A Capacited Vehicle Routing and Scheduling Problem for Passengers: A Modelling and Solution Approach

ID: 183

Abstract

Most of our daily transport requirements are made by service provider's witch must optimize their resources. These services are transportation of school children, courier services, bus tour, etc. In these services, delivery and timely are very important and they require scheduling and routing of vehicles. One of the most studied combinatorial optimization problems is the Vehicle Routing Problem (VRP) due to its directly application to many real-world cases. This paper describes a novel version of the VRP, named Capacited Vehicle Routing and Scheduling Problem for Passengers (CVRSPP). The aim of this problem is to schedule a set of buses to different services satisfying a set of constraints. This problem models a real case of the actual discretionary transport industry for groups of passengers, in which every group can hire a bus to travel from one city to any other. The travellers have some requirements that must be satisfied by the transport company and the solution must satisfy the needs from the transport company. When all these constraints are considered, the proposed problem (CVRSPP) can be considered a Capacited with Fixed Service Time, Maximum Waiting Time and No Depots VRP problem. To this end, the formal mathematical model is proposed and two metaheuristics have been developed to solve real-life instances. The empirical results show that the proposed techniques are competitive compared with other adapted approaches for solving the VRP.

Introduction

The Vehicle Routing and Scheduling Problem (VRSP) is a well-known problem studied in the literature. The interest and relevance of the VRSP comes from its directly application in real-world environments where the problem is solved by many industries in order to provide their services or to schedule their resources to optimize the associated costs to the logistic needs (Uchoa et al. 2017). The VRP is a complex combinatorial optimization problem that can be seen as a combination of 2 problems: the Travelling Salesperson Problem (TSP) and the Bin Packing Problem (BPP) which are well known NP-hard problems (Tavares et al. 2003; Korf 2002).

The basic version of the VRP (Dantzig and Ramser 1959) consists of delivering a set of packages to a set of customers distributed on a map taking into account that all delivery vehicles start and finish their service at one single depot, minimizing the cost of the routes and the size of the required fleet. This first approach of the VRP makes some assumptions in order to simplify the problem, such as: having one single depot, a homogeneous fleet of vehicles, infinite load for each vehicle, etc.

Due to high applicability of this problem to multiple contexts, these assumptions make it difficult to adapt to real life problems, so several variants of the basic VRP have been proposed. They try to adapt the formalization of the problem to the real needs of the application environment by removing the basic assumptions or by adding new ones. Some of the most studied versions of the VRP are:

- **Capacited Vehicle Routing Problem (CVRP)**(Gendreau, Laporte, and Potvin 2002a). In this problem, all vehicles have a maximum capacity that cannot be exceeded, and the whole fleet is considered to be uniform, so the maximum capacity is the same for all vehicles. A more difficult version of this approach, but also more realistic, is the Multi-Capacity VRP (MCVRP) version (Baldacci, Battarra, and Vigo 2008), in which, every single vehicle has associated a maximum-capacity, that may differ among vehicles.
- **Multiple Depot Vehicle Routing Problem (MDVRP)** (Lahyani, Coelho, and Renaud 2018). It models the real case where a delivery company may have different depots spread across the map. If the costumers are originally clustered on depots, the problem could be solved by solving multiple VRPs independently. However, when depots and customers are not related, the problem must be modeled as a MDVRP. Solving a MDVRP requires to assign each customer to a depot and then sort them in order to minimize the cost of the travel time and the size of the fleet.
- **Vehicle Routing and Scheduling Problem with Time Windows (VRPTW)**(Solomon 1987; El-Sherbeny 2010). It is an important generalization of the VRP since it can model many real-world environments. Each costumer i is associated with a time-window $[a_i, b_i]$ which defines a lower bound a_i and an upper bound b_i of time. Delivery to the customer i must be made before b_i. The vehicle can arrive to the position of the customer i before a_i but it will

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

have to wait until a_i to serve the client. In some contexts, the VRPTW has also a time window $[a_0, b_0]$ for the depot. Vehicles cannot leave the depot before a_0 and must be back before b_0.

Many techniques of different nature can be found in literature to solve VRP and its several versions. It is well-known that exact algorithms can only solve small instances of the problem (Laporte 1992) and they become unviable quickly as the problem grows. Given the intrinsic difficulty of this problem, approximation methods seem to offer the most promise for practical size problems. In (Rey et al. 2018), the authors propose a new hybrid approach based on Ant Colony Optimization (ACO) combined with Route First-Cluster Second methods and Local Search procedures to produce high quality solutions for the VRP. Furthermore, the implementation can be executed on multicore CPUs and GPUs using the computing power of modern GPUs programming technologies. It outperforms current ACO-based VRP solvers and proves to be competitive with other high performing metaheuristic solvers.

In (Wei et al. 2018), the VRP with two-dimensional loading constraints (2L-CVRP) is studied (Iori 2005). It designs a set of min-cost routes that start and finish their paths in the depot in order to serve all customers with two-dimensional rectangular weighted items. The paper proposes a Simulated Annealing algorithm with a special mechanism that allows to cooling and raising the temperature repeatedly in order to solve four different versions of 2L-CRVP. Results outperform all existing algorithms on the four versions and reach or improves the best-known solutions for most instances.

