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. *Ordering 2-opt*: TODO...

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. Often selecting a random neighbor and comparing it against the best solutions is a very good 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.

2 Experimental Setup

We tested our program on the AC computing cluster on blade center 1 (bc1), which is running an Intel Xeon E5540 processor and on Ubuntu 14.04LTS. Our code is written in C++11 without any library assistance and it was compiled with gcc 4.8.

3 Results

3.1 Results of deterministic construction heuristic

Instance	Obj	Time
automatic-1	35	0.05 ms
automatic-2	19	0.06 ms
automatic-3	426	0.09 ms
automatic-4	156	0.06 ms
automatic-5	81	0.07 ms
automatic-6	56031760	637.14 ms
automatic-7	55290	9.21 ms
automatic-8	8481181	180.70 ms
automatic-9	4523302	181.61 ms
automatic-10	740317	44.60 ms

3.2 Results of randomized construction heuristic

Instance	Best Obj	Mean Obj	Std Dev Obj	Mean Time	Std Dev Time	Runs
automatic-1	24	52.27	19.12	0.03 ms	0.01 ms	30
automatic-2	18	36.80	13.91	0.05 ms	0.01 ms	30
automatic-3	214	330.67	85.39	0.11 ms	0.02 ms	30
automatic-4	16	55.23	26.49	0.07 ms	0.02 ms	30
automatic-5	44	87.60	25.58	0.07 ms	0.01 ms	30
automatic-6	28947493	29691804.50	460496.81	859.33 ms	19.09 ms	30
automatic-7	35288	44804.30	5644.89	12.67 ms	2.21 ms	30
automatic-8	4046759	4261985.97	79548.60	172.97 ms	2.38 ms	30
automatic-9	3788437	3981221.60	98111.58	174.87 ms	2.51 ms	30
automatic-10	529589	552330.53	13061.40	64.96 ms	2.04 ms	30