Formulation and Algorithms for Creating Parking Plans in a Public Transportation Company

Thé Van Luong and Éric D. Taillard

University of Applied Sciences of Western Switzerland, HEIG-VD, Department of Industrial Systems

Abstract

Creating parking plans for the vehicles of one of the largest public transportation company of Switzerland is a difficult and long task to do by hand. This problem is formulated as an integer linear program. Commercial MIP solvers can only solve small instances. Heuristic algorithms were developed for this problem. Yet, the company uses daily the software embedding the algorithms presented in this article. For an easier management of vehicles in depots, the company requires using as few parking lanes as possible while grouping vehicles of the same type. This sub-problem is first solved and several good solutions are stored. Then, a solution of this first sub-problem is used to fix the groups of vehicles and the departure hour of each vehicle is assigned while taking into account various practical constraints. This second sub-problem is approached with taboo search metaheuristics in case an exact solution cannot be computed in a reasonable time. If no appropriate solution is found, the second sub-problem is solved again by fixing other groups.

The algorithms designed to solve both sub-problems are compared with a commercial MIP solver, assessing the difficulty of this industrial problem.

Keywords: industrial problem; metaheuristics; logistic

[1]the-van.luong@heig-vd.ch, eric.taillard@heig-vd.ch

1. Introduction

The placement of vehicles leaving a depot for performing travels of services is certainly not new, but the literature for an automated placement in the depot is very limited.

Apart from industrial problems, some depot models proposed in previous works in [1, 2, 3, 4, 5, 6] are rather simplified: unique vehicle length, single vehicle type, an unlimited length of lanes, no strict respect on departure times, no interaction between vehicles leaving the depot on adjacent lanes and so on.

Furthermore, the objective function used in algorithms from the works quoted above is mainly based on the minimization of the number of shunting movements (i.e. manœuvers) needed to reposition vehicles that are not properly placed. In practice, making a vehicle dispatching and performing such movements might be not acceptable since it represents a loss of time for agents working in a depot and it will systematically result in departure delays. In fact, the formalization proposed in these works might be restricted only to academic problems.

This limited literature might be explained by the fact each company has its own operational constraints and each depot its own characteristics.

So, we had to imagine a methodology from scratch when one of the largest Swiss public transportation companies asked us to design algorithms for the automatic placement of vehicles in its depots. The company owns several hundred different types of vehicles and has almost 10 kilometers of lanes in depots for parking them. Furthermore, many operational constraints make this industrial problem much complex than an academic one.

A key idea when solving a problem of large size with numerous constraints is to decompose it into a series of sub-problems easier to solve. This reduces the search space of the combinatorial problem which is prohibitively large. Then, the sub-problems are tentatively solved.

Two sub-problems were identified for constructing parking plan:

1. The group positioning.
2. The schedule assignment.

This decomposition is based on the good practices imposed by the company where the grouping of vehicles of the same type and the maximization of the available space are the most important objectives.

Details of the methodology leading to this decomposition are presented in [7] as well as complete information on this industrial problem An evaluation of the quality of parking plans generated by the software are also compared with those ones manually designed by the logistics manager in this work.

The present article focuses on the two sub-problems and the algorithms designed to solve them.

Section 2 briefly describes the problem. Sections 3 and 4 introduce mathematical formulations for the two sub-problems. Section 5 describes the heuristics algorithms that were implemented in our software. Section 6 provides numerical results and compares these algorithms with the solutions produced by a piece of general optimization software.

2. Problem description

Most vehicles are leaving depots at three different periods of the day: early in the morning, noon and late afternoon.

Several reference plans are designed by the manager in charge of the depots. These parking plans are adapted each day for taking into account additional services (e.g. special events or vacations) and possible changes that might occur.

Regarding the process inside a depot, hundreds of drivers are assigned to vehicles. Each vehicle must be of appropriate type and characteristics for performing the given service. Each service that must start exactly at the time specified by the timetable.

The position of each vehicle in the depot must be determined according to operational constraints (good practices) to ensure a smooth and reliable process.

To avoid an ingoing vehicle having to wait for another vehicle of a different type that is late and that must be parked in front of it on the same lane, a good practice is to have lanes composed of a single vehicle type. So, a vehicle

coming back to the depot can be immediately parked at the right place for its next departure. To avoid drivers having to search in the depot for a vehicle of the right type and characteristics, another good practice is to group the lanes containing the vehicles of same type and functionalities.

A vehicle type or model is called a *series* by the company. A physical vehicle servicing a given line according to a given timetable is referred to as a *schedule*. In addition to a set of vehicle series that must be used for a given schedule, the latter may also be specified by additional characteristics known as *schedule type*. This is, for instance, a vehicle equipped with a video camera or with an automatic ticket distributor or the fact that the vehicle must be back to the depot before 10 a.m.

The company owns several depots, located at various places. Each depot can be managed independently. Indeed, each schedule is already assigned by the company to a given depot and the vehicles are always supposed to return to the depot where they started their service.

2.1. Criteria for constructing a parking plan

As input, a file extracted from the database contains a list of schedules ordered by day type and by departure time. As output, a solution is a permutation of these schedules with the corresponding lanes that have been assigned.

Thus, a parking solution associates each schedule with a lane identifier and a position in the assigned lane.

The construction of a parking plan relies on different criteria:

- Physical hard constraints: the total length of vehicles stored on a lane, the processing of each vehicle in a lane one behind the other, the series interdiction on some lanes (e.g. megabuses that cannot be manœuvered), the equipment compatibility of vehicles on lanes (rails and overhead wires) and, for the tramways, their direction (uni- and bidirectional vehicles and lanes).

- Topological constraints of the depot: vehicles in front of some lanes must

leave before the other vehicles located in front of other lanes (trolley and tramway lanes).

- Good practices (soft constraints) imposed by the company leading to a better organization of depots by order of importance:
  1. The grouping of schedules with the same vehicle series on contiguous lanes.
  2. The maximization of free lanes and then the occupied space. Free lanes can store vehicles out of order when all repairing stations are in use.
  3. The grouping of schedule type (particularities) for the same vehicle series.
  4. The respect of minimal and ideal time periods between two departures.

In practice, there are more than 200 schedules and 60 lanes for the most constrained delivery and the largest depot, leading to over 300 possible positions on lanes for each bus schedule.

The number of combinations is huge and traditional permutation-based methods might fail to deliver a solution compatible with production requirements.

To deal with this issue, our software decomposes the problem into a series of sub-problems easier to solve. This decomposed approach is based on similar principles than a column generation and is motivated by the hierarchical objectives (hard constraints and good practices) presented above.

The first sub-problem (i.e. the group positioning problem) is to maximize the grouping of vehicle series on lanes and the number of free lanes (practices 1. and 2.). This sub-problem is exclusively based on vehicle series.

Thereafter, the schedule assignment sub-problem is to assign departure times on vehicles while trying to maximize the grouping of schedule types within the same vehicle series and respecting time periods between departures (practices 3. and 4.).

2.2. Illustration of a solution for a sub-depot

The following case gives an insight of a solution for a portion of a depot of a given morning delivery.

Figure 1(a) provides an illustration of a solution for the group positioning problem. Based on vehicle lengths, vehicles of the same series are grouped, the number of free lanes is maximized and then the occupied space is maximized.

