Evolutionary Computing

Freek Hens s1033509
Guido Klein s1021586
Floris Rossel s1010794

All code is available on Github. The corresponding files for exercises 4.1 – 4.4 are `ex1.py, ex2.py, ex3.py, ex4.py`. The figures accompanying the exercises are placed in a section at the end of the report, for clarity.

Exercise 4.1

1. See table 1 and figure 1.

Table 1: Proportional selection probabilities for functions f_i and individuals x_i.

	$x = 2$	$x = 3$	$x = 4$
f_1	0.22	0.33	0.44
f_2	0.14	0.31	0.55
f_3	0.14	0.31	0.55
f_4	0.27	0.33	0.40

2. Different fitness functions can alter the candidate selection, by either increasing or decreasing the relative probabilities of choosing a particular candidate. For example, f_2 and f_3 decrease the relative probability of the less fit candidates ($x = 2$ and $x = 3$), while increasing the relative probability of the fittest candidate ($x = 4$) compared to f_1.

Exercise 4.2

1. See figure 2.
2. In our runs, every run found the goal bit string, so 10 times. See figure 3.
3. The difference in performance when using this modification is that now, in every iteration, the random changes (dependent on mu) are set as the string for the next generation. This means that progress (fitness) is not taken into account. This results in the performance hovering around the chance level: 50 out of 100 bits correct (due to the 2 options for each bit: 0 or 1). See Figure 4

Exercise 4.3

All experiments done for assignment 4.3 use the following parameters: $mu = 0.01$, which is the probability of a mutation (in this case a swap of two cities) and $mu_{cross} = 0.1$, which is the probability of cross-over (cross-over is implemented as described in the lecture).

1. Figure 5 shows the simple EA for the given dataset (from Brightspace), and Figure 6 for the bays29 dataset [1]. Figure 7 shows the MA implementation for the given dataset, Figure 8 for the bays29 dataset [1].

   Our implementation of the MA (EA with local search) ran very inefficiently. It took around 40 minutes of runtime to complete 1500 generations with a pool of 50. For this runtime-related reason, we chose to run the MA for 100 generations with a pool size of 25. Despite this, MA is able to reach a higher fitness compared to the pure EA implementation. In 1 of the 10 runs on the given dataset, MA reached a maximum fitness of around 0.0088, while EA was only able to reach a maximum fitness of around 0.0077 (see Figures 5 and 7). Additionally, on 1 of the 10 runs on the (smaller) Bays29 dataset, MA reached a maximum fitness of around 0.000122, while EA only reached a maximum fitness of 0.000116 (see Figures 6 and 8. Moreover, the runs of the MA on both problems have less variance than the runs of the EA, which indicate that the MA performs more consistently.

   The best solutions found by both algorithms on the given data set can be seen in Figures 9 and 11. The best solutions found on the bays29 dataset can be seen in Figures 10 and 12.

   Given these results, we do see local search as a valuable addition to the pure EA.

2. We'd argue that the simple EA with the same number of generations is not a fair baseline when comparing the MA because the number of computations per generation for the EA is way lower than in the MA. For the latter, a single run would take approximately 45 minutes on our dataset, while the EA took less than a minute. A fair comparison would give both algorithms a similar "computational pool" (for example, by giving x seconds of CPU runtime to both algorithms).

3. Recent papers that evaluate memetic algorithms on the TSP problem find improved solutions with shorter run times. Gutin et al. [2] developed a MA that uses several local search heuristics for improving the solutions of each generation, including 2-opt, and uses speed-up heuristics that exclude some modifications from being applied. Another study also found that a hybrid approach [MA] outperforms both only local [heuristics] and only global search [EA or PACO] [3].

Exercise 4.4

1. Our target string is "`group two cool`", which has a length of 14. The distribution of t_{finish} has a mean of 18.65 and the shape (variance) can be seen in Figure 13.

2. Figure 16 shows that using $\mu = 0$, the GA finds the solution 18 out of 20 runs. In these 18 runs, the goal string was found using only crossover mutation. The 2 runs that were unable to find the solution within 100 generations presumably would never find the solution due to one letter that is in the target string not being in the randomly initialized strings. The distribution of t_{finish} has a mean of 44.95 and the shape (variance) can be seen in Figure 15. It should be noted that the mean and standard deviation is quite arbitrary. given the fact that some runs never converge, therefore the mean and standard deviation are influenced by the maximum number of generations.

3. Figure 18 shows that the algorithm with $\mu = 0.1$ finds the solution each run. The distribution of t_{finish} has a mean of 19.1 and the shape (variance) can be seen in Figure 13.

4. The results are shown in Figure 19 and the parameter values and target string are shown in table 2. We find the lowest median t_{finish} for $\mu = 0.1$, where the values for μ were tested from $0.0 - 1.0$ in steps of 0.1. We assume $\mu = 0.01$ or $\mu = 0.02$ is the optimal value, since the median t_{finish} is the lowest in $\mu = 0.01$, but the whiskers of $\mu = 0.02$ indicate that this value reaches good results slightly more consistent.

