NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
IT3708 – SUBSYMBOLIC METHODS IN AI

Project 2: Programming an Evolutionary Algorithm (EA)

Per Magnus Veierland
permve@stud.ntnu.no

February 25, 2016

Program implementation

The EA framework for this assignment is written with templated classes in C++14. The `individual` class holds the genotype representation, the optional phenotype representation, and the fitness score. Genotypes are represented using classes such as `dynamic_bit_vector` with a matched function to spawn an instance of the representation, e.g. `dynamic_bit_vector_creator`. There are classes to handle adult selection; `full_generational_replacement`, `generational_mixing`, etc. Truthfully, the adult selection classes are best described as "generational replacement policy" classes, since they decide how to apply a reproduction function and develop the resulting children until the next generation has been built according to the policy. Further there are parent selection classes, `fitness_proportionate`, `rank`, etc. which are used to select a single parent from a population. Reproduction classes, for which there currently is only `sexual`, are used to produce a set of children according to the reproduction policy, given a population and a parent selection policy. The reproduction class is responsible for invoking the selection of two parents, and applying crossover and mutation operators to form the resulting children. A `system` class is used to contain the parts and perform bookkeeping, but all evolution logic is handled by the parts.

As an example, the following shows how the One-Max fitness function is written as a lambda:

```
[] (const auto& genotype)
{ return static_cast<double>(genotype.count()) / static_cast<double>(genotype.size()); }
```

One-Max

When analyzing the One-Max problem with full-generational replacement, there will always be a chance of regression where fit individuals are replaced by less fit individuals. To consistently find a solution to the 40-bit One-Max problem in less than 100 generations with full-generational replacement, proportional fitness parent selection, 1 crossover point, a crossover rate of 1.0, and a mutation rate of 0.001, a population of 300 individuals was found to be appropriate. Across 100 runs, a population size of 300 individuals was found to consistently yield a valid solution. The mentioned parameter values are used as defaults in all comparisons.

The comparison of mutation rates (Figure 1) shows that a mutation rate of 0.001 clearly outperforms 0.01 and 0.005 for this problem. It appears that a mutation rate between 0.0001 and 0.0005 results in the best performance for this problem.

Comparisons of crossover rates (Figure 2) did not show any clearly superior options, and that crossover rates between 0.8 and 1.0 seems to result in the best performance.

Parent selection methods are compared in Figure 3. Rank selection clearly outperforms proportionate fitness selection, and sigma selection clearly outperforms rank selection. It can further be observed that tournament selection is the best parent selection mechanism for the problem, and that smaller tournament groups perform better than larger tournament groups.