Each rectangle in this figure represents a vehicle associated with a series. Each column corresponds to a parking lane specified by its length and its equipment (rail, electrical wire).

On the one hand, vehicle were arranged to group schedules representing the S50, S59, S37, S33 and S38 series. On the other hand, they were arranged in such a way that the number of free lanes is maximized without cutting any vehicle series.

The schedule assignment sub-problem (see Figure 1(b)), is solved by starting from the solution established during the group positioning phase.

From established positions of groups of vehicles series, the goal is to determine for each group the best permutation of schedules that maximizes the grouping of schedule types and that respects minimal and ideal time periods between two departures in the same lane.

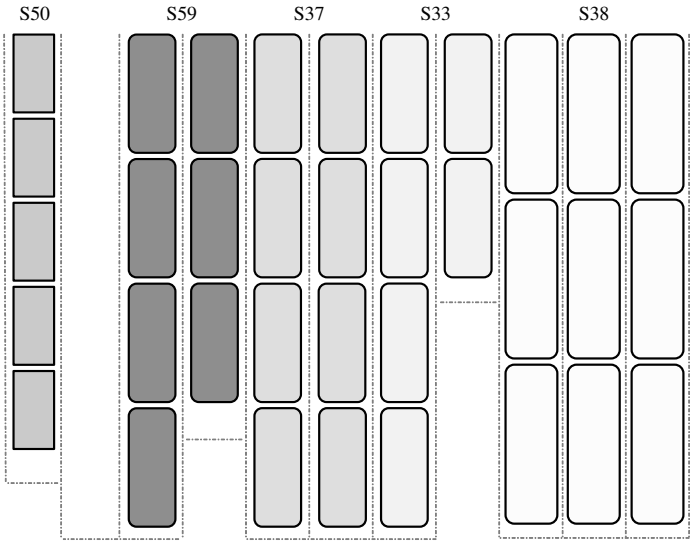
Schedules are specified by the departure hour and a color corresponding to schedule type (particularities). The other information depicted in brackets is the line number.

In this simple case (i.e. 35 schedules and 11 lanes), the number of possible combinations is already significant.

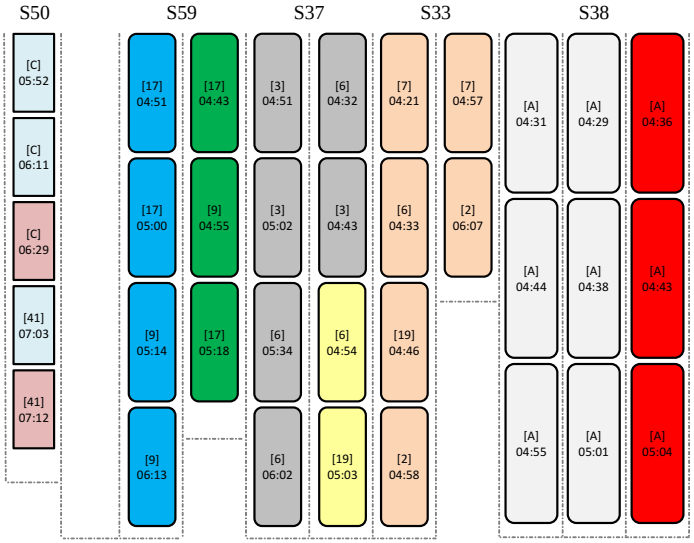
3. Group positioning problem

The group positioning problem is the first step to create a parking plan.

Solving this sub-problem aims at calculating the best position of schedules on the lanes that first maximizes the grouping of vehicles of the same series, then the number of free lanes and the occupied space.



(a) Group positioning problem. Based on vehicle lengths, the goal is to group vehicle series while maximizing the available space.



(b) Schedule assignment problem. From previous positioned groups, it consists of assigning departure times while grouping schedule types.

Figure 1: Example of a solution for a portion of a depot of a given morning delivery.

This problem is exclusively based on vehicle lengths while respecting hard constraints; other particularities such as the schedule type are discarded at this step.

While solving the problem, hard limitations, blocking topological constraints and good practices must be satisfied as well: the total length of vehicles on a lane (taking into account the spacing between the vehicles), the lane equipment and topology adapted to the vehicle series, the tramway direction and applying a single vehicle series on a lane.

3.1. Mathematical formulation

3.1.1. Sets

- I: Logical vehicles to be placed.
- S: Vehicle series (including 0 indicating that a lane is empty).
- V: Lanes where the vehicles can be parked.
- $V_s \subset V, s \in S$: Subset of lanes where vehicle series s can be parked.

3.1.2. Data

- s_i: Series of vehicle i.
- l_i: Length of vehicle i.
- c_v: Capacity (length) of lane v.

3.1.3. Variables

- x_{iv}: Vehicle $i \in I$ is on lane $v \in V_{s_i}$ ($x_{iv} = 1$) or not ($x_{iv} = 0$)
- y_{sv}: Lane $v \in V_s$ is occupied by vehicle series s ($y_{sv} = 1$) or not ($y_{sv} = 0$)

To obtain a linear objective, it is possible to introduce variables $z_{ss'v}$ whose value must be set to: $z_{ss'v} = y_{sv} \cdot y_{s'(v+1)}$. This is automatically done by a solver like *Gurobi* and does not significantly influence its computational time.

3.1.4. Constraints

- A vehicle is assigned to exactly one lane:

$$\sum_{v \in V_{s_i}} x_{iv} = 1 \quad \forall i \in I \tag{1}$$

- A lane is occupied by exactly one series:

$$\sum_{v} y_{sv} = 1 \quad \forall s \in S \tag{2}$$

- A vehicle can be placed only on a lane with the right series:

$$x_{iv} \leq y_{sv} \quad \forall i \in I, \forall v \in V_{s_i}, \forall s \in S \tag{3}$$

- The sum of vehicle lengths on a lane cannot exceed the lane capacity:

$$\sum_{i} l_i \; x_{iv} \leq c_v \quad \forall v \in V_{s_i} \tag{4}$$

3.1.5. Objectives

- First objective consists of grouping vehicle series, i.e. minimizing the number of different series in adjacent lanes:

$$obj_1: \text{ Minimize } \sum_{s \neq 0} \sum_{s' \neq s} \sum_{v=1}^{|V|-1} y_{sv} \cdot y_{s'(v+1)} \tag{5}$$

- Second objective is to minimize the number of occupied lanes (or maximizing free lanes):

$$obj_2: \text{ Minimize } \sum_{s \neq 0} \sum_{v} y_{sv} \tag{6}$$

- Third objective is to minimize the remaining space on each lane (or maximizing the occupied space):

$$obj_3: \text{ Minimize } \sum_{s \neq 0} \sum_{v} c_v \; y_{sv} \tag{7}$$

- The global and hierarchical objective for the group positioning is:

  Minimize $p_1\ obj_1 + p_2\ obj_2 + obj_3$, where p_1 and p_2 are weights such that $p_1 \gg obj_2$ and $p_2 \gg obj_3$.

  (8)

The set partitioning problem (which is NP-complete, [8]) can be polynomially transformed into this group positioning problem by considering two vehicle series having exactly the same number of vehicles and lane capacities equal to the weight of the elements of the set partitioning problem. This shows that the positioning of groups of vehicle series is NP-complete.

