Heuristic Optimization Techniques

Exercise 2

David Fankhauser, 1025876
Christoph Weiler 1029175
Gruppe 1

November 8, 2015

1 Local Search Description

Our local search framework is build in a way that we only need to extend an abstract class and implement a *nextNeighbor()* method which generates the next neighbour in the neighbourhood and a *randomNeighbor()* method which generates a random neighbour. Therefore we implemented four neighbourhood structures that are used in this local search framework.

1. *Ordering 1-shift*: This heuristic focusses on optimizing the ordering. It takes every vertex in the ordering and tries to shift it to every other position in the ordering. E.g. In the Ordering $(0, 1, 2, 3, 4)$ we could shift 0 to position 2 which results in $(1, 2, 0, 3, 4)$.

2. *Shift Edges*: This heuristic tries to improve the edge structure. It takes every edge and places it on every page other than the original one. E.g. If edge e is on page 0 we would shift it to the pages $1, \ldots, k$.

3. *Max-Crossing Neighbourhood*: As the shift edge neighbourhood is too big to be fast for big instances, we provide the max-crossing neighbourhood which tries to move all edges with the maximum crossing number of each page on all other pages. This approach is therefore much faster than the shift edges neighbourhood.

4. *Min-Max-crossing ordering*: To provide another fast heuristic we implemented the Min-Max-crossing ordering heuristic as a rather small neighbourhood. It tries to shift the vertices of the edges with the maximum number of crossings to the vertices of all edges with the minimum number of crossings. Therefore it is likely that the number of crossings gets lower on the edges with a maximum number of crossings and a little bit higher for edges with a minimum number of crossings.

The ordering 1-shift and shift edges neighbourhoods are designed as very big neighbourhood structures. We expect that they can work well together when combining them in a VND/VNS search procedure. Also we provide faster/smaller neighbourhoods that work well together. In this assignment the pure ordering of vertices or the pure shifting of edges on different pages does not have a very big effect on the objective function. But on a small test (= executing another local search procedures after the first one) we saw that this approach is promising.

For the initial solutions we used our greedy construction heuristic from the last assignment. Additionally we used another heuristic which was not covered in the last report. We call it "OrderingConstructionHeuristic": First the ordering is created by pushing the vertex with the highest degree on position 0 of the ordering. Then we collect all other vertices together with their degree, sort them ascending on their degree and insert them into our ordering. The edges are inserted one after another at a best fit style (see last GreedyConstructionHeuristic of last report). With this approach the highest degree and the second highest degree are far away from each other which is a good thing because they will not generate lots of crossings between each other.

We did not implement a complete random solution construction because it is very likely that this will perform worse than when constructing a specific solution. If there is a lot of time to compute a number of pure random solutions, some will also have a good structure that will work well with our local search heuristics. Also the No Free Lunch Theorem is an important theorem that states that it can not be stated that one heuristic is better than another in general. Theoretically random search could be better than our presented algorithms on average.

Our neighbourhoods reach better solutions and local minima in many cases for the first-improvement strategy. The best-improvement strategy did not work well in the Ordering 1-shift and Shift Edges Neighbourhoods because they are simply too big for instances with more than 100 vertices (instance 6 to instance 10). Often selecting a random neighbour and comparing it against the best solutions is a better strategy because it is very fast in comparison of performing a best-improvement search. If our first-improvement and best-improvement strategies can not find a better solution, the algorithm terminates. For the random search we say that we are probably at a local minima if 250 consecutive random neighbours could not find a better solution.

Usually the step function has a high impact on the number of iterations of our local search framework. While using a best-improvement strategy results in a few, long iterations, a first-improvement strategy leads to far more, shorter iterations. The highest number of iterations is usually implied by using a random step function. But a random neighbour is usually very quickly generated. Because of that we have lower runtimes for our random algorithms.

