CS4205 Practical 1 Report

Yağmur Karalar, Alex Bieniek

May 2019

All experiments were run under Linux Ubuntu 18.04 (64 bit) on an Intel Core i5-8250U CPU at 1.6 GHz, with 1 processor of 4 cores, with a 7 GB RAM and a 6 MB L3 cache.

1 Optimization Problem

We begin our analysis with a discussion about the function which we are optimizing. For the described function f_{P1}, we look to find the gene sequence x which provides the maximum value of f_{P1}, given some assignments of the other parameters m, k, d. We inspect the performance of our given simple genetic algorithm with the use of different variation operators and population sizes.

We aim to find the genotype x which maximizes the function f_{P1}:

$$f_{P1}(x, k, m, d) = \Sigma_{i=0}^{m-1} f_{P1}^{Sub}((x_{ik}..x_{ik+k-1}), k, d)$$

where:

$$f_{P1}^{Sub}(x, k, m, d) = \begin{cases} 1 & \text{if } u(b) = k \\ (1-d)\frac{k-1-u(b)}{k-1} & \text{otherwise} \end{cases}$$

and $u(b)$ counts the number of "true" bits in the binary sequence.

The outermost function in the fitness evaluation of an individual, f_{P1}, effectively splits the binary sequence into m substrings of size k, finds the value of f_{P1}^{Sub} for these substrings, and adds them up. This inner function f_{P1}^{Sub} has a slightly odd evaluation: for a binary string x, if all bits are "true", then the string has a value of 1. Otherwise, the value of the binary string is some other value, as shown above. We will assume that d will be between 0 and 1 because the prompt specifies that $d \in \{1/k, 1 - 1/k\}$, which will always be in this range for $k > 1$. With these conditions, we see that:

- $0 < 1 - d < 1$
- in the case where $u(b) < k$, $0 < u(b) < k - 1$, so the numerator is between $k - 1$ and 0

We see that the outcome of our "otherwise" case will always be in $[0, 1)$.

The function which we want to maximize f_{P1} is computed on the summation of evaluations the inner function f_{P1}^{Sub}, so f_{P1} is maximal when all m substrings of size k have all "true" bits. However, for a given substring that is not entirely composed of 1s, we see that the value of the substring increases as the number of "true" bits decreases. With this, we would expect that, through crossover operations of the genotypes and tournament selection, the genetic algorithm will favor members in the population with less "true" bits. However, if an individual in the population happens to have substrings of all "true" bits through random initialization or the randomized crossover of genes, these individuals will often be selected in the tournament selection and live on in the population. Our algorithm will halt if we generate such an individual with a genotype of all "true" bits.

2 Experimental Research

In our experiment, we try all possible combinations of the possible values, and collect data for each combination five times:

- $n \in \{20, 40, 60, 80, 100\}$
- $m \in \{1, 2, 4, 8, 16\}$
- $k \in \{5\}$
- $d \in \{1/k, 1 - \frac{1}{k}\}$

We would expect that the number of evaluations increase as a function of m as well as k. Due to a lack of mutations, the genotype search space will be limited by the size of the populations being sampled. It could be the case that random generation produces many of the all-true substrings which perform well, but crossover operators will destroy these substrings and begin to favor genotypes with increasingly many "false" values. The use of one-point crossover would do better at preserving these all "true" substrings, but if a given run of our algorithm effectively removes the all "true" genotype from the search space, then a uniform crossover will more quickly approach the solutions in the search space comprising of all "false" values, which may or may not have a high fitness relative to the optimum, depending on the value of d.

As a termination condition, we use progressively longer timeouts for larger values of m, at $\{3, 5, 10, 15, 20\}$ seconds. Additionally, we also make the SGA terminate if, after an evaluation, every member of the population has the same fitness.

3 Analysis

We analyze the effects of various hyperparameters on the simple genetic algorithm, given some assignments of m, k, and d. In all of our plots, we plot various metrics against the population size as well as the type of selection operator being used.

3.1 Closeness to Optimality

In our analysis of optimality, we normalize the elite fitness of a population. That is, we see how close the fitness of the elite in a population is to the optimal fitness by dividing the elite fitness by the optimal fitness. These values are plotted in figure 1.

m is the number of substrings to be evaluated in our fitness function. When we plot the elite fitneses of the SGA runs by the choice of d, we see that, as m increases, it becomes less common for our populations to find elite individuals which have optimal fitness. Higher population sizes almost always get closer to optimal elite fitness.

d can be regarded as a "discount factor" in the inner function of our fitness function, and a lower value of d decreases the magnitude of fitness of substrings with some "false" values. Intuitively, we can say that a lower "discount factor" disincentivizes convergence to the local optima with entirely "false" substrings.

