Heuristic Optimization Techniques
Exercise 2

Alexander Eisl (0250266), Peter Wiedermann (0025999)

November 7, 2015

1 Local Search

In this programming exercise, we improve the results of the construction heuristic developed in the previous exercise by using a local search. In the following subsection, we will first explain the neighborhoods that we define and then outline the step functions used.

Algorithm 1.1: Local search

```
input: Graph
output: Improved solution
begin
    sol ← calculate initial solution
    repeat:
            choose sol' ∈ N(x)
            if f(sol') ≤ f(sol) then
                    sol ← sol'
    until stopping criteria satisfied
end
```

1.1 Neighborhoods

We use the following neighborhoods:

1-node flip This neighborhood is defined as all subsets where two vertices of the initial solution are flipped.

- Size of neigborhood: $n(n-1)/2$
- Objective Function: Incremental, crossings from flipped vertices are subtracted and recalculated.

1-edge move This neighborhood consists of all solutions where one edge is moved to a different page.

- Size of neigborhood: $(pages - 1)edges$

- Objective Function: Incremental, only crossings for moved edges are recalculated.

1-node edge move This neighborhood consists of all solutions where the edges of one vertex on a specific page are moved to all different pages.

- Size of neigborhood: $(pages - 1)edges$ as worst case.[1]
- Objective Function: Incremental, only crossings for moved edges are recalculated.

1.2 Stepfunctions

We implemented the following thre stepfunctions:

first This step-function selects the first better solution found in the neighborhood.

best This step-function iterates over all elements of the neighborhood and chooses the best one.

random This step-function selects a random element of the neighborhood.

2 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 1 shows the results of our local search.

We consider a local optimum if we cannot find a better solution after 10 iterations. We have set a timout of 5 minutes for the calculations.

Best neighborhood step function strategy We compare the results for three different neighboorhoods and all the implemented step-functions.

The best performing neighboorhood structure is `1-node-flip`, as shown in Table 1. For smaller instances (1-5), the `1-node-flip` in combination with the `first` delivers the best results. For bigger instances (6-10), the `1-node-flip` finds better solutions in combination with the `random` stepfunction.

A very interesting result is the performance of the `best` stepfunction for the `1-node-flip`. For the given instances, the `best` function reaches a local optimum very quickly while `random` and `first` are calculating more iterations or even calculate until timeout for bigger instances and can find better solutions beside the local optimum of the `best` variant.

[1] depending on number of edges on respective node pages

Initial Solution The choice of the initial solution is crucial to achieve results in reasonable calculation time. Using random initialization can easily degenerate into random search or end in a local optimum with less improvement to a random solution itself.

When we compare the results with a random initial solution, we could measure an avg increase of 55% crossings. [2]

Neighborhoods Incremental objective functions implemented as described in section 1.1 are performing equally well for all 3 neighborhoods.

Subsequent (possibly random) moves in our neighborhoods can only reach valid solutions in the search space because every solution in our neighborhoods is valid.

Reaching local optimum If a local optimum is reached it is found in less than 10 iterations in most cases. As you can see in table 1 there are only a few exceptions. The results with iterations=-1 didn't reach a local optimum and ran into our timeout condition. For many graphs, we do not find an improvement. So with respect to our neighborhood, we seem to be in a local optimum.

[2] measured with 1-node flip and `first` stepfunction

	1-Node (first)	1-Node (best)	1-Node (random)	1-Edge (first)	1-Edge (best)	1-Edge (random)	1-N-E (first)	1-N-E (best)	1-N-E (random)
automatic-1.txt	***21*** 0 / 0.01	***21*** 0 / 0.00	***21*** 0 / 0.00	***21*** 0 / 0.00	***21*** 0 / 0.00	***21*** 0 / 0.01	***21*** 0 / 0.00	***21*** 0 / 0.00	***21*** 0 / 0.01
automatic-2.txt	***30*** 4 / 0.05	48 0 / 0.00	48 0 / 0.01	48 0 / 0.00	48 0 / 0.00	48 0 / 0.00	41 0 / 0.00	48 0 / 0.00	48 0 / 0.00
automatic-3.txt	94 5 / 0.22	104 0 / 0.00	97 2 / 0.06	104 0 / 0.01	104 0 / 0.00	104 0 / 0.00	104 0 / 0.00	104 0 / 0.01	***79*** 8 / 0.10
automatic-4.txt	***12*** 17 / 0.45	164 9 / 0.02	169 4 / 0.02	69 36 / 0.17	192 0 / 0.00	163 6 / 0.00	192 0 / 0.00	192 0 / 0.00	155 9 / 0.02
automatic-5.txt	***32*** 10 / 0.32	77 0 / 0.00	69 1 / 0.02	75 0 / 0.00	77 0 / 0.00	77 0 / 0.00	77 0 / 0.00	77 0 / 0.00	41 15 / 0.04
automatic-6.txt	6,151,784 -1 / 308.60	6,153,059 0 / 7.95	***5,965,084*** -1 / 308.74	6,153,059 0 / 220.01	6,153,059 0 / 9.83	6,033,148 -1 / 308.61	6,153,059 0 / 8.60	6,152,236 -1 / 319.12	6,151,720 -1 / 308.59
automatic-7.txt	140,291 -1 / 301.14	143,905 0 / 0.18	142,786 19 / 18.48	142,777 32 / 3.30	143,905 0 / 0.62	140,027 60 / 12.03	143,905 0 / 0.16	143,781 14 / 3.90	***137,629*** 315 / 204.80
automatic-8.txt	537,728 -1 / 302.71	541,969 0 / 1.84	***501,875*** -1 / 302.72	541,935 3 / 40.33	541,868 3 / 6.11	521,950 -1 / 302.73	541,969 0 / 1.83	541,598 14 / 79.82	539,307 -1 / 302.73
automatic-9.txt	1,232,469 -1 / 303.06	1,267,963 0 / 2.19	***1,186,295*** -1 / 303.06	1,268,407 4 / 2.89	1,267,642 8 / 12.85	1,231,026 -1 / 303.05	1,268,456 0 / 2.11	1,267,128 42 / 224.19	1,265,579 -1 / 303.03
automatic-10.txt	56,115 -1 / 302.27	56,260 0 / 1.34	55,134 -1 / 302.28	56,021 70 / 44.49	56,260 0 / 5.97	***53,952*** -1 / 302.27	56,260 0 / 1.35	56,260 0 / 6.43	54,848 -1 / 302.26

Table 1: Local search results. For each instance, we show the number of crossings (first row), the iteration needed to reach the local optimum (second row, first value) and the run-time in seconds of our algorithm (second row, second value). If no local optimum was found the number of iterations is -1. The best solution is highlighted in bold.