Scheduling a large fleet of rental vehicles using multi-neighbourhood local search

Tommaso Urli[1,2], Philip Kilby[1,2]

CSIRO Data61[1] & The Australian National University[2]
Tower A Level 3, 7 London Circuit, Canberra ACT 2601
tommaso.urli—philip.kilby@data61.csiro.au

Abstract

In this paper we investigate a multi-neighbourhood local search approach based on simulated annealing to solve the rental fleet scheduling problem (RVSP). This problem arises in the context of the rental of recreational vehicles (RVs). The problem consists of scheduling a set of bookings and other collateral activities on a heterogeneous fleet of recreational vehicles, so that the overall operational costs are minimised. The problem has a number of complicating features, including the ability to relocate vehicles between (sometimes distant) locations, the ability to use different vehicle types for a booking, and the necessity for some specific vehicles to be present at a given location and time for maintenance or disposal activities. This investigation is motivated by an increase in the volume of operations of one of our industrial partners, whose existing scheduling solution cannot handle adequately, and which prompts for a more scalable approach to vehicle scheduling.

1 Introduction

The *rental fleet scheduling problem* (RVSP [7]) arises in the context of the rental of recreational vehicles (e.g., camper-vans). Customers book a *product*, e.g., a four-berth camper-van, requesting departure at a given time from one of many depots owned by the company, and returning at a different time to a potentially different depot. The rentals must be carried out using a heterogeneous fleet of vehicles which are not tied to a specific depot, but are allowed to migrate. While each product has a *natural* mapping to a vehicle type, e.g., the booking of a four-berth product is preferentially satisfied by a four-berth vehicle, different vehicle types can be used, e.g., a six-berth vehicle may be assigned to a four-berth product. If such a *substitution* is needed, the company (not the customer) incurs an additional cost, which accounts both for the revenue loss and for the potential user dissatisfaction. In addition, *relocations* may be employed to effectively increase the fleet capacity and accommodate more bookings. That is, if there are insufficient vehicles at a location to satisfy the accepted bookings, and it makes economical sense to do so, vehicles can be moved across locations to meet the demand.

The RVSP is a challenging combinatorial optimisation problem, whose hardness has been often pointed out in literature. The following features are responsible for the size of the search space, which makes finding good quality solution very hard

- Many bookings start and end at different locations, so the available fleet is quite mobile.
- In general, for a given product, there might several valid *substitution schemes* differing in terms of what type of vehicle is used in place of the requested one, time needed to prepare the vehicle, and costs. The most natural vehicle type will usually have a substitution cost of zero, while other vehicle types will have non-zero substitution costs (if they are compatible).
- Relocations also have a cost, and several *relocation schemes* are available for the same set of locations and vehicles. Such schemes differ in terms of duration of the relocation and cost. For instance, a relocation between a given pair of locations may be achieved in a number of ways
  - two drivers can take turns to drive, relocating the vehicle in the shortest possible time, but at the highest cost,
  - a single driver can be employed, which will take longer due to the rest periods that the driver is required to observe, but will have a lower cost than the two-driver option, or

  - (for long relocations) a *backpacker* relocation may also be available: in this case, a backpacker is given the opportunity to take the vehicle to its desired destination. The time allowed is longer than a single-driver relocation, but at substantially reduced cost (as the backpacker pays for the fuel and the potential tolls).

- Vehicle *maintenance* requests are entered into the system, that require a specific vehicle to be available at a given location at a given time in order to perform required maintenance activities.
- Vehicle *disposal* requests may also be present. Here, a given vehicle must be available at a given time and location to be taken off-fleet. Sometimes the same vehicle will re-appear as a different (cheaper) vehicle type, and sometimes the vehicle will disappear from the system altogether.
- A *turnaround* cost may also be applied. When a vehicle arrives at a location, a certain time is required to prepare it for the next rental (e.g., to clean it). For instance a one-period turnaround means that the vehicle is available for rental the period after it returns from the previous rental. The turn-around cost depends on the vehicle type and product, and therefore is considered as part of the definition of a substitution scheme. For instance, a six-berth vehicle type may be made available for a "six-berth with fridge" product if it is fitted with a refrigerator during the turnaround period. Longer turn-arounds will usually have lower cost.
- While delays are discouraged, for practical reasons it might be beneficial to plan for them. For instance, a delay on the beginning of a rental can be introduced, but at a cost. In this case the customer may be given a substantial cash discount.

