UNIVERSITÄT PADERBORN

Operations Research B WS 2016/17

Project

Deadline: **22.02.2017, 23:59 Uhr**

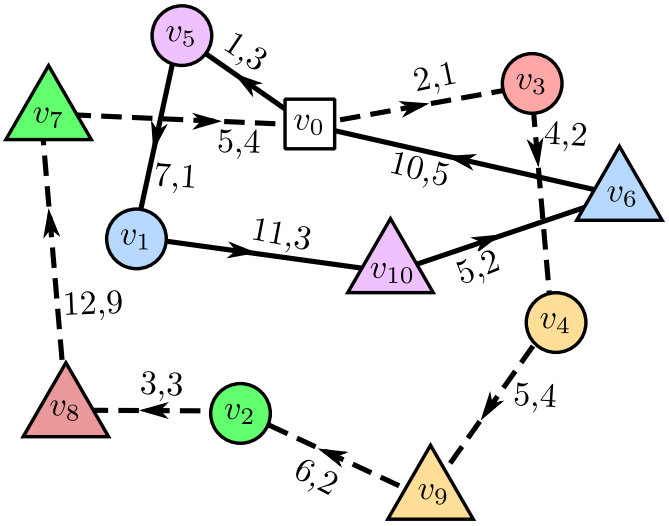
1 The dial-a-ride problem

In the static dial-a-ride problem, k cost minimal tours with k heterogeneous vehicles need to be constructed so that customers are transported between their origins and destinations under several constraints (see below). Let n be the number of customers and $G = (V, E)$ be a graph with a set of nodes $V = \{v_0, v_1, ..., v_{2n}\}$ and edges $E = \{(i, j) \mid i \neq j \text{ and } i, j \in V\}$ (G is a complete graph). The node v_0 represents the depot, the nodes $v_1 ... v_n$ are the pick-up points (origins) of each customer and $v_{n+1} ... v_{2n}$ are their destinations. This means that every customer i with origin v_i needs to get to his or her destination v_{i+n}. Each edge has transportation costs of c_{ij}^T and a travel time of t_{ij} minutes. Each customer has a time window $[e_i, l_i]$ in which they wants to be picked up and a time window $[e_{i+n}, l_{i+n}]$ for their arrival. If the vehicle arrives before a time window it must wait for the time window to begin. Each vehicle has a capacity $q_j, 1 \leq j \leq k$ (number of people). Customers have a preference for the vehicle type they travel with. If customer i is picked up by vehicle j costs of c_{ij}^P are incurred. The objective function consists of the sum of the transport costs and the customer preference costs. The parameter T describes the latest time for a vehicle to have returned to the depot.

Summary of the constraints:

- Tours start and finish at the depot. Each vehicle is assigned to exactly one tour.
- For all customers $1 \leq i \leq n$, nodes v_i and v_{i+n} must be in the same tour and v_i must be visited before v_{i+n}.
- The vehicle capacity q_j (number of passengers in the vehicle) is exceed at no point in time.
- Time windows for pick ups and drop offs must be enforced.
- All vehicles need to reach the depot before time T.
- The goal is to find the minimum cost solution in regards to transportation costs and customer preferences.

The following figure shows a sample problem instance of the dial-a-ride problem with 5 customers, 2 vehicles, one with a capacity of 2 and with a capacity of 3, and a feasible path for each vehicle. The edges are labeled with their respective travel time and transportation costs. The time windows of the customers are shown in the table below.



Time windows

Nodes (i)	1	2	3	4	5	6	7	8	9	10
e_i	5	15	0	0	0	20	30	20	10	15
l_i	12	20	10	10	5	25	35	25	15	20

Customer preferences

Customer	1	2	3	4	5
Vehicle 1 (solid)	10	5	8	2	3
Vehicle 2 (dashed)	8	3	5	10	4

Computation of the objective function:

Vehicle	Transport costs		Customer preference costs		Sum
Vehicle 1	$(3+1+3+2+5)$	$+$	$(10+3)$	$=$	27
Vehicle 2	$(1+2+4+2+3+9+4)$	$+$	$(3+5+10)$	$=$	43
Total					70

Task

Develop and implement a metaheuristic solving the aforementioned version of the dial-a-ride problem. You should describe your metaheuristic approach thoroughly in an accompanying document. You may use any metaheuristic available, or even design you own. It does not have to be a metaheuristic discussed during the course. Matheuristics are, of course, also welcome. Pure heuristic solutions and exact approaches are not allowed.

1.1 Leaderboard requirement

Every group is required to have at least one instance on the leaderboard solved by February 8th.

Bonus

Groups with solutions to all of the instances on the leaderboard will be rewarded with a bonus point. The group with the best score on the leaderboard at the time of the hand in will be rewarded with an additional bonus point.

2 Technical requirements

2.1 Execution

The code must be command line executable with the following command:

```
orbdar [maximum time] instance
```

`orbdar` is the name of the program. `[maximum time]` is an optional parameter that sets the maximum execution time **in seconds**. The program execution should not exceed the given time. If no maximum time is given the program runs until one (or no) solution is found. The parameter `instance` specifies the path to the input instance.
Please note: The program should only use one CPU core.
In other words, no threading is allowed.