In (Yi and Bortfeldt 2018), the capacited vehicle routing problem with three-dimensional loading constraints (3L-CVRP) (Gendreau et al. 2006) is solved. Authors combine existing state-of-the-art approaches in a high-level framework that solve stepwise the problem. In the first step, a Genetic Algorithm (GA) (Moura and Oliveira 2009) is proposed for solving the container loading problem to find good placements for the packages inside the vehicles. Finally, in the second step, it is solved the routing problem by means of a hybrid algorithm which combines a Tabu Search with a Tree Search Algorithm (Bortfeldt 2012). The results show that using the proposed high-level system, the computational effort can be significantly reduced.

In this paper, a new version of the VRSP to transport groups of passengers is proposed. In the literature, the main applications of the VRSP are focused on package delivery environments. Some applications over transport of passengers by bus can be found (Bowerman, Hall, and Calamai 1995; Özkan Ünsal and Yiğit 2018; Miranda et al. 2018) but normally they are focused on optimizing regular or school routes which have a cyclic nature. In the proposed approach there is not any cyclic behavior since it is focused on discretionary routes on demand by transporting groups of passengers, instead of groups of packages as the usual VRSP. To this end, some techniques are proposed and compared in order to study how to tackle this new context for the VRP.

It is also interesting to distinguish between routing problems and scheduling problems. If the customers being serviced have no time restrictions and no precedence relationships exist, then the problem is a pure routing problem. However, if there is a specified time for the service to take place, then a scheduling problem exists. Otherwise, we are dealing with a combined routing and scheduling problem (Haksever et al. 2000).In our problem, both time and precedence constraints exists which means to deal with a combined routing and scheduling problem. In (Beck, Prosser, and Selensky 2003), the authors proposed a mapping for any VRP to get an equivalent Job Shop Scheduling problem (JSP), that consists in considering the services and vehicles from VRP as jobs and machines in JSP, respectively.

Problem proposal

In this section, a new version of the VRP is proposed. The main objective of the proposed version is to schedule a set of trips/jobs in a set of vehicles/machines trying to minimize the total traveled distance. These trips come from a real case of a company that tries to optimize the passenger transport at a national level. Passengers transport entails a set of constraints that change the nature of the pure VRP and implies to consider new features: *Capacited* with *Fixed Service Time*, *Maximum Waiting Time* and *No Depots*:

- **Capacited**: if a group of N passengers need to travel from a city to another, the assigned vehicle (machine) for this service (job) must have, at least, N available seats. As the classical version of the CVRP, the selected vehicle can exceed the needs of the service. Henceforth, it is assumed that, for each group of passengers of size K, there is always one vehicle with size greater or equal to K. Furthermore, the groups cannot be partitioned by the system to travel in multiple vehicles simultaneously.
- **Fixed Service Time** (FST): if a group of passengers travels from city A to city B, the service time is fixed and no pre-emption is allowed. Notice that this is not the conventional *Time Window*(TW) in VRPTW, since the TW is defined as a temporal slot in which a package must be delivered to a customer, but within the TW the vehicle may do different deliveries to multiple customers and the vehicle has freedom to decide when to attend the customer associated with the TW. Due to the time constraint added by this feature, the VRP is transformed into a VRSP as concluded in (Haksever et al. 2000).
- **No Depots** (ND): a group of passengers may hire a bus from every city of the network. All cities are supposed to have available buses, so the concept of *depot*, which is a requirement for delivery industry, is not required for passengers transport industry. Also notice that this is not the Multiple Depot (MD) approach from MDVRP. The main difference between MD and ND is that, in MD approaches, depots are normally supposed to be a small subset of the whole set of cities but in ND every single city can indistinctly be a depot or not and it can change its condition depending on the needs of the problem. This approach implies to consider that each vehicle has its own depot corresponding to its native city.
- **Maximum Waiting Time** (MWT): all vehicles must return to its native city from which they departed for their

first service (depot), but they can perform more services before returning to their depot. To wait in a non-native city until the next service starts is allowed but it implies extra costs (diets and accommodation for the bus driver, taxes for parking on the public road, etc.). Thus, MWT is the maximum time that the vehicles are allowed to wait between services instead of returning to their native city.

If all these features are considered, the classical VRP is transformed into the new proposed version of the problem: *Capacited Vehicle Routing and Scheduling Problem for Passengers* (CVRSPP).

Problem specification

An instance of the proposed problem is the combination of four elements: a graph $G = (V, E, C)$, representing the map, a specification of the customers demand D, a maximum waiting time MWT and a set B of available vehicles. Each element is formalized as follows:

