186.112	**Heuristic Optimization Techniques**	WS 2012/13
Programming Exercises	Institute of Computer Graphics and Algorithms Algorithms and Data Structures Group	23.10.2012

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, 4.12.2012. The second interviews will be together with the oral exam on Tuesday, 15.1.2013. 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-ws12@ads.tuwien.ac.at`.

For both exercises you will have to solve the *Stamina Aware Sightseeing Tour Problem* with various heuristics, 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, 11.12.2012, the three best teams will be rewarded with glory & fame and some small presents. The criterion will be the best solution of the instance `sastp_1000.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 Stamina Aware Sightseeing Tour Problem (SASTP)

The SASTP considers the following situation: Suppose you are a tourist in a city on a sightseeing trip. You want to visit various points of interest (POIs), however you only have a limited amount of time and stamina, so you cannot visit every single spot. Therefore, you need to choose the most interesting POIs and their order. Additionally, every POI offers different sightseeing methods for enjoying them (e.g., a guided tour, a look behind the scenes, ...), so you need to decide which method you want to use (you can visit every POI at most once and use at most one method).

For the SASTP, there are three quantities you need to keep track of: satisfaction, time and stamina.

Satisfaction This is the quantity you want to maximize. Every sightseeing method you use to enjoy a POI gives you some satisfaction (depending on the method). However, traveling between POIs incurs a satisfaction penalty since you are not interested in viewing industrial districts while in transit. The penalty of a single transfer between locations is the travel distance (euclidean distance) between the locations multiplied with a scaling factor α.

Time You only have a finite amount of time available for sightseeing. The travel time is the sum of all transfers between locations (from hotel to first your POI, between POIs and from last POI back to hotel). The time for a transfer is the distance between the locations divided by your movement *speed*. Of course, the actual sightseeing at the POIs also needs time and has to be accounted for. The sum of travel time and sightseeing time is the overall time required by your sightseeing schedule.

Stamina You need stamina for sightseeing. When leaving the hotel, you have some initial amount. Visiting a POI costs a certain amount of stamina, depending on the sightseeing method you choose. You are not allowed to have negative stamina, so you need to be back at the hotel before you run out of it. However, after you have spent stamina for sightseeing at a POI, you can rest and recover stamina before you travel to the next POI. The amount of stamina you recover is the time spent resting multiplied by the recuperation factor called *habitus*. There is an upper limit to your stamina, so after some time additional resting does not result in additional stamina. The time you spend resting at each POI also counts towards the time required by your sightseeing schedule. Traveling between locations does not change your stamina.

Table 1: Number of POIs (Spots), satisfaction of best known solution and maximum possible improvement to best known solution (Gap).

Name	Spots	Best Known	Gap [%]
sastp_10.prob	10	43.2979	0
sastp_20.prob	20	72.4877	0
sastp_50.prob	50	212.6247	0
sastp_100.prob	100	505.3261	0
sastp_200.prob	200	981.3208	0.41
sastp_500.prob	500	2496.4770	6.25
sastp_1000.prob	1000	3608.0643	≥ 53.2

SASTP Input: An SASTP instance specifies the maximum sightseeing time (maxtime), the initial stamina (initst), the maximum stamina (maxst), the three factors α, *speed*, and *habitus*, and the location of the hotel (start) in a two dimensional plane. It also specifies the locations of each POI (2D Cartesian coordinates) and the sightseeing methods available there. Each method has an associated satisfaction gain, time requirement, and stamina cost. The units (e.g., for time, speed, distance) are compatible without any conversion.

SASTP Output: An SASTP solution is a schedule for visiting POIs, together with the associated sightseeing method chosen for each place and the time spent resting at the specific spots so that none of the constraints are violated.

Exercise Package

The exercise package contains (in addition to this specification) some other helpful files. The most important ones are the SASTP instances you will need to solve for the exercises. They reside in the folder "Instances" and have the extension ".prob". The number contained in each file-name is the number of POIs. Table 1 shows some properties of the instance files. The file format of the instance files is easy to parse and mostly self-explanatory. A point of interest is introduced with "spot", followed by an id, the name of the spot, and two floating point values which specify its location. After a spot follow the sightseeing methods for this spot, introduced by the keyword "method", followed by its name, the generated satisfaction, the time required to use this method, and its stamina cost.

The format of SASTP solutions is even simpler, as it is a list of spots to visit, together with the method to use and the resting time. The file "sastp_10.sol" is an optimal solution to "sastp_10.prob" and is included for reference.

For the exercises, you will have to implement various heuristics to solve the SASTP. You can choose the programming language you want to use freely. If you choose C++, the folder "Framework" contains classes for SASTP problem instances and solutions, so you do not have to parse and write the files yourself. The classes need C++11 features, so remember to set the appropriate flag during compilation. Even if you choose another language, the class SASTPSolution might still be of interest because it includes code to check a solution for validity. There is also an external application available to check the validity of a solution. If you are logged in to one of our servers you can execute `/home1/share/SASTP/SASTPChecker sastp_10.prob sastp_10.sol` to check if the file sastp_10.sol is a valid solution for sastp_10.prob for example. It will also tell you the satisfaction and tour time of the solution, so you can check if your implementations calculate those quantities correctly. Beware, if solution and problem do not match, you're gonna have a bad time! It goes without saying that you are *not* allowed to use any libraries or frameworks that help you with implementing the required heuristics.

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 both Exercises

For both exercises, you have to write a two page abstract (excluding tables and figures). This abstract has to contain a short description of the algorithms you implemented, followed by an in-depth evaluation, and discussion. For evaluation, run each algorithm in every required variant for the provided instances 30 times (if the result is not deterministic), i.e., for the first exercise this means at least 889 separate runs. In your abstract, include at least a table (or tables) containing for each algorithm variant and SASTP instance the best found satisfaction, the average satisfaction, its standard deviation, the average time required to find the best solution, and the total run time. Structure the data in a way to allow comparisons between algorithms. Also include one figure with the development of the average satisfaction over time for the instance "sastp_1000.prob" for all algorithm variants. Draw conclusions from the collected data in the discussion segment of your abstract. Bring your abstract in printed and digital form as pdf to the interviews (or send the pdf in advance per email). Also bring (or send) your best solution to "sastp_1000.prob" *for both exercises* in digital form. Make sure the solutions are actually valid by using the SASTPChecker.

Exercise 1

Considering the SASTP, design and implement the following single solution based metaheuristics for your first exercise.

1. Design a useful *greedy* construction heuristic to solve the SASTP 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. Combine your neighborhood structures to create a *Variable Neighborhood Descent* (VND) algorithm and test this approach with the three step functions. Is this combination performing better than the single neighborhoods within the local search?
4. Now take your best local search or VND settings and embed it alternatively into 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) Create shaking neighborhood structures for a *Generalized Variable Neighborhood Search*.

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.

Memetic Algorithm: First, create an efficient *Genetic Algorithm* (GA) to solve the SASTP. 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 1 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 1.