Heuristic Optimization Techniques 2016W

Book Embedding Problem

First assignment – construction heuristics

Magdalena Malenda

Kornel Kis

Problem:

Develop your own construction heuristic for the K-page crossing number minimization problem. The subtasks were the following:

- Develop a deterministic construction heuristic.
- Develop a randomized/multi-solution construction heuristic. It is allowed that separately created solutions influence the execution of your algorithm.
- Run experiments and write a report as discussed in the problem description.

Approach and solution:

Our solution is written is C++ and uses the provided C++ framework for input parsing and output file writing. We developed one type of heuristic construction heuristic, which contains a possible deterministic solution as a special case.

Our solution is of two main parts: the first is the spine-order pre-calculation phase, the second is the actual calculation of placements of edges on the different pages. We argue that calculating it separately is more beneficial than doing it together, since for larger, denser graphs optimizing the spine order is not really beneficial. For large, sparse graphs, it is probably more beneficial to optimize the spine order more.

For the spine-order calculation, we first calculate the vertex with the largest degree, then use the commonly known DFS algorithm to discover the graph from that root. It is worth mentioning that during the DFS calculation, we firstly choose those vertices, which are the furthest from the current vertex.

Once we have the spine order, we do the following algorithm:

- Sort the edges with regard to their length (descending order)
- Place the longest edge on the first page (named [0]). The length is calculated with regard to the previously calculated spine order.
- For the remaining edges, we place them onto a page, on which it causes the least number of new crossings (deterministic case), or choose randomly from a set of pages on which the new edge causes the least number of new crossings (the size of the set is determined by *group_size* variable, which is a tunable input, non-deterministic scenario)
- We go through the list of the edges with this strategy, and thus, the pages with the edges are generated

It is important to see that the deterministic case is nothing more than a case where the group_size variable is equal to one 1. With this way, we put some random effects into the algorithm which may improve results, but we do not turn the whole calculation into a random search.

In the following table the run times of the algorithm are presented. Each value is calculated from 5 individual runs with the given parameters. The run times are always given in the format of *(mean,standard deviation),number_of_crossings*. All measurement result is given in seconds. If at a given run, *group_size* exceeds the number of pages, the number of total pages are used as *group_size*.

The tests were completed on Ubuntu 14.04, on a normal laptop computer with Intel Core i7 5500U CPU and 8 GB of RAM.

group_size	Runtime [s] (small, instance-1), K = 3	Runtime [s] (medium, instance-7), K = 2	Runtime [s] (large, instance -6), K = 8
1	(0.0012,0.00079),14	(0.590,0.00374),126216	(125.69,0.4863),4897287
2	(0.0098,0.00295),14	(0.586,0.01479),150491	(128.4244,4.2304),5483973
7	(0.0043,0.00021),30	Same as above	(130,340,6,521),7668496