Vehicle maintenance and disposal requests are assumed to be mandatory, i.e., they are required to be performed at the location and time specified.

The objective of the RVSP is to find the assignment of bookings and other activities to vehicles so that the sum of operational costs, including delay costs, substitution costs, relocation costs, and lost revenue, is minimised. In this paper we propose a multi-neighbourhood local search to produce high quality solutions at the request of an industrial partner.

The paper is organised as follows. In Section 2 we briefly summarise the existing literature on the RSVP and its variants. In Section 3 we formally describe the problem as addressed in this work. Section 4 presents our approach to solve the RSVP. Section 5 presents some scalability results on synthetic instances. Finally Section 6 concludes the paper with the future directions for this project.

2 Related Work

The RVSP has been studied extensively in the past 10 years, mostly at the request of private rental companies seeking to streamline their operations and increase their revenue. In this section we briefly summarise the most important contributions our work builds upon.

In [7], the authors present the RVSP for the first time, describing how the problem was derived from their industrial partner's (Tourism Holdings Limited, or THL) requirements, and providing a first mathematical formulation of the problem, including the main cost items. The authors of [7] identify two use cases for the RVSP, reflecting the operations at THL. The first use case (*static* scheduling) consists in building a schedule from scratch, or improving an existing one, starting from a complete set of the activities (e.g., booking and maintenance) and a description of the fleet. The second use case (the *dynamic* scheduling) deals with inserting new bookings into an existing schedule or checking whether such an insertion is possible. This second use case reflects the booking procedure at THL. A customer rings the call centre and asks whether it is possible to book a certain type of vehicle for the given dates and locations. An operator queries the system to check whether the request can be accepted, and if so finalises the booking. In [6], the authors propose two different approaches to solve the RVSP, tackling both the static and the dynamic use cases. The first approach consists in mapping the RVSP onto one or more assignment components (problems), one for each vehicle type. Each assignment component aims at finding a "successor" for each activity and only includes activities compatible with the vehicle type at

hand. The cost of an assignment combines the cost items identified in [7]. A set of additional constraints enforces each booking to be only satisfied in one of the assignment components. A bespoke successive shortest path heuristic [4] is used to find cycles with a negative cost, corresponding to improvements to the schedule. The heuristic is an anytime algorithm, in that once a cycle of negative cost is found, the heuristic proceeds to find the next ones. Because of this, the approach is particularly suited for the dynamic use case, which requires real-time performance. The second approach consists in mapping the RVSP on one or more network-flow problems, again one for each vehicle type. Each network-flow problem considers a time-layered network with a node for each location and a time index for each time step in the horizon. Activities are represented by arcs of bounded capacity. In [5] the authors present an evolution of the assignment model for the RVSP, and present a Lagrangean relaxation-based heuristic (RFSP) to solve it. The goal of the work is to "handle large real-life instances of the RVSP in less than one hour of computational time". Notably, the authors explore parallelisation techniques, which appear to yield significant results on hardware with up to 8 cores. More recently, [12] proposed another variation of the assignment model from [6]. The notable feature in this work is that the nodes of the assignment components are aggregated by similarity. The advantage of such formulation is of course model compactness (the nodes are "converted" to domain values that are dealt with numerically, and therefore the number of decision variables is reduced). The usual set of linking constraints forces a booking to only appear in one of the assignment components. The proposed approach consists in relaxing the linking constraints, thus obtaining a number of independent maximum flow problems. Since the coefficient matrix of the resulting problem is unimodular, the LP relaxation yields a number of integer solutions. An iterative procedure is then used to satisfy the linking constraints.

