186.112	**Heuristic Optimization Techniques**	WS 2014/15
Programming Exercise 1	Institute of Computer Graphics and Algorithms Algorithms and Data Structures Group	11.11.2014

General Information

This course requires two mandatory programming exercises. These exercises are meant to be solved in teams of two. In special cases individuals or teams of three are possible, but only after consulting the lecture's organization. The group interviews for the first exercise are scheduled for Tuesday, 10.12.2014. The second interviews will be together with the oral exam on Tuesday, 20.1.2015. For both interviews please register via TISS for a time-slot. Arrangements for specific time-slots are done during the lecture or via email to `heuopt-ws14@ads.tuwien.ac.at`.

For the first exercise you will have to solve the *Time-Constrained Bipartite Vehicle Routing Problem* with various heuristics, see below. For some additional motivation we will organize our own competition for the first (and only the first) exercise. You will participate automatically by handing in your approach on time. During the lecture on Tuesday, 16.12.2014, the three best teams will be rewarded with fame & glory and some small presents. The criterion will be the best solution of the instance `tcbvrp_700.prob` presented during your interviews. We do not consider the running time your algorithm needs to find the solution. *Your tournament rank has no influence on your grading for this lecture.*

The Time-Constrained Bipartite Vehicle Routing Problem (TCBVRP)

The BVRPDC considers the following situation: Suppose you are part of a team organizing a large-scale Christmas charity event. Your aim is to supply a number of care facilities (e.g. orphanages, nursing homes, etc.) with Christmas dinner. To this end you were able to gather a number of suppliers (e.g. company cafeterias, hotel kitchens, etc.) through extensively advertising and promoting your event. Furthermore, you have a certain number of vehicles for delivering the meals to the care facilities at your disposal. Because of the expected rather large number of prepared meals that need to be delivered to each care facility, each vehicle can only transport exactly one batch of meals from a supplier to one facility. Due to the common generosity around the Christmas season (and the possible prospect of positive publicity) you could convince even more suppliers than necessary to provide meals to all the considered care facilities. In order to guarantee that no facility has to wait too long for the delivery you are given a time limit within which all deliveries must be finished. Furthermore, since you and your co-workers are planning to also enjoy the holiday yourselves, you want to finish the distribution as fast as possible. Hence, you are now faced with the problem of efficiently delivering the meals from the suppliers to the care facilities, i.e. you need to decide on the suppliers you choose and on the routes for the vehicles s.t. all care facilities receive their meals as fast as possible.

Formally, the TCBVRP can be stated as follows: We are given

- a complete graph $G = (N, A)$, where
  - N is the set of nodes consisting of the set of *supply nodes*, S, and the set of *demand nodes*, D, with $|S| \geq |D|$, and a designated node 0, denoting the *depot*.
  - A is the set of arcs containing an arc for each pair of nodes $i, j \in N$, weighted by the time, t_{ij}, needed to travel from i to j.
- the number of *available vehicles* m;
- the *time limit* (per vehicle) T.

Note that the instances are in general not symmetric, i.e. $t_{ij} \neq t_{ji}$.

A feasible solution is a set of at most m tours, where over all tours each demand node is visited exactly once and each supply node is visited at most once. Each tour starts and ends at the depot, where the first visited

node must always be a supply node and the last visited node must always be a demand node. Furthermore, only arcs (i, j) with either $i \in S$ and $j \in D$, or $i \in D$ and $j \in S$ may be used. More precisely, each tour has the form $0 \to s_1 \to d_1 \to s_2 \to d_2 \to \ldots \to s_k \to d_k \to 0$, with all $s_i \in S$ and all $d_j \in D$. Finally, for each tour the total time necessary for completing it must not exceed T.

The goal is to find a feasible solution for which the total time over all tours is minimal. Formally: Let Π be the set of tours. The objective can then be formulated as

$$\sum_{\pi \in \Pi} \sum_{(i,j) \in \pi} t_{ij} \tag{1}$$

TCBVRP Input: A TCBVRP instance specifies the supply and demand nodes, the number of available vehicles, a matrix of travel times between the nodes and the global time limit.

TCBVRP Output: A TCBVRP solution is a set of tours covering each demand node, s.t. none of the above-mentioned constraints are violated.

Exercise Package

The exercise package contains (in addition to this specification) some other helpful files. The most important ones are the TCBVRP instances you will need to solve for the exercises. They reside in the folder "Instances" and have the extension ".prob". Each file consists of the following parts:

- Three lines constituting the header of the instance file stating the number of nodes, the global time limit and the number of vehicles.
- $|N| - 1$ lines containing for each node $i \in N \setminus \{0\}$ its number i followed by either 'S' or 'D', stating that node i is a supply or a demand node respectively.
- A time matrix of size $|N| \times |N|$, where each row starts in a new line and the columns are separated by spaces. The entry in row i, column j states the time t_{ij} needed to travel from the node with number i to the node with number j.

