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 neighbor in the neighborhood and a *randomNeighbor()* method which generates a random neighbor. Therefore we implemented three neighborhood 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 so 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 Neighborhood*: As the shift edge neighborhood is too big to be fast for big instances, we provide the max-crossing neighborhood 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 neighborhood.

4. *Min-Max-crossing ordering*: To provide another fast heuristic we implemented the Min-Max-crossing ordering heuristic as a rather small neighborhood. It tries to shift the vertices of the edges with the maximum number of crossings on each page 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 neighborhoods are designed as very big neighborhood structures. We expect that they can work well together when combining them in a VND/VNS search procedure. Also we provide faster/smaller neighborhoods 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 hightest 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. But when we have a lot of time some to compute a lot of pure random solutions, some will 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 neighborhoods reach better solutions and local minima in many cases for the first-improvement strategy. The best-improvement strategy did not work so well in the Ordering 1-shift and Shift Edges Neighborhoods because they are simply too big for huge instances. Often selecting a random neighbor 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 propably at a local minima if 250 consecutive random neighbors 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 neighbor 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 neighborhood
- moe = move edge neighborhood
- map = Max crossing page neighborhood
- mmc = Min-Max crossing ordering neighborhood
- f = first-improvement step function
- r = random step function

Instance	g+sho+f Obj	g+sho+f Time	g+moe+f Obj	g+moe+f Time	g+map+f Obj	g+map+f Time	g+mmc+f Obj	g+mmc+f Time
automatic-1	21	500.15 ms	21	358.23 ms	21		21	
automatic-2	11	818.47 ms	13	500.93 ms	15		14	
automatic-3	125	812.99 ms	70	523.08 ms	80		133	
automatic-4	12	1.06 s	69	800.01 ms	74		51	
automatic-5	20	2.07 ms	24	1.50 s	24		35	
automatic-6	7489931	15 min	7524000	15 min	7524000		7512123	
automatic-7	25725	10.5 min	25418	15 min	26602		27251	
automatic-8	605470	15 min	612736	15 min	404367		615945	
automatic-9	869237	15 min	972760	15 min	968508		858008	
automatic-10	29006	15 min	28763	15 min	27214		29094	

3.2 Results of randomized local search heuristic

Instance	g+sho+r best Obj	g+sho+r mean Obj	g+sho+r Std-Dev	g+sho+r Time	g+moe+r best Obj	g+moe+r mean Obj	g+moe+r Std-Dev	g+moe+r Time
automatic-1	21	21	0		21	21	0	
automatic-2	11	11.4	0.7		13	13	0	
automatic-3	125	127.0	1.2		66	72	3.24	
automatic-4	12	13.6	1.9		69	69	0	
automatic-5	20	20	0		24	24.3	0.3	
automatic-6	7061987	7344789	25095.3		7524000	7524000	0	
automatic-7	17046	23756.7	3499.1		25485	25612.7	124.2	
automatic-8	568146	606308.3	35446.8		331656	488932.0	166397.7	
automatic-9	368012	628114.7	250787.7		461877	477389.1	17208.5	
automatic-10	28940	29002.5	85.2		28840	28978.4	102.6	

	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		21	21	0	
automatic-2	15	11.4	0		14	14	0	
automatic-3	112	114.3	2.0		133	133	0	
automatic-4	74	74.5	0.5		31	42	11.0	
automatic-5	20	24.6	4.2		35	35	0	
automatic-6	7524000	7524000	0		7493976	7502285.2	7976.6	
automatic-7	26602	27008.4	358.7		27251	27251	0	
automatic-8	395485	406022.4	10473.6		613464	615204.6	1708.7	
automatic-9	964135	970884.7	6200.5		897044	932076.1	3284.1	
automatic-10	27099	28864.6	1544.8		29094	29094	0	