186.112	**Heuristic Optimization Techniques**	WS 2011/12
Programming Exercises	Institute of Computer Graphics and Algorithms Algorithms and Data Structures Group	25.10.2011

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 deadline and the group interviews for the first exercise are scheduled for Tuesday, 6.12.2011. The second interviews will be together with the oral exam on Tuesday, 17.1.2012. Arrangements for specific timeslots are done during the lecture or via email to `heuopt-ws11@ads.tuwien.ac.at`.

Both exercises deal with the *Traveling Tournament Problem*, see below. As a little additional motivation we will organize our own tournament for the first (and only the first) exercise. You will participate automatically by handing in your approach on time. During the lecture on Tuesday, 20.12.2011, the three best teams will be rewarded with glory & fame and some small presents. The criterion will be the best solution of the instance `NL8` 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 Traveling Tournament Problem

The Traveling Tournament Problem (TTP) is a sports timetabling problem that abstracts two issues in creating timetables: home/away pattern feasibility and team travel.
For more information visit the TTP website: `http://mat.gsia.cmu.edu/TOURN/`
You can find the full problem description at: `http://mat.gsia.cmu.edu/TOURN/ttp_final.ps`

For test purposes use the following instances from the website: `NL4, NL6, NL8, NL10`

Exercise 1

Considering the Traveling Tournament Problem (TTP), design and implement the following single solution based metaheuristics for your first exercise. The solutions you create might be infeasible according to the *consecutive home* and *consecutive road* games parameters. If so you would need a repair mechanism for your algorithms. For the first exercise you can relax these parameters, if you want to, but keep in mind that this is not possible for the second exercise.

1. Design a useful *greedy* construction heuristic to solve the TTP and implement your approach.
2. Define at least two different neighborhood structures for this problem and use each of them in a simple local search procedure. Use your construction heuristic to generate an initial solution. Implement the step functions *random neighbor*, *next improvement* and *best improvement*. Keep efficiency in mind! Is an incremental evaluation possible?
3. Test and compare your local search variants with the benchmark instances mentioned above. Document the average performance of the neighborhoods in combination with each step function. Take into account that for stochastic algorithms you have to average over a certain amount of runs (e.g., 30 runs).
4. Combine your neighborhood structures to create a *Variable Neighborhood Descent* (VND) algorithm and test this approach. Is this combination performing better than the single neighborhoods within the local search?
5. Now choose alternatively one of the following options, which you should again test and compare with the tested approaches so far:
   (a) Randomize your construction heuristic to create a *Greedy Randomized Adaptive Search Procedure*.

(b) Take your best local search or VND settings and use it within a *Generalized Variable Neighborhood Search*.

You can choose the programming language for your implementation freely.

It is not part of the exercise to design a nice user interface. Nevertheless, your program must generate a protocol file containing the following information:

- the iterations where a better solution was found and the objective value of this solution
- the total number of iterations and the considered neighbors

Bring to the interview the solutions and protocol files of your best runs for the instances mentioned above for all variations of your algorithms.

Write an **abstract** of two pages about your algorithms, draw conclusions from your various approaches and bring a printed version to the interview. Describe especially the neighborhood structures and their characteristics. You should also bring an executable version of your program to the interview, e.g., on a USB stick, on your notebook, or online.

Exercise 2

For the second exercise create a population based metaheuristic, either a *Memetic Algorithm* (MA) or an *Ant Colony Optimization* (ACO), according to the following specifications. Keep in mind that this time you must consider the *consecutive home* and *consecutive road* games parameters.

Memetic Algorithm: First, create an efficient *Genetic Algorithm* (GA) to solve the TTP. Design an appropriate chromosome encoding for your candidate solutions. Be sure that your search space is not getting too large. You have to design variation operators for the recombination and mutation (at least two of them for each). It is important that during the recombination useful parts of the parent solutions are passed to the offspring.

The second task is to combine your GA with one or more local search procedures of exercise one to create an MA. Make sure that you incorporate the local search efficiently so that the running time of your program is still adequate.

Ant Colony Optimization: An important issue for an effective ACO is a meaningful pheromone model. Therefore, you have to find a compromise between a complex but strong model and an efficient, compact one (concerning memory usage and speed). Another important factor is the use of heuristic information during the construction process and the use of local search.

Therefore, you should implement an ACO variant of your own choice for which you design at least two ant construction heuristics and incorporate one or more local search procedures from exercise one.

The modality of the second interview is the same as for the first.

Test your approach with the instances mentioned above. Do not only optimize your algorithm just for some special cases but also for general use. Therefore, you should test with other instances, too. Again, your program must create a protocol file, which contains the best solutions and their objective values. You must also log the average objective value of each iteration.

Again, write an abstract of two pages and bring a printed version to the interview. You should discuss your algorithm and the problems you had during the implementation. The abstract must include a table which documents your results. Compare the different operators/heuristics you used and document your results within the abstract. Keep in mind that you should take the average value of multiple test runs to get a meaningful result.