1 Implementation and testing

1.1 Implementation issues

Throughout the process of implementation, one of the main issues, that hindered further implementation and experimentation, was trying to find the optimal settings and for the evolutionary algorithm. This lead to lots of testing happening trying to find better setting for the algorithm, however this caused further issues, as not only does the randomness of the algorithm make it hard to whether one program is better optimized than another without statistical analysis but also made testing for bugs through manual testing became tedious and hard to repeat, given the same results was unlikely to be produced several times. There was no good way to solve or adapt to these issues, other than manually trying to find the bugs and optimize the algorithm, which slowed down the implementation of the overall program significantly.

The agile development approach used caused issues, and was not well suited to something like this research project, specifically the use of sprints, meant that not all elements of the program, were planned out before earlier part integral to the structure, had been implemented. This caused issues in two different ways, some of the earlier parts had to be completely overhauled which wasted project time, whilst others were just kept making the algorithms less effective or producing more bugs later on.

The issue of randomness, was particularly bad for the hybrid method, as it was already quite inconsistent, this made testing to see whether any adjustments made to the algorithm, generally did not show results without large amounts of testing, especially for the 9x9 sudokus, which take significantly longer to run than the 4x4s. The other issue was making small adjustments to program for improvements was hard to track on the 4x4 puzzle, as the number of generations was generally so small, that it couldn't be improved much, so the only changes that would show is when the number of generations greatly increases.

For the multi-objective EA implementation, there was a misconception on how to use and compare the fitness values of two different puzzles, which took several weeks to realise that the wrong approach was being used. A similar problem happened for the overall design of the selection and mutation functions which were being applied in a inefficient manner, however when the repair class was changed to apply with the new method, it could no longer produce results for the puzzles, so these changes were reverted for the repair based solver only, meaning that it is less effective than it could be.

Several simple issues were made during the experiment phase, specifically surrounding writing the results to files, with results being incorrectly recorded, overwritten and deleted, which lead to experiments having to be run several times. This admittedly could have been avoided by properly testing out the experiment code on a smaller set of data using test file locations.

1.2 Testing

There was no define approach to how the testing was created, some of the tests were created after the methods as a way to check their behaviour, and some tests were done in the test driven development style, where tests were made first and methods were created to pass the tests. Due to the nature of the project, the testing itself, only really has to cover finding errors and checking method behaviour, given that there is no user or for the system. The two different types of tests used in the project have been automatic and manual testing.

1.2.1 Automatic testing

The automatic testing, uses the 3rd party library JUnit and covers most of the important methods in the Puzzle, Solver, RepairBasedSolver and MultiObjectiveSolver classes. It was also used to do the initial tests for the getSolution methods, which in this program is the integration tests given that the two solver classes getSolution use most of the components.

1.2.2 Manual testing

The manual testing for the project was done, when behaviours in the outputs of the different solvers were found that were not intentional. The testing was mostly trying to recreate the conditions of the bug, however difficult it was given the randomness and using the breakpoints and debugger in the IntelliJ IDE to identify what was going wrong.