Table 2: Parameter values and target string for experiment 4.4.4

runs	max generations	p_{cross}	K	target string
20	100	1	2	`group two cool`

5. Figure 20 shows the results for the same experiment with $K = 5$ (and all other parameters equal to 4.4.4). This means that the tournament selection is done from a larger pool of 5 random candidates, as opposed to 2 from which the fittest candidate is chosen to become the parent. The optimal μ is still around 0.01 or 0.02 in our 20 runs, but the other values for μ are now more evenly distributed: there is no clear up-curving trend like in experiment 4.4.4; it seems more linear. This can be explained by the larger tournament selection pool: since K is larger, there is a larger chance that the fittest individual out of all individuals is in the pool. This, together with the trend that a lower μ gets better results, indicates that a larger μ (i.e. more random mutations) hinders the convergence performance of the algorithm.

This shows that some exploration is necessary to cover the complete search space (as $\mu = 0$ does not converge every time), but it is also important (especially when selection pressure is low) to exploit the found results.

References

[1] *Bays29 dataset for TSP*. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`. Accessed: 13-03-2023.

[2] Gregory Gutin and Daniel Karapetyan. “A memetic algorithm for the generalized traveling salesman problem”. In: *Natural Computing* 9 (2010), pp. 47–60.

[3] Yuezhong Wu, Thomas Weise, and Raymond Chiong. “Local search for the Traveling Salesman Problem: A comparative study”. In: *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*. 2015, pp. 213–220. DOI: 10.1109/ICCI-CC.2015.7259388.

Figures

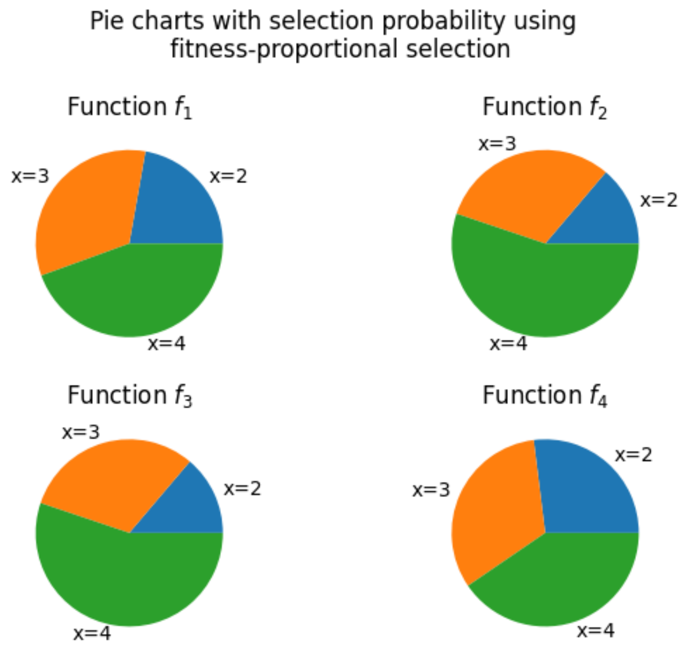

Figure 1: Pie charts of the three individuals x_i with their selection probability.

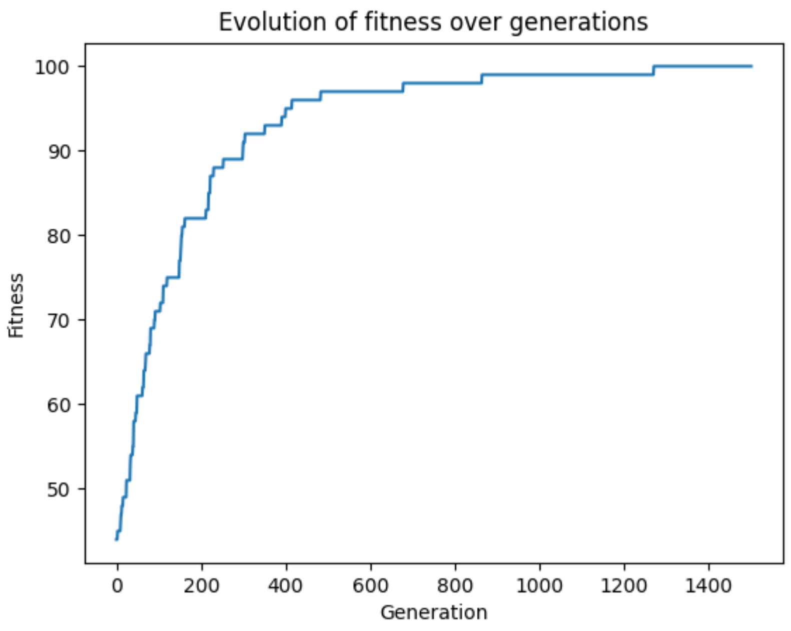

Figure 2: The best fitness against the number of generations for bit strings with $l = 100$, $\mu = 1/l$.