2 Experimental Setup

We tested our program on our local computer because in this assignment we used cmake for crossplattform development and the boost library. The needed tools were not installed on Eowyn nor Behemoth therefore we decided to test our application on our computers. The test environment for the results are a Intel Core i7-2600K processor on Ubuntu 15.10. Our code is written in C++11 and it was compiled with gcc 5.2.1.

3 Results

3.1 Results of deterministic local search heuristic

for the evaluation we use the following keys:

- g = greedy construction heuristic
- sho = shift ordering neighbourhood
- moe = move edge neighbourhood
- map = Max crossing page neighbourhood
- mmc = Min-Max crossing ordering neighbourhood
- f = first-improvement step function
- r = random step function

	g+sho+f		g+moe+f		g+map+f		g+mmc+f	
Instance	Obj	Time	Obj	Time	Obj	Time	Obj	Time
automatic-1	21	1 ms	21	2 ms	21	1 ms	21	3 ms
automatic-2	11	16 ms	13	5 ms	15	1 ms	14	2 ms
automatic-3	125	69 ms	70	153 ms	80	11 ms	133	2 ms
automatic-4	12	238 ms	69	7 ms	74	1 ms	51	4 ms
automatic-5	20	152 ms	24	20 ms	24	2 ms	35	2 ms
automatic-6	7489931	15 min	7524000	15 min	7524000	281 ms	7512123	8 s
automatic-7	25725	15 min	25418	60 s	26602	27 ms	27251	583 ms
automatic-8	605470	15 min	612736	15 min	404367	27 s	615945	28 s
automatic-9	869237	15 min	972760	15 min	968508	1 s	858008	128 s
automatic-10	29006	15 min	28763	15 min	27214	2 s	29094	12 s

3.2 Results of randomized local search heuristic

	g+sho+r				g+moe+r			
Instance	best Obj	mean Obj	Std-Dev	Time	best Obj	mean Obj	Std-Dev	Time
automatic-1	21	21	0	2.0ms	21	21	0	2.9ms
automatic-2	11	11.4	0.7	4.6ms	13	13	0	3.9ms
automatic-3	125	127.0	1.2	8.4ms	66	72	3.24	24.2ms
automatic-4	12	13.6	1.9	19.4ms	69	69	0	5.5ms
automatic-5	20	20	0	8.9ms	24	24.3	0.3	9.9ms
automatic-6	7061987	7344789	25095.3	32.3s	7524000	7524000	0	119.1ms
automatic-7	17046	23756.7	3499.1	2.1s	25485	25612.7	124.2	1048.6ms
automatic-8	568146	606308.3	35446.8	6.9s	331656	488932.0	166397.7	174.0s
automatic-9	368012	628114.7	250787.7	90.3s	461877	477389.1	17208.5	15min
automatic-10	28940	29002.5	85.2	10.5s	28840	28978.4	102.6	23.1s

	g+map+r				g+mmc+r			
Instance	best Obj	mean Obj	Std-Dev	Time	best Obj	mean Obj	Std-Dev	Time
automatic-1	21	21.2	0	3.8 ms	21	21	0	4.2 ms
automatic-2	15	11.4	0	3.5 ms	14	14	0	6.8 ms
automatic-3	112	114.3	2.0	18.7 ms	133	133	0	9.5 ms
automatic-4	74	74.5	0.5	6.8 ms	31	42	11.0	10.5 ms
automatic-5	20	24.6	4.2	9.1 ms	35	35	0	8.0 ms
automatic-6	7524000	7524000	0	370.6 ms	7493976	7502285.2	7976.6	4.9 s
automatic-7	26602	27008.4	358.7	168.8 ms	27251	27251	0	218.2 ms
automatic-8	395485	406022.4	10473.6	28.2 s	613464	615204.6	1708.7	984.3 ms
automatic-9	964135	970884.7	6200.5	1.7 s	897044	932076.1	3284.1	1.5 s
automatic-10	27099	28864.6	1544.8	4.5 s	29094	29094	0	1.9 s