State Encoding

In the following, we elaborate our encoding design of the initial state s_0, defining the individual's genotype. The initial state is defined by the agent's initial position. In most environments, the initial state is either fixed or randomly chosen (by sampling from μ), as is the goal state. Fixed and randomly selected initial states and goals have a high impact on the behavior, a trained agent is going to display. For this reason, initial states and goals should be taken into account when deciding the configuration of the environment. This has the benefit of controlling random behavior of the agent by reducing random factors in the environment. Additionally, it creates the possibility of an unlikely initial state for the agent to start in. This gives us the chance to see the agent act in states, that would otherwise not be encountered in a normal episode.

We used a binary encoding to represent the altered initial state, since it represents a simple way of encoding information and can easily be used for recombination and mutation operators. However, the challenge we face with a binary encoding is that with m bits, exactly 2^m possible states can be represented. Considering our example gridworld with a 9x9 grid, we have 81 states representing possible intial positions. A binary encoding of 6 bits could only represent 64 states while 7 bits can represent 128 states. We solved this challange by appling inverse normalization. Every dimension d of the startup state space is represented by a binary encoding e of length m. As shown in 1, we first map each part of the binary encoding to an integer, followed by normalizing these integers. The normalized values are then mapped to possible states of their respective dimensions. The state values build the final initial state for the environment.

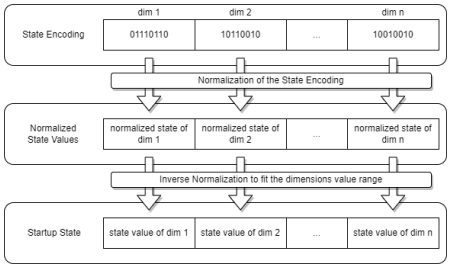


Figure 1: Mapping of State Encoding to Startup State

The benefit of this mapping from binary encoding to intial state value is that we can represent discrete state spaces with different sizes, as well as continuous state spaces with a specified precision. For discrete and continuous state spaces, we have a slightly different calculation of the normalization of the state encoding and the inverse normalization to the actual state value.

Discrete State Space Cutting the state encoding in $|d|$ equally sized encodings e_d, we can compute the corresponding value as follows: The binary encoding e_d with length m is translated to its corresponding integer value. This integer value is then normalized by dividing it by the number of possible integer values for the encoding length: $s_{norm} = \frac{int(e_d)}{2^m}$. The normalized state s_{norm} is a real number in $[0, 1[$. In the next step it will be mapped to the dimensions value range. To map the normalizedState to an integer, that lies in the dimensions value range, we require the maximal and the minimal possible values of the dimensions state space. We then reverse the normalization using the new value range and receive a real number, that is then rounded off to the next lower integer value: $s = \lfloor s_{norm} * (v_{max} + 1 - v_{min}) + v_{min} \rfloor$. In a discrete state space, we have to choose an encoding length which can represent more states, than are actually present in the state space. This ensures, that each state can be represented by at least one encoding. Since we cannot map the states of the encodings uniformly to the states of the state space, we accept the compromise that there are slightly different probabilities of occurring for the different states. By choosing a longer encoding, the difference in the probabilities of occurrence between the states is reduced. We can compute the difference in probabilities of occurrences depending on the chosen encoding length $p_{low} = \frac{1}{2^m} \lfloor \frac{1}{r} \rfloor$ and $p_{high} = \frac{1}{2^m} \lceil \frac{1}{r} \rceil$, with the interval range of values $r = \frac{1}{2^m}(v_{max} + 1 - v_{min})$. For our gridworld environment, this means that for a grid of 9x9 cells, the initial position of the agent would be encoded by two encodings of at least length 4. Each encoding respresents one dimension of the gridworld, the row and the column, which each have 9 possible states with $v_{max} = 8$ and $v_{min} = 0$.

Continuous State Space For a continuous state space, we do not require the state values to be integers. For this reason, we have a slightly different calculation. Here, we choose the encoding length based on the precision of the state values that we want to receive. The state encoding is normalized as $s_{norm} = \frac{int(e_d)}{2^m - 1}$. The difference to the calculation in the discrete space is a range of $[0, 1]$. In the inverse normalization, we do not round the value off, which is why we do not need to add 1 to the v_{max}: $s = s_{norm} * (v_{max} - v_{min}) + v_{min}$ For continuous state spaces, a longer encoding equals a higher precision. We can calculate the difference between two encodings values as $s_{dist} = \frac{v_{max} - v_{min}}{2^m - 1}$.