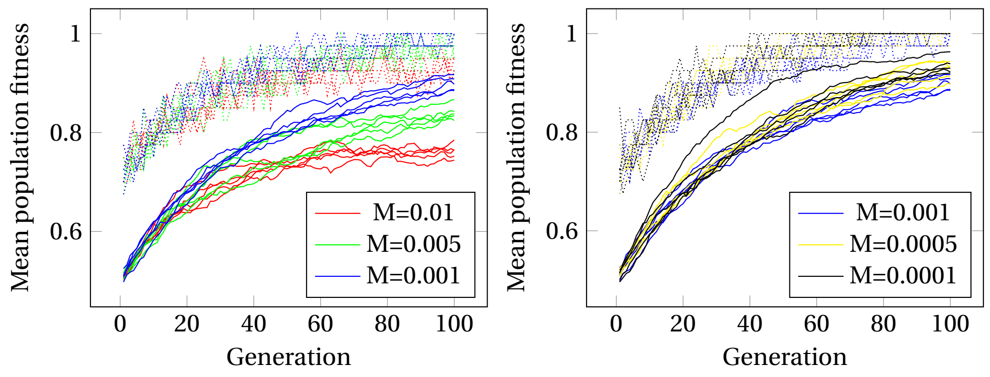

Figure 1: ONE-MAX mutation rate comparison

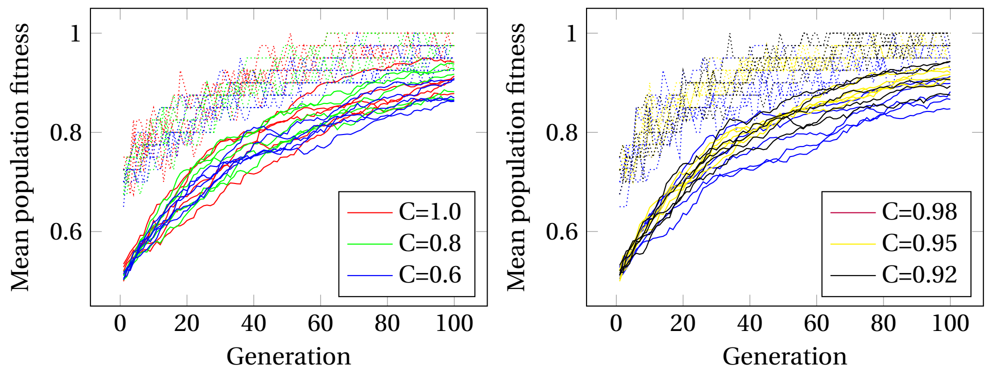

Figure 2: ONE-MAX crossover rate comparison

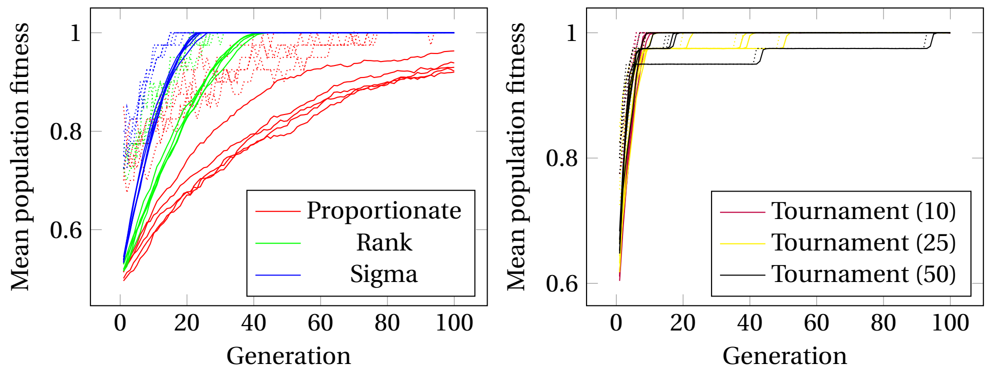

Figure 3: ONE-MAX parent selection comparison

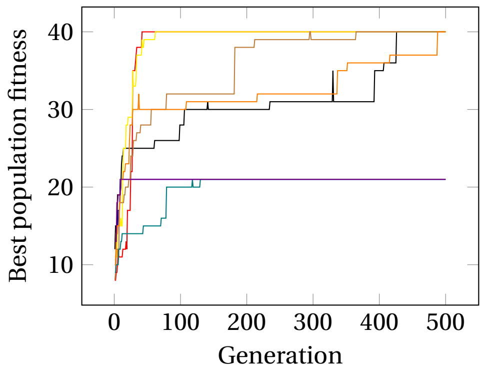

Figure 4: LOLZ comparison of 8 runs

Leading-One Leading-Zero (LOLZ)

The results from evaluating the LOLZ problem 8 times with the configuration described for One-Max is shown in Figure 4. The plots show that for 2/8 attempts, the EA gets stuck within the local maxima of the “zero genotype”. Initially the population genotypes will be filled with both zeros and ones. Once evolution starts, a majority of either will steer a positive feedback loop which will further favor one of the sides, which will split in roughly 50/50 of runs. Up until the Z-point; both ones and zero genotypes will be able to score the same fitness. However, once stuck in the local maximum where longer zero genotypes will not continue to increase in fitness, building “one”-strings will be very difficult, as they will score worse than “zero”-strings until they can cross the Z-point.

Surprising Sequences

Figure 5 shows the integer vector genotype used to represent individuals for the surprising sequence problem. In addition to representing an individual as an integer vector, the symbol set size S is also stored. This value is used when mutating the individual, such that mutating any position in the vector involves setting its value to a random member of the symbol set.

As no obvious advantage was seen in using a separate phenotype representation, individuals use the same representation for its genotype and effective phenotype.