The plots help us understand some of the behavior of our SGA:

- more genotype substrings make it harder for our algorithm to converge to genotypes with all "true" bits because there is a lower probability to find the genotype of all "true" bits as the fitness function causes our algorithm to converge toward all "false" bits.
- Higher populations will create more random individuals, so we increase the odds of creating elite individuals with optimal genotypes from the start, and more diversity in genotypes will no longer favor converging towards the local optimum of all "false" bits.
- the elite fitness of the populations are generally better when the discount factor is lower, but only when the algorithm doesn't converge to the optimal solution. Higher discount values will disincentivize the algorithm from converging to the local optimum of all "false" values, but because it's so hard to find the global optimum as m increases, the achieved elite fitness will be lower in most cases.

The plots produced to measure optimality don't seem to divulge especially strong trends relating to the type of variation operator used, so we refrain from deriving any intuitions here.

3.2 Time Until Convergence

We can observe that, as expected, the time until convergence increases as the number of subsections m increases as well. We plot time until convergence in figure 2.

Generally, as the initial population size increases, times until convergence tend to increase. The plot seems to show a drop in time until convergence when $p = 100$, but this is because the plot ignores runs which did not converge, and the sample of runs with $p = 100$ happens to have some instances which converged quite early. With that, the plot is a bit flawed, but the intuition remains that higher population sizes lead to longer times until convergence.

Regarding the comparison of the two different crossover types, there is no clear superiority of one over the other. The two methods give approximately the same results for all m values until population size is 60. For larger population sizes, one point crossover gives slightly better results overall compared to uniform crossover for $m = 1$, 2 and 4 with two d values.

3.3 Frequency of Convergence

There are 2 graphs in the appendix regarding the convergence frequency for the 2 different discount factor values, seen in figure 3 and figure 4. We'll discuss the common traits and differences. For both d values, the optimal solution is reached only when the number of subsections is 1 or 2. We believe that, as the number of substrings m increases, it becomes less likely that the SGA finds the optimal solution. The SGA also converges less frequently because there are more unique ways in which the genotypes can crossover, so a locally maximal fitness optimum will take longer to find.

Furthermore, for m values 1 and 2, the convergence to optimal ratio increases as the population size increases. This likely occurs because higher population sizes are able to randomly start with or produce optimal genotypes.

The ratio of convergence to equal fitnesses of individuals is higher compared to the convergence to optimality, as expected. We observe that, as the population size increases, the SGA is less likely to converge because there are more genotypes being used in population variation, so the search space is larger and slower to converge.

For the two different discount values, it can be seen that the algorithm reaches a full convergence where all the solutions have equal fitness. Regarding the larger population sizes, the relationship is not entirely clear. This ambiguity can be resolved with more experiments.