4. Schedule assignment problem

Once all the groups of vehicle series are placed, the second sub-problem to obtain a parking plan is the assignment of schedules.

The goal is to assign schedules on positioned groups in such a way schedule types within a vehicle series are grouped. As a second hierarchical objective, another good practice is to respect as much as possible minimal and ideal time periods separating two departures.

In addition, a feasible solution must respect the remaining hard and topological constraints: chronological departure times on each lane, tramway direction and weak blocking that may happen (i.e. a vehicle in front of a lane blocks the vehicles in front of adjacent lanes).

4.1. Mathematical formulation

4.1.1. Sets

- I: Schedules (or physical vehicles) to be placed.
- V: Lanes where the vehicles must be parked.
- $V_i \subset V, i \in I$: Subsets of lanes where vehicle i can be parked.
- $N \subset V$: Lanes for which the vehicle in front must leave before those in front of the next adjacent lane.

- $P \subset V$: Lanes where the vehicle in front must leave before those in front of the previous adjacent lane.

4.1.2. Data and parameters

- h_i: Departure time of vehicle i ($i \in I$).
- h_{min}: Minimal difference of time between departures on the same lane ($h_{min} = 10$).
- h_{ideal}: Ideal difference of time ($h_{ideal} = 20$).
- n_v: Number of vehicles to be placed on lane v ($v \in V$)
- t_{ij}: Schedule type of vehicle i and j are similar ($t_{ij} = 1$) or not ($t_{ij} = 0$).
- w_{ij}: Reward for respecting departure time between h_{min} and h_{ideal} for vehicles i and j (with $h_j > h_i$)

$$w_{ij} = \begin{cases} 1.5 \cdot h_{min} & h_{min} \leq h_j - h_i \leq h_{ideal} \\ h_{min} & h_j - h_i > h_{ideal} \\ -4 \cdot (h_{min} - (h_j - h_i)) & h_j - h_i < h_{min} \end{cases}$$

4.1.3. Variables

- $x_{ivp} = \begin{cases} 1 & \text{if schedule } i \text{ is on lane } v \text{ at position } p \\ 0 & \text{otherwise} \end{cases}$
  $(i \in I, v \in V_i, 1 \leq p \leq n_v)$

4.1.4. Constraints

- A vehicle is assigned to exactly one position:

$$\sum_{i|v \in V_i} x_{ivp} = 1 \quad \forall v \in V, \quad 1 \leq p \leq n_v \tag{9}$$

- A position is occupied by exactly one vehicle:

$$\sum_{p=1} n_v x_{ivp} = 1 \quad \forall i \in I, \forall v \in V_i \tag{10}$$

- The departure time of any vehicle must be prior to the vehicle following it:

$$h_i \; x_{ivp} \leq \sum_{j \neq i} h_j \; x_{jv(p+1)} \quad \forall i \in I, \forall v \in V_i, 1 \leq p < n_v - 1 \tag{11}$$

- Weak blocking constraints:

$$h_i \; x_{iv1} \leq \sum_{j \neq i} h_j \; x_{j(v+1)1} \quad \forall v \in N, \forall i \in I \tag{12}$$

$$h_i \; x_{iv1} \leq \sum_{j \neq i} h_j \; x_{j(v-1)1} \quad \forall v \in P, \forall i \in I \tag{13}$$

4.1.5. Objectives

- Maximize the number of schedules of the same type (color) on each lane.

$$obj_1: \text{ Maximize } \sum_{i} \sum_{j \neq i} \sum_{v \in V_i} \sum_{p=1}^{n_v - 1} t_{ij} \; (x_{ivp} \cdot x_{jvp_{+1}}) \tag{14}$$

- Maximize the number of schedules of the same type between successive lanes (i.e. comparing the last vehicle of a lane with the first vehicle of the next lane):

$$obj_2: \text{ Maximize } \sum_{i \in I} \sum_{j \neq i} \sum_{v, v+1 \in V_i} t_{ij} \; (x_{ivn_v} \cdot x_{j(v+1)1}) \tag{15}$$

- Maximize best practices between any two departures (i.e. penalizing departures that do not respect a minimal time and rewarding departures that respect an ideal time):

$$obj_3: \text{ Maximize } \sum_{i} \sum_{j \neq i} \sum_{v \in V_i} \sum_{p=1}^{n_v - 1} w_{ij} \; (x_{ivp} \cdot x_{jv(p+1)}) \tag{16}$$

- The global and hierarchical objective for the schedule assignment problem is:

$$\text{Maximize } \; m \; (obj_1 + obj_2) + obj_3, \text{ where } m \text{ is such that } m \gg obj_3 \tag{17}$$

The respect of good practices is taken into account by defining a hierarchical objective function that assigns a penalty for each good practice that is not respected. A large penalty is given if schedule types are not clustered (mixed schedules on the same lane or all vehicles of a schedule type not on contiguous lanes). A smaller penalty is given if the time spacing between two departures is not between minimal and ideal values. The values of penalties are determined in such a way one has hierarchical objectives.

For a few events, the manager needs to regroup particular bus lines that go to the district of international organizations [7]. Another similar hierarchical objective taking line groupings might be added between obj_2 and obj_3.

5. Heuristic solution

Since a piece of general optimization software might not be adapted to solve this industrial problem, this section describes the algorithms that were implemented in our software. The goal is to provide as many details as possible, so that the solutions to group positioning and schedule assignment can be reproduced.

The philosophy beyond the algorithms is simple: if an exact procedure fails to rapidly solve a sub-problem, then a dedicated heuristic is executed.

5.1. Group positioning problem with contiguous blocks

For positioning groups of vehicle series, the key idea of the proposed algorithm consists of creating contiguous blocks of schedules with the same vehicle series and finding the best position of these blocks on the available lanes. Blocks of schedules are considered as an inseparable entity.

Algorithm 1 provides the steps of the group positioning heuristic. It requires all_series, the set of all groups of vehicle series that must be parked in the lanes of a depot, specified by the set $lanes$. First, for all series of vehicles, all the possible positions for placing a compact block grouping all the vehicles of a series are computed. This approach is feasible for the problem instances under

consideration since the number of different positions is limited to a few dozens. Then, the best position for each group is calculated by solving a kind of set partitioning problem with a backtracking algorithm (Algorithm 2).

Algorithm 1 Group positioning

Require: *all_series* and *lanes*

1: **for** each *current_group* of *all_series* **do**
2: *group_lanes* ← *lanes*
3: Filter in *group_lanes* the lanes for which *current_group* is forbidden
4: *group_position* ← ∅
5: **for** $i \leftarrow 1, \ldots, |group\_lanes|$ **do**
6: **if** all vehicles of *current_group* can be placed in an inseparable block starting from lane i while verifying hard constraints for *current_group* **then**
7: *current_position* ← smallest inseparable block starting from lane i that contains all vehicles of *current_group*
8: *group_positions* ← *group_positions* ∪ *current_position*
9: **end if**
10: **end for**
11: *all_positions*[*current_group*] ← *group_positions*
12: **end for**
13: *series_groups* ← Algorithm 2(*all_series*, *all_positions*, *lanes*)

Ensure: *series_groups*

5.1.1. Backtracking algorithm to position all groups

Once all the possible block positions are generated, an enumerative method based on a depth-first search is performed to position all groups optimally. This is done by Algorithm 2 which requires *all_positions* provided by Algorithm 1 in addition to *all_series* and *lanes*.

