Heuristic Optimization Techniques

Exercise 1

David Fankhauser, 1025876
Christoph Weiler 1029175
Gruppe 1

October 25, 2015

1 Construction Heuristic Description

Our construction heuristic focueses on placing vertices next to each other that are adjacent to each other. This approach can not lead to crossings because the definitons states that in order to create a crossing between the edges $E_1 = \{v_1, v_2\}$ and $E_2 = \{v_3, v_4\}$, one of v_3 and v_4 must be inbetween of v_1 and v_2 and the other one outside. This can not happen if v_1 and v_2 are adjacent in the resulting ordering.

Therefore we created the following construction heuristic:

```
// Create ordering
Ordering o = { first vertex given }
while (not all vertices in o) {
    v_l = last inserted vertex in o
    v_a = find vertex adjacient to v_l
    if(v_a exists) {
        add v_a to o after v_l
    } else {
        v_a = find next vertex not yet in o
        add v_a to o after v_l
    }
}

// Create edge list
for (Edge e in instance) {
    for (Page k in instance) {
        crossingNr = calculate created crossing if we would add e to k
    }
    add e to the page k with the lowest crossingNr
}
```

As one can see we create the ordering first and then create the edge list. We use this approach because it is much faster than other construction heuristics that will adapt the ordering

to the inserted edges. In our opinion one should not invest a lot of time in constructing a good solution because further improvement heuristics do this job very well and they should have more time to investigate the search space further. On the other hand the starting solution influences the structure of the solution and potentially it will keep this structure (or parts of it) when improving the solution. From this point of view a good starting heuristic would be preferable. Due to the six assignments in this course we assume that our improvement methods will most likly also change structure of the resulting solution, therefore we went with the first approach.

Our solution is represented by a solution class file that stores two vectors that are kept synchronous, namely a vector that stores the ordering of each vertex and a vector that stores the vertex for each ordering. This makes calculations for the crossing number easier. We also store a hashmap (unordered_map) that maps the page number to a vector of edges. This datastructure represents all edges that are stored on one page. We do not compute the number of crossings seperately, but when inserting/deleting one edge from our solution class, we compute the difference that this insertion/deletion caused and add it to our already computed value. This approach will make improvement heuristics such as local search much easier because the complexity of recalculating the whole objective function decreases drastically.

Our construction heuristic that generates multiple solutions is an adaption of the previous shown heuristic. Instead of inserting the first adjacient vertex to the previously inserted one, we search for all adjacient vertices and choose a random one of them. The advantage of this approach is that we are now able to construct multiple solutions with the same heuristic. Some solutions might be better and some might be worse but the solutions are not the same any more. When we improve given solutions it is often better to have multiple solutions. We can then do a more extensive search in the search space (e.g. Local search on multiple constructed solutions) or combine these solutions (e.g. Evolutionary algorithms).

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 is running 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