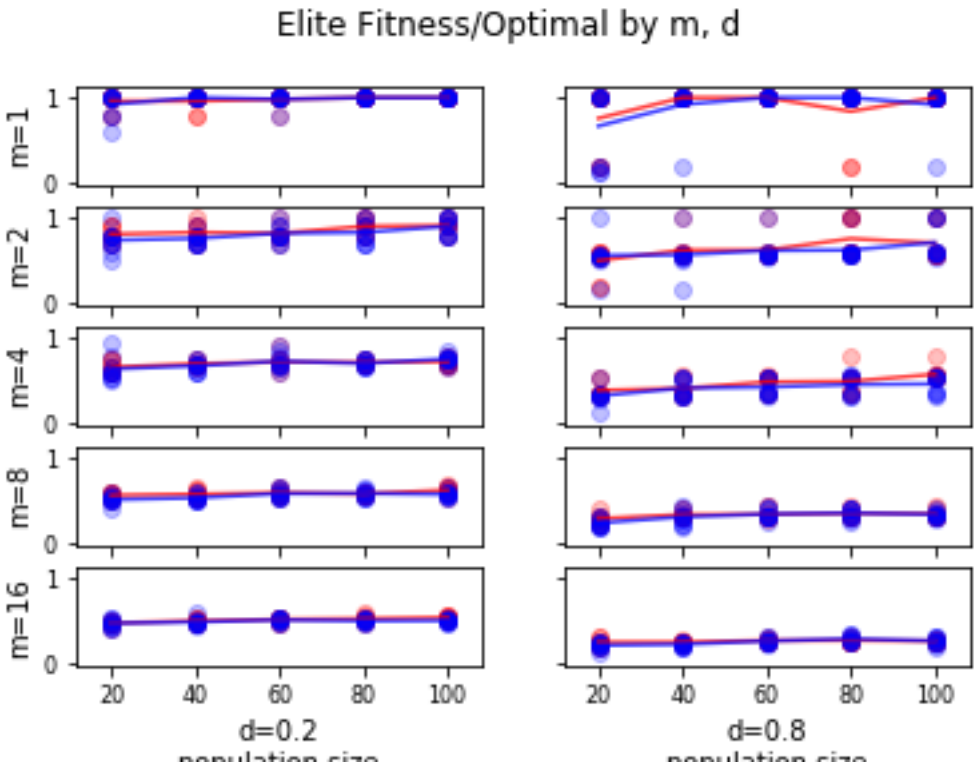

Figure 1: fitness of elites, as a fraction of the optimal fitness, against population by value of m, d, and variation operator
red - uniform crossover
blue - one-point crossover

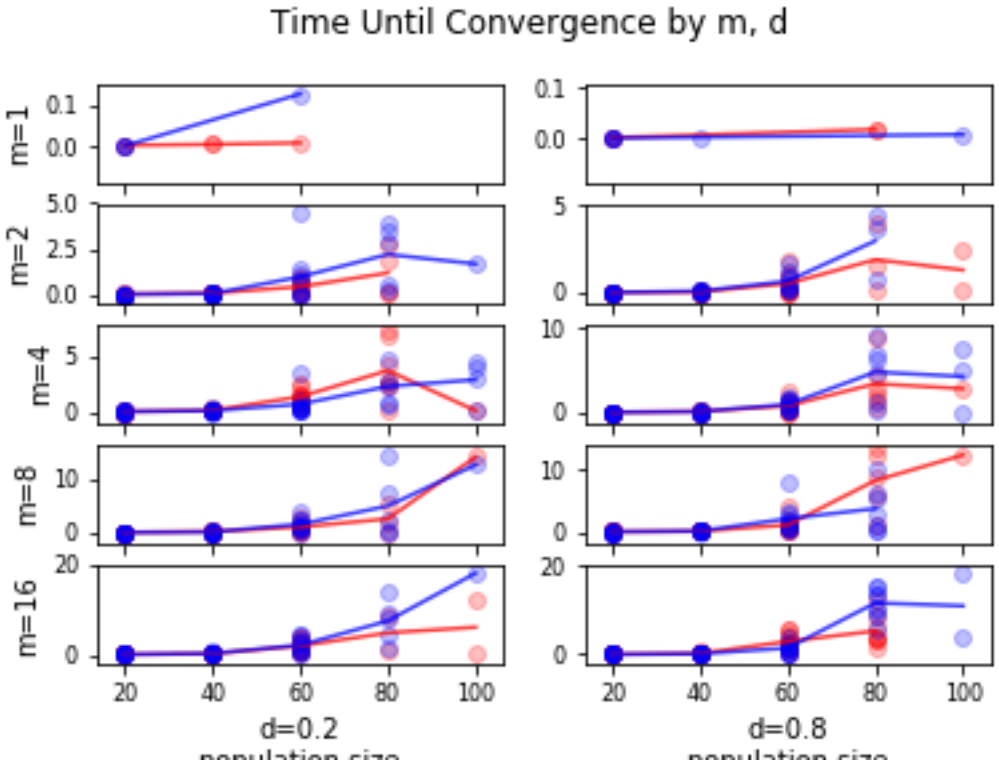

Figure 2: Time until convergence of solution against population
by value of m, d, and variation operator
red - uniform crossover
blue - one-point crossover