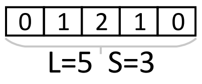

Figure 5: Surprising sequences integer vector genotype

The fitness function for both local and global surprising sequences is based on the function `collisions`, which takes a sequence parameter, the symbol set size (S), and the

S	L	Gen.	Sequence
3	10	0	2, 0, 0, 2, 2, 1, 0, 1, 1, 2
5	26	39	4, 4, 0, 2, 1, 3, 4, 3, 3, 1, 2, 3, 2, 4, 1, 1, 0, 1, 4, 2, 2, 0, 3, 0, 0, 4
10	100	2825	2, 0, 8, 9, 1, 5, 5, 0, 0, 1, 4, 3, 6, 8, 2, 8, 8, 7, 5, 9, 6, 2, 9, 7, 3, 2, 5, 3, 5, 6, 9, 8, 1, 8, 5, 1, 9, 3, 1, 7, 6, 6, 4, 0, 2, 2, 3, 9, 0, 5, 8, 0, 7, 1, 6, 3, 3, 8, 4, 7, 9, 4, 5, 4, 4, 2, 6, 0, 3, 4, 6, 7, 8, 6, 5, 2, 1, 0, 9, 2, 7, 4, 1, 2, 4, 8, 3, 0, 4, 9, 5, 7, 7, 0, 6, 1, 1, 3, 7, 2
15	215	2435	0, 10, 10, 14, 11, 13, 6, 8, 8, 1, 12, 1, 2, 12, 2, 13, 0, 7, 9, 11, 1, 14, 7, 2, 9, 4, 2, 0, 1, 0, 3, 13, 3, 10, 5, 0, 11, 4, 14, 13, 9, 13, 14, 8, 9, 0, 14, 2, 6, 10, 12, 14, 6, 9, 9, 12, 3, 0, 2, 7, 3, 14, 12, 5, 5, 9, 6, 0, 5, 3, 6, 14, 3, 11, 12, 13, 1, 3, 1, 13, 12, 12, 0, 12, 11, 11, 7, 14, 4, 1, 9, 8, 6, 2, 4, 12, 8, 2, 14, 5, 12, 6, 7, 4, 4, 0, 9, 3, 4, 11, 0, 0, 6, 4, 8, 4, 7, 10, 11, 3, 3, 5, 7, 6, 1, 5, 8, 10, 3, 8, 7, 5, 2, 11, 14, 9, 7, 1, 8, 12, 9, 14, 0, 8, 0, 4, 10, 2, 8, 11, 9, 5, 4, 13, 7, 7, 13, 13, 8, 3, 2, 10, 7, 12, 10, 4, 9, 10, 0, 13, 10, 6, 3, 9, 2, 1, 6, 11, 5, 14, 1, 1, 10, 8, 5, 13, 2, 2, 3, 7, 11, 6, 12, 7, 8, 14, 14, 10, 1, 4, 6, 5, 6, 6, 13, 11, 2, 5, 10, 13, 5, 1, 11, 8, 13
20	380	2923	7, 6, 14, 3, 7, 8, 13, 3, 9, 12, 16, 0, 13, 19, 19, 7, 16, 2, 9, 9, 19, 16, 10, 10, 7, 15, 12, 6, 8, 18, 18, 9, 14, 12, 19, 12, 18, 15, 14, 9, 6, 11, 17, 10, 11, 10, 1, 2, 11, 11, 5, 17, 9, 2, 12, 5, 8, 16, 19, 2, 17, 1, 14, 19, 3, 16, 12, 2, 8, 0, 6, 12, 12, 9, 1, 0, 11, 4, 0, 12, 10, 5, 13, 0, 19, 0, 15, 11, 16, 7, 1, 13, 7, 5, 15, 6, 17, 18, 3, 14, 2, 3, 10, 9, 7, 9, 3, 11, 3, 19, 1, 18, 5, 4, 13, 11, 19, 5, 5, 6, 2, 15, 5, 10, 12, 14, 7, 12, 17, 17, 5, 3, 3, 18, 2, 2, 13, 13, 9, 17, 12, 7, 18, 16, 8, 1, 3, 15, 19, 6, 0, 0, 17, 14, 16, 1, 8, 10, 8, 6, 9, 5, 1, 1, 16, 6, 3, 13, 16, 17, 8, 9, 18, 11, 12, 3, 12, 0, 16, 13, 1, 11, 15, 4, 17, 4, 14, 17, 6, 16, 3, 6, 1, 4, 2, 16, 14, 4, 8, 7, 14, 10, 18, 14, 11, 1, 6, 19, 11, 0, 1, 12, 11, 8, 17, 19, 9, 15, 2, 18, 6, 18, 0, 9, 16, 15, 7, 11, 7, 17, 13, 8, 5, 16, 16, 5, 18, 8, 12, 1, 7, 3, 8, 15, 8, 4, 16, 9, 0, 5, 11, 9, 11, 14, 6, 15, 0, 2, 10, 3, 2, 4, 19, 14, 15, 18, 10, 19, 4, 15, 1, 19, 15, 16, 18, 1, 9, 4, 6, 5, 19, 8, 2, 6, 6, 4, 12, 15, 9, 8, 14, 1, 15, 13, 12, 4, 1, 10, 6, 7, 19, 10, 13, 10, 0, 3, 4, 18, 7, 0, 18, 13, 5, 7, 4, 11, 18, 17, 0, 8, 11, 2, 0, 7, 7, 2, 14, 8, 8, 3, 1, 17, 16, 4, 4, 5, 2, 19, 13, 17, 11, 6, 13, 6, 10, 2, 1, 5, 9, 13, 15, 17, 15, 3, 17, 3, 0, 10, 17, 2, 5, 14, 5, 0, 14, 18, 19, 18, 4, 7, 13, 18, 12, 13, 14, 14, 0, 4, 3, 5
3	7	0	1, 0, 2, 2, 1, 2, 0
5	12	5	3, 4, 4, 0, 1, 2, 0, 3, 0, 2, 1, 4
10	25	386	4, 7, 2, 6, 8, 0, 8, 4, 6, 9, 1, 0, 3, 1, 5, 8, 2, 2, 5, 7, 4, 9, 3, 8, 6
15	37	6042	2, 5, 11, 8, 6, 10, 4, 7, 13, 4, 9, 12, 10, 1, 0, 9, 7, 14, 3, 3, 13, 12, 2, 9, 5, 4, 11, 1, 11, 0, 10, 6, 12, 8, 14, 6, 7
20	50	7135	1, 12, 15, 17, 14, 0, 3, 19, 6, 9, 11, 4, 1, 13, 2, 3, 5, 2, 19, 10, 17, 8, 7, 10, 14, 18, 13, 0, 8, 4, 5, 18, 0, 16, 13, 9, 6, 7, 6, 15, 5, 19, 11, 14, 12, 17, 3, 1, 1, 16