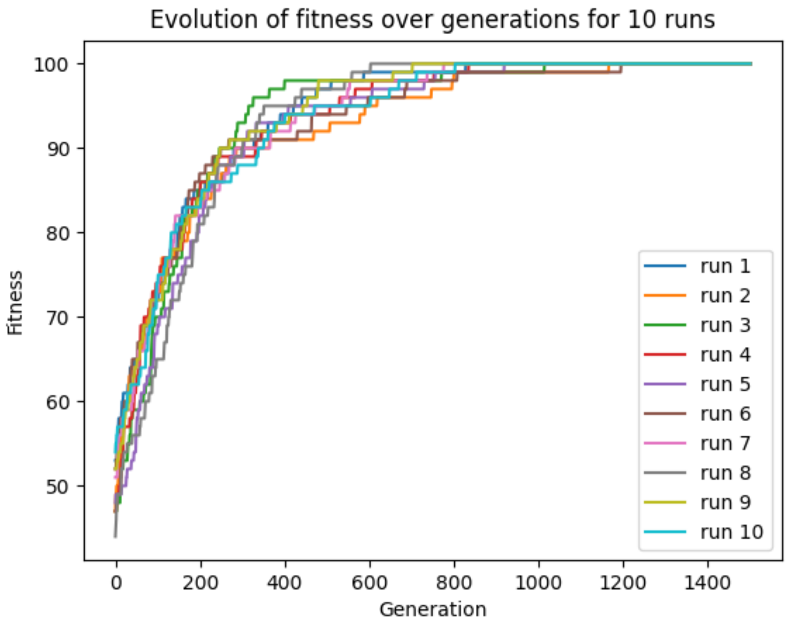

Figure 3: The best fitness for $n = 10$ runs.

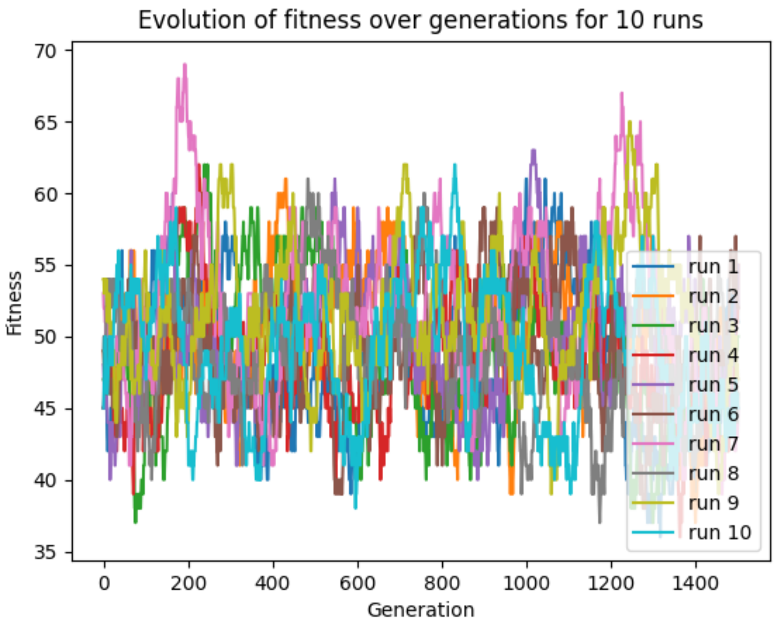

Figure 4: The best fitness for $n = 10$ runs, always replacing x with x_m.

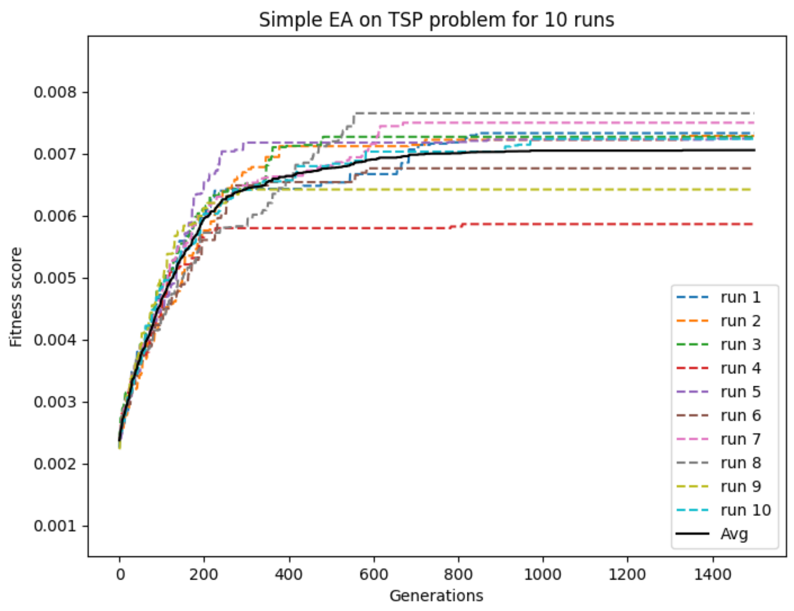

Figure 5: EA on given dataset. 1500 generations, pool of 50.

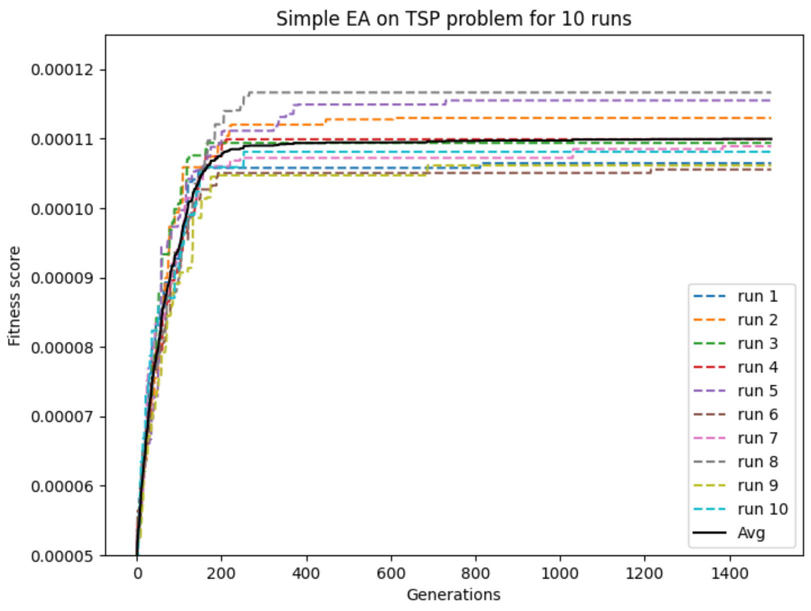

Figure 6: EA on the bays29 dataset [1]. 1500 generations, pool of 50.

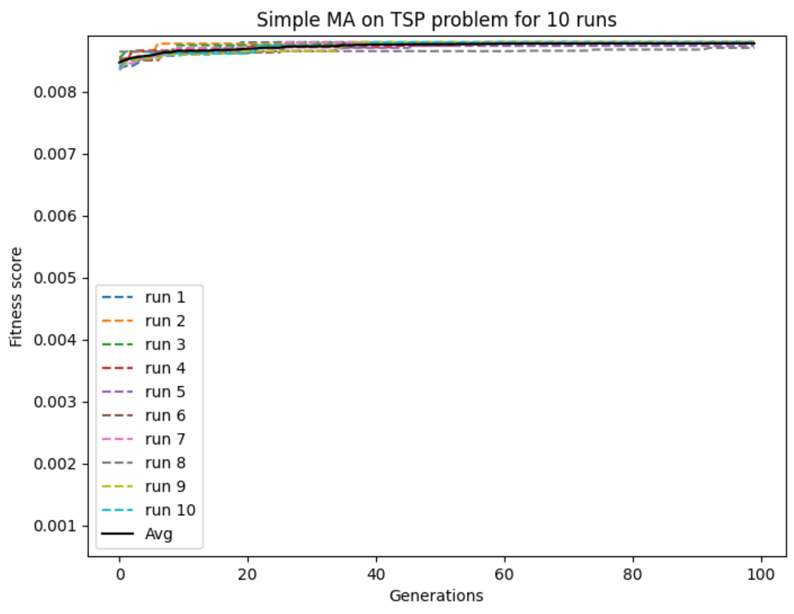

Figure 7: MA on given dataset. 100 generations, pool of 25.

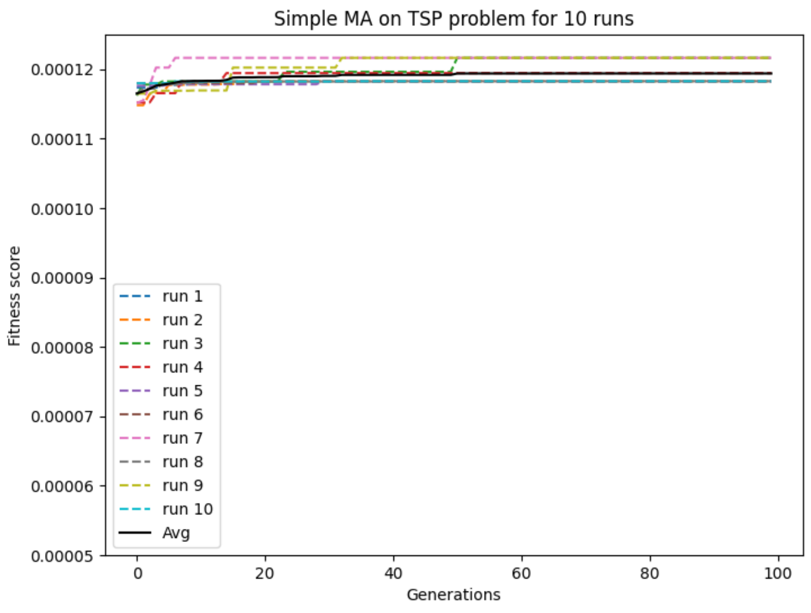

Figure 8: MA on the bays29 dataset [1]. 100 generations, pool of 25.

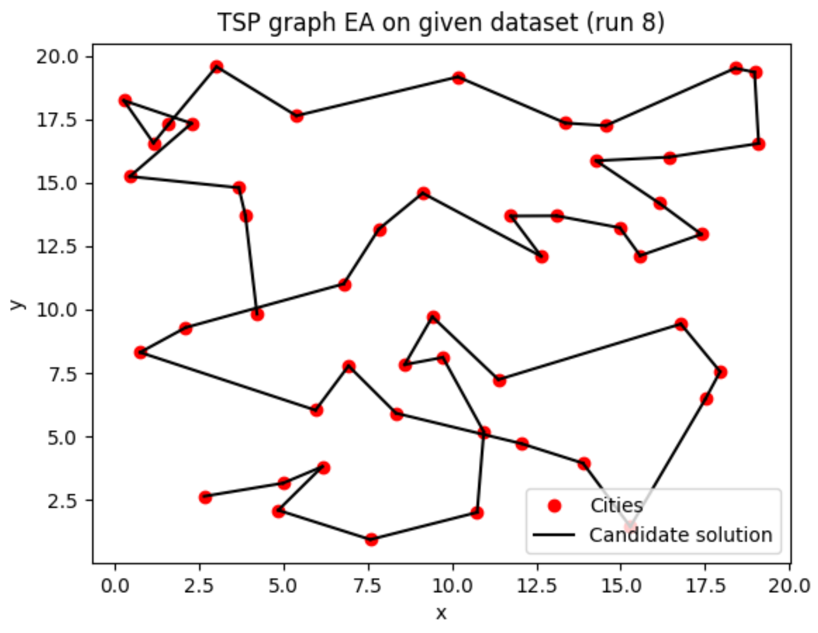

Figure 9: Solution found by EA on given dataset after 1500 generations, with a pool of 50.

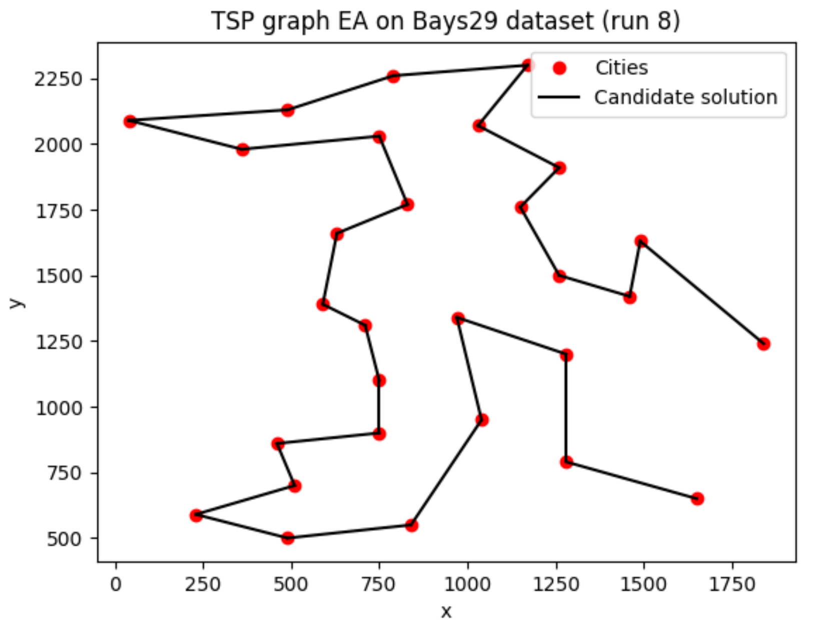

Figure 10: Solution found by EA on bays29 dataset after 1500 generations, with a pool of 50.

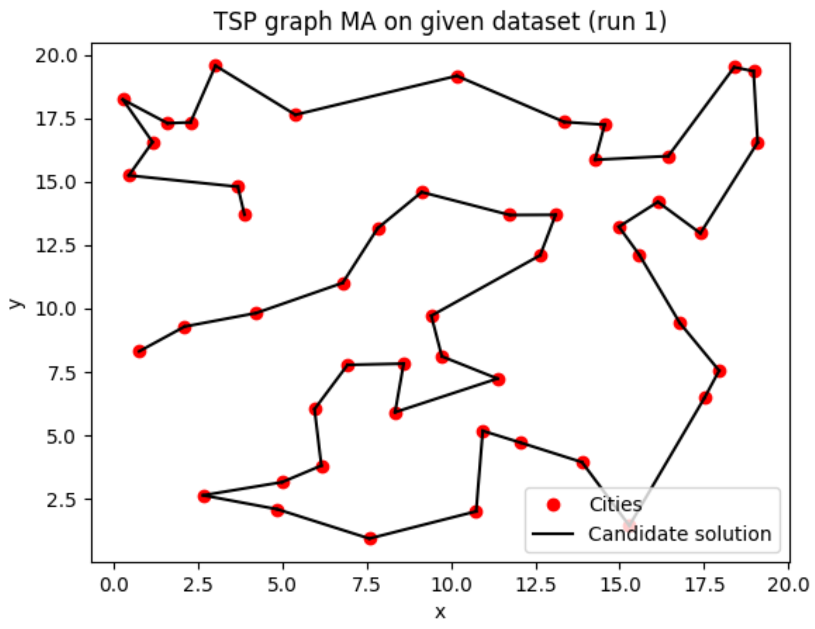

Figure 11: Solution found by MA on given dataset after 100 generations, with a pool of 25.

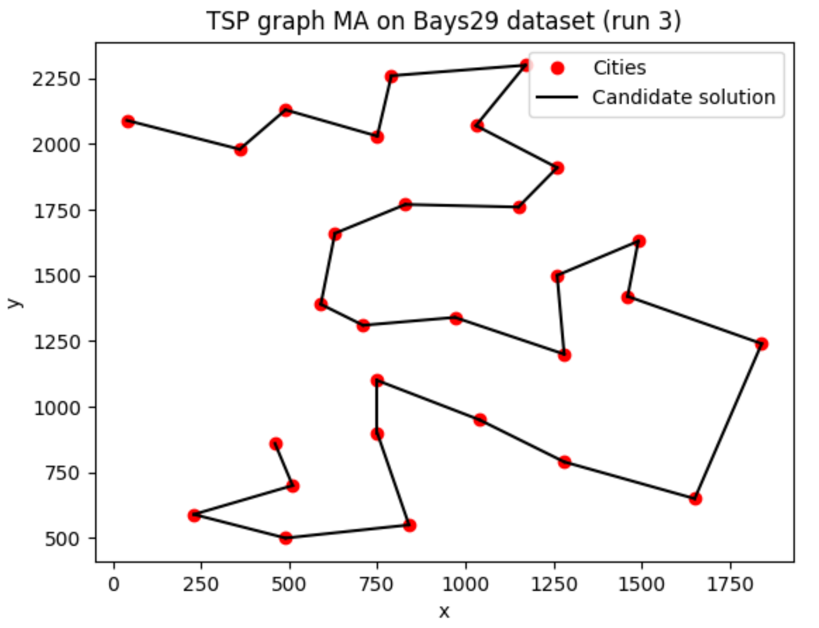

Figure 12: Solution found by MA on bays29 dataset after 100 generations, with a pool of 25.

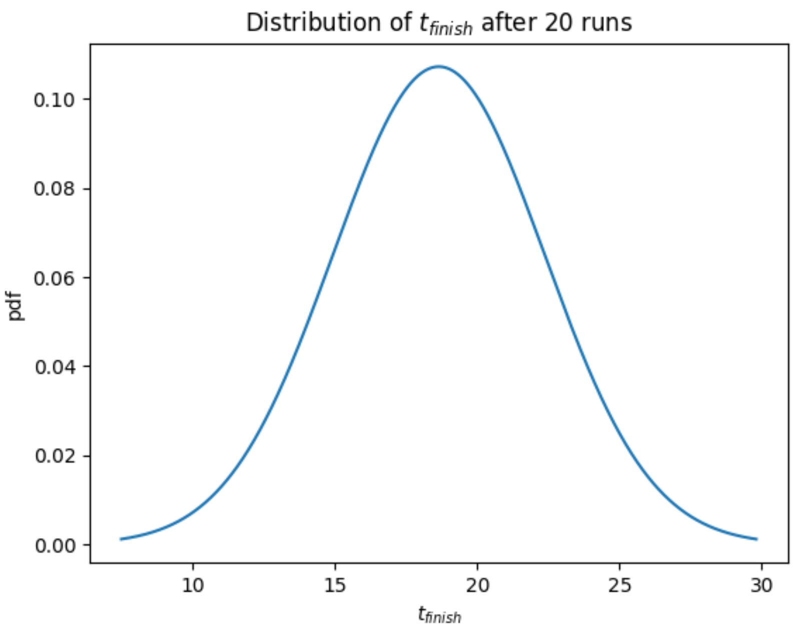

Figure 13: Distribution of t_{finish} for the string search GA with $K = 2, \mu = 0.01$.

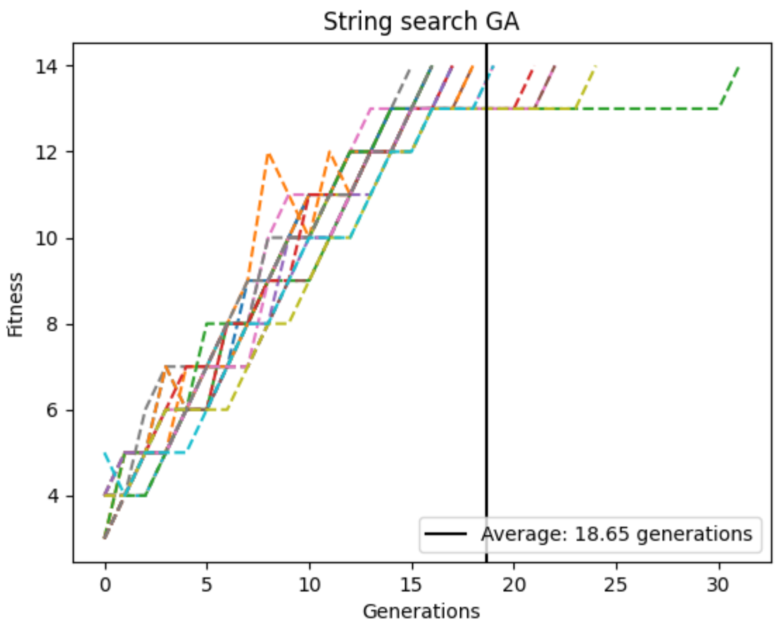

Figure 14: String search GA using $\mu = 0.01$ and $K = 2$. Maximum generations were set to 100. All runs find the target string.

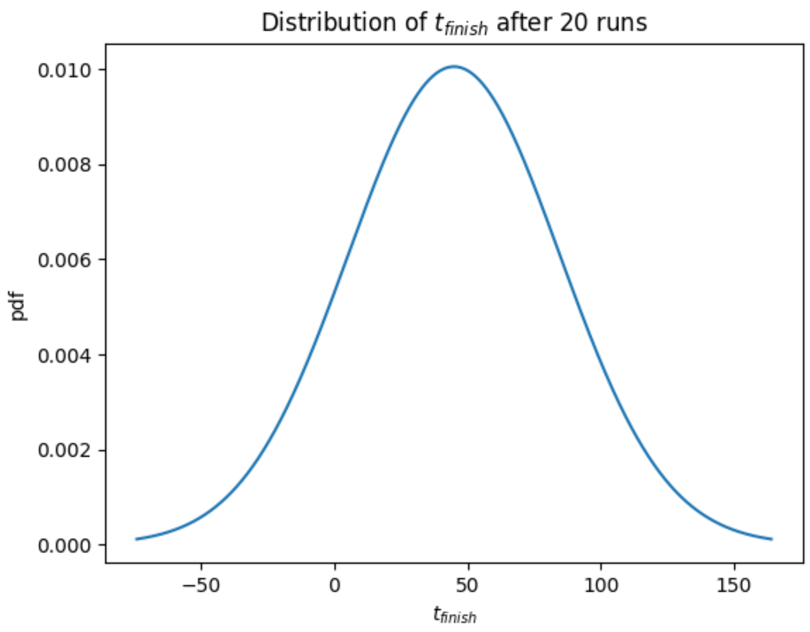

Figure 15: Distribution of t_{finish} for the string search GA with $K = 2, \mu = 0$.

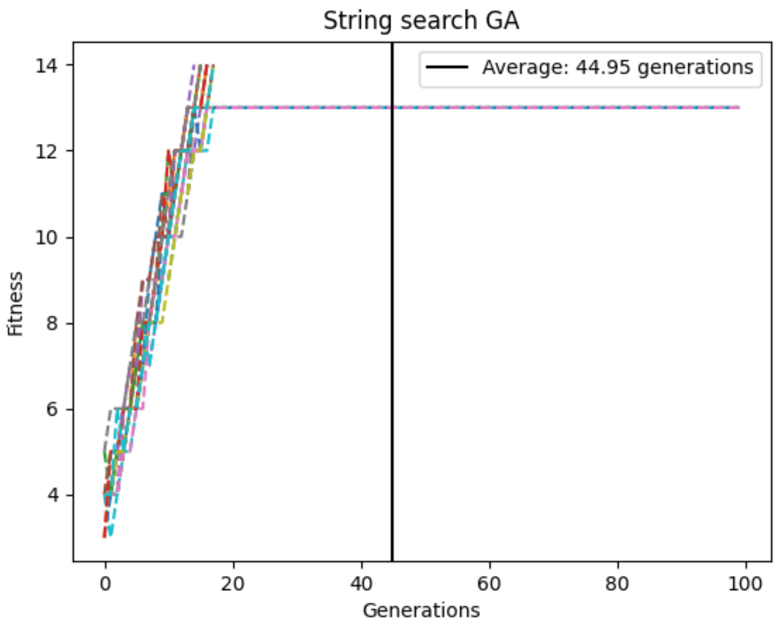

Figure 16: String search GA using $\mu = 0$ and $K = 2$. Maximum generations were set to 100. 18 of 20 runs find the target string.

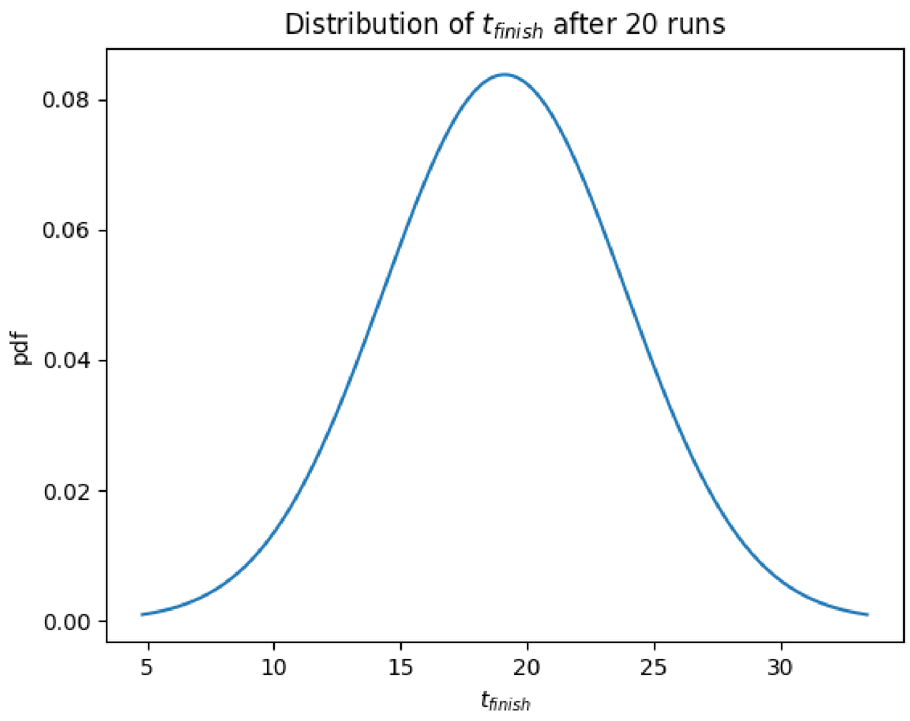

Figure 17: Distribution of t_{finish} for the string search GA with $K = 2, \mu = 0.1$.

