a unique deadhead edge for every commodity. Because of the high number of deadhead connections, Kliewer et al. introduced the time-space network in [11] to solve the MDVSP. This model aggregates deadhead edges, resulting in a much smaller problem that is easier to solve. The number of deadhead edges of the BRP model in subsection 2.3 can be decreased in the same way.

In our test instances, we used a time-space network equivalent of our BRP model. We generated our daily schedules by solving a time-space network model on random instances given by a method described in [8]. The disruption time was set to 0, which means that we considered the whole daily schedule in every case. We modelled the scenario when a new trip is introduced to the daily schedule at the beginning of the day. This new trip is our disrupted trip, which is generated as a single short trip by the same random method referenced above.

We tested the methods on different instances with 12, 100, 500 and 800 trips in their original schedule. Several different test cases were generated for each instance. Table 1 shows the average results of 10 randomly generated cases for every instance.

Table 1 Test results of the heuristic methods

Instance	Trips	Depots	Opt.(s)	Rec.(s)	Gap (Rec.)	Loc.(s)	Gap(Loc.)
R1	13	2	1.03	0.02	0 %	0.001	0%
R2	101	4	1.12	0.05	0 %	0.004	0%
R3	501	4	132	0.08	0 %	0.01	0%
R4	801	4	801	0.08	0.04 %	0.05	0%

The number of trips and depots of the BRP can be seen in columns 2 and 3. The running time of the solution of the exact model is given in column 4. The running time of the heuristics and their gap from the optimal solution are

represented in columns 5-6 and 7-8 respectively. All tests were carried out on a PC with and Intel Core i5 2.80GHz CPU and 4 GB RAM. The IP model was solved using the COIN-OR Symphony solver.

We could not solve the IP for instances with higher number of trips, because the model itself was too big to be contained in memory. The heuristics provided results for significantly larger input as well. We tested the two heuristic methods on both random and real-life instances, and both of them returned multiple solution suggestions. The biggest real-life instance we used contained 2674 trips, while the biggest random instance had 3500 trips.

The solution time of the heuristics remained fast, and took at most around 15 seconds. We also hand-tailored some of the bigger test instances, in which we knew the best solution from the point of view of operational planning (e.g. trivial insertion of a trip to a duty, or a certain trip has to be moved/delayed to insert the disrupted trip). The algorithms found the desired solutions in every case.

The results of the heuristic algorithms are promising, as they give multiple solutions even for larger instances in a short time, while the number of modified schedules and moved trips stay low. Their good speed and solution quality, and the multiple given solutions make them suitable for a decision support system described in the previous sections.

5 Conclusions and future work

In this paper, we considered the multiple depot BRP, which deals with rescheduling the disrupted daily schedule of a transportation company. This problem is important, as disruptions happen in the daily schedules of every company, and the order of transportation should be restored as soon as possible. We described the restrictions of the problem, and defined a mathematical model based on the arising needs.

Such a problem requires a real-time solution, because the results must be processed by operators and communicated to the bus drivers, which also takes time. As the size of the model is too big to be solved in such short time, we proposed two fast heuristic algorithms to produce results in a couple of seconds. Our tests on randomly generated instances showed that the heuristics give a solution that is close to the optimum. While we could not measure the quality of the algorithms on bigger real-life instances, the running time still remained fast, and both methods gave the expected results for artifical disruption scenarios for these inputs.

Because of their ability to produce multiple good quality solutions in a short time, these algorithms seem suitable for a decision support system that helps the operators of a transportation company in the rescheduling process by giving them possible solution suggestions for the problem. However, there are still questions for future research.

The size of the mathematical model is too big to be contained in memory for bigger instances. To get exact solutions for real-life problems, we can use

decomposition methods (e.g. column generation), or heuristic size reduction of the model. Both approaches should be investigated in future works.

Acknowledgements This work was partially supported by the European Union and co-funded by the European Social Fund through project HPC (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0010).

References

1. Békési, J., Brodnik, A., Krész, M., Pas, D.: An integrated framework for bus logistics management: Case studies. Logistik Management **5**(1), 389–411 (2009)
2. Bertossi, A., Carraresi, P., Gallo, G.: On some matching problems arising in vehicle scheduling models. Networks **17**(1), 271–281 (1987)
3. Bodin, L., Golden, B.: Classification in vehicle routing and scheduling. Networks **11**(1), 97–108 (1981)
4. Bodin, L., Golden, B., Assad, A., Ball, M.: Routing and scheduling of vehicles and crews: The state of the art. Computers and Operations Research **10**(1), 63–212 (1983)
5. Bunte, S., Kliewer, N.: An overview on vehicle scheduling models. Journal of Public Transport **1**(4), 299–317 (2009)
6. Clausen, J., Larsen, A., J. Larsen, J., Rezanova, N.J.: Disruption management in the airline industry-concepts, models and methods. Computers & Operations Research **37**(5), 809–821 (2010)
7. Clausen, J., Larsen, A., Larsen, J.: Disruption management in the airline industry - concepts, models and methods. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark, DTU (2005)
8. Dávid, B., Krész, M.: Application oriented variable fixing methods for the multiple depot vehicle scheduling problem. Acta Cybernetica **21**(1), 53–73 (2013)
9. Desrochers, M., Lenstra, J., Savelsbergh, M., Soumis, F.: Vehicle routing with time windows: Optimization and approximation. Vehicle routing: Methods and studies **16**, 65–84 (1988)
10. Jespersen-Groth, J., Potthoff, D., Clausen, J., Huisman, D., Kroon, L.G., Maróti, G., , Nielsen, M.N.: Disruption management in passenger railway transportation. Tech. rep., Erasmus University Rotterdam (2007)
11. Kliewer, N., Mellouli, T., Suhl, L.: A time-space network based exact optimization model for multi-depot bus scheduling. European Journal of Operational Research **175**(3), 1616–1627 (2006)
12. Li, J.Q., Borenstein, D., Mirchandani, P.B.: A decision support system for the single-depot vehicle rescheduling problem. Computers and Operations Research **34**(4), 1008–1032 (2007)
13. Li, J.Q., Mirchandani, P.B., Borenstein, D.: The vehicle rescheduling problem: Model and algorithms. Networks **50**(3), 211–229 (2007)
14. Li, J.Q., Mirchandani, P.B., Borenstein, D.: A lagrangan heuristic for the real-time vehicle rescheduling problem. Transportation Research Part E: Logistics and Transportation Review **45**(3), 419–433 (2009)