The algorithm enumerates all possible positions for the first group, then the second one with remaining lanes and so on. Since groups are considered

as inseparable blocks and their number is limited, the procedure checking if a group can be positioned on remaining lanes is very fast.

The exact method is able to rapidly find an optimal solution for small instances. For larger ones, the algorithm execution might be stopped after a certain period of time (user-defined parameter).

In any cases, a maximum of 10 best solutions found so far is saved. They are identified using equation (6) in Section 3 as first objective and then equation (7). Keeping several solutions allows using different vehicle series grouping if the best grouping solution reveals to be infeasible during the schedule assignment phase.

5.2. Group positioning with group cuts

Cutting groups of vehicle series occurs in few situations when the combination of group positioning and schedule assignment has failed. For example, if it is not possible to have only inseparable blocks of vehicle series or if the decisions taken at a step may over-constrain the problem for the next step.

In all cases, the first objective is to maximize the number of free lanes when cutting groups.

Then, as a second hierarchical objective, the cuts are done in such a way that: a) the distance of grouping of vehicle series in the lanes is minimized b) the occupied space on each lane is maximized.

Algorithm 3 provides details about the backtracking algorithm that position group cuts.

According to what is done by the company, the groups often cut are the series involving the highest number of vehicles and the simplest constraints to satisfy. In practice, this corresponds to buses. As a consequence, the first groups to be sorted must be the ones with the vehicles with the hardest constraints (the lane equipment and the tramway direction).

Thereafter, the main part of the algorithm aims at determining the best position (Algorithm 4) for the first group on all available lanes, then the second one on remaining lanes, and so on.

Algorithm 2 Backtracking to position all groups

Require: *all_series*, *all_positions* and *lanes*

Sort *all_series* by increasing order of possible positions on *lanes* (it quickly allows pruning unwanted nodes and detecting if no solution is feasible)
for each *position* of *all_positions*[1] **do**
  Create *node* with *node.current_group* $\leftarrow$ *all_series*[1], *node.current_position* $\leftarrow$ *position*, *node.remaining_lanes* $\leftarrow$ *lanes* and *node.used_lanes* $\leftarrow \emptyset$
  Push *node* to *pool*
end for
while *pool* $\neq \emptyset$ AND NOT *end_condition* **do**
  Pop *node* from *pool*
  if *node.current_position* is available on *node.remaining_lanes* **then**
    Update *node.used_lanes* and *node.remaining_lanes* with *node.current_position*
    if *node.current_group* is the last of *all_series* (all vehicules placed) **then**
      Compute the *score* of the feasible *solution* (number of free lanes given by $|node.remaining\_lanes|$ (equation (6), Section 3) and the remaining space in *node.used_lanes* (equation (7))
      all_solutions $\leftarrow$ *all_solutions* $\cup$ *solution*
    else
      for each *position* of *all_positions*[*node.current_group* + 1] **do**
        Create *new_node* with *new_node.current_position* $\leftarrow$ *position*, *new_node.current_group* $\leftarrow$ *node.current_group* + 1, *new_node.remaining_lanes* $\leftarrow$ *node.remaining_lanes*, *new_node.used_lanes* $\leftarrow$ *node.used_lanes*
        Push *new_node* to pool
      end for
    end if
  end if
end while
series_groups $\leftarrow$ 10 best solutions from *all_solutions*

Ensure: *series_groups*

Algorithm 3 Backtracking to position all cut groups

Require: *all_series* and *lanes*

Sort *all_series* by decreasing constraint difficulty and then by increasing number of vehicles
Create *node*:
  $node.current\_group \leftarrow all\_series[1]$,
  $node.remaining\_lanes \leftarrow lanes$, $node.used\_lanes \leftarrow \emptyset$
Push *node* to *pool*
while $pool \neq \emptyset$ **do**
  Pop *node* from *pool*
  if *node.current_group* is the last of *all_series* (all vehicules placed) **then**
    Retrieve the feasible *solution* from *node.used_lanes* and *all_series*
    $all\_solutions \leftarrow all\_solutions \cup solution$
  else
    if vehicle lengths of *node.current_group* not higher than length of *node.remaining_lanes* **then**
      $group\_positions \leftarrow$ Algorithm 4$(node.current_group, node.remaining_lanes)$
      if $group\_positions \neq \emptyset$ **then**
        for each *position* of *group_positions* **do**
          Create *new_node*:
            $new\_node.used\_lanes \leftarrow node.used\_lanes$,
            $new\_node.remaining\_lanes \leftarrow node.remaining\_lanes$,
            $new\_node.current\_group \leftarrow node.current\_group + 1$
          Update *new_node.used_lanes* and *new_node.remaining_lanes* with *position*
          Push *new_node* to *pool*
        end for
      end if
    end if
  end if
end while
series_groups $\leftarrow$ 10 best solutions from *all_solutions*

Ensure: *series_groups*

For some groups, the number of best positions (partial solutions) can be multiple (Pareto Front). In practice, this number is very low for the first groups. Hence, the whole front of solutions of each group can be used for subsequent computations.

Then, the list of global solutions is sorted according to the second objective of the hierarchical evaluation function (minimization of distance within a cut group and maximization of the occupied space on used lanes) for each combination of groups, keeping a maximum of 10 solutions.

5.2.1. Backtracking to generate all best positions

Given a group and a set of lanes, the internal backtracking (Algorithm 4) is able to generate all best cut positions.

The key idea is to construct all best possible positions in an enumerative way by starting to check the feasibility of all possible sub-lanes of size one, then all the sub-lanes of size two and so on.

Determining in advance the minimal number of lanes required allows quickly pruning non-promising nodes.

The second hierarchical objective (equations (5) and (7)) is used to generate a set of Pareto solutions with a maximum number of free lanes, limiting the number of solutions with cut positions. Note that a solution with the minimum distance between 2 sub-groups of the same series, corresponds to a group that is not cut.

The exact algorithm finds an optimal solution or a Pareto optimal set for small instances. For larger instances, the algorithm might be stopped after a certain time (parameter set by the user).

5.3. Schedule assignment problem

Once the positioning of groups of vehicles series on the lanes is determined, the next problem is to find for each group the best assignment of schedules on these lanes. The best solution from the group positioning problem is extracted

Algorithm 4 Generate best positions for group cuts

Require: *group* and *lanes*

1: *minimal_number_lanes* is determined with a greedy heuristic by sorting lanes by decreasing order of lengths and then by placing vehicles on them
2: **for** each *lane* of *lanes* **do**
3: Create *node*: $node.sub\_lanes \leftarrow lane$
4: Push *node* to *pool*
5: **end for**
6: **while** $pool \neq \emptyset$ AND NOT *end_condition* **do**
7: Pop *node* from *pool*
8: **if** total vehicle length is above the length of $node.sub\_lanes$ **then**
9: **if** *group* is compatible with hard constraints of $node.sub\_lanes$ AND $|node.sub\_lanes| < minimal\_number\_lanes$ **then**
10: **for** each $lane \in lanes \setminus node.sub\_lanes$ **do**
11: Create *new_node*: $new\_node.sub\_lanes \leftarrow node.sub\_lanes \cup lane$
12: Push *new_node* to *pool*
13: **end for**
14: **end if**
15: **else**
16: For the feasible *solution*, compute obj_1, the grouping distance objective between sub-groups of vehicles of the same series (equation (5)) and obj_3, the occupation of lanes (equation (7))
17: **if** *solution* non-dominated according to obj_1 and obj_3 **then**
18: Update the archive of non-dominated solutions and *best_positions*
19: Save *current_position* associated with *group* and $node.sub\_lanes$
20: $best\_positions \leftarrow best\_positions \cup current\_position$
21: **end if**
22: **end if**
23: **end while**

Ensure: *best_positions*

and assignments of schedules are done group by group independently with Algorithm 5. Since there might be possible interactions between groups, several global assignments are tentatively performed by changing the order in which each group is treated.