All of the approaches proposed so far in literature, solve the RVSP by

1. modelling the decisions at the vehicle type level, rather than the vehicle level, and then using a post-procedure to extract the vehicle schedules and generate a solution to the problem,
2. stripping some of the problem details, e.g., the disposal of specific vehicles, substitutions, etc., from the model, and dealing with them in a complex and heuristic post-processing stage, and
3. decomposing the problem in a number of independent components that must be made consistent in a successive step.

Many of the reasons behind these choices are driven by the fact that solving MIP problems is computationally hard, and therefore simplifying them is a sensible strategy to obtain good quality solutions in a reasonable amount of time. However, our industrial partners have stated that the MIP-based approach is not able to consistently produce good solutions with a growing number of bookings, and particularly with large numbers of fixed assignments.

In this work, we adopt a completely different strategy. We model the RVSP directly as a stochastic local search (SLS, [9]) problem, retaining all of the original details of the formulation, e.g., substitutions, relocations, disposals, and maintenance requests. Solving the complete problem directly, instead of relying on simplification, decomposition, and post-processing, gives us the confidence that we are considering as many ways of minimising the cost as possible. At the same time, using multiple neighbourhood relations decreases the probability of getting trapped in a local minimum [3, 8]. To explore the search space we use a combination of simulated annealing (SA, [11]) and stochastic hill climbing (HC). Simulated annealing has been applied successfully to a range of combinatorial optimisation problems, including circuit design, vehicle routing, scheduling [13], and time-tabling [1]. Hill climbing is used as a final refinement procedure to handle situations in which simulated annealing terminates before converging. One of the advantages of local search-based meta-heuristics is that they are extremely parsimonious in terms of memory requirements. This is a an important condition for obtaining the level of scalability needed to find good quality solutions for large-scale problems (more than ten thousand bookings, and more than one thousand vehicles), an issue which has been often raised in the context of RVSP [5]. Moreover, while in this work we target the static case of the RVSP, we believe local search to be a promising method to deal with schedule repair, and hence with the dynamic case as well.

3 Problem definition

In this work, we consider the problem formulation presented in [7], barring minor details that are described below. It should be noted that the formulation has since then evolved to include a variety of cost components and requirements of our client. Not all of these changes have been considered in this work, but are the subject of future iterations of the project.

A problem instance of the RVSP defines what vehicles and rental locations are available, and what activities have to be scheduled. The bulk of the activities will normally consist of bookings, i.e., requests to pick up a *product* at a certain location and date, and return it at a (possibly) different location and date. A product represents one or more *preferred* vehicle types and one or more *compatible* vehicle types. A request can be feasibly carried out using any vehicle of a preferred vehicle type without additional cost, or using a vehicle of a compatible vehicle type, but incurring a cost which depends both on the vehicle being used instead and on the time needed to get it ready. Replacing a vehicle with a compatible one is called a *substitution* and is one of the two principal sources of cost. The other main source of cost is a *relocation*. A relocation is needed when a vehicle needs to begin an activity at a location different from the end location of its previous activity. There are different ways to relocate a vehicle, each one providing a different trade-off between cost and duration. In addition to bookings, that can be scheduled on a variety of vehicles, there are *fixed* activities that have necessarily to be scheduled on a specific vehicle. Among these are a nominal *start of service* activity represents the current location of the vehicle, one or more maintenance activities, and (optionally) a *disposal* activity, representing where and when the vehicle will be decommissioned[1]. Necessarily, no further activities can be scheduled after a disposal. Table 1 summarises the entities used in the following formal description of the problem. For each activity $a \in B \cup F$ we denote by $sl_a, el_a \in L$ the locations at which the activity has to start