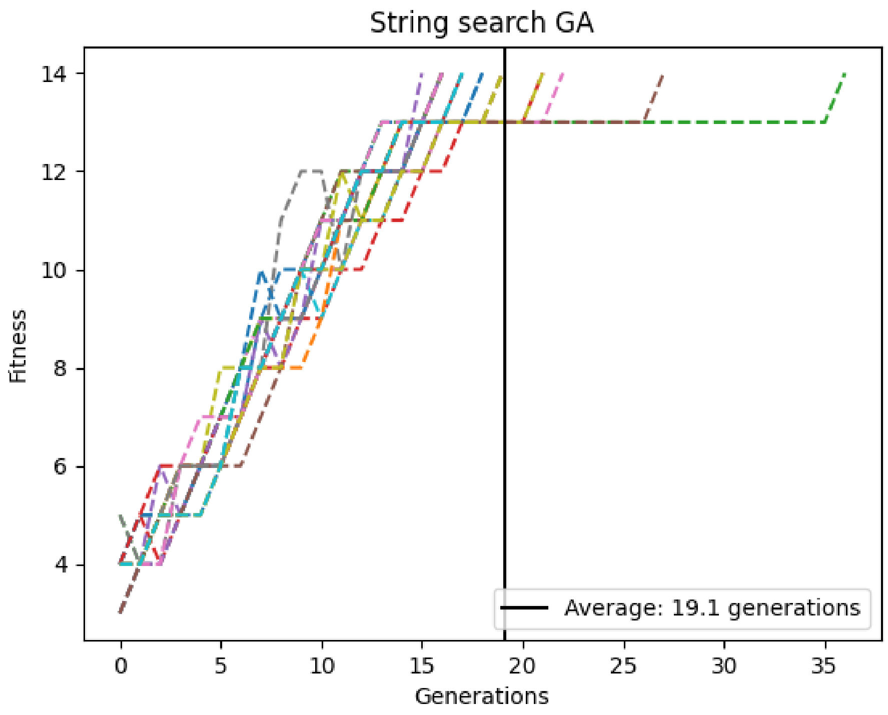

Figure 18: String search GA with $\mu = 0.1$ and $K = 2$. All 20 runs find the target string.

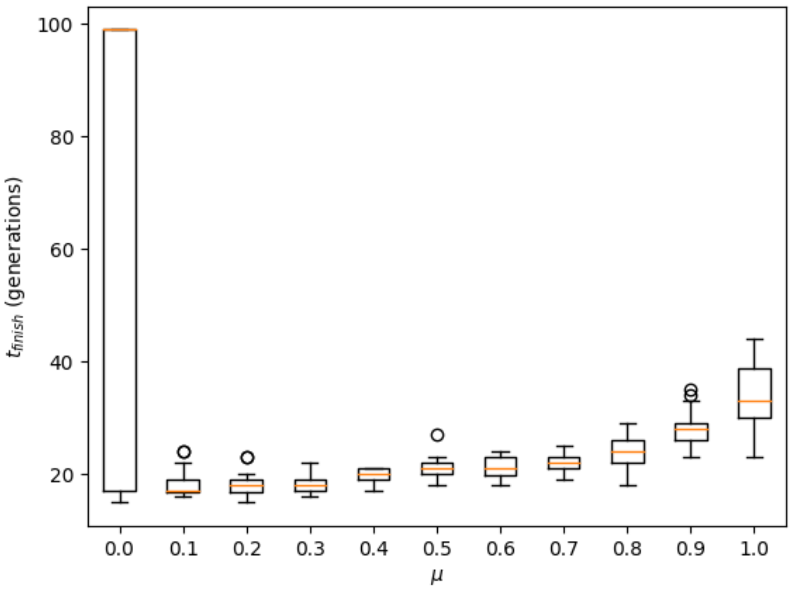

Figure 19: Boxplots of the relationship between μ and t_{finish} for $K = 2$.

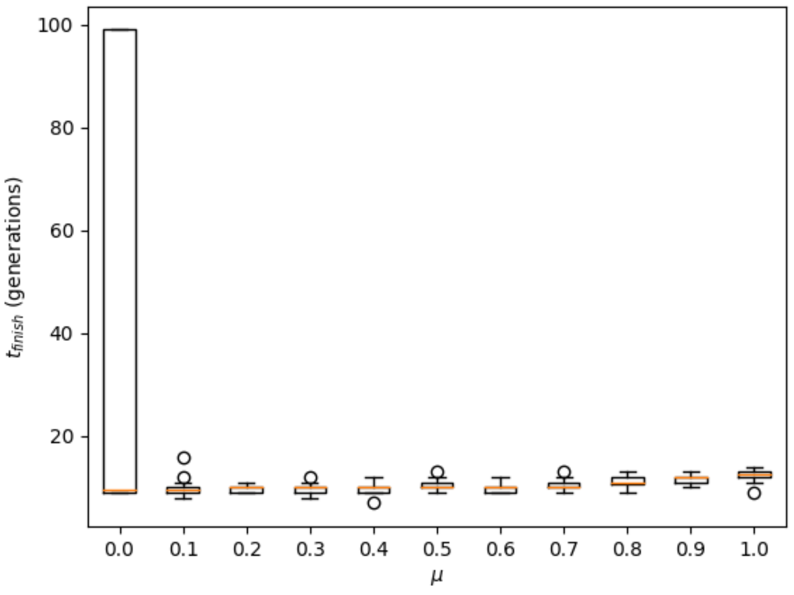

Figure 20: Boxplots of the relationship between μ and t_{finish} for $K = 5$.