5.3.1. Separation of groups of vehicle series

Groups of vehicle series can be divided into two categories:

- Independent groups for which the schedules can be assigned in an independent manner without interacting with other vehicle series (buses in most cases).

- Dependent groups for which the schedule assignment could have an impact on the future assignment of other group positions on adjacent lanes (mostly for trolleybuses and tram-cars). In this case, the algorithm tries different combinations until reaching one that works. It is not necessary to try all combinations exhaustively: it is sufficient to think about only combinations involving adjacent groups.

  If it is not possible to construct valid parking plans because of dependent groups, the assignment is restarted with the second best solution from the group positioning problem and so on.

Algorithm 5 shows how assignments are processed for both dependent and independent groups.

5.3.2. Process of schedule assignment

Algorithm 6 details the different steps to assign schedules of a given group. Basically, a time-limited branch-and-bound is used to construct a solution.

For dependent groups, the procedure needs to take into account schedules that have been previously assigned to other groups (*tentative_parking_plans*). As said above in Algorithm 5, the order of execution of groups may lead to a non-feasible solution.

Algorithm 5 Global schedule assignment

Require: *series_groups*

1: Get one *group_solution* from *series_groups*
2: Separate *group_solution* into: *dependent_groups* and *independent_groups*
3: **for** each *i_group* of *independent_groups* **do**
4: $parking\_plan \leftarrow$ Algorithm 6$(i_group, \emptyset)$
5: $parking\_plans \leftarrow parking\_plans \cup parking\_plan$
6: **end for**
7: Determine *combination_list*, containing all the permutations of adjacent *dependent_groups*
8: **repeat**
9: Get *current_combination* from *combination_list*
10: $tentative\_parking\_plans \leftarrow \emptyset$, $error \leftarrow false$
11: **for** each *d_group* of *current_combination* AND NOT *error* **do**
12: $tentative\_parking\_plan \leftarrow$ Algorithm 6$(d_group, tentative_parking_plans)$
13: **if** $tentative\_parking\_plan = \emptyset$ **then**
14: $error \leftarrow true$
15: **else**
16: $tentative\_parking\_plans \leftarrow tentative\_parking\_plans \cup tentative\_parking\_plan$
17: **end if**
18: **end for**
19: **until** NOT *error* OR $combination\_list = \emptyset$
20: **if** NOT *error* **then**
21: $parking\_plans \leftarrow parking\_plans \cup tentatives\_parking\_plans$
22: **end if**

Ensure: *parking_plans*

If the sub-problem cannot be solved in an exact manner, a taboo search metaheuristic is used to improve the solution obtained.

Algorithm 6 Schedule assignment for a single group

Require: *current_group* and *tentative_parking_plans*

Apply a (time-limited) branch-and-bound for *current_group* with *tentative_parking_plans* to obtain *schedule_solution*
if *schedule_solution* $= \emptyset$ **then**
  tentative_parking_plan $= \emptyset$
else
  if *schedule_solution* not optimal **then**
    Apply a taboo search to refine *schedule_solution*
  end if
  tentative_parking_plan $\leftarrow$ *tentative_parking_plan* $\cup$ *schedule_solution*
end if

Ensure: *tentative_parking_plan*

Once the schedule assignment is found for the group at hand, the process is repeated for other groups until resulting parking plans are completed.

Templates of branch-and-bound and taboo search are not detailed since they do not present any implementation particularities or difficulties. Specific details are emphasized below though.

5.3.3. Branch-and-bound to build a schedule assignment

For each group of vehicle series, a set of lanes is initially available with locations for which all the schedules of the group must be assigned. The method tries in an enumerative way to put a schedule on the first location, another schedule on the second one and so on. A depth-first search quickly allows constructing a feasible solution.

The evaluation function is hierarchical (implemented by aggregating objectives with a strong difference of weights):

1. Grouping by schedule type (equations (14) and (15)). This part evaluates

the number of groups done on the same lane and on adjacent lanes.

2. Respect of the minimal time period between two departures (equation (16)).

The lower bound computation is based on the potential maximum grouping of schedule types that could be obtained if the exploration goes further. This estimate is often lower than the best solution found so far and allows pruning a very high number of unwanted solutions.

The algorithm also includes another natural pruning procedure based on the equipment compatibility, the tramway direction, schedule compatibility and topological constraints on adjacent lanes (trolleybus and tramways).

The exact algorithm quickly finds an optimal solution for small instances. For larger ones, the algorithm stops after a user-defined time limit and returns the best feasible solution found.

5.3.4. Taboo search to improve the schedule solution

The taboo search is based on moves that exchange two schedules. This exchange operator does not allow transforming a feasible solution into a non-feasible one. Taboo conditions are implemented with a simple array that stores the iteration at which a move can be used again. In practice, this method provides better results (for a time-limited execution) than taboo conditions that prevent from replacing the two same schedules to the same position.

The taboo duration is set to the number of moves divided by 16 (determined by a parameter tuning heuristic). A simplified variant of the pruning procedure aforementioned is used to rapidly determine if a tentative move leads to an infeasible solution. Both taboo search and branch-and-bound use the same evaluation function. The algorithm stops after a user-defined time limit as well.

6. Experiments

All the algorithms presented in the previous section were embedded in a software that was designed for the company.

Table 1: Configuration of the different depots. Information gathered on the lane length gives an estimation of the instance size.

	Depot				
	DB	DJ	DC	DK	DV
# lanes	29	70	43	6	11
Total lanes length [m]	2982	4649	993	405	589
Average lane length [m]	102	66	23	67	53
Tram (22, 31, 42, 44 and 53 meters)	√		√	√	
Trolleybus (18, 19 and 24 meters)		√			
Bus (12, 18 and 24 meters)		√	√		√
# topological and blocking constraints	24	20	28	1	1

When we started designing the application, the transportation company shared with us a number of test instances. These instances correspond to the complete constructions of morning deliveries, which are the hardest problems.

The numerical data of 30 group positioning and 90 schedule assignment instances are available on http://mistic.heig-vd.ch/taillard/problemes.dir/problemes.html. These instances are given under the form of the mathematical formulations of Sections 3 and 4.

The quality of the parking plans generated by our software are discussed in another article [7]. Very shortly, it can be said that the solutions produced by our software totally satisfy the production requirements of the company. The last runs the software daily for a few years.