Symbol	Meaning
L	Set of locations where vehicles can be picked up and returned
V	Set of vehicles in the fleet
T	Set of vehicle types available
R	Set of relocation schemes available
S	Set of substitution schemes available
V_t	Set of vehicles of type $t \in T$
P	Set of products available
B	Set of bookings to schedule
F	Set of activities to be scheduled on a specific vehicle
F_v	Set of fixed activities for vehicle $v \in V$

Table 1: Entities that constitute a problem instance.

and end, respectively. Moreover, we denote by $st_a, et_a \in \mathbb{N}$ the start and the end time of the activity, respectively. In practice, each day in the planning horizon, i.e., one year, is split in two periods, morning and afternoon, and each time index corresponds exactly to a day and a period. For each booking $b \in B$, we denote by v_b the *value* of b, i.e., the revenue gained by successfully scheduling it with one of the vehicles. We denote by a_v^i the i-th activity scheduled on vehicle v. Activities in F can be either bookings for a specific vehicle, or maintenance or disposal requests that have to be scheduled. In practice, if any of these activities are not scheduled successfully, the lost revenue for a fixed booking is handled in the same way as for a normal booking, and the penalty for not scheduling a maintenance is a value v_m that is provided as a parameter to the solver. A substitution scheme $s \in S$ is defined by a vehicle type $t_s \in T$, a product type $p_s \in P$, a fixed cost c_s, a turnaround time tt_s and a turnaround cost tc_s[2]. Moreover, a substitution has a feasibility window $[st_s, et_s]$[2] within which the substitution process needs to start.

[1] Disposal activities were not part of the original formulation of the problem, but were introduced as a successive extension.
[2] This constitutes an extension of the formulation in [7].

Note that there might be multiple substitution schemes for the same vehicle type and product pair, but different trade-offs between turnaround cost and time. The scheme is assumed to include zero-time, zero-cost "substitutions" for the preferred vehicle type(s). We denote by s_a the substitution scheme selected for activity a. A relocation scheme $r \in R$ is composed by a start and and end location $sl_r, el_r \in L$, a duration d_r in days, a cost c_r and, optionally, a vehicle type t_r. If no vehicle type is specified, then the relocation scheme can be used for any vehicle. As for the substitution schemes, each relocation scheme has a feasibility window $[st_r, et_r]^2$ within which it has to start. Again, there might be different relocation schemes for the same locations and vehicle types, but different trade-offs between cost and duration. For completeness, R includes zero-cost, zero-time "relocations" from every location to itself. In this way, a relocation cost is available for every legal end location/start location pair. We denote by r_a the substitution scheme selected for activity a. In some situations it might make sense to allow for small delays in the starting time of an activity. An activity $a \in B \cup F$ with a delay $0 \leq d_a \leq D$, where D is typically equal to one day, will generate a cost proportional to the delay and dependent on the vehicle type being used. Each vehicle type $t \in T$ has a unit cost dc_t for each unit of delay. The delay $d_{a_v^i}$ on the commencement of a_v^i is computed in accordance to the formulation in [7]

$$d_{a_v^i} = \max\left(st'_{a_v^i} - st_{a_v^i}, 0\right), \text{where} \tag{1}$$

$$st'_{a_v^i} = et_{a_v^{i-1}} + tt_{s_{a_v^i}} + d_{r_{a_v^i}} \tag{2}$$

and the cost of such delay is $d_{a_v^i} \times dc_t$, where t is the type of the vehicle v where the delayed activity a is scheduled. For each vehicle $v \in V$ we denote by $A_v \subseteq B \cup F$ the set of activities assigned to it. A solution of the problem, or *schedule* consists of a set of assigned activities $A = \bigcup_{v \in V} A_v$ and a set of unassigned bookings $U = B \setminus A$ (all fixed activities must be assigned).