2.2 Input

Your program should accept instances of the following form: (Whitespaces might be one or more blanks, tabs or newlines)

###customers	n
###max-time	T
###vehicles	k
###vehicle-capacity	$q_1\ q_2\ \cdots\ q_k$
###sym-transit-times	
$t_{0,1}\ t_{0,2}\ \cdots\ t_{0,2n}$	
$t_{1,2}\ t_{1,3}\ \cdots\ t_{1,2n}$	
$\vdots\ \ddots$	
$t_{2n-1,2n}$	
###sym-transit-costs	
$c^T_{0,1}\ c^T_{0,2}\ \cdots\ c^T_{0,2n}$	
$c^T_{1,2}\ c^T_{1,3}\ \cdots\ c^T_{1,2n}$	
$\vdots\ \ddots$	
$c^T_{2n-1,2n}$	
###time-windows	
$e_1 \dots e_{2n}$	
$l_1 \dots l_{2n}$	
###preferences	
$c^P_{1,1}\ c^P_{1,2}\ \cdots\ c^P_{1,n}$	
$\vdots\ \qquad\qquad\ \vdots$	
$c^P_{k,1}\ c^P_{k,2}\ \cdots\ c^P_{k,n}$	

All parameters are 64 bit signed **integers**. Transit times and costs are symmetric.

The input file of the sample instance looks as follows:

```
###customers 5
###max-time 50
###vehicles 2
###vehicle-capacity 2 3
###sym-transit-times
4 3 2 6 1 10 5 2 2 7
2 7 2 7 11 7 1 1 11
9 1 3 9 8 3 6 12
4 5 3 2 2 3 11
4 2 4 2 5 7
4 5 4 7 4
3 6 10 5
12 9 3
8 4
8
###sym-transit-costs
3 1 1 5 3 5 4 2 5 4
4 5 8 1 9 2 1 2 3
2 2 5 3 7 3 2 6
2 3 4 5 4 1 12
2 5 2 3 4 9
3 4 7 4 7
2 6 11 2
9 8 4
9 3
7
###time-windows
5 15 0 0 0 20 30 20 10 15
12 20 10 10 5 25 35 25 15 20
###preferences
10 5 8 2 3
8 3 5 10 4
```

2.3 Output

There are three different cases for the output of your solution procedure. Your output must conform to the following specification or it will be marked as incorrect: (Note that vehicles are 1-indexed!)

Case 1: Maximum runtime is reached without finding a solution:

```
###RESULT: Timeout.
```

Case 2: The instance is infeasible, thus there is no solution:

```
###RESULT: Infeasible.
```

Case 3: A solution exists:

```
###RESULT: Feasible.
###COST: Objective Value
###VEHICLE 1: node1 node2 ... node$m_1$
###...
###VEHICLE k: node1 node2 ... node$m_k$
```

Additionally the cpu time for the computation of the solution has to be provided in the following format:

```
###CPU-TIME: cpu-time (in seconds)
```

Note: The depot can be excluded from the path of each vehicle.

2.3.1 Example

Example output for the given sample (Section 2.2):

```
###RESULT: Feasible.
###COST: 68
###VEHICLE 1: 5 1 10 6
###VEHICLE 2: 3 4 9 2 8 7
###CPU-TIME: 4.00
```

Note: Line breaks must be strictly adhered to for your solution to be accepted. Whitespaces or tabs are treated the same (e.g. $\#\#\#COST : 68$ is the same as $\#\#\#COST : \quad 68$)

3 Miscellaneous

If the output of your program does not match the output format described in Section 2.3, you will be receive 0 points for the implementation without any exception.

3.1 Submission

The solution (implementation and accompanying document) of this assignment must be handed in before February 22, 2017 23:59 (midnight). Please create a single archive (.zip or .tar.gz only) of your source code and a PDF document describing your approach and upload it to Koala. Hand written documents are not allowed. You will also be required to give a short presentation about your project at a date after the hand in (to be determined).

Hand-in document

The document you hand in should contain at least the following components:

- How to compile and run the code
- Approach description
- Citations and discussion of relevant work in the literature
- Experimental investigation of your approach's components (**!!!**)
- Experimental investigation of the approach's overall performance (**!!!**)

Please keep the document short, concise and to the point. Do not put any actual code into the document! Doing so will result in 0 points for the document. Pseudo code is, however, allowed and encouraged. Note that the experimental investigation should include analysis and will not receive full points unless analysis is included. You can use tables, plots, etc. to present the performance of your technique.

3.2 Collaboration policy

Students are allowed to work in groups of up to 3 students. Exceptions to this will be granted if necessary. Code may not be shared between groups. Discussion of high level concepts is allowed and encouraged.

Code re-use policy

Standard libraries as well as libraries/code snippets from the internet may be used as long as they fulfill all of the following conditions:

1. They are credited in the submission document, as well as in the submitted code,
2. They do not solve the given task directly (i.e., a solver accepting a model is OK)
3. They do not carry out any metaheuristic search strategy (i.e., you must write the metaheuristic yourself),
4. They do not completely encompass all operations of a single heuristic.
5. The code license permits its use in 3rd party works.

In other words, third party code can be used as part of the instance reading, to assist in objective evaluation, or can be used within neighborhood/initial solution/termination heuristics, but cannot completely replace these components. The overall search strategy itself, however, must be completely the own work of the student.

4 Grading

Projects will be evaluated based on the following criteria:

- Quality of the initial solution heuristic and its solutions (if applicable)
- Quality and performance of the neighborhood heuristics (if applicable)
- Code quality (in terms of the metaheuristic implementation)
- Document quality (in terms of the metaheuristic implementation)
- Insight and analysis of description document and presentation
- Final solution quality