Exercise 2
IT3708

Simon Borøy-Johnsen
MTDT

February 25, 2016

1 Implementation

1.1 Description

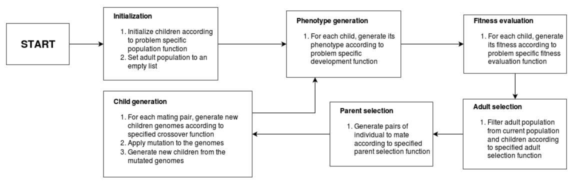
START
Initialization
1. Initialize children according to problem specific population function
2. Set adult population to an empty list
Phenotype generation
1. For each child, generate its phenotype according to problem specific development function
Fitness evaluation
1. For each child, generate its fitness according to problem specific fitness evaluation function
Adult selection
1. Filter adult population from current population and children according to specified adult selection function
Parent selection
1. Generate pairs of individual to mate according to specified parent selection function
Child generation
1. For each mating pair, generate new children genomes according to specified crossover function
2. Apply mutation to the genomes
3. Generate new children from the mutated genomes

Figure 1: Overview of the evolutionary loop

The solution to this exercise was programmed using Python 3.4. The library *matplotlib* was used for the fitness plots.

The evolutionary loop follows the flow from Figure 1. It is pretty simple, and follows the pattern from the EA appendix.

In addition to the main loop, some data is collected at the end of each generation. For each generation, the most fit phenotype, and best, worst, and average fitness values are stored and logged. The standard deviation in the fitness values is also stored and logged.

1.2 Modularity and re-usability

To run the simulation, all you need is a set of parameters. The parameters contain information about the world (population size, genome size, crossover and mutation rates, selection mechanisms, etc.) in order to properly configure the simulation, and the problem to solve.

A problem is an instance of the Problem Python class. A Problem class contains the functions specific to the problem, like genome and phenotype representations, fitness evaluation, and population initialization.

Instances of the Problem class, selection mechanisms, and world information as numeric values can all be sent to the algorithm as parameters, so the algorithm core does not need to change in order to implement other problems or selection mechanisms. Figure 2 shows some of the Problem class code.

```
class Problem:
    def __init__(self, name,
        **kwargs):
        self.name = name

    @staticmethod
    def generate_population(
        population_size,
        genome_size, **kwargs):
        raise NotImplemented

    def fitness_function(self,
        phenotype,
        **kwargs):
        raise NotImplemented

    @staticmethod
    def genome_to_phenotype(
        genome, **kwargs):
        raise NotImplemented

    def represent_phenotype(
        self, phenotype, **kwargs):
        raise NotImplemented

    @staticmethod
    def crossover_function():
        raise NotImplemented

    @staticmethod
    def mutation_function():
        raise NotImplemented
```

Figure 2: Example code from the problem class

2 One-Max

2.1 Population size

With the crossover rate set to 80% and mutation rate set to 1%, the algorithm was able to solve the One-Max problem in under 100 generations 50 of 50 times using a population size of 400.

2.2 Crossover and mutation rate

Further experimentation showed that increasing both the crossover rate and mutation rate to 90% decreased the maximum number of generations used to the range [8, 15] (over 10 runs). Figures 3, 4, 5, and 6 show some example trials.

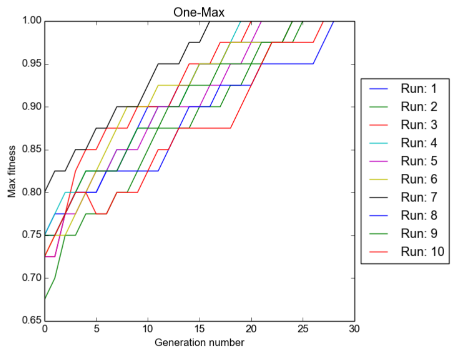

Figure 3: Crossover rate: 0.1, mutation rate: 0.1

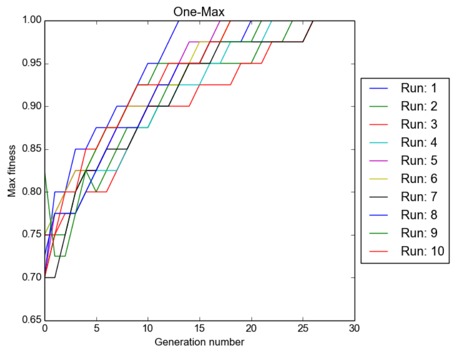

Figure 4: Crossover rate: 0.5, mutation rate: 0.1

2.3 Parent selection mechanism

Experimenting with the four different parent selection mechanisms (using different group sizes and epsilon values for tournament selection) showed that

