Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Tobias Friedrich, Timo Kötzing, Andrew Sutton **Summer Term 2015**

Project 2 – "Heuristic Optimization"

https://hpi.de/friedrich/teaching/ss15/heuristic-optimization.html

Swarm algorithms (algorithms based on mimicking swarm intelligence) are useful (and used) as heuristic approaches to many NP-hard combinatorial problems, such as TSP. In this project we will implement a simple ACO (ant colony optimization) algorithm for TSP. To that end, look at TSPLib (a collection of problem instances for TSP) at

`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

In the category *Symmetric traveling salesman problem (TSP)* you will find a lot of TSP data (instances of TSP), as well as the optimal tour lengths for these instances. The file names hint at how many vertices are in the graph (we call the number of vertices in a graph n). We will only consider metric symmetric TSP instances on complete graphs.

You can find more details regarding ACO at `http://de.wikipedia.org/wiki/Ameisenalgorithmus`. We will implement an algorithm called *Max-Mint Ant System* (MMAS) defined as follows. We attach to each edge of the graph a value, the *pheromone value*. We start with a value of $1/n$ on each edge; MMAS also uses maximal and minimal values for pheromone values (thus Max-Min), given as parameters $\tau_{\min}$ and $\tau_{\max}$. We now imagine an ant starting on some vertex. She chooses an unvisited neighboring vertex at random with a probability proportional to the pheromone value on the edge to this neighbor and goes to this neighbor; then she chooses a neighbor of this new vertex again as before and so on, until she has no more vertices to visit and returns home, thus constructing a tour of the graph. If we want to additionally use *heuristic information*, then we let the length of the edge (given as $w(u,v)$ if the edge is from u to v) influence the decision. Finally we compute the length of the tour which the ant chose.

MMAS always remembers the best (= shortest) solution, initialized as random tour. In each iteration a new tour is constructed by an ant as described above. This new solution is compared with the best so far, which is then replaced if necessary. After this the interesting bits start: The pheromones are being updated. We iterate over all pheromones on all edges. If the best-so-far solution is x^* and it uses the edges $E(x^*)$, then the new pheromone value of an edge e is computed as follows.

$$\tau'(e) = \begin{cases} \min\left\{(1-\rho)\cdot\tau(e)+\rho, \tau_{\max}\right\}, & \text{if } e \in E(x^*); \\ \max\left\{(1-\rho)\cdot\tau(e), \tau_{\min}\right\}, & \text{otherwise.} \end{cases}$$

$\rho < 1$ is a parameter of the algorithm. As long as we do not find better solutions, we always use the same old tour (the best so far) to update the pheromones (and they will eventually reach the borders).

The pseudocode for MMAS is given as follows (with N we denote the unvisited neighbors). We use α and β as further parameters of the algorithm.

Algorithm 1: The algorithm MMAS.

```
1 τ(e) ← 1/n, for all e ∈ E;
2 x* ← construct(τ);
3 update(τ, x*);
4 while optimum not found do
5 |   x ← construct(τ);
6 |   if TSP(x) < TSP(x*) then
7 |   |   x* ← x;
8 |   τ ← update(τ, x*);
```

Algorithm 2: The method `construct`.

```
1 Let v_1 be the start vertex;
2 for k = 1 to n − 1 do
3 |   R ← Σ_{z∈N} τ(v_k, z)^α · w(v_k, z)^{−β};
4 |   Choose one neighbor v_{k+1} ∈ N where the probability of selection of any
  |   fixed z ∈ N is τ(v_k,z)^α · w(v_k,z)^{−β} / R;
5 return (v_1, ..., v_n);
```

For a first run of the algorithm you should use the following parameter settings: $\rho = 1/n$, $\tau_{\min} = 1/n^2$, $\tau_{\max} = 1 - 1/n$, $\alpha = 1$ and $\beta = 0$ (no use of heuristic information).

Assignment. Implement MMAS as well as parsing a TSPLib-file (you may use code from the web for parsing a TSPLib-file).

Also use heuristic information (for example $\beta = 4$). Try out a number of parameter settings (or use some automatic parameter tuning, if you feel like that). Document your test runs for figuring out some good parameter settings. Implement at least the two following options as stopping criteria: reaching the optimum or 10 minutes of wall-clock time have passed. Test your algorithm on some small TSP-instances (from TSPLib, only symmetric instances). How large can the instances be so that your algorithm can still find a good solution which is only some percent above the

optimum? You may modify your algorithm as you like, for example by sending multiple ants each round and then choosing the best tour.

Each team submits a zip-File with the following content.

(a) Your code;

(b) A pdf in which you discuss your findings and display the performance of your algorithm.

Extra Credit: Lets have some fun! We give bonus points for:

- Solving an instance from TSPLib with at least 40 cities optimally;
- Solving an instance from TSPLib with at least 100 cities at most 1% worse than the optimum tour length;
- Solving an instance from TSPLib with at least 1000 cities at most 30% worse than the optimum tour length;
- Being the team which solved the highest-number-of-cities instance from TSPLib at most 50% worse than the optimum tour length.

You might need to tune your algorithms some to make them good enough for these extra credit items (and there is no time limit on these). Feel free to google for ideas for how to improve ACO algorithms (I can recommend the paper by Stützle and Hoos from 2000, *Max-Min Ant System*). Document your strategies, your results and your resources used ("four cores at 2.3GHz for 24 hours...").