- $G = (V, E, C)$ where
  - $V = \{v_1, v_2, \ldots, v_m\}$ is the set of vertices of the graph, each one representing a city, where $m = |V|$ is the total number of cities.
  - $E = \{(v_i, v_j) \mid i \neq j;\ v_i, v_j \in V\}$ is a set of edges of the graph. An edge between 2 cities means that it is possible to travel between them.
  - $C = \{c_{v_i,v_j} = [t_{i,j}, d_{i,j}],\ \forall (i,j) \in E\}$ is the costs of the edges in E. Notice that each edge has two different associated costs:
    - $t_{i,j}$: is the time needed for traveling from v_i to v_j.
    - $d_{i,j}$: is the distance between cities v_i and v_j.
- $D = \{d_1, d_2, \ldots, d_N\}$ is set of N requested services. Each service $d_i = [p_i, q_i, r_i, s_i]$ is composed of four parameters:
  - $p_i \in V$: is the departure city for service i.
  - $q_i \in V$: is the arrival city for service i.
  - r_i: is the departure time for service i. The service i must start at r_i in p_i and must end at $r_i + t_{p_i q_i}$ in q_i. Delays are not allowed.
  - s_i: is the size (number of passengers) of the service i.
- MWT: is the global parameter for the whole instance indicating the Maximum Waiting Time allowed between services.
- $B = \{b_1, b_2, \ldots, b_a\}$: is the set of available vehicles. A vehicle b_i can transport a maximum of c_i passengers. The number of total available buses is $a = |B|$.

  In many environments, where this problem can be applied, the set B is not consider. For example, let's suppose a travel agency that wants to hire buses for the transport of its customers to different airports from multiple cities during a large period of time. In this context, the agency can hire as many buses as it needs from multiple bus companies around the country. Henceforth this approach will be used, which, in practice, only implies to consider B as an infinity set, or at least, a large enough set to assign one different vehicle to each service.

An instantiation of the problem is a tuple $s = [x_1, x_2, \ldots, x_N] \mid x_i \in B$ where x_i is the vehicle/machine assigned to service/job i. Notice that, in the proposed problem, the departure and arrival time is fixed, so the objective is not to sort the services/jobs, but how to schedule them in vehicles/machines in order to save resources. Thus, a complete solution is an assignation of machines to all jobs. Jobs assigned to the same machine are sorted by their departure time. An instantiation s is a *solution* S if the following constraints are satisfied:

1. All services/jobs assigned to the same vehicle/machine are compatible. Two services/jobs are compatible if:
   - They do not overlap in time, and also there is enough time for traveling from the arrival city of the first service to the departure city of the second service (eq. 1).

     $$r_j > r_i + t_{p_i q_i} + t_{q_i p_j} \quad \forall i, j \in D \mid x_i = x_j \wedge r_j \geq r_i \tag{1}$$

   - The waiting time between two services is less or equal to MWT (eq. 2).

     $$\begin{gathered} r_j - (r_i + t_{p_i q_i} + t_{q_i p_j}) \leq MWT \\ \forall i, j \in D \mid x_i = x_j \wedge r_j > r_i \end{gathered} \tag{2}$$

2. The capacity of the vehicle is not exceeded (eq. 3).

   $$c_{x_i} \geq s_i \ \forall x_i \in s \tag{3}$$

The cost of a solution $S = [x_1, x_2, \ldots, x_M] \mid x_i \in B$ can be measured in terms of multiple factors, such as: the size of the fleet needed or the total unused kilometres (unused kilometres are the kilometres that the vehicles need for traveling between jobs without passengers). The size of the fleet can be defined as:

$$|F| : F = \{x_i \in S\} \tag{4}$$

To formally define unused kilometres some auxiliary definitions will be used:

- $SV_v = \{i \mid x_i \in S \wedge x_i = v\}$: is the set of services assigned to the vehicle v.
- $SB_j = \{i \in \{1, N\} \mid r_i + tp_i q_i < r_j \wedge x_i = x_j\}$: SB_j: is the set of services assigned to the same vehicle than service j but scheduled before it.
- $JP_j = \underset{i \in SB_j}{\operatorname{argmax}} \ r_i$ is the job that immediately precede job j.

Using these definitions, the total unused kilometres (UK) traveled by all the vehicles of a given solution can be formally defined as:

$$UK = \sum_{v \in F} \Big(\sum_{j \in SV_v} d_{q_{JP_j} p_j} \Big) + d_{q_{y_v} p_{z_v}} \tag{5}$$

where:

- $y_v = \underset{i\,:\,x_i \in s \,\wedge\, x_i = v}{argmax} \ r_i$: is the last service assigned to the vehicle v.
- $z_v = \underset{i\,:\,x_i \in s \,\wedge\, x_i = v}{argmin} \ r_i$: is the first service assigned to the vehicle v.

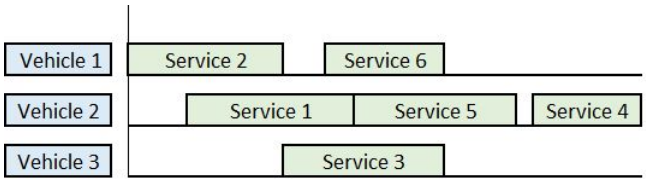

Figure 1: Solution example

- $d_{q_{y_v} p_{z_v}}$: is the distance between last visited city by vehicle v and the first one. This distance must be added to the evaluation since returning to its own depot is mandatory for every vehicle.

The objective of the search process could be minimize both factors $|F|$ and UK in a multi-objective way in order to minimize both the needed fleet and the total unused kilometres traveled by the vehicles. However, in some contexts, one of these factors could not be relevant. In this work, we will focus on minimizing UK since the real world case under investigation does not have any constraint on the number of available vehicles.

Solving Techniques

Different techniques to solve the proposed problem have been proposed and tested. In this section, two different metaheuristics for solving the proposed problem are presented.