3.1 Constraints

There are four hard constraints that a schedule has to honour according to [7]. These are

- **H1** each vehicle must perform all of the *fixed* activities assigned to it,
- **H2** each activity can be scheduled at most on one vehicle,
- **H3** the activities assigned to each vehicle must be temporally ordered, and
- **H4** there is an upper limit on any delay in commencing an activity.

Even if not explicitly stated in [7], in practice not scheduling an already booked rental can have impact that goes beyond the lost revenue, e.g., loss of reputation. Therefore, we also consider an additional hard constraint which can be turned on or off depending on the situation and which, if active, enforces the scheduling of all the activities

- **H5** each activity must be scheduled on some vehicle.

3.2 Costs

The quality of a schedule is evaluated according to the following cost items

- **LR** the lost revenue from rentals that are not included in the schedule, measured as the sum of revenues v_b from unscheduled bookings $b \in B$,
- **RC** the relocation costs, computed as the sum of c_r for all of the activities needing a relocation $r \in R$,
- **SC** the substitution costs, computed as the sum of the c_s and tc_s $s \in S$, and
- **DC** the costs due to delays on the commencement of an activity, computed as the sum of $dc_t \cdot d_a$ for all activities $a \in B \cup F$ whose start time delay is $0 \leq d_a \leq D$.

4 Our approach

We model the RVSP and solve it using stochastic multi-neighbourhood local search. In this section, we first describe our model, then we discuss the search strategy.

4.1 Model

We describe our model in terms of the solution encoding, the invariants we enforce, and the objective.

4.1.1 Solution encoding

Given a problem instance, a solution is represented by as a list of schedules, one for each vehicle $v \in V$. Schedules are ordered lists of *activities*, where each activity is defined by a start time, and end time, a start location, and an end location. Activities encode different entities of the problem. Booking request and maintenance requests are mapped directly on schedule activities. Additionally, for each vehicle in the fleet, we consider two nominal activities, a *start of service* activity, and an *end of service* activity. These activities cannot be scheduled; instead they are pinned to the start and end of each vehicle's schedule and cannot be moved during the search. If a vehicle becomes available after the beginning of the scheduling horizon, i.e., because of an acquisition, then the end time and location of the start of service activity reflect those of the acquisition. Similarly, if a disposal request is issued for a particular vehicle, the start time and location of the end of service activity reflect those of the disposal request. If a vehicle is not involved in an acquisition, its start of service activity reflects the beginning of the scheduling horizon and the last position of the vehicle. If a vehicle does not need to be disposed, then the end of service activity is set to begin at the end of the scheduling horizon and at a wildcard *anywhere* location. Additionally to schedules, a solution contains mapping from each activity to a relocation and a substitution scheme. To handle situations in which an activity does not require a substitution or a relocation, or in which a substitution or a relocation are required but not available, we use special *null* and *forbidden* substitutions and relocation schemes. The activities that cannot be feasibly scheduled by the algorithm are automatically assigned to a special *dummy vehicle*, which is also used as a buffer for temporarily unscheduled activities during search. Activities in this schedule do not need relocations, substitutions, and their delay is not penalised.

4.1.2 Invariants

By limiting the changes that can be made to a schedule once a solution is initialised, we maintain a number of invariants, which allows us to automatically guarantee the satisfaction of some of the constraints in Section 3.1

- all activities are always part of a schedule,
- start end end of service activities are pinned at the start and end of the respective vehicle's schedule,
- maintenance activities and bookings are always assigned to their respective vehicle,
- activities have an associated (non *forbidden*) relocation scheme,
- bookings have an associated (non *forbidden*) substitution scheme,
- activities are always in temporal order, except for the ones in the dummy schedule.

Some invariants can be violated after initialisation but once satisfied they cannot be violated again.

4.1.3 Objective function

The objective function consists of a linear combination of the costs presented in Section 3.2, the number of violations of the hard constraints **H4** and **H5**[3], and the number of forbidden substitutions and reloca-

[3]Constraints **H1-3** are automatically guaranteed by the above invariants.

tions in the schedule. Note that the algorithm is not allowed, by construction of the local search moves, to introduce violations to any of the constraints **H1-4** or forbidden substitutions and relocations.