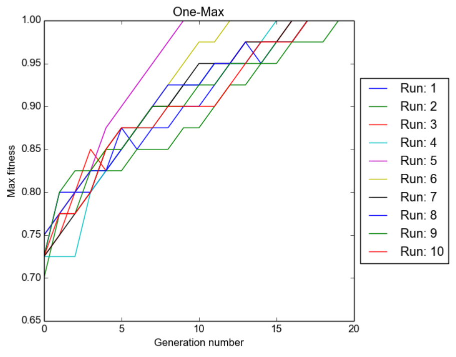

Figure 5: Crossover rate: 0.6, mutation rate: 0.3

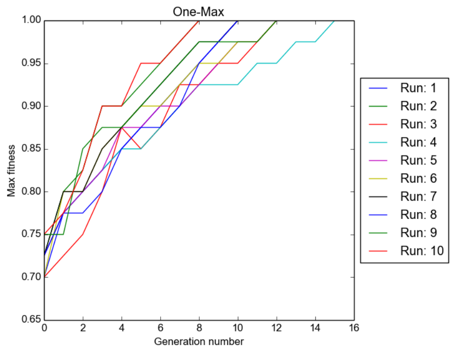

Figure 6: Crossover rate: 0.9, mutation rate: 0.9

using the tournament selection with a group size of 50 and an epsilon value of 0.1 provided the best results. The maximum number of generations used was decreased to the range [3, 6] (over 10 runs). Figures 7, 8, 9, and 10 show some example trials.