The goal of this section is to compare the performance of these algorithms with a piece of general optimization software like *Gurobi*.

6.1. Instances and depot configuration

Table 1 provides details on the different depots of the company.

For each depot, basic information such as the number of lanes, the total lanes length of the depot or constraints about the lanes is reported. This gives

an insight of the difficulty of the instances for each depot.

The number of topological and blocking constraints is composed of the lanes that go to a specific direction for tramways, weak blockings that might occur between vehicles in adjacent lanes (tramways and trolleybuses) and vehicle series that are forbidden on some lanes (e.g. mega buses hard to maneuvered).

6.2. Configuration used for the experiments

All the experiments of this article were executed on a MacBook Pro Intel Core i7-3840QM 2.8 Ghz with 16 GB ram using Mac OS X with *Gurobi* version 8.1. Although this is a quad core laptop, only one CPU core was used for the experiments. This is in accordance with the setup of the transportation company.

The parameters of our software were set up in such a way that the construction of all plans for a complete week (sixteen deliveries) never exceeds an hour on a desktop computer.

6.3. Group positioning and schedules assignment

To evaluate our software for a regular usage, we asked the transportation company to provide us all the data to build new reference plans from scratch. This set is composed of 29 instances.

All the instances were directly extracted from the *HASTUS* timetable system that the transport company uses [9].

There are 5 depots, leading to 145 group positioning instances. Since there are different series to handle for each depot (not all series are present in the 29 instances), this represents a total of 420 schedule assignment instances.

6.3.1. Mathematical programming evaluation

Both group positioning (mathematical model given in Section 3) and schedule assignment problems (mathematical model given in Section 4) were tentatively solved with *Gurobi* for 29 instances.

Only one execution and a time limit of 1000 seconds are considered for each instance (a complete run of our software never exceeds this time limit). Since the

objective function of each problem is implemented as a weighted sum of multiple objectives, the best solution returned by *Gurobi* might not have the best value for each of the three objectives. Hence, the *PoolSearchMode* parameter was set to do a systematic search for a maximum of 10 solutions (i.e. when *Gurobi* finds the best solution, it continues finding the second best solution and so on).

To evaluate the solution quality, the best hierarchical objective values found by *Gurobi* and our software are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

Since some instances for specific depots are identical, results are grouped and the average computational time is also reported. The standard deviation is not represented since its value is close to 0.

A number of results are reported in Tables 2 to 5. Supplementary results are presented in Appendix, Tables A.6 to A.12.

In Table 2 and Table 3, we see that *Gurobi* instantly finds optimal solutions (0.1 seconds) for the group positioning sub-problems for depots DC, DK and DV. These are the easiest instances since there are at most three different series of vehicles (e.g. S06b, S50 and S59) in these depots.

In Table 4, we see *Gurobi* is not able to reach an optimal positioning after 1000 seconds for the depot DB where there are 6 different series (1000^1 means only one solution is returned after a time-limit of 1000 seconds). Compared to our software, the difference in terms of *obj2* (equation 6: minimization of occupied lanes) is significant (up to 2 lanes for the instance #8). Furthermore, not minimizing *obj1* (equation 5) for instances #23-29 indicates that a cutting vehicle series happens whereas it is possible to have inseparable blocks of vehicle series.

In Table 5, the same observation occurs for the depot DJ regarding instances #10-14 and #16-19 (7 different series): No optimal solution could be found. For instances #3, 6, 7, 22 (6 different series), the unique solution returned by *Gurobi* is the same that the one generated by our software.

Since *Gurobi* is not able to provide an optimal solution in a reasonable time

Table 2: Main results for depots DC and DK. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
DC 43 lanes - 993 m									
Instances #3-7									
Group positioning	33	2	33	849	0.1	2	33	849	
Schedules assignment									
S06b	11	0	9	0	210.7	0	9	0	1.6
S50	7	0	3	0	0.1	0	3	0	
S59	15	0	11	0	1000^{+}	0	11	0	
Instances #9-14, 16-19									
Group positioning	35	2	35	885	0.1	2	35	885	
Schedules assignment									
S06b	11	0	10	0	1000^{+}	0	10	0	2.0
S50	7	0	3	0	0.1	0	3	0	
S59	17	0	13	0	1000^{+}	0	13	0	
DK 6 lanes - 405 m									
Instances #1-29									
Group positioning	4	0	1	180	0.1	0	1	180	
Schedules assignment									0.1
S05	4	3	0	45	0.1	3	0	45	

Table 3: Main results for the depot DV. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DV 11 lanes - 589 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #9-14									
Group positioning	30	0	11	589	0.1	0	11	589	
Schedules assignment									197.8
S59	30	18	10	237	1000[1]	18	10	**265**	
Instances #20									
Group positioning	30	0	11	589	0.1	0	11	589	
Schedules assignment									189.6
S59	30	18	10	245	1000[1]	18	10	**260**	
Instances #23-29									
Group positioning	30	0	11	589	0.1	0	11	589	
Schedules assignment									191.7
S59	30	18	10	225	1000[1]	18	10	**255**	

Table 4: Main results for the depot DB. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DB 29 lanes - 2982 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #1-7, 15, 22									
Group positioning	40	5	19	2171	1000[1]	5	19	**2125**	
Schedules assignment									
S03-04	14	7	4	90	1000$^+$	7	4	90	
S03-04b	5	2	1	25	0.8	2	1	25	194.9
S05	5	1	2	20	165.3	1	2	20	
S05b	12	7	3	105	147.3	7	3	105	
S06	1	0	0	0	0.1	0	0	0	
S06b	3	2	0	-40	0.1	2	0	-40	
Instance #8									
Group positioning	40	5	20	2164	1000[1]	5	**18**	**2071**	
Schedules assignment									
S03-04	14	7	4	90	1000$^+$	7	4	90	
S03-04b	5	2	1	25	0.8	2	1	25	216.4
S05	4	2	0	20	221.7	2	0	20	
S05b	12	8	2	100	173.7	8	2	100	
S06	2	1	0	10	0.1	1	0	10	
S06b	3	1	0	-6	0.1	1	0	-6	
Instances #23-29									
Group positioning	53	6	27	2825	1000[1]	**5**	27	2825	
Schedules assignment									
S03-04	16	7	7	100	1000[1]	7	7	**105**	
S03-04b	6	0	4	0	0.1	0	4	0	24.9
S05	6	4	1	21	0.1	4	1	21	
S05b	16	9	5	130	264.2	9	5	130	
S06	7	4	2	55	54.8	4	2	55	
S06b	2	1	0	10	0.1	1	0	10	