Specifically, let n_{FR} be the number of activities with forbidden relocations, n_{FS} the number of forbidden substitutions, n_D the number of activities with illegal delays (representing the degree of violation of **H4**), and n_U the number of unscheduled activities (representing the degree of violation of **H5**). Then the objective of our problem is

$$\textbf{minimise } (LR + RC + SC + DC) + (n_{FR} + n_{FS} + n_D + n_U) \cdot w_H \tag{3}$$

where w_H is a weighting factor for the violations to hard constraints, which we set to 10^6.

4.2 Search strategy

Our search strategy is based on a combination of simulated annealing as a primary search strategy, and stochastic hill climbing as a final refinement step. In particular, we adopt the simulated annealing with *cutoffs* described in [10] and which has been used successfully to solve other hard combinatorial optimisation problems, e.g., [1]. Simulated annealing starts with an initial temperature t_{max} which is slowly decreased by multiplying it by a cooling rate cr until it reaches a final temperature t_{min}. In the simulated annealing with cutoffs, the temperature is decreased when a certain number of neighbours n_s have been sampled *or* when a certain number of neighbours $n_a < n_s$ have been *accepted*, whichever happens first. Depending on the cost landscape, this has the potential to save time at the beginning of the search to exploit it later when the algorithm settles on lower cost solutions. Similarly to [1], in our approach we control the number of neighbours that have to be accepted using a parameter $\rho = n_a/n_s$ of the solver. Since the simulated annealing phase is run with a timeout, the final refinement step helps the algorithm to converge if the simulated annealing has not reached t_{min}. In practice, we control the fraction of time budget used for hill climbing with a parameter $t_{HC} \in [0, 1]$ of the solver.

4.2.1 Initialisation

Initialising the solver with a good quality solution is fundamental to obtain good performance, especially when operating on a time budget. The idea behind our initialisation strategy is to schedule as many activities as possible, so that the initial penalty for the hard constraint **H5**, which in our experience is the harder to satisfy, is small. Since start and end of service activities are automatically pinned to their vehicle's schedule, the initialisation only needs to deal with maintenance and rental requests. In the considered formulation, maintenance requests are mandatory, and therefore we can safely schedule each maintenance activity on its vehicle and only consider the rental requests. In order to fit as many activities as possible, it makes sense to have a compact schedule. To achieve this, we consider the *transition time* of an activity as the sum of substitution and relocation time, and its *slack time* as the time "wasted" between the activity an its predecessor. The slack time can be computed as the time left to the start of the activity after the substitution and relocation activities have been executed. To *greedily* build a schedule that has a small total sum of transition and slack time, we first sort the activities temporally (considering start time and end time in a lexicographic way). Then, starting from the first unscheduled activity, we place each activity in the position that minimises the sum of transition and slack time and is legal *wrt.* time, available substitution schemes, and available relocation schemes. Activities that cannot be scheduled are placed in the dummy schedule.

4.2.2 Neighbourhood relations

Both simulated annealing and hill climbing use the same neighbourhood relations. In order to provide multiple orthogonal ways to modify a solution, we define three neighbourhoods (local search moves)

> **Reassign**$(v_1, p_1, r_{os}, v_2, p_2, s, r, s_{ns}, r_{ns})$ this move reassigns the activity at position p_1 of vehicle v_1 at position p_2 of vehicle p_2, moving the activity current at the position to position p_{2+1}. The

moved activity is associated with new a compatible substitution and relocation schemes $s \in S$ and $r \in R$. Since, in principle, the move could break the substitution and relocation schemes feasibility for the *old successor* and the *new successor* of the moved activity, the parameters r_{os}, s_{ns}, and r_{ns} specify new compatible ones. After the move is executed, any affected activity has compatible substitution and relocation schemes.

