Heuristic Optimization Techniques
Exercise 3

Alexander Eisl (0250266), Peter Wiedermann (0025999)

November 21, 2015

1 General Variable Neighborhood search

In this programming exercise, we implement a generalized variable neighborhood search heuristic, re-using the neighborhoods from the previous exercise. We implement the algorithm as shown on page 11 of the lecture slides on VNS + VDS.

1.1 Neighborhoods

We re-use the neighborhoods from Assignment 2 and add a parameterized node-move variant:

X-node-move Defined as all subsets where X vertices of the initial solution are moved to other positions.

- Size of neigborhood: $(n-1)^2 \cdot X$
- Objective Function: Incremental, only crossings for moved edges are recalculated.

The neighborhoods from the previous assignment:

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 re-calculated.

1.2 Neigborhoodstructures in General VNS

We define two non-overlapping sets of neigborhoods for the outer and inner loop:

VNS: used for the shaking, we decided to take the complex neighborhoods here, because it would be costly to search them completely. We use the following order:

1-node-move, 2-node-move, 3-node-move

VND: used for variable neighborhood descent in the inner loop. We use use the following order:

1-edge-move, 1-node-edge-move

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.

We set a timeout of 5 minutes for the calculations. Because we are relying on random calculations, we provide the summary statistics of the results over 30 independent runs.

Table 1 shows the number of crossings for the GVNS using the order of neighborhoods as described in Section 1.2. The corresponding run-times are reported in Table 4.

We find that for the small instances, the GVNS finds better solutions than the local search we used in the previous exercise. The timing in Table 4 shows that for larger instances (except for instance 7) the maximum run-time is usually exhausted and, as a result, we do not find large improvements compared to the local search in the previous exercise.

Order of the neighborhoods We run two additional experiments where we change the order of the neighborhoods.[2]

Tables 2 and 5 show the results when we reverse the order of the VNS neighborhoods used for the shaking.[3]

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

[2] see Section 1.2 for the original order

[3] Unfortunately, our script which executed the experiments did not include a character string specifying the experiment and, thus, re-used the filenames of our output files. As a result, we have to re-run

As we can see, the reverse order of the VNS neighborhoods improves the minimum number of crossings for many instances. If we are looking at the timing results of instance 7, in Table 5, we see that the variance in the runtime is higher than in the original order. This might imply that we get a better minimum for the cost of stability of the solution, originating from the fact that we start exchaning 3 random vertices, than just 1 as in the original order. However, the fact that we hit the maximum run-time in many cases makes this difficult to interpret.

our computations to get all the output files necessary for the submission on the cluster. Runtimes and crossing numbers were recorded separately and are correct. Because of the random nature of our algorithm, the crossing numbers of the solutions that we will submit on the cluster tomorrow might be different. Please note that the submission deadline on TUWEL seems to be too early (Sunday, 00:00) and different from the one mentioned in the assignment (Sunday, 23:55).

	var	avg	sd	min
automatic-1.txt	1.51	9.42	1.23	9
automatic-2.txt	35.90	41.68	5.99	36
automatic-3.txt	20.88	67.53	4.57	59
automatic-4.txt	234.26	97.05	15.31	84
automatic-5.txt	0.95	40.68	0.98	39
automatic-6.txt	0.00	6,153,059.00	0.00	6,153,059
automatic-7.txt	714,058.88	136,518.53	845.02	135,111
automatic-8.txt	581,888.04	541,495.47	762.82	539,317
automatic-9.txt	687,002.37	1,266,526.95	828.86	1,265,326
automatic-10.txt	24,946.19	55,289.26	157.94	55,035

Table 1: Number of crossings using default neighborhood order

	var	avg	sd	min
automatic-1.txt	2.53	9.68	1.59	9
automatic-2.txt	41.53	39.21	6.44	25
automatic-3.txt	39.41	64.53	6.28	50
automatic-4.txt	13.62	127.53	3.69	112
automatic-5.txt	107.82	45.84	10.38	30
automatic-6.txt	0.00	6,153,059.00	0.00	6,153,059
automatic-7.txt	1,033,834.45	136,568.16	1,016.78	134,450
automatic-8.txt	0.00	541,969.00	0.00	541,969
automatic-9.txt	1,096,844.32	1,267,338.32	1,047.30	1,265,185
automatic-10.txt	76,894.93	55,855.74	277.30	55,425

Table 2: Number of crossings using reverse order of VNS neighborhood

	var	avg	sd	min
automatic-1.txt	2.35	16.47	1.53	12
automatic-2.txt	19.15	37.89	4.38	36
automatic-3.txt	27.64	64.79	5.26	56
automatic-4.txt	116.13	137.37	10.78	108
automatic-5.txt	30.85	47.32	5.55	40
automatic-6.txt	954,027,528.83	6,011,850.26	30,887.34	5,962,740
automatic-7.txt	52,922.03	137,904.84	230.05	137,553
automatic-8.txt	5,544,075.63	504,782.95	2,354.59	501,545
automatic-9.txt	13,573,379.20	1,188,287.47	3,684.21	1,185,819
automatic-10.txt	28,478.53	55,428.00	168.76	55,178

Table 3: Number of crossings using reverse order of VND neighborhood

	var	avg	sd	min
automatic-1.txt	0.00	0.03	0.00	0
automatic-2.txt	0.00	0.02	0.01	0
automatic-3.txt	0.05	0.61	0.22	0
automatic-4.txt	0.00	0.08	0.01	0
automatic-5.txt	0.00	0.06	0.01	0
automatic-6.txt	0.26	309.22	0.51	309
automatic-7.txt	712.56	277.61	26.69	228
automatic-8.txt	0.01	302.82	0.09	303
automatic-9.txt	0.00	303.16	0.06	303
automatic-10.txt	0.00	302.35	0.04	302

Table 4: Runtime in seconds using default neighborhood order

	var	avg	sd	min
automatic-1.txt	0.00	0.02	0.00	0
automatic-2.txt	0.00	0.02	0.01	0
automatic-3.txt	0.22	0.74	0.47	0
automatic-4.txt	0.00	0.05	0.02	0
automatic-5.txt	0.00	0.12	0.04	0
automatic-6.txt	0.01	309.07	0.11	309
automatic-7.txt	894.86	275.49	29.91	216
automatic-8.txt	0.00	302.81	0.03	303
automatic-9.txt	0.06	303.23	0.24	303
automatic-10.txt	0.00	302.34	0.01	302

Table 5: Runtime in seconds using reverse order of VNS neighborhood

	var	avg	sd	min
automatic-1.txt	0.00	0.01	0.00	0
automatic-2.txt	0.00	0.04	0.01	0
automatic-3.txt	0.12	0.74	0.35	0
automatic-4.txt	0.00	0.06	0.02	0
automatic-5.txt	0.00	0.08	0.01	0
automatic-6.txt	0.90	309.33	0.95	309
automatic-7.txt	0.00	301.15	0.01	301
automatic-8.txt	0.00	302.81	0.02	303
automatic-9.txt	0.06	303.22	0.24	303
automatic-10.txt	0.00	302.34	0.02	302

Table 6: Runtime in seconds using reverse order of VND neighborhood