Solution representation

All the proposed techniques will use the same representation of an instantiation. Let's suppose a problem with 6 services that can be carried out by 3 vehicles as shows figure 1. This instantiation could be expressed unambiguously in the terms described in the previous section as $s = [2, 1, 3, 2, 2, 1]$. This representation is formally useful because of its simplicity but it presents some computational problems: it requires a lot of computation cost in order to determinate whether an instantiation s is feasible or not.

In order to find a computationally affordable representation, a procedure based on *push forward* proposed in (Solomon 1987) have been developed. With this new procedure, an instantiation s takes the form of a list containing all the services IDs in any order. Feasibility of a solution is not compromised by the selected order for the services, that is, all possible permutations of the services represent a feasible solution, which also means that every possible instantiation s is also a solution S . Using the proposed procedure, the instantiation showed in figure 1 could be represented as:

$$s = [2, 6, 1, 5, 4, 3]$$

Notice that there is no separator to indicate when the next service is assigned to a new vehicle. This is because, if a separator exists, some combinations could be unfeasible. So, given a list of services IDs in any order (a solution, by definition), it is necessary to carry out a technique to build a schedule and then evaluate it. Thus, 2 different modifications of the *push forward* technique have been developed: *Light Evaluation* (LE) and *Heavy Evaluation* (HE).

Algorithm 1: Compatibility checking

input : Pair of jobs=(s1,s2)
output: True if s1 and s2 are compatible. False otherwise.

1 **Function** *compatibles(s1,s2)*:
2 condition_1 = $r_{s2} \geq r_{s1} + t_{p_{s1} q_{s1}} + t_{q_{s1} p_{s2}}$ //(eq. 1) ;
3 condition_2 = $r_j - (r_i + t_{p_i q_i} + t_{q_i p_j}) \leq MWT$ //(eq.2) ;
4 return condition_1 $\wedge$ condition_2 ;

Light Evaluation (LE) Let's suppose a solution $S = [9, 2, 1, 3, 5, 7, 4, 10, 6, 11, 8, 12]$ for an instance problem of 12 jobs (see fig. 2). LE technique divides, by a Greedy Algorithm, the whole set of services in multiple subsets, each one corresponding to a different vehicle. This Algorithm (alg. 2) checks, for each used vehicle (line 4), if the job could fit the last position in that vehicle (line 6) and add it in that position (line 7). If the job does not fit any available vehicle (line 10), a new vehicle is added and the job is introduced on it (line 11). Figure 2 clarifies the positions that LE checks for introducing a new service into the current vehicle. Shaded squares are the tested position. If a service does not fits any shaded square according to the constraints (lines 2 and 3 from alg. 1), then a new vehicle is added with the service inside itself (line 11).

Algorithm 2: LE algorithm

input : A list of services IDs: solution S
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

1 vehicles = [ ] ;
2 **for** $s \in S$ **do**
3 introduced = False ;
4 **for** $b \in vehicles$ **do**
5 last_position = Length(b) - 1 ;
6 **if** *compatibles(b[last_position], s)* **then**
7 b.append(s) ;
8 introduced = True ;
9 break;
10 **if** $\neg introduced$ **then**
11 vehicles.append([s]) ;
12 return vehicles;

Heavy Evaluation (HE) LE is a fast way to build a feasible schedule from any solution s because it only checks one single position for each available vehicle. This procedure can be modified to check all positions in the vehicle. This variation will probably find better solutions but the computational cost is higher. This alternative is presented as *Heavy Evaluation*. A comparison between these 2 procedures will be carried out in evaluation section. Algorithm 3 shows the HE procedure and figure 3 clarifies the positions that HE checks for introducing a new service into the current vehi-