Swap$(v_1, p_1, v_2, p_2, s_1, r_1, s_2, r_2, s_{os1}, r_{os1}, s_{os2}, r_{os2})$ this move swaps the activity at position p_1 on vehicle v_1 with the activity at position p_2 on vehicle v_2. The swapped activities are assigned new substitution (s_1 and s_2) and relocation (r_1 and r_2) schemes, and so are their former and new successors. As for the *Reassign* move, after the move, every affected activity has compatible substitution and relocation schemes.

ChangeMode(v, p, s, r) this move changes the substitution and relocation scheme of the activity at position p on vehicle v with $s \in S$ and $r \in R$ respectively. No other activity is affected.

Illegal moves. Not all of the moves are legal. In practice, when we generate a move from one of the neighbourhoods, we skip any move that would introduce violations of the hard constraints. For instance, if changing the relocation scheme of an activity would yield a forbidden delay, that move is skipped.

Biased moves. In order to focus the search towards the elimination of hard constraint violations, we also consider two additional "biased" neighbourhood relations: *Reassign*$^+$ and *Swap*$^+$. The difference from their unbiased counterparts is that only the activities that cause at least one hard constraint violation are considered (in the case of *Swap*$^+$) at least one of the two moves needs to be causing a violation). When a *Reassign* or *Swap* relation is chosen, the biased version of the neighbourhood is chosen with probability p_{bias}, which is a parameter of the solver, if the probability test fails, the normal neighbourhood is chosen to generate the move.

5 Experimental analysis

We tested our solver both on synthetic and real-world instances. As we are not allowed to disclose the real-world data, the results presented here are only based on the synthetic instances, which we make publicly available along with the instance generator used to produce them[4].

5.1 Parameter tuning

The parameters of our solver (see Table 2) were tuned using *F-Race* [2]. For the tuning process, we generated 50 different instances with 1500 vehicles and 7500 rental requests each, with each vehicle having a probability of 0.2 to have a maintenance request and the same probability of having a disposal request assigned. The initial parameters ranges were selected based on previous experiments with the solver and with SA in general. The best parameter settings found are reported in Table 2.

5.1.1 Validation

Our goal is to build a local search solver which compares favourably with the exiting solution deployed by our client. In this sense, a comparison between the two approaches would be in order. In practice, however, such a comparison is non trivial, as the model (and particularly the objective function) used by the existing solution differs from ours, and some details of it are undocumented. We plan on conducting such comparison in the future, but decided to focus on testing the scalability of our approach in the present work. We therefore considered different instance sizes (number of vehicles $n_v \in \{100, 300,$

[4]Code and instances are available at `https://gitlab.com/hauthor/rvsp-instances`.

Parameter	Meaning	Best value
p_{bias}	Probability of choosing a biased neighbourhood for *Reassign* and *Swap*	0.75
t_{max}	Initial temperature in SA	10^7
t_{min}	Final temperature in SA	0.1
cr	Cooling rate (temperature decrease multiplier) in SA	0.99
n_s	Neighbours sampled before reducing the temperature in SA	10^6
ρ	$n_s \cdot \rho =$ neighbours accepted before reducing the temperature in SA	0.1
t_{HC}	Proportion of time budget dedicated to the final refinement step	0.1

Table 2: Parameters of our solver.

$500, 700, 900, 1100, 1300, 1500, 1700\}$ and number of rentals $n_r = 3.9 \cdot n_v$), and generated 50 random instance for each size. In order to identify interesting instances among the generated ones, we ran the tuned solver 10 times for each instance using a timeout of 30 minutes, and we identified for each instance size the single instance maximising the gap between the minimum and the maximum cost reached. We used this value as a measure of instance hardness or "interestingness".

