Premature Convergence Prevention for Genetic Algorithms through Population Diversity Maintenance

Wayne Wang, Cliff Yang
Middlebury College

Abstract

Premature convergence problem remains a key obstacle for traditional Genetic Algorithms. In order to overcome premature convergence, the population diversity needs to be maintained high. In this project, we apply different approaches that maintain the population diversity throughout generations, with the objective to avoid premature convergence problem in GA. Namely, we apply dynamic mutation, adaptive mutation, tournament crossover, and weighted crossover. These methods are applied to select the participants of crossover and mutation before applying the identical crossover and mutation operator. The GA is analyzed in a 2D-basketball gameplaying environment built with Unity with the intention to visually demonstrate the effects of such prevention methods. The result shows that the tournament crossover approach has a significant effect on preventing premature convergence, while the weighted crossover method improves the overall performance. The two mutation methods demonstrate insignificant influence on the performance when applied alone.

Introduction

Metaheuristic algorithms have been applied in numerous fields to solve real-world problems. The single-solution-based metaheuristic algorithms "utilize single candidate solution and improves this solution by using local search," but could get stuck in local optima. (Katoch et al., 2021) Population-based algorithms, exemplified by the Genetic Algorithm (GA), could alleviate the problem. However, a population-based metaheuristic algorithm still faces the challenge of premature convergence, where a suboptimal solution dominates the reproduction of the population and leads us away from the optimal solution. In this project, we intend to explore the effectiveness of different premature convergence prevention techniques, as well as the combinations of the ones that take place at different stages of GA. In order to make sure a premature convergence does not occur, the population's diversity needs to be maintained throughout the generations, and different methods have been proved to be effective in maintaining population diversity and preventing premature convergence. Namely, we intend to compare techniques for diversity maintenance in crossover and mutation operators and techniques to apply these operators.

Different environments in which the genetic algorithms are applied have a great influence on the variables within the GA. Many previous scholars have studied premature convergence prevention methods in the setting of the Traveling Salesman Problem. In this project, we developed a 2D basketball shooting game using Unity as a gameplaying environment to study different premature convergence prevention methods. The novelty brought by implementing GA in a gameplaying environment over a traditional TSP is that it amplifies the effect of premature convergence, as well as the difficulty for the gameplaying agent to escape the local optimum. In

our project, a gameplaying environment also provides the benefit of its visual representation of scenarios to reflect premature convergence, as well as its ramifications on students learning GA.

We will first elaborate on existing works on various methods of preventing premature convergence in GA and the comparisons on them. Then, we will describe our method, in which we define our learning environment, present the development process, and elaborate on the GA operators and metrics that we chose. After discussing methods, we will describe the process of gathering the data result and present the result. Last, we will end with a discussion on the take-aways from the results, as well as on future works on the topic.

Related Work

Different approaches and methods to prevent premature convergence in different stages of GA have been discussed by scholars. