Acta Cybernetica **23** (2018) 783–800.

Integrated Vehicle Scheduling and Vehicle Assignment

József Békési[a], Balázs Dávid[a], and Miklós Krész[a]

Abstract

The vehicle scheduling problem has been extensively studied in the past decades. Yet, most models and methods given in the literature consider only a theoretical scenario where vehicles just have to service the timetabled trips of the input. However, schedules created this way cannot be used in real life, as they should also consider constraints such as refueling, parking, and maintenance, which are all connected to the vehicle servicing the trips. In this paper, we give a set partitioning model for the multi-depot integrated vehicle scheduling and vehicle assignment problem. This model can also be used as a general framework, which can integrate multiple activities based on the rules or regulation of the different possible input scenarios. We give a column generation-based solution method, and demonstrate its efficiency on randomly generated test instances, which treat the refueling of vehicles with two different fuel types as the vehicle-specific activity.

Keywords: vehicle scheduling, vehicle assignment, integrated model, alternative fuel, refueling, maintenance

1 Introduction

The vehicle scheduling problem (VSP) is one of the most basic optimization problems arising in public transportation. It has been fairly well researched over the past few decades, but state-of-the-art models and methods mostly treat the problem as a mathematical one. The usual models and solution methods given for the VSP are interesting from a theoretical point of view, but many of the solutions cannot be applied directly in real life. They mainly focus on assigning timetabled trips to the vehicles of the company. However, there are other constraints that need to be considered while creating a vehicle schedule.

The vehicle schedules of a transportation company are not just virtual sets of tasks that have to be executed in the given sequence, but they also have a real vehicle (or at least a vehicle brand/type) assigned to the schedule as well. Knowing

[a]University of Szeged, Department of Applied Informatics, E-mail: {bekesi,davidb,kresz}@jgypk.szte.hu

DOI: 10.14232/actacyb.23.3.2018.4

the vehicle responsible for the execution of the trips also means that there are special needs that have to be taken into account while the vehicle is in service. For instance, it can run out of fuel, and has to be refueled, or spends too much time in service, and has to be sent for short maintenance during the day. While similar constraints are important from the perspective of a real-life application, they are not widely studied in the literature. We will refer to these requirements as vehicle-specific activities. More application-oriented approaches for the VSP started appearing with the rise of electric and alternative-fuel vehicles. These are both cheap and environmentally friendly to operate. However, their major drawback is that they can only run a limited distance, so refueling events have to be considered when creating their schedules [19].

While refueling is an important constraint that has to be included in a VSP model, other vehicle-specific activities should also be considered, such as parking and maintenance [15, 7]. Constraints like these receive significantly less attention than refueling. In this paper, we present a set partitioning model for the integrated vehicle scheduling and assignment problem with vehicle-specific activities. If such activities are included in a vehicle schedule, they also determine different tasks that the vehicle has to execute besides its timetable trips. This results in a vehicle assignment combined with scheduling. Our goal is to give a general framework where most vehicle-specific activities can be integrated, and provide a flexible model where many of these application-oriented constraints can be readily included. While there may exist other models and methods that deal with these problems separately, to our knowledge such activities have not been considered together in the same problem before.

We give a column generation-based solution method for this model, and demonstrate its efficiency on randomly generated test instances. To showcase the model, we use refueling as the vehicle-specific activity for these instances, and also study the concept of multiple fuel types, which is also rarely considered in the literature.

2 Vehicle scheduling and vehicle-specific activities

For the introduction of the VSP, we will follow our terminology from [10]. The input of the VSP is a set V of vehicles and set T of *timetabled trips*. Every trip has a departure and arrival time, a starting and ending location, and the distance that they cover. Trips also have a set of vehicles that are able to service them. 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'. Traveling between two locations without servicing a timetabled trip is called a *deadhead trip*. Deadheads also have a starting and ending location, distance and duration in time.

The VSP gives an assignment between the timetabled trips and the vehicles in such a way that every trip is executed exactly once, and trips assigned to the same vehicle are pairwise compatible. As mentioned before, trips can determine the set of vehicles that can execute them, and this is achieved through the concept

of *depots*. Vehicles may belong to depots. These are determined by the features of the trips; namely, certain trips might be served only by vehicles that satisfy given constraints (e.g. is the vehicle wheelchair accessible, does the vehicle have air conditioning, or a given passenger capacity?). Such constraints are important, and are centered around the services you want to provide to passengers taking the given trip (e.g. people with a wheelchair should have no problem getting on any bus along the route of this trip), or a regulation connected to the trip (e.g. every bus covering a trip that is longer than a given distance must have air conditioning),

These depots are usually determined by a combination of two features; namely the type of vehicle (e.g. with/without air conditioning, solo, or articulated), and its starting location at the beginning of the day. When we talk about vehicles belonging to the same depot in this paper, we mean vehicles sharing the same vehicle type and having the same starting/ending geographical location at the beginning/end of the day. If the problem has at least two depots, every trip is also assigned a depot-compatibility vector that corresponds to the depots that can execute it.

Besides depots, *vehicle characteristics* will also be addressed. These also represent different attributes of the vehicles, but only those that do not have an influence on servicing trips. For example, the fuel type of the vehicle may be such a characteristic; vehicles belonging to the same depot can run on different fuels, but can still service the same trips.

The total cost of the problem usually consists of a one-time daily cost for each vehicle in service, and an operational cost proportional to the distance traveled by these vehicles. Both of these costs may be different depending on the depot of the vehicle. The result of the VSP is a daily vehicle schedule, which consists of vehicle blocks. A vehicle block is basically a sequence of tasks that are executed by the same vehicle.

If the problem has only one depot, it is called the single depot vehicle scheduling problem (SDVSP), and it can be solved in polynomial time. A formulation for the SDVSP is presented in [4]. A problem with at least two depots is referred to as a multiple depot vehicle scheduling problem (MDVSP). The MDVSP was introduced by Bodin et al. in [5], and its NP-hardness was shown by Bertossi et al. [3]. There are several different models and solution methods available for the problem, and Bunte et al. provide a good overview of these in [8].

Solution methods based on the basic concepts of the SDVSP and MDVSP cannot be applied directly in practice, as they only deal with covering all timetabled trips. In real life, however, vehicles have to execute several types of activities during the day. These come from different vehicle-specific needs (parking, refueling, maintenance, etc.), and they usually have to be executed after a vehicle has covered a certain distance (refueling, maintenance), or spent a certain amount of time without performing any activities (parking). In all of these cases, the total length of certain consecutive activities is limited, and vehicle-specific events have to be scheduled after such work pieces.

In general, extensions of the VSP that have either a time or distance constraint on the length of the vehicle blocks are called the Vehicle Scheduling Problem with Time/Route Constraints [13, 8]. These alone cannot satisfy the constraints for

vehicle-specific activities, as they limit the total length of the flow representing a block, while activities only need a limit for certain parts of this flow. However, most problems that consider the above vehicle-specific activities are special cases of this group.

In this paper, we introduce a set partitioning-based model for the integrated vehicle scheduling problem with vehicle-specific activities. The purpose of this model is to provide a general framework capable of producing vehicle schedules that can be used in practice. Most papers dealing with the VSP do not consider vehicle-specific activities, although they are really important real-life constraints for vehicles. One such activity that has gained increasing popularity in recent years is the scheduling of alternative fuel (natural gas, hybrid) or electric vehicles.

Vehicle scheduling for alternative fuel vehicles (AF-VSP) is hard because of the limited distance they can cover. Vehicles have to be refueled during their daily blocks, and there are usually very few special refueling stations, with a limited number of refueling pumps. Because of this, location problems for refueling stations are also important [17]. Li presented a flow network-based model for both alternative fuel and electric vehicles in [18]. Here, a single depot VSP is considered with a single fuel type, and a single refueling station at the depot. Several column generation-based solution techniques are presented on instances with up to 947 trips. Adler studied the problem with multiple depots in [1, 2]. He used a set partitioning model for alternative fuel vehicle scheduling, and also used column generation to solve instances with up to 50 trips. Larger instances are solved by applying heuristic methods.

Electric vehicle scheduling (E-VSP) has also been getting a lot of attention recently. Both the above-mentioned paper by Li [18] and the dissertation of Adler [1] present models and solution methods for the E-VSP. Their solution approaches are similar to those that were used for the AF-VSP. They mainly deal with battery swapping, and their solution approaches are also based on column generation. A time-space network-based model that allows the charging of the vehicles is given in Reuer et al. [23]. In [25], multiple models are presented for the E-VSP that consider battery charge. The solution is once again obtained using a column generation approach.

Another important vehicle-specific activity is the assignment of small maintenance tasks to vehicles during their daily blocks. According to Haghani and Shafahi [15], they can be of three types: daily, preventive and emergency maintenance. Daily inspections can be built into a vehicle schedule at the beginning or the end of the blocks, while emergency maintenance is only issued as part of a disruption management process when something unexpected happens to a vehicle. However, preventive maintenance has to be executed by vehicles after a set distance or time interval. Such maintenance activities were examined for rolling stock rotations by Borndörfer et al. in [7], while Haghani et al. also gave a model for inserting preventive maintenance into existing bus schedules in [15].

It can be seen that none of the above papers deal with multiple vehicle-specific activities. Moreover, the vehicle-specific activities that they study usually consider a single vehicle characteristic (e.g. vehicles will all have the same fuel type, or

require the same type of maintenance). In this paper, we introduce the multiple depot integrated vehicle scheduling and assignment problem with vehicle-specific tasks. This model acts as a general framework, where multiple activities with time or distance constraints can be included depending on the requirements of the specific problem. The model can provide feasible solutions with these constraints, while also taking capacity constraints into account. While all the above-mentioned papers solve the vehicle scheduling problems with a single specific vehicle activity, our goal was to provide a general model that can handle multiple activities simultaneously.

We will define the integrated vehicle scheduling and assignment problem with vehicle-specific activities (VSAP-VS) in the following way. Let the input of the problem be a set T of timetabled trips, set D of depots and set V of vehicles. These concepts here are exactly the same as those introduced for the VSP at the beginning of this section; namely, trips have a departure and arrival time, a starting and ending location, and the vehicles can also be assigned to depots in the same way. Trips also determine the set of depots that are able to serve them by considering the features of the vehicles in the given depot.

In addition to the input given for the VSP, there are n different vehicle-specific activities represented by set R. These activities are connected to vehicle characteristics rather than depots (e.g. refueling vehicles with natural gas and preventive maintenance of hybrid vehicles), and certain activities can only be carried out by given vehicles. Let set $R_j \subseteq R$ denote the possible tasks belonging to activity j. Similar to timetabled trips, task $r \in R_j$ has a starting and ending time, and departure and arrival locations (although these two are usually the same, as activities like parking or refueling are stationary). In order to model the capacities of the particular activities, they can only carried out in fixed, discrete time intervals instead of continuous availability. Each activity has its own set of rules and regulations, but these are usually connected to a set timespan or distance limit. For instance, a vehicle cannot travel more than a given distance without carrying out a refueling task, or it has to begin a parking task if it remains idle for more than a given amount of time. Compatibility between trips and activities can also be defined in a similar way to the VSP. Consider a pair of tasks $a, a' \in T \cup R$, which are compatible if:

- the same vehicle $v \in V$ is able to service both of them (the depot of v is compatible with any trips in $\{a, a'\}$),
- they both satisfy the vehicle characteristics of v (any vehicle-specific task in $\{a, a'\}$ can be carried out by v; e.g. we cannot assign battery recharging to a vehicle running on gas), and
- vehicle v can service a' after finishing a with respect to the running time and distance between the arrival location of a and the departure location of a' (also considering the possible deadhead trip between a and a', where necessary).

The aim is to provide a feasible vehicle schedule that includes both timetabled trips and vehicle-specific activities, and tasks in any vehicle block are pairwise compatible. The cost of such a vehicle schedule is a linear combination of three different terms; namely, a one-time daily cost for each vehicle covering a block, a distance proportional cost for covering timetabled trips and deadheads, and costs of the vehicle-specific activities included in the blocks.

In the remainder of this paper, we will first introduce a mathematical model and solution method for the VSAP-VS, then demonstrate its efficiency by solving randomly generated test instances of different sizes and types.

3 Modeling and solution of the problem

In this section, we introduce our model for creating vehicles schedules that also take vehicle-specific activities into account. Then we present our solution method for it.

3.1 A set partitioning mathematical model

For each depot $d \in D$, let B_d be the set of feasible b blocks that start from $dt(d)$ and also return there. A block is a sequence of compatible trips and activities that can be executed by the same vehicle, and it satisfies all activity rules connected to this vehicle. Let

$$B = \bigcup_{d \in D} B_d$$

be the set of all such blocks. For each $b \in B_d$, let y_b^d be the following binary variable

$$y_b^d = \begin{cases} 1, & \text{if } b \in B_d \text{ block is part of the solution} \\ 0, & \text{otherwise} \end{cases}$$

Furthermore, let $a_{e,b}^d$ be the following

$$a_{e,b}^d = \begin{cases} 1, & \text{if } b \in B_d \text{ block contains activity edge } e \\ 0, & \text{otherwise} \end{cases}$$

Let T denote the set of trips for the problem, and R give the set of tasks belonging to all vehicle-specific activities. Tasks that are connected to a single activity of a given vehicle characteristic are given by $R_i \subseteq R (1 \leq i \leq n)$, where n is the number of all such activities. The capacity of a depot $d \in D$ is denoted by k_d, and the maximum number of vehicles that can simultaneously perform a vehicle-specific task $r \in R$ is given by k_r. Let c_b be the cost associated with block $b \in B_d$. Then we can formulate the problem in the following way:

$$\text{minimize} \sum_{d \in D} \sum_{b \in B_d} c_b y_b^d, \tag{1}$$

s.t.

$$\sum_{d \in D} \sum_{b \in B_d, e=(dt(t),at(t)) \in E_d} a_{e,b}^d y_b^d = 1, \forall t \in T \tag{2}$$

$$\sum_{d \in D} \sum_{b \in B_d, e=(dt(r),at(r)) \in E_d} a_{e,b}^d y_b^d \leq k_r, \forall r \in R \tag{3}$$

$$\sum_{b \in B_d} y_b^d \leq k_d, \forall d \in D \tag{4}$$

$$y_b^d \in \{0,1\}, \forall d \in D, \forall b \in B_d \tag{5}$$

$$a_{e,b}^d \in \{0,1\}, \forall d \in D, \forall b \in B_d, \forall e \in E_d \tag{6}$$

Constraint (2) ensures that every trip is covered exactly once. Blocks simultaneously containing certain vehicle-specific activities are given by constraint (3), each task $r \in R$ having a maximum capacity k_r. Note that this constraint only ensures the vehicle limits on each task, as we suppose that every block is feasible, also meaning that they satisfy the distance and time constraints or any other rules connected to the activities. Managing this feasibility will be addressed later on. Finally, constraint (4) limits the capacities of each depot $d \in D$.

3.2 A column generation approach

Due to the huge number of possible blocks (resulting in a large amount of decision variables), the model cannot be solved directly by a MIP solver. Moreover, generating all blocks is also problematic, as the number of possible combinations is huge. Instead of giving all the possible blocks in the model, only the most important ones have to be generated to achieve a good quality solution.

Column generation [11, 20] is a classical method that is usually applied to such problems, relaxing the integer constraints of the variables. This relaxed problem is also known as the master problem. The usual steps taken during the solution process are the following:

1. Create an initial solution. The resulting schedule will provide the starting set of columns.
2. Solve the relaxed problem (master problem) on the actual set of columns, store the lower bound, and duals.
3. Solve a pricing problem in order to look for new columns that have a negative reduced cost.
4. Add the new columns to the master problem, and erase any old columns that are obsolete, and have a large cost.

5. Check the termination criteria. If none apply, go to step 2.

6. Create the final schedule based on the current columns of the problem.

Creating the final vehicle schedule in step 6 can be achieved by solving the resulting problem as an IP using a solver. A solver can also be used in step 2 for the solution of the master problem LP. The process can terminate in step 5 if a given iteration count is reached, or the solution has not improved significantly in the past few iterations. The most important part of the algorithm is the solution of the pricing problem in step 3.

3.3 Initial solution

Next, we present our heuristic for creating an initial solution for the column generation process. Its pseudo code can be seen in Algorithm 1.

The input of the algorithm is the set T of trips and set V of vehicles. The process iterates over the input trips in ascending order of their departure times, and assigns the current trip to an existing block with the cheapest cost. If there is no block where the trip can be inserted, then a new block is created for the trip. The vehicle chosen for this block is the one with the smallest cost that can execute the trip. After each trip assignment, the current block is checked to see whether its assigned vehicle has to undergo a task belonging to any of its vehicle-specific activities.

The function $checkActivity(A, b)$ checks block b to see if it could execute any of the remaining trips without violating the rules of activity A. If any of the rules are violated, b is sent to carry out the given activity, and is assigned the cheapest compatible task. If the activity was connected to a resource (e.g. distance or time), this is also replenished for the vehicle servicing the block.

As an example, we present a function for managing refueling activities in Algorithm 2. Vehicles are sent for refueling tasks if their remaining distance (denoted by *remDist*) does not allow them to head out for any trip, service it, and then head back to any location. For this, we count the distance of two deadheads as *maxDeadheadTime* (with the maximal possible distance), and the distance of the trip as *maxTripTime* (taking the maximal remaining trip distance into account). If refueling is needed, the vehicle is assigned to the next available possibility with the cheapest cost. After the refueling task is completed, the remaining distance that the vehicle can cover is again set to its maximum value.

When the initial solution is created, its blocks are used as the starting columns of the relaxed master problem.

3.4 Pricing problem

After solving the master problem, information about its duals is used to create new columns that can improve its current objective. Each such column corresponds to a legal vehicle block. These blocks are created with the use of a generation network.

Algorithm 1 Initial vehicle schedules with activities.

Funct buildSchedule(T, V)

Let the set of blocks be $B := \{\}$
Order T by ascending trip departure times
for $(t \in T)$ **do**
  Let $f := b \in B$ with the cheapest insertion cost for t
  if $f = \emptyset$ **then**
    $f := v \in V$ that can serve t with the smallest cost
    Assign t to f
    $B := B \cup \{f\}$
  else
    Assign t to f
  end if
  for all activities $R_i \subseteq R$ **do**
    checkAtivity(R_i, f)
  end for
end for
return B

Funct checkActivity(A, b)

$P :=$ all available tasks in A for b
$d :=$ resource (time/distance) needed for b to serve any trip t
$v :=$ is an activity rule violated servicing any trip t?
if $d \geq remainingResource(b)$ **or** $v =$ **true then**
  $p :=$ cheapest compatible task from P
  Assign p to b
  $remainingResource(b) := MAX$
else if v = true **then**
  $p :=$ cheapest compatible task from P
  Assign p to b
end if

Algorithm 2 Function for checking refueling activities.

Funct checkFuel(b)

$R :=$ all available refueling times for b
$d := 2 \cdot maxDeadheadTime + maxTripTime$
if $d \geq remDist(b)$ **then**
  $r :=$ cheapest compatible refueling possibility for b
  Assign r to b
  $remTime(b) := MAX$
end if

This network provides the basis of the pricing problem, and it is used to build vehicle blocks with a negative cost that also satisfy all the above-mentioned vehicle-

specific activities. This basically means that after a vehicle has consumed enough of a given resource, the appropriate events for its vehicle-specific needs also have to be scheduled. This will be done by solving a resource-constrained shortest path problem.

We use a time-space generation network similar to the one presented by [24], and a separate network is created and solved for every pair of depot and activity type. Each network is given by the possible tasks that can be assigned to the vehicles, namely timetabled trips, deadhead trips and other vehicle-specific events. The nodes of the network correspond to the arrival and departure time of these tasks on their given time-lines. The edges of the network can either correspond to their respective tasks, or be waiting edges on the time-lines.

Formally, let $G = (N, E)$ represent a general duty generation network. Such a network is created for each combination of depot d and activity a, resulting in networks $H_a^d = (N_a^d, E_a^d)$. The nodes in N_a^d are the following: *depot_source, depot_sink, trip_start, trip_end, activity_start, activity_end*. Edges in E_a^d connect these nodes; *trip_start* and *trip_end* nodes belonging to the same trip are connected by *trip edges*, *activity_start* and *activity_end* nodes belonging to the same activity are connected by *activity edges*. Any end node is connected to the start nodes of other tasks; *deadhead edges* connect those in different locations, while *waiting edges* run between nodes in the same location. The only exception to this is activities. Namely, the end node of an activity task is not connected to the start node of the same activity type, as there is no point in executing two similar vehicle-specific activities after each other. *Block_start edges* connect the *depot_source* node to every *trip_start* node, and every *trip_end* node is connected to the *depot_sink* node with *block_end edges*.

An example of such a network can be seen in Figure 1. Here, refueling is the vehicle-specific activity of the network, and it is performed in 20-minute tasks.

To ensure feasibility with respect to vehicle-specific events, so-called resources are also associated with each vehicle on the network. A single resource is allocated for each vehicle-specific activity, which calculates the time/distance (depending on the resource) covered by the vehicles with the help of a resource extension function. These will filter out blocks whose total consumed resources violate any of the vehicle-specific requirements. Tasks belonging to the activity replenish the appropriate resource capacity of the executing vehicle.

As described in [24], negative reduced cost vehicle blocks are generated on these networks using a dynamic programming approach presented in [12]. Blocks with the lowest negative reduced cost are added to the master problem from each generation network.

3.5 Creating the final schedule

After the column generation steps have terminated, the resulting master problem only gives us a solution for the LP-relaxed scheduling problem. To obtain a final integer solution, a second phase is usually executed. This can be done in several ways.

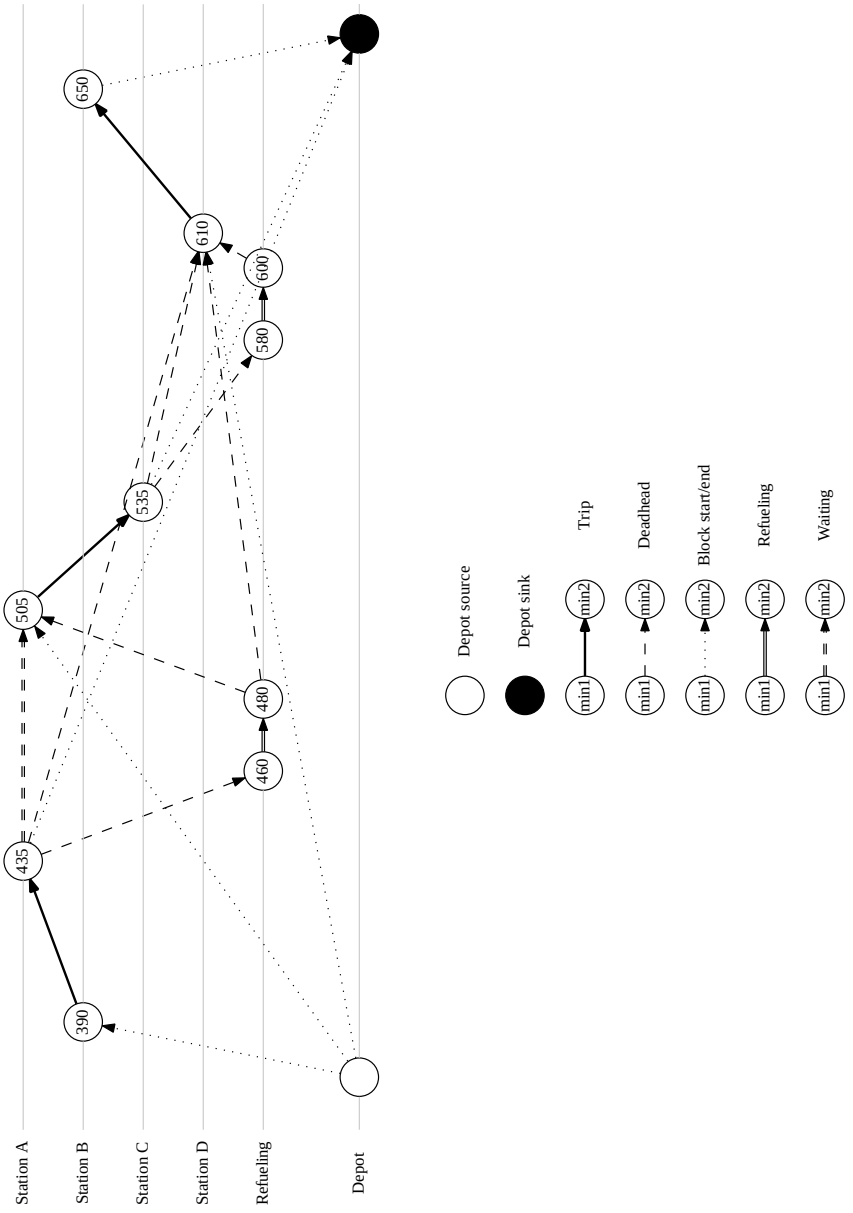

Figure 1: Example of a generation network

One approach is to use Lagrangian relaxation [16], as in the case of integrated vehicle and crew scheduling problems, where the linking constraints between vehicle and crew are an ideal candidate to be relaxed.

Another possibility is to embed the column generation process in a branch-and-bound framework that searches for the optimal integer solution [6, 21]. This method is also referred to as branch-and-price.

Integer solutions can also be obtained by the use of truncated column generation, which was used for the MDVSP by Pepin et al. [22]. After the column generation has terminated, this approach attempts to obtain an integer solution of the master problem by performing a rounding/variable fixing heuristic that fixes the values of fractional variables.

Our approach is similar to truncated column generation, but instead of using a heuristic approach to round the variables of our problem, we simply re-introduce the integrality constraint to our master problem, and solve it using an IP solver. A similar approach to this was also applied in [14]. To control the time of the solution process, we also set a time limit for the solver.

4 Test results

We tested our method on random input generated based on [9]. Both single and multiple-depot test cases were created in different sizes; namely 50, 250, 500, 1000, 1500 and 2000 trips. While the model we presented can be used as a general framework for handling different vehicle-specific activities, we chose refueling to showcase our test instances, as this is the most widely studied vehicle activity. However, while papers in the literature usually deal with only a single fuel type for a problem, our test instances use two different fuel types. Moreover, we allow vehicles belonging to the same depot to have different types of fuel, which is rarely considered in other studies.

A total of 24 test cases were solved both for the single- and multiple depot problems; we generated 4 instances for each problem size. The ILOG CPLEX solver was used during the solution process, with a limit set on its running time: i.e. the limit on the column generation phase was 7.5 hours, while the IP phase ran for 3 hours in the single depot case, and 5 hours in the multi-depot case. This gave a total maximum running time of 10.5 hours (37800 seconds) for single depot problems, and 12.5 hours (45000 seconds) for multi-depot problems. These limits have to be included if we consider the practical aspects of such a solution method, as planners should have an estimate on the required running time if they want to have a feasible solution.

The following types of data are presented for our results: the number of vehicle blocks given by the initial heuristic (initial blocks), the final number of blocks given by the resulting IP (IP blocks), the final optimality gap (%) given by CPLEX, and the solution time of the instance.

Table 1 shows results obtained for the single depot test instances. It can be seen that good quality solutions are achieved for all 24 problems, the optimality gap is

Table 1: Single depot test instances.

Instance	Trips	Initial blocks	IP blocks	MIP gap (%)	Time (s)
input_01	50	16	16	0.00%	95
input_02		16	10	0.00%	80
input_03		16	11	0.00%	86
input_04		16	16	0.00%	85
input_05	250	58	71	0.01%	258
input_06		56	56	0.00%	335
input_07		58	58	0.01%	329
input_08		56	69	0.00%	257
input_09	500	99	131	0.08%	4938
input_10		107	104	0.43%	5473
input_11		103	107	0.49%	5527
input_12		96	137	0.18%	5014
input_13	1000	184	290	1.51%	11228
input_14		183	231	2.16%	19808
input_15		187	219	2.03%	19361
input_16		185	288	1.54%	11073
input_17	1500	260	611	0.01%	21028
input_18		268	529	3.56%	21629
input_19		264	612	0.01%	18488
input_20		262	595	2.05%	21612
input_21	2000	374	731	2.70%	21643
input_22		364	663	0.76%	21615
input_23		358	383	0.00%	18105
input_24		358	593	0.01%	21007

generally under 1% (with only a single instance being above 3%, and six instances lying between 1% and 3%). The maximum runtime limit was exceeded only by the 1500 and 2000 trip instances. One notable feature of all single depot results is that the IP solution of the problem uses significantly more buses than the initial heuristic. The reason for this can be found by examining the different cost factors of the input. Most of the vehicles generated for the input instances ended up having a relatively low daily cost, and a more significant distance proportional cost factor. Because of this, the IP solution attempted to minimize the distance traveled by its vehicles (which meant trying to schedule as few deadhead trips as possible). The initial heuristic is essentially a greedy assignment of trips to vehicles, which does not take the distance traveled into account, and only introduces new vehicles to the schedules if it really has to.

Overall, the solutions we obtained for a single depot and two fuel types look

favorable, even for larger instances. The maximum problem size presented by the test results for similar problems in other papers in the literature rarely exceed 1000 trips, while we obtained good quality solutions for several instances with 1500 and 2000 trips as well, taking into account vehicles with two different refueling constraints.

Table 2: Multiple depot test instances.

Instance	Trips	Initial blocks	IP blocks	MIP gap (%)	Time (s)
input_1	50	26	9	0.00%	10
input_2		25	8	0.00%	16
input_3		10	9	0.01%	26
input_4		9	9	0.00%	13
input_5	250	51	32	8.06%	19028
input_6		46	34	6.42%	19301
input_7		47	35	7.30%	19253
input_8		45	35	6.49%	18998
input_9	500	84	64	11.43%	22801
input_10		100	63	10.84%	23184
input_11		81	57	12.73%	24836
input_12		86	58	12.48%	24452
input_13	1000	181	135	15.29%	45067
input_14		164	122	23.59%	45045
input_15		177	124	21.65%	45087
input_16		182	124	23.48%	45064
input_17	1500	281	216	25.11%	45081
input_18		270	217	26.64%	45033
input_19		291	221	27.02%	45024
input_20		284	222	27.59%	45112
input_21	2000	366	320	26.35%	45113
input_22		376	329	26.10%	45160
input_23		378	336	26.19%	45119
input_24		335	313	27.60%	45061

Table 2 gives the test results for the multi-depot test cases. Here we used vehicles belonging to two different depots, and we also considered two fuel types (both depots had a mix of vehicles with both fuel types). This results in a more complicated problem (multiple depots, two fuel types, instances with a large number of trips) than those that are usually examined in the literature.

The results for small, 50-trip instances look promising, as we also found near optimal solutions in a short time. However, these were the only cases where both the column generation phase and the IP phase terminated well before its time limit.

Even for the 250 trip instances, the IP solution process was aborted prematurely as it ran out of time. For the 250 and 500 trip instances, the column generation process terminated with no more columns to generate, but it also ran out of time for the larger input.

The effect of reaching the time limit can clearly be seen in the results. While the average gap given by the solver was 9.47% for the 250- and 500-trip instances, where the column generation finished without any problems, the three remaining instance sets (with 1000, 1500 and 2000 trips), where both the column generation and IP solution phases were aborted because they exceeded the time limit, had an average gap of 24.72%. While these results might not seem particularly promising, the quality of the solution obtained only depends on the time allocated for the two phases. Based on the single depot, and small multi-depot results, the model will provide better quality solutions, but the time limit has to be increased significantly. However, this would invalidate the very reason that we introduced the time limit in the first place; namely, to ensure that the results are useful in practice, which means that they have to be produced in a reasonable time.

5 Conclusions and future work

In this paper, we presented a general framework for the integrated vehicle scheduling and assignment that also takes into account tasks for vehicle-specific activities. This framework provides a daily vehicle schedule that also includes the special needs of the vehicles executing it; activities such as refueling and parking are considered in the resulting vehicle blocks. We presented a set partitioning-based mathematical model for the problem, where most vehicle-specific activities can easily be integrated based on the desired constraints. This model is then solved using a column generation approach. We demonstrated the efficiency of the proposed model on randomly generated test instances, using refueling to showcase vehicle-specific activities.

Instances with one and two depots were both generated, and all of them had two fuel types with different constraints (also allowing the possibility that the same depot can contain vehicles with different fuel types). A time limit was set on both phases of the solution process to guarantee that a feasible result could be achieved in a reasonable time. Test runs for a single depot and two fuel types resulted in good quality solutions, as did the smaller-sized multi-depot ones. The results for the larger multi-depot instances had a bigger optimality gap, but this was due to one or both of the phases terminating because of the time limit set. Based on the results of the other test cases, these gaps should also be smaller if more time is provided for finding the solution.

While the results are promising, there is still room for improvement of the process. Implementing a proper branch-and-price framework could result in a better quality solution. Because of the limited running time, a truncated column generation approach with a rounding heuristic as the second phase should also be considered. Another way of reducing the solution time might be the parallelization

of the column generation process. Implementing these approaches and comparing their results should be the next phase of this work.

The model we created is a general, flexible framework that can handle multiple vehicle activities. Although we presented problems with two different fuel types as our test cases, we did not generate instances with two or more completely different activity types. Our future research work will also include experimenting with different vehicle constraints connected to these activities.

Acknowledgements

József Békési was supported by the EU-funded Hungarian grant EFOP-3.6.2-16-2017-00015.

Balázs Dávid was supported by the project "Integrated program for training new generation of scientists in the fields of computer science", no EFOP-3.6.3-VEKOP-16-2017-0002. The project has been supported by the European Union and co-funded by the European Social Fund.

Miklós Krész was supported by National Research, Development and Innovation Office - NKFIH Fund No. SNN-117879.

References

[1] Adler, Jonathan D. *Routing and scheduling of electric and alternative-fuel vehicles*. PhD thesis, Arizona State University, 2014.

[2] Adler, Jonathan D and Mirchandani, Pitu B. The vehicle scheduling problem for fleets with alternative-fuel vehicles. *Transportation Science*, 51(2):441–456, 2016.

[3] Bertossi, Alan A, Carraresi, Paolo, and Gallo, Giorgio. On some matching problems arising in vehicle scheduling models. *Networks*, 17(1):271–281, 1987.

[4] Bodin, Lawerence and Golden, Bruce. Classification in vehicle routing and scheduling. *Networks*, 11(1):97–108, 1981.

[5] Bodin, Lawerence, Golden, Bruce, Assad, Arjang, and Ball, Mark. Routing and scheduling of vehicles and crews: The state of the art. *Computers and Operations Research*, 10(1):63–212, 1983.

[6] Borndörfer, Ralf, Löbel, Andreas, and Weider, Steffen. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. *Lecture notes in economics and mathematical systems*, 600:3, 2008.

[7] Borndörfer, Ralf, Reuther, Markus, Schlechte, Thomas, Waas, Kerstin, and Weider, Steffen. Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877, 2015.

[8] Bunte, Stefan and Kliewer, Natalia. An overview on vehicle scheduling models. *Journal of Public Transport*, 1(4):299–317, 2009.

[9] Dávid, Balázs and Krész, Miklós. Application oriented variable fixing methods for the multiple depot vehicle scheduling problem. *Acta Cybernetica*, 21(1):53–73, 2013.

[10] Dávid, Balázs and Krész, Miklós. The dynamic vehicle rescheduling problem. *Central European Journal of Operations Research*, 25(4):809–830, 2017.

[11] Desaulniers, Guy, Desrosiers, Jacques, and Solomon, Marius M. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[12] Desrosiers, Jacques, Dumas, Yvan, Solomon, Marius M, and Soumis, François. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

[13] Freling, Richard and Pinto Paixão, José M. *Vehicle Scheduling with Time Constraint*, pages 130–144. Springer Berlin Heidelberg, 1995.

[14] Gintner, Vitali, Kliewer, Natalia, and Suhl, Leena. *A Crew Scheduling Approach for Public Transit Enhanced with Aspects from Vehicle Scheduling*, pages 25–42. Springer Berlin Heidelberg, 2008.

[15] Haghani, Ali and Shafahi, Yousef. Bus maintenance systems and maintenance scheduling: model formulations and solutions. *Transportation Research Part A: Policy and Practice*, 36(5):453–482, 2002.

[16] Huisman, Dennis. *Integrated and dynamic vehicle and crew scheduling*. PhD thesis, Erasmus University Rotterdam, 2004.

[17] Kuby, Michael and Lim, Seow. The flow-refueling location problem for alternative-fuel vehicles. *Socio-Economic Planning Sciences*, 39(2):125–145, 2005.

[18] Li, Jing-Quan. Transit bus scheduling with limited energy. *Transportation Science*, 48(4):521–539, 2013.

[19] Li, Jing-Quan. Battery-electric transit bus developments and operations: A review. *International Journal of Sustainable Transportation*, 10(3):157–169, 2016.

[20] Lübbecke, Marco E and Desrosiers, Jacques. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[21] Mesquita, Marta, Paias, Ana, and Respício, Ana. Branching approaches for integrated vehicle and crew scheduling. *Public Transport*, 1(1):21–37, 2009.

[22] Pepin, Ann-Sophie, Desaulniers, Guy, Hertz, Alain, and Huisman, Dennis. A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12(1):17–30, 2009.

[23] Reuer, J, Kliewer, N, and Wolbeck, L. The electric vehicle scheduling problem: A study on time-space network based and heuristic solution approaches. In *Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT), Rotterdam*, 2015.

[24] Steinzen, Ingmar, Gintner, Vitali, Suhl, Leena, and Kliewer, Natalia. A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382, 2010.

[25] van Kooten Niekerk, M. E., van den Akker, J. M., and Hoogeveen, J. A. Scheduling electric vehicles. *Public Transport*, 9(1):155–176, 2017.

Received 7th May 2018