Table 1: Local (top) and global (bottom) surprising sequences (population: 200)

pair distance to test for; d. The function counts collisions by iterating over each position, i, in the sequence. If the pair consisting of the symbol at position i and the symbol at position $i + d + 1$ has not been encountered, it is marked as seen in a lookup table. However, if the pair has already been seen as part of the `collisions` function invocation, it is recorded in the sum of collisions which the function ends up returning.

The local fitness function (Equation 1) is based on a single call to the `collisions` function for the case $d = 0$, while the global fitness function (Equation 2) invokes the `collisions` function once for each d in the range $[0, L-3]$. Each fitness function will result in a fitness of 1 in the case of no collisions, and scoring any addition in the number of collisions with a worse fitness.

$$fitness_{local} = \frac{1}{1 + \texttt{collisions}(sequence, S, 0)} \tag{1}$$

$$fitness_{global} = \frac{1}{1 + \sum_{d=0}^{L-3} \texttt{collisions}(sequence, S, d)} \tag{2}$$

DIFFICULTY

The ONE-MAX problem was clearly the easiest problem, as there are no local maxima and only one correct solution for a given length. It is easy to write a good fitness function which reflects individual fitness well.

LOLZ is a harder problem than ONE-MAX since it is designed to have a distinct local maxima, and because representations in the local maxima are exact opposites to the global maxima; it will be very hard to escape this local maxima for an EA.

Surprising sequences is the hardest problem of the three due to the complexity and cost of the fitness evaluation, and due to the complexity of valid sequences. Global sequences are harder than local sequences since the number of pairs which must be considered rises much faster for the global case than for the local case.