Table 1 shows some properties of the provided instance files.

The format of TCBVRP solutions is even simpler, as it contains for each used vehicle a line listing the nodes visited in the respective tour separated by spaces. The final line in the file lists the objective value of the solution, i.e. the total time over all tours. The file "tcbvrp_10.sol" (also found in the instance folder) is an optimal solution to "tcbvrp_10.prob" and is included for reference.

For the exercises, you will have to implement various heuristics to solve the TCBVRP. You can choose the programming language you want to use freely.

GridEngine: For these exercises, you will get accounts to log onto our servers. From there you will be able to submit jobs to our cluster for evaluating your heuristics. You do not have to use our cluster, if you do not want to you can skip this section. However, we encourage you to use it because it allows for meaningful runtime comparisons between teams. If you decide to use our cluster, the folder GridEngine contains helpful scripts (which can be run on our servers). The scripts qhostr, qstatr and qstatra are replacements for the qhost and qstat utilities of the grid engine and print information about the compute servers a bit more nicely. Use qhostr to list the available servers with information about used CPUs and memory, flag -e will add additional information about running jobs. The script qstatr lists jobs submitted by you, with -r just your running jobs, with -p only pending jobs, and -rpc counts your running and pending jobs. The script qstatra gives the same information, just for all users currently running jobs. The script runSolve.sh submits jobs to the cluster (when run from one of our servers), you will need to modify it to fit your needs. SolveArray.sh is what the execution hosts (the PCs in the cluster) run. This script needs to call your actual application with the necessary parameters.

Deliverables for the Exercise

For this exercise, you have to write a short abstract of two to three pages (excluding tables and figures). This abstract has to contain a short description of the algorithms you implemented, followed by an in-depth

Table 1: Total Number of nodes (excluding the depot), time limit and number of vehicles for the provided instances. Note that a '-' in the vehicles column denotes that the number of vehicles is unlimited.

Name	Nodes	Time Limit	Vehicles
tcbvrp_10.prob	10	240	2
tcbvrp_30.prob	30	240	4
tcbvrp_60.prob	60	360	6
tcbvrp_90.prob	90	480	8
tcbvrp_120.prob	120	480	8
tcbvrp_180.prob	180	720	10
tcbvrp_300.prob	300	360	-
tcbvrp_400.prob	400	480	-
tcbvrp_500.prob	500	720	-
tcbvrp_700.prob	700	1200	-

evaluation, and discussion. A problem description is not necessary. For evaluation, **run each algorithm in every required variant** for the provided instances 30 times (if the result is not deterministic). In your abstract, include at least a table (or tables) containing for each algorithm variant and TCBVRP instance the best found total travel time over all tours, and the total run time of the computation.

Structure the data in a way to allow comparisons between algorithms. Draw conclusions from the collected data in the discussion segment of your abstract. Submit your abstract via email to `heuopt-ws14@ads.tuwien.ac.at` **not later than two days before the date of your interview**! Also bring (or send) your best solution to "tcbvrp_700.prob" in digital form.

Exercise

Considering the TCBVRP, design and implement the following single solution based metaheuristics.

1. Design a useful *greedy* construction heuristic to solve the TCBVRP and implement your approach. Note that your greedy heuristic might yield infeasible solutions which can but need not be repaired.
2. Using your construction heuristic to obtain an initial solution, implement one of the following alternatives.
   (a) Define at least three different neighborhood structures for this problem and combine them in a *Variable Neighborhood Descent* (VND) algorithm. Make sure to implement suitable step functions for your neighborhoods.
   (b) Implement a *Very Large Neighborhood Search* (VLNS) approach. Bear in mind that a good VLNS consists not only of a well-defined large neighborhood structure but also an efficient way of searching it.
   (c) Implement a *Tabu Search* (TS) based on a suitable neighborhood structure. Use attribute-based tabu list(s) and at least one aspiration criterion.

   How do you deal with infeasible solutions?
3. Take your implementation of a VND, VLNS or TS and embed it into one of the following alternatives:
   (a) Randomize your construction heuristic to create a *Greedy Randomized Adaptive Search Procedure.*
   (b) Create shaking neighborhood structures for a *General Variable Neighborhood Search.*
   (c) We are also open to your own ideas for extending the approach. However, please discuss them with us in advance, in case you want to implement them instead.

   Is your adapted approach performing better than the original one?

Remarks:

- Keep efficiency in mind! Is an incremental evaluation possible?
- A nice graphical user interface is **not necessary**, a textual output is sufficient!
- It goes without saying that you are not allowed to use any libraries or frameworks that assist you in the implementation of the actual metaheuristics. However, you may use libraries that implement basic graph data structures and algorithms.