ORTEC
OPTIMIZE YOUR WORLD

The VeRoLog Solver Challenge 2017

Information for participants

This document describes the problem for the VeRoLog solver challenge 2017. In addition, the formats of the instance and solution data are presented.

Problem description

The problem of the VeRoLog solver challenge 2017 is based on a routing challenge faced by a large international herd improvement company (see [GHK2015] for more details). The task at hand is to regularly measure the quality of milk samples at a number of farm locations, the customers. For this, special measuring tools are needed, and those have to be delivered to the customers at their request. After the measurement, the tools have to be picked up again. The planning of these deliveries and pickups will be the challenge for the competition.

Our planning horizon consists of a period of consecutive days, numbered 1, 2, and so on. There are different kinds of tools, each having its own size, and of each tool kind a fixed number of tools is available.

There are tool requests from customers that have to be satisfied. A *request* asks for a number of tools of one kind, that need to be present at the customer for a given number of consecutive days. The delivery of the tools has to fall within a certain time window. The tools of the request have to be picked up by one vehicle the day after the request is completed. A customer may need more than one tool kind: these are then defined as separate requests.

For example, there can be a request for 4 days for 2 tools of kind 6 to be delivered on day 1, day 2 or day 3. If it is delivered on day 2, it will stay at the customer from day 2 to day 5 (inclusive) and both tools have to be picked up by one vehicle on day 6.

There is one depot location where all tools are located at the beginning and the end of the planning horizon. A vehicle can load a tool at the depot and unload it at a customer. However, after the first day, a vehicle can also pickup a tool at one customer and deliver it to another customer without visiting the depot inbetween. The daily route of a vehicle *must* begin and end at a depot, even if there are no tools to be loaded or unloaded. At the end of a day, all tools on board of a vehicle are unloaded at the depot, and are thus available for the next day.

We have to hire vehicles to carry out the requests, we can hire any amount that we want. The available vehicles all have the same capacity (with regard to the tool sizes). During any part of a route, the total size on board of a vehicle may not exceed its capacity. As a result, when doing a pickup task, a vehicle must have enough capacity left to load the tools. When doing a delivery task, a vehicle must have the tools of the requested kind on board.

The distance that a vehicle can travel in one day is limited to a given maximum. A vehicle is allowed to return to the depot several times during a day, to unload tools and/or load (extra)

tools as long as the maximum is not exceeded. It is not allowed to exchange tools between vehicles during a day: the tools unloaded at the depot are not available for other vehicles on this day. However, a vehicle could collect two tools at a customer, and drop one at the depot to pick it up later on the day.

In order to determine the traveled distances, we provide coordinates for each customer location as well as the depot. The distance between the coordinates (x_1, y_1) and (x_2, y_2) is defined to be the floor of the Euclidean distance, i.e., $\lfloor \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \rfloor$. For your convenience, we provide the complete distance matrix for some smaller test instances. However, this should be regarded as *optional* information. For instances with, say, 1000 locations providing the full matrix is cumbersome. Hence, your solver should be able to run using only the locations' coordinates.

The main objective is to serve all requests at a minimum cost: there are costs per distance traveled, costs for using a vehicle for a day, and costs for using a vehicle at all. Additionally, there are costs associated with the tools in use. Each tool has a cost, depending on the tool kind. Per tool kind we can calculate the minimum number of tools needed to execute the routes: for each day we can calculate the daily use, i.e. the number of tools of this kind that are at the customers. The minimum number needed is the maximum daily use.

To our knowledge, this extension of the vehicle routing problem has received little attention in the literature. Gromicho et al. [GHK2015] devised a heuristic solution approach for the original problem. However, they only considered a daily planning, whereas we now consider a longer horizon. The longer horizon adds an extra dimension to the problem, namely handling the time relation between deliveries and pickups: while we do know *what* pickups have to be done, we do not know *when* they have to be done in advance.

Additional remarks

- Once the days for the deliveries of the requests are decided, finding a feasible solution to the problem is not difficult (if it exists). Per task, delivery or pickup, we can select one vehicle carrying out only this task. This leads to a feasible solution if each customer location can be visited within the maximum trip distance and the size of each tool does not exceed the vehicle capacity. Both are very reasonable assumptions and can be checked beforehand. All instances we provide will satisfy these assumptions.

- The planning horizon is added for convenience. In the provided instances a request will have a delivery interval from at earliest day 1, and at latest the penultimate day. So even the pickup of the tools of the requests will fall within the planning horizon.

- The delivery of a request can not be split: if several units are requested, all have to be delivered by one vehicle on the same day, and all tools have to be picked up together after the requested number of days. No reuse for other requests at the same address is allowed. In the provided instances there will be no time overlapping requests at the same location for the same tool kind.