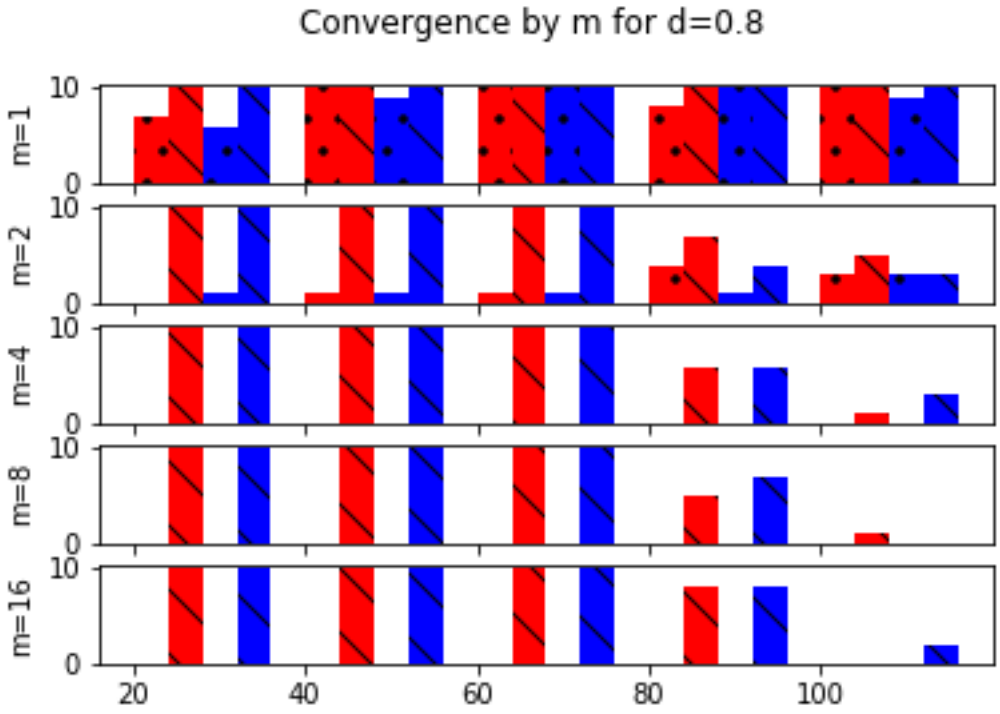

Figure 3: Frequency of finding optimal solution and convergence, by variation operator
red - uniform crossover
blue - one-point crossover
dotted - optimal solution found
lined - converged to equal fitnesses of individuals

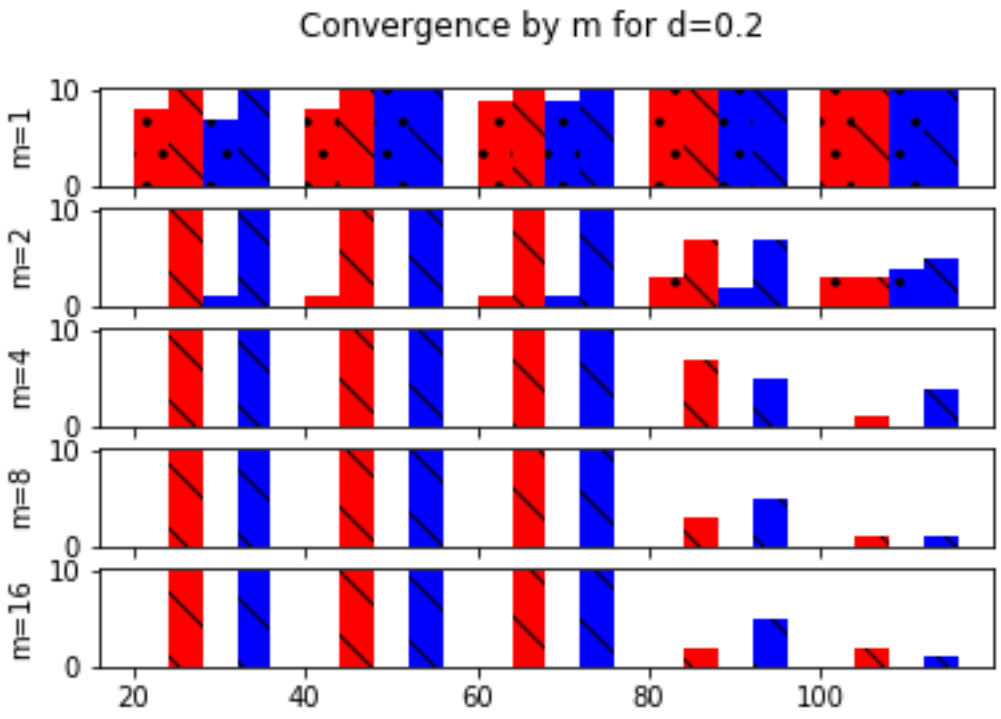

Figure 4: Frequency of finding optimal solution and convergence, by variation operator
red - uniform crossover
blue - one-point crossover
dotted - optimal solution found
lined - converged to equal fitnesses of individuals