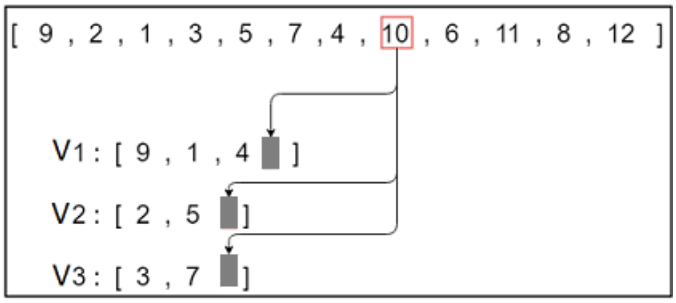

Figure 2: LE checked positions

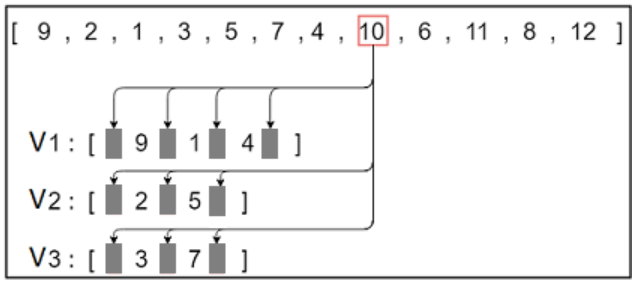

Figure 3: HE checked positions

Algorithm 3: HE algorithm

input : A list of services IDs: solution S
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  vehicles = [ ] ;
2  for s ∈ S do
3  |   introduced = False ;
4  |   for b ∈ vehicles do
5  |   |   for s1 ∈ b do
6  |   |   |   if compatibles(s1, s) then
7  |   |   |   |   b.append(s, position(s1)) ;
8  |   |   |   |   introduced = True ;
9  |   |   |   |   break;
10 |   |   if introduced then
11 |   |   |   break;
12 |   if ¬introduced then
13 |   |   vehicles.append([s]) ;
14 return vehicles;
```

cle.

GRASP

Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic commonly used to solve combinatorial optimization problems. The GRASP metaheuristic proposes a constructive phase, where a solution is found in a randomized greedy way and a local search phase where the solution is improved. The first phase is executed until a timeout is reached and the process returns the best solution found. Afterwards the second phase tries to improve the solution. Two variants of the GRASP algorithm have been implemented: GRASP with local search after the constructive process (GRASP after) and GRASP with local search during the constructive process (GRASP during). Algorithm 4 shows the GRASP procedure. Line 4 is used only in "GRASP during" version and line 8 is used only in "GRASP after" version. Evaluation in line 5 is carried out in terms of UK (see eq. 5)

GRASP constructive phase works as follows:

1. Services are introduced, sorted by their r_i (input).
2. A process looks for a subset of compatible services (lines 7-8) and schedule them in the same vehicle (line 10).

Algorithm 4: GRASP algorithm

input : A list S of sorted services and a timeout T
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  best_eval = inf;
2  while ¬timeout do
3  |   sched = constructive_phase(S);
4  |   sched = local_search(sched) //GRASP
   |     DURING;
5  |   if evaluation(sched) < best_eval then
6  |   |   best_eval = evaluation(sched);
7  |   |   best = sched;
8  best = local_search(best) //GRASP AFTER;
9  return best ;
```

Scheduled services are removed from the list (line 11) and the process continues with the remaining services (line 7). The probability of linking two services (line 9) i, j is inversely proportional to the distance that separates them:

$$1 - \frac{d_{q_i p_j}}{max\ d_{q_w p_z} \forall w, z \in [1, d]} \quad (6)$$

3. The process continues until the list of services remains empty (line 3). Algorithm 5 shows the GRASP constructive phase procedure.

Local search phase is executed after the constructive phase or after the timeout assigned to the constructive phase, depending on the GRASP version that is being executed. The algorithm that implements the local search phase is a simple process that iteratively tries to swap the position of two services, to analyze if the new solution improves the previous one, according to eq. 5. This procedure is proposed in alg. 6. Swap function in line 4 is a simple script that swaps the position in the schedule of the 2 services passed as parameters. All possible pair of jobs are checked for swapping (lines 1-2) in a O(n^2) algorithm and the GRASP process ends giving as output an improved solution or the original one.

Simulated Annealing (SA)

Simulated Annealing (Gendreau, Laporte, and Potvin 2002b) is a metaheuristic that tries to emulate the behavior of hot materials cooling down slowly until reaching regular

Algorithm 5: GRASP constructive phase

input : A list S of sorted services
output: A division of the services IDs in subsets, each one corresponding to a vehicle.

```
1  vehicles = [ ] ;
2  max_distance = max d_{q_w p_z} ∀w, z ∈ [1, d] ;
3  for s ∈ S do
4  |  last = s ;
5  |  new_vehicle = [s] ;
6  |  S.remove(s) ;
7  |  for s2 ∈ S do
8  |  |  if compatibles(last, s2) then
9  |  |  |  if random() ≤ 1 − d_{last_i s2_j} / max_distance then
10 |  |  |  |  new_vehicle.append(s2) ;
11 |  |  |  |  S.remove(s2) ;
12 |  |  |  |  last = s2 ;
13 |  vehicles.append(new_vehicle) ;
14 Function compatibles(s1,s2):
15 |  condition_0 = s1 ≠ s2 ;
16 |  condition_1 = r_{s2} ≥ r_{s1} + t_{p_{s1} q_{s1}} + t_{q_{s1} p_{s2}}
   |   //(eq. 1 ;
17 |  condition_2 =
   |   r_j − (r_i + t_{p_i q_i} + t_{q_i p_j}) ≤ MWT //eq.2 ;
18 |  return condition_0 ∧ condition_1 ∧ condition_2
```

solid structures. It is supposed that the slower the material cools, the more regular and perfect will be the solid structure reached. An iteration of the SA consists on transforming a current solution s_t into a new solution s'_t by making random minor changes on s_t. If s'_t is better than s_t, then s_{t+1} will be s'_t, otherwise s'_t is accepted as s_{t+1} with a probability that is usually decreasing as execution progresses. Formally:

$$s_{t+1} = \begin{cases} s'_t & \text{if f}(s'_t) > \text{f}(s_t) \\ s'_t \text{ with probability } p_t & \text{if f}(s'_t) \leq \text{f}(s_t) \\ s_t & \text{otherwise} \end{cases}$$

where:

- f(x) is the application of equation 5 to the solution x.
- p_t is the probability to accept a solution that worsens the current solution. This probability is normally defined as:

$$p_t = exp\left(-\frac{f(s_t) - f(s'_t)}{\theta_t}\right)$$

  where θ_t is the current time-step of the algorithm execution. This time-step allows that, as the execution progresses, it is increasingly difficult to accept solutions that worsen the current solution.