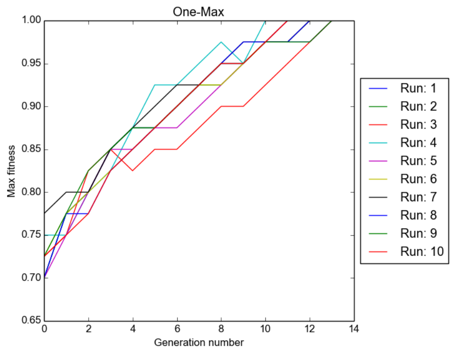

Figure 7: Fitness proportionate

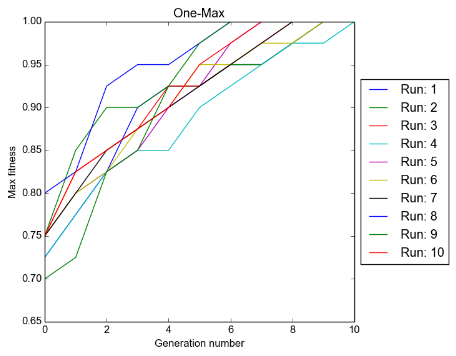

Figure 8: Sigma scaled

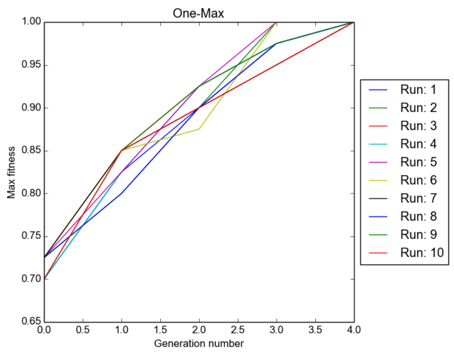

Figure 11: One-Max random target vector

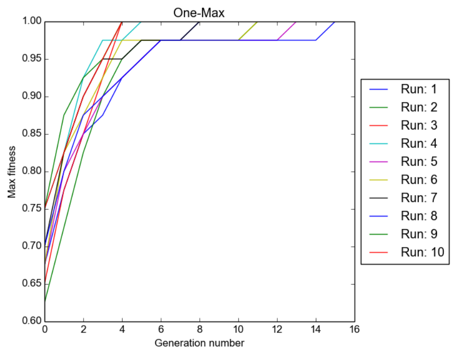

Figure 9: Tournament selection, k=50, $\epsilon = 0.1$

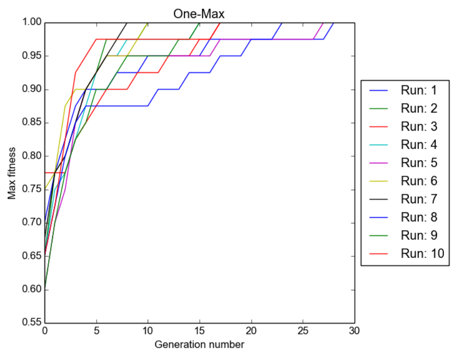

Figure 10: Tournament selection, k=20, $\epsilon = 0.3$

2.4 Random target vector

I expect the difficulty of solving the random bit vector version of the One-Max problem to be no different from the original problem. The two problems are basically the same, with the only difference being the string to search for. As can be seen in Figure 11, the results are very similar.

3 LOLZ

When running the LOLZ problem, one can see that the algorithm sometimes flattens completely. This happens when it gets stuck at local maximums. This can be seen in Figure 12. The reason why this happens is explained in the last section of the report.

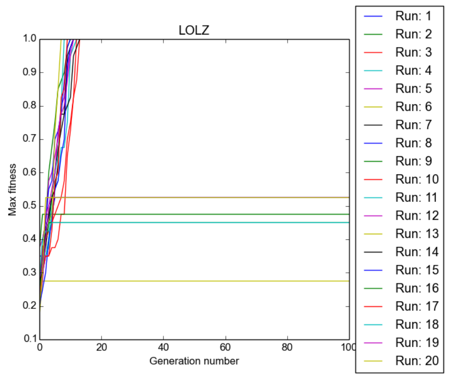

Figure 12: Plot from running the LOLZ problem using the same configuration as for the One-Max problem

4 Surprising sequences

4.1 Genetic encoding

The genotype is represented as a string, where a single genome component is represented as a character. The phenotype is represented as a tuple version of the string. See Figure 13.

ABACA $\longrightarrow$ ('A', 'B', 'A', 'C', 'A')

Figure 13: Genotype and phenotype representation in the surprising sequences

S	Population size	Generations	L
3	200	1	10
5	200	48	26
10	200	80	101
15	200	438	225
20	200	740	390

Table 1: Longest locally surprising sequences.

S	Population size	Generations	L
3	200	1	7
5	200	9	12
10	200	99	25
15	200	306	39
20	200	25	51

Table 2: Longest globally surprising sequences.

4.2 Fitness evaluation

The fitness of a phenotype is calculated by iterating the phenotype, listing all sequences AX_nB. For globally surprising sequences, n can take on all possible values. For locally surprising sequences, n=0 (only comparing two subsequent characters). The fitness is calculated by dividing the number of unique sequences AX_nB by the number of total sequences found, using the following equation:

$$fitness = \frac{set(list(sequences)) - 1}{list(subsequences) - 1} \tag{1}$$

If the number of unique sequences equals the total number of sequences, the string is totally surprising, and the fitness=1. If the string consist of only repeating characters, the fitness will be calculated to; 0.0 for locally search, and a low number > 0 for globally search.

4.3 Longest sequences found

The strings found can be found in the files `local.txt` and `global.txt`.

5 Difficulty

The fitness landscape for the One-Max problem is pretty simple. It has only a single global maximum, no local ones. Every iteration increments towards the goal. The One-Max problem is therefore the easiest one.

The LOLZ problem is the second easiest problem. The Z value sets a local maximum. The algorithm can start to trend towards the local maximum. If the algorithm is "trapped" there, the sub optimal solutions with leading zeros will win over the potentially better solutions starting with ones, and the algorithm will never reach the global maximum.

Another reason that the LOLZ problem is hard is that two fit mating parents will not necessarily produce a fit offspring. The parents might be fit in two completely different ways, and the offspring might therefore be useless. Also parents like "01111111111" will have potential to produce really good offspring, but will only rarely get the chance because of its bad fitness.

The two surprising sequence problems are the hardest ones. The fitness landscape for these problems are more advanced than for the two others, with up to many local and global maximums. The path to a final goal is tricky, and a lot of trials end up in dead alleys. The complexity of the fitness itself also adds to the difficulty of the problem. A random mutation is not necessarily as beneficial as in the other two problems.

The global problem is the most difficult, as this has to fulfil all the criteria for the local problem, plus some additional criteria.