Table 5: Main results for the depot DJ. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DJ 67 lanes - 4649 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #3, 6, 7, 22									
Group positioning	140	5	39	3054	1000^{1}	5	39	3054	
Schedules assignment									
S301	30	19	7	209	1000^{1}	**20**	**7**	**273**	
S37	10	5	1	-5	1000^{+}	5	1	-5	353.0
S38	8	4	1	70	1000^{+}	4	1	70	
S50	9	6	1	35	0.4	6	1	35	
S57	4	2	1	30	5.3	2	1	30	
S59	79	**infeasible**			1000^{1}	**54**	**19**	**678**	
Instances #10-14									
Group positioning	187	6	54	4062	1000^{1}	6	**52**	**3954**	
Schedules assignment									
S301	28	20	4	224	1000^{1}	19	5	**275**	
S35	7	4	1	-7	7.4	4	1	-7	
S37	28	18	5	188	1000^{1}	17	6	**248**	211.0
S38	9	4	2	80	1000^{+}	4	2	80	
S50	7	4	1	28	1000^{+}	4	1	28	
S57	5	3	1	40	9.9	3	1	40	
S59	103	51	10	-501	1000^{1}	**70**	**28**	**931**	
Instances #16-19									
Group positioning	189	7	54	3996	1000^{1}	**6**	**53**	4012	
Schedules assignment									
S301	28	20	4	224	1000^{1}	19	5	**275**	
S35	7	4	1	-7	7.6	4	1	-7	
S37	28	18	5	188	1000^{1}	17	6	**248**	218.3
S38	9	4	2	80	1000^{+}	4	2	80	
S50	7	4	1	36	478.1	4	1	36	
S57	5	3	1	40	9.5	3	1	40	
S59	105	29	14	464	1000^{1}	**71**	**29**	**1000**	

for large instances of the group positioning sub-problem, the schedule assignment problem was started from the group positioning found by our software.

In Table 2, we see that *Gurobi* can optimally solve the smallest 231 (out of 420) the schedules assignment sub-problems instances, especially for DC and DK series. A time written 1000^{+} seconds means that not all the best 10 solutions have been returned within 1000 seconds.

In Table 3 and Table 4, we see that *Gurobi* can not solve the schedules assignment sub-problems instances for depot DV and the instances #23-29 for the depot DB. Since *obj3* (equation 16: penalizing and rewarding departures) is not maximized, it means that there are departures that do not respect a minimal time and an ideal time.

More important, for the depot DJ, the difference in terms of *obj1* and *obj2* (equations 14 15: maximization of groupings of schedules of the same type) with our software is consequent for every sub-problems involving the S59 series. It represents more than 30 schedules that cannot be grouped. It will lead to solutions with many mixed schedule types, that are not acceptable for the company in terms of resulting parking plans. Worse, *Gurobi* was not able to generate a feasible solution within 1000 seconds for some of these instances involving the S59 series.

Generally, *Gurobi* cannot properly solve schedule assignment sub-problems in a reasonable time when the number of schedules is higher than 15.

Regarding our software, the reported execution time accounts for the group positioning and schedule assignment problems. Default parameters were set up to have an equivalent amount of time during each phase. Our software is able to produce optimal and satisfactory solutions to all instances in a reasonable time.

Solving small group positioning instances (DC, DK and DV) with *Gurobi* is faster and simpler in terms of implementations (number of lines of code). It could have been easily integrated in our application.

7. Conclusions

For the large instances that the company faces, a piece of general optimization software is not able to provide a solution in a computational time compatible with production requirements, at least when using simple mathematical models. Indeed, after one hour of computation, for a single delivery, no optimal solution is found for the group positioning problem which is the first part of the general problem to solve. This highlights the difficulty of solving this industrial problem. In contrast, the software described in this article is able to provide in one hour a solution that meets all the expectations of the company for the entire problem and for all 16 deliveries of a week.

The key to success of the proposed algorithm is a right decomposition of the problem into sub-problems that can be solved efficiently with an exact method in most cases and with heuristics in other cases. Since the solution of a sub-problem may over-constrain the next one at a further stage, several sub-optimal solutions are kept during the process, increasing the probability one of them leads to a feasible solution for subsequent stages.

The software has been used daily with satisfaction by the company for several years. The company has produced over 3000 departure schedules with the software, none of them needing a manual treatment.

In this article, algorithms for creating parking plans are presented for the general case. However, in practice, the problem is constantly evolving, and there are additional constraints that might occur in few specific situations.

For instance, the company bought new vehicles. The indoor garages are not large enough to park all vehicles for morning deliveries. So buses are parked outside the garage building, in front of the doors. These buses must be first processed to avoid blocking all the other vehicles in the depot.

Another good practice is the refinement of departure times for bus of a given schedule type. For each block of vehicles, the company operates a Z distribution of bus departures on adjacent lanes. Such a method achieved by the company ensures a smooth process for vehicles leaving the depot.

Furthermore, another new constraint is about sets of three adjacent lanes in given depots that must be released before a given hour during the morning delivery. These lanes can then be used by the staff as an angle parking for their private cars.

Finally, during the weekend, a maximal time interval should separate two departures in the same lane. This constraint prevents the last schedule to block a lane when all the other vehicles are already outside since fewer vehicles are in service.

All these issues are related in [7] that focus on the complete description of the industrial problem.

Acknowledgement

The funding body will be acknowledged following peer review.

[1] U. Blasum, M. R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, T. Winter, Scheduling trams in the morning, Math. Meth. of OR 49 1 (1999) 137–148.

[2] T. Winter, U. T. Zimmermann, Real-time dispatch of trams in storage yards, Annals of Operations Research 96 (1-4) (2000) 287–315. doi:10.1023/A:1018907720194.

[3] G. Gallo, F. D. Miele, Dispatching buses in parking depots, Transportation Science 35 3 (2001) 322–330.

[4] M. Hamdouni, G. Desaulniers, O. Marcotte, F. Soumis, M. van Putten, Dispatching buses in a depot using block patterns., Transportation Science 40 3 (2006) 364–377.

[5] R. Freling, R. Lentink, L. Kroon, D. Huisman, Shunting of passenger train units in a railway station., Transportation Science39 2 (2005) 261–272.

[6] L. G. Kroon, R. M. Lentink, A. Schrijver, Shunting of passenger train units: An integrated approach, Transportation Science 42 4 (2008) 436–449.

[7] Luong, Taillard, A methodology for creating parking plans in a public transportation company, Tech. rep., HEIG-VD (2019).

[8] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.

[9] J.-M. Rousseau, J.-Y. Blais, Hastus: An interactive system for buses and crew scheduling., Computer Scheduling of Public Transport–2 (1985) 45–60.

Appendix A. Supplementary results

Table A.6: Supplementary results for the depot DC. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DC 43 lanes - 993 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #1, 2, 15, 20									
Group positioning	34	2	34	867	0.1	2	34	867	
Schedules assignment									
S06b	11	0	9	0	209.9	0	9	0	1.6
S50	7	0	3	0	0.1	0	3	0	
S59	16	0	12	0	1000^{+}	0	12	0	
Instance #8									
Group positioning	34	3	34	867	0.1	3	34	867	
Schedules assignment									
S05b	3	0	2	0	9.5	0	2	0	1.6
S06b	8	0	6	0	2.4	0	6	0	
S50	7	0	3	0	0.1	0	3	0	
S59	16	0	12	0	1000^{+}	0	12	0	
Instance #21									
Group positioning	32	2	32	831	0.1	2	32	831	
Schedules assignment									
S06b	11	0	10	0	1000^{+}	0	10	0	1.2
S50	7	0	3	0	0.1	0	3	0	
S59	14	0	11	0	1000^{+}	0	11	0	
Instances #22, 29									
Group positioning	33	2	33	849	0.1	2	33	849	
Schedules assignment									
S06b	11	0	9	0	232.9	0	9	0	8.9
S50	7	0	3	0	0.1	0	3	0	
S59	15	0	11	0	1000^{+}	0	11	0	
Instance #23-25									
Group positioning	35	3	35	885	0.1	3	35	885	
Schedules assignment									
S05b	5	0	4	0	0.1	0	4	0	1.9
S06b	6	0	5	0	0.2	0	5	0	
S50	7	0	3	0	0.1	0	3	0	
S59	17	0	13	0	1000^{+}	0	13	0	

