Heuristic Optimization Techniques
Exercise 1

Alexander Eisl (0250266), Peter Wiedermann (0025999)

October 25, 2015

1 Deterministic Construction Heuristic

In the first part of this programming exercise, we develop a deterministic construction heuristic to solve the K-page crossing number minimization problem. The following listing shows the pseudo-code of our algorithm.

Algorithm 1.1: Deterministic construction heuristic

```
input: Graph with given spine-order and set of edges
output: Graph with new spine-order and edge partition

begin
    start with empty solution
    spine-order ← sort(spine order)
    while solution not complete
        add edge to page that creates the fewest crossings
    end
end
```

Our heuristic construct the solution in two parts:

- In the first part, we create a new spine-order by sorting the vertices according to the number of neighbors of each vertex. The idea of this approach is to get vertices with many neighbors closer together to minimize the number of crossings the edges of these vertices can potentially produce.
- In the second part of the heuristic, we add edges to the pages of our solution using a greedy approach. Edges are added in the order of the adjacency matrix (arbitrary). Each edge is added to the first page where it does not introduce additional crossings. If we cannot find such a page, we add the edge to the page where we introduce the minimum number of crossings.

We decided to generate the spine-order in the beginning and then decide on the edge-partinioning independently. Separating the two problems makes the algorithm simpler and computationally faster.

We will mix spine-order and edge-page allocation during our randomized variant desribed in Section 2.

Variable	Type	Description
Pages	Array	Number of pages is constant; allows direct access to each element
Page	List of edges	dynamic; only sequential access needed; each time we add an edge we incrementally update the number of crossings
Spine Order	Array	does not change often in our implementation; easy access. To be able to calculate the number of crossings efficiently, we use an additional spine-order map.

Table 1: This table shows the structure of our solution.

2 Randomization

In this section, we explain how we extend our approach to a randomized/multi-solution heuristic. We do this by randomly permutating a subset of the vertices on the spine. Thus, the result of our spine-sorting algorithm is subject to random variations.

We first start with a spine-porder as described in our deterministic variant and apply the following randomized variant:

- We randomly permutate n vertices k times, to create k different initial spine-orders. For each of these k spine-orders we can run the same edge-allocation algorithm as described in section 1.
- We then compute the number of crossing and – from the list of generated solutions – select the one resulting in the lowest number of edge-crossings.

Our algorithm does not degenerate into random search because we try to improve our deterministic approach.

3 Solution Representation

Table 1 shows the basic structure of our solution representation.

4 Results

We executed the code on a desktop computer with a Core i7 Quad-Core CPU with 2.67Ghz and 24 GB of main memory. Table 3 shows the results of our randomized algorithm, where we have choosen to perform $n = 4$ permutations of vertices to generete $k = 5$ different solutions.

The solutions we generate using the random approach are slightly better than the solutions generated by our deterministic routine. This is not surpring, as we can make sure that the randomized heuristic produces at least the same result as the deterministic algorithm by including the deterministic spine-order in our set of solution candidates.

	Crossings			Runtime		
	min	avg	sd	min	avg	sd
automatic-1.txt	17.00	17.97	0.18	0.00	0.00	0.00
automatic-2.txt	48.00	48.00	0.00	0.00	0.00	0.00
automatic-3.txt	152.00	152.00	0.00	0.00	0.00	0.00
automatic-4.txt	181.00	181.00	0.00	0.00	0.00	0.00
automatic-5.txt	74.00	74.00	0.00	0.00	0.00	0.00
automatic-6.txt	5,146,883.00	5,543,757.41	127,168.49	50.14	52.39	3.47
automatic-7.txt	141,516.00	143,304.17	783.35	0.43	0.43	0.00
automatic-8.txt	522,514.00	551,906.28	10,771.76	10.13	10.23	0.13
automatic-9.txt	1,105,586.00	1,234,003.31	44,577.44	11.29	11.40	0.13
automatic-10.txt	51,835.00	54,339.69	1,193.40	3.34	3.39	0.07

Table 2: This table shows the results of our randomized construction heuristic. The first three columns show the minimum, mean and standard deviation of the number of crossings, the last three columns show the minimum, mean and standard deviation of the runtime in seconds.

Table 3: This table shows the results of our randomized construction heuristic.