- It is allowed to visit one location several times on a day, but only to pickup tools, or to deliver a request. It is not allowed to visit a location, and leave there some tools that are not requested.

Format of the instance data

We will now describe the format of the instance data, that will be available in both a text file and XML file format. We explain how it is constructed for the text file case and refer to an XML schema file that defines the structure of the XML file.

As a first note, all IDs in the input are positive integers, except the depot ID, which is 0. Furthermore, within each section, the entries are sorted by their (consecutive) IDs. This applies to both the text files and the XML files.

We do not discuss the XML files since we believe that the schema files give sufficient information.

Text file

An example of a instance file in text format follows below.

```
DATASET = VeRoLog solver challenge 2017
NAME = testInstance

DAYS = 50
CAPACITY = 6
MAX_TRIP_DISTANCE = 25000
DEPOT_COORDINATE = 0

VEHICLE_COST = 100000
VEHICLE_DAY_COST = 1000
DISTANCE_COST = 1

TOOLS = 4
1 1 20 100
2 1 20 100
3 1 20 100
4 1 20 100

COORDINATES = 7
0 10 50
1 20 10
2 50 5
3 33 7
4 40 40
5 70 40
6 1 35

REQUESTS = 6
1 1 1 10  5 1 1
2 2 5 25  3 3 1
3 3 20 30  4 4 2
4 4 25 45  5 1 3
5 5 40 45 2 2 1
6 6 15 30 7 4 1

DISTANCE
0 41 60 48 31 60 17
41 0 30 13 36 58 31
60 30 0 17 36 40 57
48 13 17 0 33 49 42
31 36 36 33 0 30 39
60 58 40 49 30 0 69
```

```
17 31 57 42 39 69 0
```

Instance text file explanation

Here we give an explanation of the data file. The different sections in the text file will always appear in this given order. However, additional line breaks or spaces may be present. Entries in a line are separated by tabs.

`DAYS` (integer) gives the number of days in the planning horizon. The days are numbered starting from 1.
`CAPACITY` (integer) denotes the capacity of one vehicle.
`MAX_TRIP_DISTANCE` (integer) is the maximum distance that one vehicle can travel on a day.
`DEPOT_COORDINATE` Gives the ID of the depot. In the supplied instances the depot is always located at coordinate 0.

The next section defines the weights for the vehicle related costs. In this particular example, the total number of needed vehicles is by far the most important objective, but note that this does not always need to be the case.

`VEHICLE_COST` (integer) The cost for a needed vehicle of the complete planning horizon. This cost should be mutiplied by the maximum number of vehicles needed on a day.
`VEHICLE_DAY_COST` (integer) The cost per route, i.e. the cost per used vehicle per day.
`DISTANCE_COST` (integer) The cost per unit of distance.

The folllowing three sections start with a name, followed by the number of entries in that section. For example, `TOOLS = 4` indicates that there are 4 tool kinds, of which their details will follow in the next 4 lines.

`TOOLS` lists the tool kinds by their IDs (the first entry). The second entry gives the size of one tool of this tool kind (integer) and the third entry is number of tools that is available. The last entry is the cost for using one tool of the tool kind (integer).

Under `COORDINATES`, the depot and customer locations are given. The first entry is the location ID: in our instances ID 0 will represent the depot location, and the next locations are customer locations. The second and third entries are the actual x and y coordinates (integers).

Under `REQUESTS`, one can find all delivery requests by the customers. The first entry is the request ID. The second entry denotes the location ID of the customer that created this request. The next two entries specify the first and last day of the time window for the delivery of this request. The fifth entry specifies the number of days that the tools of the request have to stay at the customer. The last two entries give the tool kind ID and the number of requested tools.

The `DISTANCE` section is optional: it contains redundant information and is provided in some test instances for your convenience. Note that if the distance matrix is present, it contains Euclidean distances between locations. The jth entry of the ith row of this matrix denotes the travel time from location i to location j. Because the depot has ID `0`, the matrix starts with a 0th row, containing the travel times from the depot to customer locations.

