Heuristic Opt. Techniques - Assignment 2 Report

Martin Blöschl and Cem Okulmus

1 Implementation

To build a framework that can use different neighbourhood structures and different step functions on these, we first tried to think of three separate neighbourhoods.

1.0.1 Neighbourhood structures

- **One Edge Swap**
  This neighbourhood provides all solutions that differ to the current solution in only one edge swap - i.e one edge is moved to a different page.
- **One Node Swap**
  This neighbourhood provides all solutions that differ to the current solution in one pair of nodes swapped in the spine order - i.e two nodes swap places another in the spine order.
- **Node Neighbour Swap**
  This neighbourhood provides all solutions where the order of the nodes differs in one swap of two neighbouring nodes, i.e nodes that are next to each other in the spine order. This is essentially a smaller neighbourhood of the One Node Swap neighbourhood which is thus faster, but has less potential to escape local optima.

None of these neighbours (on its own) is complete, in the sense that all possible instances in the solution space are covered. We created neighbourhoods that either only focus on the edges or on the node order. So clearly only a combination of neighbourhoods would reach any valid solution in the search space.

1.0.2 Step Functions

- **Best Improvement**
  This step function is quite trivial. Every neighbour is evaluated and only the best improvement (if any!) is accepted.
- **First Improvement**
  This step function accepts the first neighbour with a lower crossing count than the original solution (where we created our neighbours from).
- **Random Choice**
  This step function chooses a random instance from the neighbourhood. To implement this, we actually instruct the neighbourhood itself to generate a random, but valid, neighbour for the given initial solution. To make this still sensible, we

Furthermore, to increase the performance of our crossing counting, we implemented a new mechanism altogether, and extended the existing mechanism from our last assignment.

The new mechanism, which we called IntegerColission, is used when the spine order is changed, and it consists of an integer array that counts at which nodes which edge is “active”,i.e., passing over it. This required that the edges are first sorted by their highest (w.r.t the ordering) end point.

The old mechanism, which we called ActiveEdge, almost the same, except it also inserted the edges itself in a search tree, sorted by their “span” (the distance between the current node at which it is passing, and its highest end point). We added the possibility to also remove edges from it, which allows an incremental evaluation of solutions, which only differ in their edge partition.

Therefore, the incremental evaluation is only used for the Edge Swap neighbourhood. Since we could not make our ActiveEdge mechanism incremental for changes in the spine order, we reduced the needed memory usage by replacing it with a simple integer array.

2 Evaluation

For the testing, we used the eowyn cluster (eowyn.ac.tuwien.ac.at) under the given login credentials. For our testing, we let each of the ten instances (automatic-1 to automatic-10) run for each of three construction heuristics. The first two are the Deterministic and Randomized construction heuristics from assignment 1. We furthermore added a completely Random construction “heuristic”, it simply generates a totally random solution. Its purpose is simply to observe how the local search improves it compared to the other two. (The Randomized was run for 30 seconds and the best one taken as the initial solutions).

Then we used one the three neighbourhoods defined above and one of the three step functions from above. Altogether this gives us 27 possible test scenarios. Each specific instance was limited to 15 minutes, with a hard timeout. At the timeout, the last found solution was output.

For the larger instances (6 to 10), we only used the first improvement step functions, since the other two almost always stopped without finding anything within 15 minutes. Ideally, this could be solved in the case of Best Improvement by drastically increasing the performance of creating and evaluating new solutions.

The results can be seen in the table below. Each entry is the best (lowest) result that was achieved using that combination of heuristic for initial solution, step function and search in the neighbourhood.

3 Observations

We observerd, that using random solutions did not yield a gain, in any of the instances. Our construction heuristics preform much better than just random initial solutions. Since the construction heuristics are relatively cheap, they should be used over just random initial solutions.

To measure how many iterations it took to find local optima, we run a specific test for the smaller instances (automatic-1 to automatic-5), while also keeping track of iteration count. Here the average is 1, with a few outliers to 5 and 8. On the other hand, the larger instances, took up to 10 or 20 iterations before the timeout hit. Since we couldn’t find optimal solutions on the smaller ones, but had plenty of time to spare, it might be a good idea to use larger neighbourhoods for them.

We believe the next logical improvement of our approach would be a combination of separate neighbourhoods, such in a Variable Neighbourhood Search (VNS).

Scenario	automatic-1	automatic-2	automatic-3	automatic-4	automatic-5	automatic-6	automatic-7	automatic-8	automatic-9	automatic-10
DFE	18	40	108	148	71	7326516	139578	576106	1401287	54087
DFN	21	40	112	148	71	7523974	139604	571005	1377295	54151
DFNE	21	50	112	193	89	753489	140553	364272	1400563	54157
DBE	18	40	108	148	71	-	-	-	-	-
DBN	21	40	112	148	71	-	-	-	-	-
DBNE	21	50	112	193	89	-	-	-	-	-
DRE	21	51	112	193	93	-	-	-	-	-
DRN	21	50	112	196	93	-	-	-	-	-
DRNE	21	54	112	197	93	-	-	-	-	-
R1FE	9	5	51	44	25	1880924	127389	246871	358671	44290
R1FN	9	5	52	44	25	1880948	128023	250084	352748	44305
R1FNE	9	5	52	50	27	1880948	127853	239841	364272	44309
R1BE	9	5	51	44	25	-	-	-	-	-
R1BN	9	5	52	44	25	-	-	-	-	-
R1BNE	9	5	52	50	27	-	-	-	-	-
R1RE	9	7	53	51	28	-	-	-	-	-
R1RN	9	7	53	51	28	-	-	-	-	-
R1RNE	9	7	53	50	28	-	-	-	-	-
R2FE	25	78	293	133	143	8596350	1205472	12156016	1790171	233865
R2FN	26	78	293	133	143	8588458	1143775	12156016	1682728	234061
R2FNE	28	104	342	182	177	8566426	1164516	12156016	1705438	234257
R2BE	25	78	293	133	143	-	-	- -	-	-
R2BN	26	78	293	133	143	-	-	-	-	-
R2BNE	28	104	342	182	177	-	-	-	-	-
R2RE	28	112	352	193	184	-	-	-	-	-
R2RN	28	104	325	193	155	-	-	-	-	-
R2RNE	28	108	350	193	184	-	-	-	-	-

Table 1: Code for scenario name: [Construction name] + [Step function name] + [Neighbourhood name]
[Construction name] = { D := Deterministic, R1 := Randomized, R2 := Random }
[Step function name] = { B := Best Improvement, F := First Improvement, R := Random }
[Neighbourhood name] = { E := Edge Swap, N := Node Swap, NE := Node Neighbour }