The proposed SA (alg. 7) transforms a solution s_t into s'_t by swapping the position of two randomly selected services (lines 8-14). Evaluation of a solution is carried out by using equation 5 after the LE or HE procedures (line 15). The SA needs a initial solution to start its execution. In order to build this initial solution, two different approaches have been developed:

Algorithm 6: GRASP local search phase

input : A schedule SCH = list of services separated in vehicles
output: An improved schedule, if found. Otherwise, the input schedule

```
1  for s1 ∈ vehicles do
2  |  for s2 ∈ vehicles do
3  |  |  eval1 = evaluation(SCH);
4  |  |  SCH.swap(s1,s2);
5  |  |  eval2 = evaluation(SCH);
6  |  |  if eval2<eval1 then
7  |  |  |  break;
8  |  |  else
9  |  |  |  SCH.swap(s2,s1) //undo swap
10 return SCH;
```

- The initial solution is the result of randomly shuffle all services.
- The initial solution comes from sorting all services by their departure time r_i.

Evaluation

To evaluate the proposed techniques, it is necessary to create some problem instances. Since the CVRSPP is a new version of the problem, there is no benchmark available in the literature to test different solving techniques. Thus, some small real data instances provided by the company have been analyzed to generate a new synthetic but realistic benchmark. This benchmark tries to respect the nature of the real data. An instance of the problem is composed of 2 parts: a map and a set of services to be carried out by vehicles. The generation of maps and services are independent processes. In fact different instances may have the same map or even a set of services can be tested with different maps.

A map is actually the graph $G = (V, E, C)$ defined in *problem specification* section . The map has been created following the next steps:

1. A 2-dimensional Euclidean grid of size 100x100 is created.
2. 50 points, each one representing a city, are randomly located on the grid. Each city is a vertex of the graph.
3. Vertices are located in a Euclidean space, so there exists an Euclidean distance between each pair of vertices. This also means that you can travel from every city to any other city in the map in a straight line. The edges of the graph represent these straight lines.
4. Distances between two cities $d_{i,j}$ in C are the Euclidean distances in the grid. Assuming that distances are measured in kilometres and that vehicles travel at v km/h, the times $t_{i,j}$ in C between cities are defined as:

$$t_{i,j} = \frac{d_{i,j} + random(-1, 1) * 0.25 * d_{i,j}}{v}$$

   The random component can add or subtract up to 25% of the real distance between cities (function $random(a, b)$

Algorithm 7: Simulated Annealing

input : A List S of services and initial temperature K
output: A scheduled solution

```
best_s = S ;
current_s = S ;
current_eval = evaluation(LE(current_s)) // or HE ;
best_eval = current_eval ;
iterations = 0 ;
while T>1 do
    iterations += 1 ;
    pos1 = random.int(0, length(S)) ;
    pos2 = random.int(0, length(S)) ;
    new_s = copy(current_s);
    //swap positions ;
    aux = new_s[pos1] ;
    new_s[pos1] = new_s[pos2] ;
    new_s[pos2] = aux ;
    new_eval = evaluation(LE(new_s)) // or HE ;
    if new_eval < current_eval then
        best_s = new_s ;
        current_s = new_s ;
        current_eval = new_eval ;
    else
        energy = exp(-(current_eval-new_eval)/iterations) ;
        if energy<random() then
            current_s = new_s ;
            current_eval = new_eval ;
    T = T*0.99 ;
return best_s ;
```

returns a float number ∈ [a,b]). This causes that, as happens in the real-world cases, smaller distances could need more time to be traveled than bigger ones, so, in terms of needed time, it could be cheaper to link two routes for traveling from A to B instead of using the direct route from A to B, even though more kilometres are traveled.

The set of services $D = \{[p_i, q_i, r_i, s_i] \mid \forall i \in [1, d]\}$ is also generated in a randomized way, but in this case, some features of the real-world have been emulated in order to have more realistic instances:

- p_i: In real-world instances, it has been observed that about 10% of cities on the map were chosen by 62% of the services as departure cities. This commonly occurs with important cities of a country. In order to emulate this behavior, 10% of the cities are randomly selected as *important cities* and 62% of the p_i in services are select from the set of *important cities* while the 38% of the remaining p_i are randomly selected.
- q_i: The *important cities* that are commonly selected by passengers as departure cities are also selected as arrival cities by the 32% of the services. Again, this situation is emulated by assigning 32% of the services to q_i from the set of *important cities*. Again, the remaining services are randomly assigned are arrival city.
- r_i: The time horizon for each instance is set to 15 days. In order to have a discrete quantization of the 15 days, they are measured in "number of quarter hours". Accordingly, the departure time (r_i) for each service is randomly selected in the interval [0, 1440] (1440 is the total number of quarter hours in 15 days). For example, if a service i has its $r_i = 136$ means that its departure time is 10:00 AM of the second day.
- s_i: The most commonly vehicles used for passengers transport have one of the following sizes: 30, 54, 55 or 70 seats. In real-world cases, 70% of the services use vehicles with 54 or 55 seats, so s_i is assigned respecting this percentage. Remaining 30% is randomly assigned.