Additional Experimental Results

Encoding Length We already approached the topic of choosing the right encoding length in Appendix . The encoding of each dimension has to have a sufficient length such that every state in the state space can be represented by it. However, in an environment with a discrete state space, some startup states could be represented by more encodings than other startup states. With our approach, this leads to a higher probability of occurring. The difference in probabilities depends on the length of the state encoding. Considering a grid size of 11×11, the possible startup state are only represented by a grid size of 9x9. We did not consider cells on the surrounding wall as possible startup states. Each of the two dimensions has 9 possible values, that need to be encoded by a binary encoding with at least 4 bits. We concluded a simple experiment, to be able to better recommend an encoding length. We compared five different encoding lengths per dimension: $n \in [4, 5, 6, 7, 8]$. For each encoding length, we created 81000 random bit encodings of length $2 * n$. We transformed them to states in the environment using the method described in Appendix and plotted them in a heatmap to be able to show the difference in the occurrence of states. Looking at Figure 2, we can see a significant difference in the length of encodings. Especially for an encoding length of 4 or 5, the difference in occurrence is clearly visible. The darker cells are less probable to occur then the lighter cells. This supports the computation of the difference in probability described in Appendix . For an encoding length of 4, some states are 2 times as probable as others. For an encoding length of 5, this is reduced to 1.33 times. Looking at the heatmaps and the calculation of the difference in probability, we recommend to use a state encoding length, where a state is less than 1.25 more likely than another state. In FlatGrid11, this is the case for an encoding length of 6. There only exist a few states that are 1.14 more likely than other states. This difference in probability can be regarded as insignificant for our algorithm.

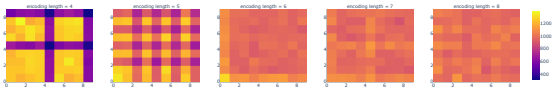
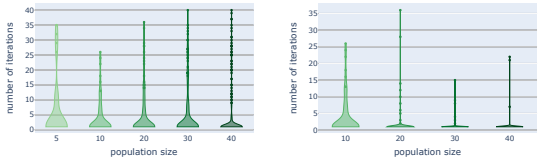


Figure 2: Comparing 81000 random 11x11 state encodings

Population Size The population size determines how many demonstrations are shown to the user. On one side it is important to show enough different behaviors for the user to get a good impression on how the agent behaves. Also, we could execute the genetic algorithm using a greater population size, but only display the 10 best individuals. However, this has several drawbacks. First, a greater population size makes it easier for individuals to survive in the population. A larger population means, there will also be more individuals with a low fitness value. This also applies to individuals that are not diverse. Second, a greater population size makes it more difficult for individuals, to be considered diverse in comparison to other individuals. In most cases the following applies: the more individuals to compare to, the less diverse the individual can be. This makes it more difficult for a new individual to have a high enough fitness value to consider it one of the best individuals in the population.



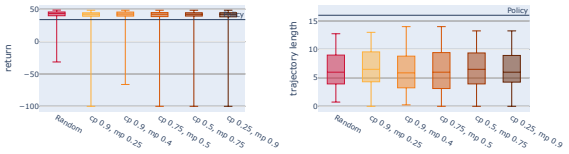
(a) all individuals (b) best 10 individuals

Figure 3: New individuals joining the entire population and only the best 10 individuals of the population

This reasoning is supported by Figure 3, where we can see all newly added individuals that survived at least one generation. In each one of the plots, we compared the development across different sizes of the population. The displayed data contains individuals from 4 runs of the genetic algorithm. In Figure 3a, we look at all individuals of the respective population size. We can determine, that with a larger population we also have more individuals survive at least one generation at a later point in the algorithm. This shows, that with a larger population new individuals have higher chances of surviving to the next generation. A larger population therefore reduces the selection pressure and leads to a slower convergence of the algorithm. Especially in an environment with few startup states, it is more difficult to find new individuals that are considered more diverse than already evaluated individuals. Since the diversity is based on the comparison to other individuals, the initially created individuals have an advantage over newly created individuals in later iterations. Older individuals are already considered highly diverse, while newer individuals have to be compared to them. The newer individuals have more and better competition than the initially generated individuals. Since the user is not interested in viewing all episodes of a large population, we reduced the population of the last generation

to only show the best 10 individuals. The resulting plot can be seen in Figure 3b. While looking at all individuals of the population shows more new individuals joining the population, the larger the population is, we can see a clear difference to that when only looking at the best 5 individuals. The last iteration where a new individual is ranked as one of the best 5 individuals of the population is sinking for an increasing population size. Only looking at the best individuals, we now see that with a larger population the algorithm converges significantly faster. We should not choose a too small population size. Important information could get lost. When viewing the episodes selected by the genetic algorithm, obviously we want to see the most diverse individuals, but we also want to see a few individuals that do not have an outstanding behavior. It is important to choose a population size that does not only show the most drastically diverse behavior, but instead shows as much of the behavior as possible without showing redundant behavior. The goal is, to see only those individuals, that are strictly necessary to present diverse behavior of the agent. Therefore, we compared the return and trajectory length distribution of all individuals in the population, with the reward and trajectory length distribution of only the best 10 individuals.

Crossover and Mutation Probability For every problem, that should be solved by a genetic algorithm, there are two hyperparameters that need to be tuned: the crossover probability and the mutation probability. A higher crossover probability usually leads to more exploitation of individuals with a high fitness value, while a high mutation probability explores the search space to avoid getting stuck in a local optimum. By applying mutation, new diverse individuals are evaluated, that could lead to new insights. We explored several different settings for the two probabilities: a high crossover probability of 0.9 with a low mutation probability of 0.25, a slightly lower crossover probability of 0.75 with a higher mutation probability of 0.5, and both of these settings with switched probabilities. Additionally, we added the setting with a crossover probability of 0.9 and a mutation probability of 0.4, which we used in the previous experiments. With this experiment, we mean to investigate, if the search space requires more exploitation of the already evaluated individuals or more exploration of the search space. The results in Figure 4 show, that a low crossover probability paired with a high mutation probability does not lead to significantly better results in terms of the distribution of trajectory lengths than random search. However, looking at the distribution of returns, we can state, that our

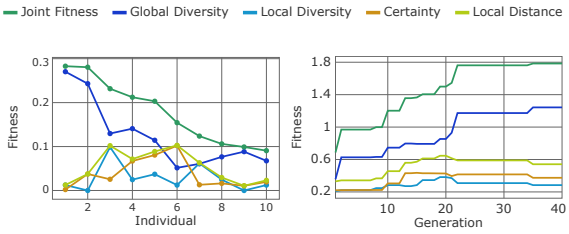


(a) final returns (b) final trajectory lengths

Figure 4: Diversity of the individuals of the last generation compared by different operator probabilities

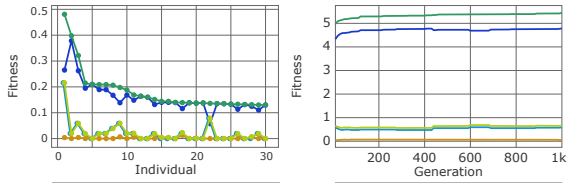
algorithm shows more diverse behavior than random search. Random search does not always manage to find the startup states that lead to a very interesting behavior, namely the agent's standstill.

Fitness Impact Analysis

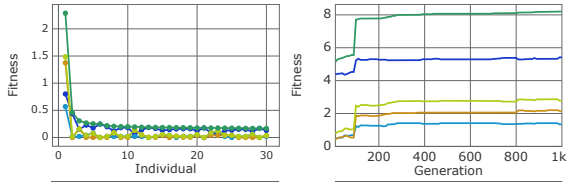


(a) Population Analysis (b) Generation Analysis

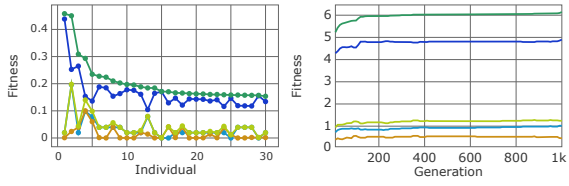
Figure 5: HoleyGrid *JointFitness* Analysis



(a) Population Analysis (b) Generation Analysis



(c) Population Analysis (d) Generation Analysis



(e) Population Analysis (f) Generation Analysis

Figure 6: FetchReach *JointFitness* Analysis