Format of the solution data

The solution file specifies the routes for each day. As with the instance files, solutions can be submitted as either text files or XML files. For the text file format, we will now explain how it should be constructed. For the XML format, an XML schema file is included as was done for the instance files.

```
DATASET = VeRoLog solver challenge 2017
NAME = testInstance

MAX_NUMBER_OF_VEHICLES = 1
NUMBER_OF_VEHICLE_DAYS = 12
TOOL_USE = 3 1 1 3
DISTANCE = 1028
COST = 113828

DAY = 1
NUMBER_OF_VEHICLES = 1
1 R 0 1 0

DAY = 5
NUMBER_OF_VEHICLES = 1
1 R 0 2 0

DAY = 6
NUMBER_OF_VEHICLES = 1
1 R 0 -1 0

DAY = 8
NUMBER_OF_VEHICLES = 1
1 R 0 -2 0

DAY = 15
NUMBER_OF_VEHICLES = 1
1 R 0 6 0

...
```

Solution text file explanation

Here we give an explanation of the data file.

The first two lines `DATASET` and `NAME` refer to the instance that the solution file belongs to.

The next section containing `MAX_NUMBER_OF_VEHICLES`, `NUMBER_OF_VEHICLE_DAYS`, `TOOL_USE`, `DISTANCE`, and `COST` is optional. It lists some characteristics of the solution, which make it easy to check the total cost of the solution. These results are also returned by the validator and hence can be checked by the user.

After these sections the solution is given, day by day. The section starting with `DAY` first lists by the header `NUMBER_OF_VEHICLES` the number of vehicles used in this day. The next lines start with an index referring to the different vehicles, and list the routes. This is explained with the example below:

```
1 R 0 2 7 -5  0
```

- '1' refers to the index of the vehicle. Since vehicles are exchangeable, we can simply number them by $1, 2, 3, \ldots$.
- 'R' refers to the route of this vehicle. In addition to the routes it is possible to add additional data, see below.

- '0' refers to a visit of the depot. Remember that a route starts and ends at the depot and this is represented explicitly here.
- '2' refers to the delivery of the request with ID 2. So vehicle 1 travels from the depot to the location of request 2, and delivers the tools of the request. Hence we assumed implicitly that the needed tools were loaded to the vehicle at the depot.
- '7' refers to request 7, which is delivered.
- '-5' refers to the pickup of the tools of the request with ID 5. Clearly one of the routes on a previous day should contain the delivery task for request 5.

Note that the loading of tools at the depot is not explicitly mentioned in the solution. It is assumed that we load exactly the tools needed for the route; it is of no use to load a tool at the depot that is not used for a request.

Extra information in the solution text file

It is possible to add extra data on the state you think the system is in. This can be very helpful for debugging your code. We explain this again by an example.

```
DAY = 10
NUMBER_OF_VEHICLES = 1
START_DEPOT = 2 1 1 3
FINISH_DEPOT = 2 1 1 3
5 R 0 1 0
5 V 1 -1 0 0 0
5 V 2 0 0 0 0
5 D 82
```

Here we give the solution of day 10 to some instance with extra lines starting with `START_DEPOT`, `FINISH_DEPOT`, `5 V` and `5 D`.

The line starting with `START_DEPOT` represents the number of tools at the depot at the end of day 9, minus all tools that will be delivered from the depot on day 10. The numbers `2 1 1 3` represent that there are 2 tools of kind 1, 1 tool of the kinds 2 and 3, and 3 tools of kind 4.

The line starting with `FINISH_DEPOT` refers to the number of tools at the depot, after all vehicles returned to the depot. The numbers `2 1 1 3` represent that there are 2 tools of kind 1, 1 tool of the kinds 2 and 3, and 3 tools of kind 4. These number are the same as in `START_DEPOT`, since no tools were returned on this day.

The line starting with `5 V 1` refers to the first visit (visit 1) of vehicle 5 to the depot. The remaining 4 numbers represent how many tools of the 4 tool kinds the vehicle delivered during this visit. Here `-1 0 0 0` tells that -1 tool of kind 1 was delivered to the depot, i.e. that 1 tool of kind 1 is loaded to the vehicle.

The line starting with `5 V 2` refers to the second visit of vehicle 5 to the depot, at the end of the day. Here `0 0 0 0` indicates that no tools were returned.

Finally `5 D 82` tells that the distance traveled by vehicle 5 was 82: the distance from the depot to location 1 is 41.

Using the validator

The validator is written in Python, which is freely available at `https://www.python.org/`. The validator can be downloaded from `https://verolog.ortec.com/`. The typical use is

```
>> SolutionCVRPTWUI.py -i instance.txt -s solution.txt
```

It will generate a cost overview of the solution represented in `solution.txt` to the instance in `instance.txt`. The validator has several options, that are explained by

```
>> SolutionCVRPTWUI.py -h
```

References

[GHK2015] Joaquim Gromicho, Sameh Haneyah, and Leendert Kok, *Solving a Real-Life VRP with Inter-Route and Intra-Route Challenges*, Available at SSRN 2610549 (2015).