Table 3 reports our average results on 10 runs of 30 minutes on the 9 selected instances using the parameters found through the tuning. The left half of the table reports the instance information for comparison purposes. The right half of the table reports the median objective (revenue minus the operating costs), the median number of unscheduled activities (which constitute violations), the median number of other violations (e.g., excessive delays, or forbidden relocations), and the median slack per vehicle, i.e., the number of days each vehicle was idle on average (over one year). While the number of unscheduled

#	n_v	n_r	Seed	Objective	Unscheduled	Other violations	Slack
1	100	390	24	429,549.75	62.50	0	0.1
2	300	1170	8	1,737,004.00	37.50	0	0.1
3	500	1950	8	3,188,447.50	62.00	0	0.1
4	700	2730	28	4,762,141.25	76.50	0	0.2
5	900	3510	8	6,031,555.00	118.50	0	0.1
6	1100	4290	28	7,728,633.50	159.00	0	0.2
7	1300	5070	28	9,189,134.25	182.50	0	0.2
8	1500	5850	28	10,507,894.50	207.50	1	0.2
9	1700	6630	3	11,442,380.00	300.50	0	0.2

Table 3: Average results of 10 repetitions of 30-minutes on the validation instances.

activities seems high, in all cases it represents less than 2% of the activities. It should be noted that this is likely an artefact of the synthetic instances, which have been generated using a rather simple procedure. The solver, tuned on the synthetic instances, obtained fewer unscheduled activities on the real-world instances, e.g., less than 5, in scenarios with features comparable to the ones of instance #9.

6 Conclusions and future work

We developed a first multi-neighbourhood local search approach for the RVSP. Our goal was to build a single optimisation strategy to optimise the full problem formulation without recurring to simplifications and post-processing, as opposed to most approaches in literature. Our choice of local search is driven by the typical degree of scalability obtained by this class of approaches, which a requirement to solve problems of the size that our industrial partner faces.

The first results obtained using our solver are encouraging, but there is a lot of room for improvement. As the next steps of this project we plan to: *i)* explore techniques to parallelise our search algorithm,

ii) move to a meta-heuristic that is suitable for long optimisation runs, e.g., simulated annealing with reheats or iterated local search (ILS), *iii)* consider smarter approaches for the choice of the neighbourhood at each step, e.g., variable neighbourhood search (VNS), and *iv)* explore feature-based parameter tuning techniques, that would allow us to perform better on a wider range of problem instances, by choosing the most suitable set of parameters based on the problem at hand.

As a part of our contribution, we also open-sourced our instance generator and the full set of benchmark instances, so as to promote the research on this interesting combinatorial optimisation problem.

References

[1] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83 – 92, 2016.

[2] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[3] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 262–275. Springer, 2002.

[4] Michael Engquist. A successive shortest path algorithm for the assignment problem. *INFOR: Information Systems and Operational Research*, 20(4):370–384, 1982.

[5] Andreas T. Ernst, Elena O. Gavriliouk, and Leorey Marquez. An efficient lagrangean heuristic for rental vehicle scheduling. *Computers & Operations Research*, 38(1):216–226, 2011.

[6] Andreas T. Ernst, M. Horn, P. Kilby, and M. Krishnamoorthy. Dynamic scheduling of recreational rental vehicles with revenue management extensions. *Journal of the Operational Research Society*, 61(7):1133–1143, 2010.

[7] Andreas T. Ernst, Mark Horn, Mohan Krishnamoorthy, Philip Kilby, Phil Degenhardt, and Michael Moran. Static and dynamic order scheduling for recreational rental vehicles at tourism holdings limited. *Interfaces*, 37(4):334–341, 2007.

[8] Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[9] Holger Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[10] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; Part I, Graph partitioning. *Operations research*, 37(6):865–892, 1989.

[11] Scott Kirkpatrick, C. Daniel Gelatt, Mario P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[12] Surhuta Kulkarni, Andreas T. Ernst, A. Ranade, and Mohan Krishnamoorthy. A linear programming based iterative heuristic for the recreational vehicle scheduling problem. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 784–788, 2016.

[13] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143–1160, 2006.