Different sizes of instances were generated by modifying the parameter d (number of total services). 3 classes of instances were built, depending on the size, with 50 instances each class. Table 1 shows a description of the benchmark.

The proposed benchmark for the CVRSPP were solved with 3 different techniques: GRASP and SA from the previous section and an adaptation of the Genetic Algorithm (GA) proposed in (Baker and Ayechew 2003). The parametrization for this GA can be summarized as:

1. the representation of an individual is made according to LE and HE techniques. Both versions will be compared.
2. the size of the population is fixed to 600 individuals.
3. 300 individuals are selected to be crossed.
4. each individual has 30% of mutation probability. Mutation consists on swapping two randomly selected services.
5. crossover is carried out by 2-Point Crossover (Kora and Yadlapalli 2017).
6. fitness of a solution is calculated in terms of unused kilometres (UK) (eq. 5).
7. regarding the substitution method, all new individuals are inserted into the population are ordered by fitness and the best 600 individuals are selected.

All techniques have been executed with a 300 seconds timeout or a convergence criterion. This criterion stops the search if it performs 1000 iterations without improvement.

Table 2 shows the unused kilometres reached for each technique with all techniques and variations. The table shows the arithmetic average of the 50 instances per class. GRASP has no LE and HE versions because LE and HE are procedures to build the schedule from a representation, but GRASP has its own procedure (algorithm 5). The annealing versions tagged with "+sort" means that the initial solution for the SA was created sorting the services by their r_i while SA versions without this tag were executed initializing the solution randomly.

Class name	Size	Number of instances
I_250	250 services	50
I_500	500 services	50
I_1000	1000 services	50

Table 1: Benchmark

		Unused Kilometres (eq.5)		
Technique	Variant	I_250	I_500	I_1000
GA	LE	128695,44	268615,94	547433,06
	HE	80089,4	158643,53	304641,56
SA	LE	79879,88	170450,52	346754,82
	HE	82983,04	163806,78	309075,5
	LE+sort	65660,04	117246,74	196706,14
	HE+sort	65961,04	117.022'32	196.462'4
GRASP	After	**48763,74**	**87410,34**	**148574,12**
	During	50839,06	93497,38	159622,9

Table 2: Unused kilometres for different techniques

		Execution time (s)		
Technique	Variant	I_250	I_500	I_1000
GA	LE	306,33	315,39	343,87
	HE	356,5	523,5	647,57
SA	LE	5,23	15,38	45,38
	HE	6,12	26,07	106,668
	LE+sort	6,93	19,9	59,16
	HE+sort	68,85	229,68	836,96
GRASP	After	5,7	7,68	16,43
	During	**5,14**	**5,62**	**7,6**

Table 3: Execution time for different techniques

As shown in table 2 it is not easy to determinate which of the HE or LE procedures are the best option for the proposed problem because on each technique, they produce different results: on GA the best results come from HE but with SA the best results come from LE, depending on the instance size. It is also interesting to point out that, in all techniques, when the problem doubles its size; the evaluation function (unused kilometres) has a similar behavior since it doubles its value, approximately.

Table 3 shows the average execution time for solving each instance size. It can be observed that SA and GRASP had a better behavior that GA in all instances. It is also observed that GA exceeds the timeout (300 s.) in all its executions. This is due to the fact that GA spends all the time evaluating the initial population (the process cannot be interrupted during the initialization). When the initialization is finished, the search process should start but the timeout has been exceeded, so the process finishes its execution returning the best value in the initial population. The main problem of the GA is that the search process needs, at least, a medium-size population to find good solutions but this population need a lot of computational cost to be maintained (evaluated and checked for feasibility). Notice that SA and GRASP finished the execution by the convergence criterion without reaching the timeout. Finally, GRASP maintained the best behavior minimizing the unused kilometres and converging in low time. Particularly, depending of the location of the local search in the GRASP algorithm, the best results are obtained in unused kilometres or in execution time. In any case, a GRASP algorithm is considered a competitive metaheuristic for solving this class of problems.

Figure 4 shows the number of buses and the number of unused kilometres for solving all instances of class I_1000. It can be observed that most of the instances used between 175 and 210 buses to solve their problems. The number of unused kilometres was ranged between 142000 and 154000 kms for most instances. However there is no relationship between both parameters. Using more buses does not represent a lower amount of unused kilometres. Similar results were obtained for I_250 and I_500.