Table A.7: Supplementary results for the depot DB. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DB 29 lanes - 2982 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #9-14, 16-21									
Group positioning	53	5	28	2908	1000[1]	5	28	**2885**	
Schedules assignment									
S03-04	16	7	7	100	1000[1]	7	7	**105**	
S03-04b	6	0	4	0	0.1	0	4	0	25.2
S05	10	6	3	85	9.2	6	3	85	
S05b	17	11	4	140	1000[1]	11	4	140	
S06	3	1	1	15	0.3	1	1	15	
S06b	1	0	0	0	0.1	0	0	0	

Table A.8: Supplementary results for the depot DV. Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DV 11 lanes - 589 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #16-19, 21									
Group positioning	30	0	11	589	0.1	0	11	589	
Schedules assignment									193.6
S59	30	18	10	250	1000[1]	18	10	**260**	

Table A.9: Supplementary results for the depot DJ (1). Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DJ 67 lanes - 4649 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instance #4									
Group positioning	140	4	39	2976	1000[1]	4	39	**2958**	
Schedules assignment									
S301	27	18	6	261	1000[1]	18	6	261	
S37	13	7	2	30	1000[+]	7	2	30	299.0
S38	8	4	1	70	1000[+]	4	1	70	
S50	9	6	1	35	0.4	6	1	35	
S59	83	44	11	46	1000[1]	**56**	**22**	**653**	
Instance #5									
Group positioning	142	4	39	3034	1000[1]	4	39	3034	
Schedules assignment									
S301	27	18	6	261	1000[1]	18	6	261	
S37	13	7	2	22	1000[+]	7	2	22	295.1
S38	8	4	1	70	1000[+]	4	1	70	
S50	9	6	1	35	0.4	6	1	35	
S59	85	28	9	256	1000[1]	**59**	**21**	**710**	
Instances #8									
Group positioning	161	5	45	3460	1000[1]	5	45	3460	
Schedules assignment									
S301	25	17	5	240	1000[1]	17	5	**245**	
S37	22	14	4	126	1000[1]	14	4	**140**	321.2
S38	8	4	1	70	1000[+]	4	1	70	
S50	11	8	1	84	1000[+]	8	1	84	
S57	5	3	1	40	9.8	3	1	40	
S59	90	37	13	-144	1000[1]	**62**	**23**	**910**	

Table A.10: Supplementary results for the depot DJ (2). Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DJ 67 lanes - 4649 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #1, 2									
Group positioning	161	5	45	3460	1000^{1}	5	45	3460	
Schedules assignment									
S301	25	17	5	240	1000^{1}	17	5	**245**	
S37	22	14	4	126	1000^{1}	14	4	**140**	329.8
S38	8	4	1	70	1000^{+}	4	1	70	
S50	11	8	1	84	1000^{+}	8	1	84	
S57	5	3	1	40	9.9	3	1	40	
S59	90	29	12	463	1000^{1}	**62**	**23**	**900**	
Instance #9									
Group positioning	187	8	53	4034	1000^{1}	**7**	53	4034	
Schedules assignment									
S301	28	20	4	224	1000^{1}	19	5	**275**	
S35	7	4	1	-7	7.3	4	1	-7	
S36	1	0	0	0	0.1	0	0	0	333.3
S37	28	18	5	188	1000^{1}	17	6	**248**	
S38	8	5	2	75	1000^{+}	5	2	75	
S50	7	4	1	28	1000^{+}	4	1	28	
S57	5	3	1	40	9.6	3	1	40	
S59	103	51	10	-501	1000^{1}	**70**	**28**	**931**	
Instance #15									
Group positioning	172	5	48	3656	1000^{1}	5	48	3656	
Schedules assignment									
S301	25	17	5	240	1000^{1}	17	5	**245**	
S37	22	14	4	126	1000^{1}	14	4	**140**	323.0
S38	8	4	1	70	1000^{+}	4	1	70	
S50	12	9	1	76	10.9	9	1	76	
S57	5	3	1	40	9.5	3	1	40	
S59	100	**infeasible**			1000^{0}	**69**	**26**	**995**	

Table A.11: Supplementary results for the depot DJ (3). Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DJ 67 lanes - 4649 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instance #20									
Group positioning	187	6	54	4062	1000^{1}	6	**52**	**3954**	
Schedules assignment									
S301	28	20	4	224	1000^{1}	19	5	**275**	
S35	7	4	1	-7	7.3	4	1	-7	
S37	28	18	5	188	1000^{1}	17	6	**248**	209.1
S38	9	4	2	80	1000^{+}	4	2	80	
S50	7	4	1	36	481.4	4	1	36	
S57	5	3	1	40	9.7	3	1	40	
S59	103	51	8	-416	1000^{1}	**70**	**27**	**985**	
Instance #21									
Group positioning	181	6	51	3932	1000^{1}	6	**50**	**3838**	
Schedules assignment									
S301	28	20	4	224	1000^{1}	19	5	**275**	
S35	7	4	1	-7	7.3	4	1	-7	
S37	28	18	5	188	1000^{1}	17	6	**248**	223.1
S38	9	4	2	80	1000^{+}	4	2	80	
S50	7	3	1	36	1000^{+}	3	1	36	
S57	5	3	1	40	9.6	3	1	40	
S59	97	**infeasible**			1000^{0}	**68**	**24**	**951**	
Instances #23, 24									
Group positioning	188	5	53	3980	1000^{1}	5	53	3980	
Schedules assignment									
S301	37	23	5	205	1000^{1}	**24**	**9**	**370**	
S35	7	4	1	-7	7.3	4	1	-7	378.8
S37	28	18	5	188	1000^{1}	17	6	**248**	
S50	7	4	1	36	475.7	4	1	36	
S57	5	3	1	40	9.9	3	1	40	
S59	104	52	9	-593	1000^{1}	**70**	**29**	**980**	

Table A.12: Supplementary results for the depot DJ (4). Hierarchical objective values are reported for the group positioning problem (minimize equations 5, then 6, then 7) and for each vehicle series of the schedule assignment problem (maximize equations 14 and 15, then 16).

		Gurobi				Our software			
DJ 67 lanes - 4649 m	#sched	obj1	obj2	obj3	time [s]	obj1	obj2	obj3	time [s]
Instances #25-29									
Group positioning	188	6	54	3996	1000[1]	6	**53**	**3980**	
Schedules assignment									
S301	28	20	4	224	1000[1]	19	5	**275**	
S35	7	4	1	-7	7.7	4	1	-7	
S37	28	18	5	188	1000[1]	17	6	**248**	208.9
S38	9	4	2	80	1000[+]	4	2	80	
S50	7	4	1	36	491.4	4	1	36	
S57	5	3	1	40	10.0	3	1	40	
S59	104	52	9	-593	1000[1]	**70**	**29**	**980**	