

Heuristic Opt. Techniques - Assignment 2 Report

Martin Blöschl and Cem Okumus

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 Move**

This neighbourhood provides all solutions that differ to the current solution in only one edge move - 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.

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

The new crossing count method, 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 crossing count method, 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 Move 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 has the following information: best value / difference to mean value / standard deviation using that combination of heuristic for initial solution, step function and search in the neighbourhood.

3 Observations

We observed, that using random solutions did not yield a gain, in any of the instances. Our construction heuristics perform much better than just random initial solutions. In all test runs our “intelligent” randomized construction heuristic was better than the completely random initial solution. 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.

Entries as $X/Y/Z$ where X = best value and Y = absolute difference from mean and Z = std. deviation (last two where applicable).

Scenario	automatic-1	automatic-2	automatic-3	automatic-4	automatic-5	automatic-6	automatic-7	automatic-8	automatic-9	automatic-10
DPE	18/0/0	40/0/0	108/0/0	115/0/0	71/0/0	7326516/0/0	139578/0/0	576106/0/0	1401287/0/0	54087/0/0
DFN	21/0/0	40/0/0	112/0/0	116/0/0	71/0/0	7523974/0/0	139604/0/0	571005/0/0	1377295/0/0	54151/0/0
DFNE	21/0/0	50/0/0	112/0/0	113/0/0	89/0/0	753489/0/0	140553/0/0	364272/0/0	1400563/0/0	54157/0/0
DBE	18/0/0	40/0/0	108/0/0	107/0/0	71/0/0	-	-	-	-	-
DBN	21/0/0	40/0/0	112/0/0	116/0/0	71/0/0	-	-	-	-	-
DBNE	21/0/0	50/0/0	112/0/0	113/0/0	89/0/0	-	-	-	-	-
DRE	21/3.25/3.40	51/4.50/3.14	112/0/0	113/0/0	93/0/0	-	-	-	-	-
DRN	21/3.65/3.43	50/2.48/1.97	112/0/0	113/0/0	93/0/0	-	-	-	-	-
DRNE	21/0/0	54/6.48/7.32	112/0/0	118/0/0	93/0/0	-	-	-	-	-
R1FE	9/0/0	5/0/0	51/5.86/3.57	7/0/0	18/1.21/3.13	1880924/?/? ^a	127389/?/? ^a	246871/?/? ^a	358671/?/? ^a	44290/?/? ^a
R1FN	9/0/0	5/0/0	52/4.26/3.71	7/0/0	25/2.35/3.04	1880948/?/? ^a	128023/?/? ^a	250084/?/? ^a	352748/?/? ^a	44305/?/? ^a
R1FNE	9/0/0	5/0/0	52/4.14/3.68	7/0/0	25/2.24/3.92	1880948/?/? ^a	127853/?/? ^a	239841/?/? ^a	364272/?/? ^a	44309/?/? ^a
R1BE	9/0/0	5/0/0	51/5.66/3.84	7/0/0	13/1.58/3.85	-	-	-	-	-
R1BN	9/0/0	5/0/0	52/3.26/2.15	7/0/0	20/2.61/3.71	-	-	-	-	-
R1BNE	9/0/0	5/0/0	52/3.66/3.25	7/0/0	27/2.84/3.69	-	-	-	-	-
R1RE	9/0/0	7/3/2.91	53/2.76/3.14	7/0/0	18/1.91/3.54	-	-	-	-	-
R1RN	9/0/0	7/1.54/1.12	53/2.36/2.91	7/0/0	26/2.31/3.47	-	-	-	-	-
R1RNE	9/0/0	7/2.84/1.74	53/2.41/3.51	7/0/0	28/2.38/2.49	-	-	-	-	-
R2FE	25/4.11/4.40	78/8.48/4.54	293/23.1/36.3	133/18.48/4.54	143/13.35/16.37	8596350/?/? ^a	1205472/?/? ^a	12156016/?/? ^a	1790171/?/? ^a	233865/?/? ^a
R2FN	26/2.04/1.37	78/4.45/3.48	293/35.3/37.1	133/14.45/5.48	143/15.42/17.12	8588458/?/? ^a	1143775/?/? ^a	12156016/?/? ^a	1682728/?/? ^a	234061/?/? ^a
R2FNE	28/3.81/2.66	104/8.78/9.64	342/23.1/31.4	182/18.89/9.64	177/13.17/11.44	8566426/?/? ^a	1164516/?/? ^a	12156016/?/? ^a	1705438/?/? ^a	234257/?/? ^a
R2BE	25/3.45/3.78	78/2.45/1.87	293/33.5/33.6	133/12.31/9.49	143/13.59/13.69	-	-	-	-	-
R2BN	26/2.00/1.41	78/5.48/3.54	293/43.4/32.8	133/15.82/3.46	143/13.42/12.87	-	-	-	-	-
R2BNE	28/1.15/	104/6.87/4.87	342/53.3/34.7	182/16.87/4.87	177/13.35/14.74	-	-	-	-	-
R2RE	28/1.25/1.25	112/10.87/7.78	352/39.7/35.9	193/10.77/7.17	184/19.77/15.95	-	-	-	-	-
R2RN	28/2.30/2.17	104/13.16/7.75	345/28.5/36.1	193/13.66/7.36	155/18.59/16.16	-	-	-	-	-
R2RNE	28/2.32/2.54	108/4.49/6.97	350/32.4/36.0	193/14.49/10.97	184/12.44/16.07	-	-	-	-	-

Table 1: Code for scenario name: [Construction name] + [Step function name] + [Neighbourhood name]

[Construction name] = { D := Deterministic, R1 := Randomized, R2 := Completely Random }

[Step function name] = { B := Best Improvement, F := First Improvement, R := Random Neighbour }

[Neighbourhood name] = { E := Edge Swap, N := Node Swap, NE := Node Neighbour }

^aOnly one measurement here, since 15 timeout hit