Heuristic Opt. Techniques - Assignment 4 Report

Martin Blöschl and Cem Okulmus

1 Implementation

We chose to implement a Genetic Algorithm (GA) for this assignment. For this we set out to find an encoding that is easier for the GA to work with, and then implement the different operators for Selection, Recombination, Mutation and Replacement. For the last element, we decided early on to stick with a form of generational replacement, but where only the better half (w.r.t. fitness) of the previous generation is replaced.

1.1 Encoding

Since the spine-order is already stored as just an array of integers, we simply thought of it as its own "chromosome". For the edge partitions, we encode a list of pairs of edges (which itself is an integer pair) and a page (as an integer). For decoding it, we create the edge partition as an array of the length of the pages, each cell containing the corresponding list of edges.

1.2 Initial population

First, we used some solutions constructed by our advanced construction heuristics (see previous assignment reports) as initial population. However it turned out that those solutions were mostly very similar, and thus the initial population was not diverse enough. We then used a set of purely random initial solutions (random page assignment, random spine order). This turned out to be worse in the beginning, but eventually after some generations lead to better results.

1.3 Selection

We implemented a simple selection method that just selects the better half of the population for recombination. "Better" here means less crossings. To get more precise, we sort the population by crossings ascending and pick all individuals from 0 to half the size of the population (rounded down).

1.4 Recombination

1.5 Mutation

We mutate approximately 33% of the individuals and from those exactly one gene. Of course we could spend some time optimizing these parameters. In the future, we could try mutating less individuals but in those selected for mutation we mutate multiple genes, or we could try mutating even more individuals. Since our parameter setting (33%, mutate one gene) turned out to give quite good results, we did not spend additional time adjusting these parameters.

The mutation of a gene is in our case changing the page assignment of an edge. Every edge must be assigned to one page. If such an assignment is mutated, we assign this edge to a random other page.

1.6 Replacement

As stated above, we replace the worse half of our population with the children of the selected population. The size of the populations always stays the same.

2 Evaluation

The evaluation was run, once again, on the eowyn cluster. For instances 1 through 5, we used a population size of 10000 and 200 generations. For instances 6 through 10, we reduced the population size to 1000 and generations to 100. This was for performance reasons, as the increase in memory size reduced the performance drastically. The results for 10 runs each can be seen on table **??**

3 Observation

For the smaller instances (1 through 5), the GA almost always found either optimal or compared with the other solution submissions from the other groups, the best known solutions (the exception being 5). For the larger instances this was not the case. The smaller population size reduced the variation and therefore effectiveness of the meta heuristic.

Instance	Global best	Mean (10 runs)	std. dev.	runtime mean (sec.)	runtime std. dev.
automatic-1	9	9	0.0	?	?
automatic-2	0	0.2	0.42	?	?
automatic-3	26	41.7	6.98	?	?
automatic-4	0	0.3	0.67	?	?
automatic-5	2	5.6	2.22	?	?
automatic-6*	8360244	8404827	39653.48	?	?
automatic-7*	92502	100305	7597.2	?	?
automatic-8*	947961	989561	30022.91	?	?
automatic-9*	1347025	1453652	82271.8	?	?
automatic-10*	202479	207943	4081.1	?	?

Table 1: Test results for GVNS. Entries marked with Star (*) use different population sizes as described in the report.