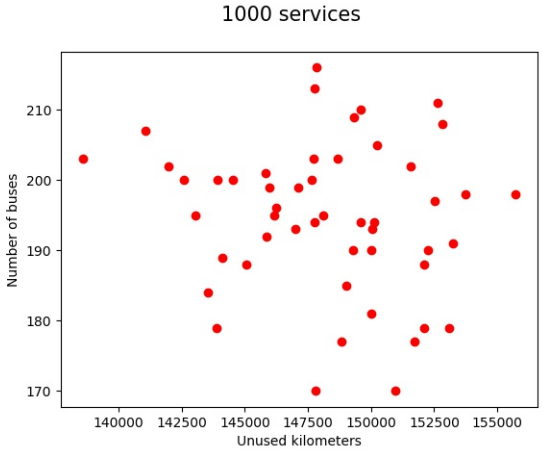

Figure 4: Unused kilometres vs Number of buses for I_1000

Conclusions and Future Work

This paper proposes a novel version of the VRP, named Capacited Vehicle Routing and Scheduling Problem for Passengers (CVRSPP). The main objective of this problem is to schedule a set of buses to different services satisfying a set of constraints. When all these constraints are considered, the proposed problem can be considered a Capacited with Fixed Service Time, Maximum Waiting Time and No Depots VRP problem. This problem models a real case of the actual discretionary transport industry for groups of passengers. The formal mathematical model has been presented and two metaheuristics have been developed to solve this problem. A benchmark for the proposed problem has been developed and it will be available at the research group webpage. The generated benchmark respects the nature of real-world cases providing realistic instances. Results shows that GRASP is a competitive technique compared with other adapted approaches for solving the VRP. It is able to save up to 30% of unused kilometers with respect the solutions obtained by experts in real life instances.

As future work it is proposed to add more constraints to the problem in order to make it more realistic such as: include a maximum driving time per driver, including some special features in some buses and services (fridge, access for the handicapped, TV, etc.). It is also a future work to tackle the dynamic rescheduling of this problem since, in real-world cases vehicles are often damaged during a service or a traffic jam delays arrival times. These situations can transform a feasible solution into a non-feasible solution and there is a need to solve these problems in a dynamic way.

References

Baker, B. M., and Ayechew, M. 2003. A genetic algorithm for the vehicle routing problem. *Computers and Operations Research* 30(5):787 – 800.

Baldacci, R.; Battarra, M.; and Vigo, D. 2008. *Routing a Heterogeneous Fleet of Vehicles*. Boston, MA: Springer US. 3–27.

Beck, J. C.; Prosser, P.; and Selensky, E. 2003. Vehicle routing and job shop scheduling: What's the difference? In *ICAPS*.

Bortfeldt, A. 2012. A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Comput. Oper. Res.* 39(9):2248–2257.

Bowerman, R.; Hall, B.; and Calamai, P. 1995. A multi-objective optimization approach to urban school bus routing: Formulation and solution method. *Transportation Research Part A: Policy and Practice* 29(2):107 – 123.

Dantzig, G. B., and Ramser, J. H. 1959. The truck dispatching problem. *Management Science* 6(1):80–91.

El-Sherbeny, N. A. 2010. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science* 22(3):123 – 131.

Gendreau, M.; Iori, M.; Laporte, G.; and Martello, S. 2006. A tabu search algorithm for a routing and container loading problem. *Transportation Science* 40(3):342–350.

Gendreau, M.; Laporte, G.; and Potvin, J.-Y. 2002a. *6. Metaheuristics for the Capacitated VRP*. 129–154.

Gendreau, M.; Laporte, G.; and Potvin, J.-Y. 2002b. *6. Metaheuristics for the Capacitated VRP*. 129–154.

Haksever, C.; Render, B.; Russell, R. S.; and Murdick, R. G. 2000. *Service Management and Operations (2nd Edition)*.

Iori, M. 2005. Metaheuristic algorithms for combinatorial optimization problems. *4OR* 3(2):163–166.

Kora, P., and Yadlapalli, P. 2017. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications* 162(10).

Korf, R. E. 2002. A new algorithm for optimal bin packing. In *Eighteenth National Conference on Artificial Intelligence*, 731–736. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Lahyani, R.; Coelho, L. C.; and Renaud, J. 2018. Alternative formulations and improved bounds for the multi-depot fleet size and mix vehicle routing problem. *OR Spectrum* 40(1):125–157.

Laporte, G. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59(3):345 – 358.

Miranda, D. M.; de Camargo, R. S.; Conceição, S. V.; Porto, M. F.; and Nunes, N. T. 2018. A multi-loading school bus routing problem. *Expert Systems with Applications* 101:228 – 242.

Moura, A., and Oliveira, J. F. 2009. An integrated approach to the vehicle routing and container loading problems. *OR Spectrum* 31(4):775–800.

Rey, A.; Prieto, M.; Gómez, J. I.; Tenllado, C.; and Hidalgo, J. I. 2018. A cpu-gpu parallel ant colony optimization solver for the vehicle routing problem. In Sim, K., and Kaufmann, P., eds., *Applications of Evolutionary Computation*, 653–667. Cham: Springer International Publishing.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* 35(2):254–265.

Tavares, J.; Pereira, F. B.; Machado, P.; and Costa, E. 2003. Crossover and diversity: A study about gvr. In *In Proceedings of the Analysis and Design of Representations and Operators (ADoRo'2003*, 27–33.

Uchoa, E.; Pecin, D.; Pessoa, A.; Poggi, M.; Vidal, T.; and Subramanian, A. 2017. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* 257(3):845 – 858.

Wei, L.; Zhang, Z.; Zhang, D.; and Leung, S. C. 2018. A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 265(3):843 – 859.

Yi, J., and Bortfeldt, A. 2018. The capacitated vehicle routing problem with three-dimensional loading constraints and split delivery—a case study. In Fink, A.; Fügenschuh, A.; and Geiger, M. J., eds., *Operations Research Proceedings 2016*, 351–356. Cham: Springer International Publishing.

Özkan Ünsal, and Yiğit, T. 2018. Using the genetic algorithm for the optimization of dynamic school bus routing problem. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience* 9(2):6–21.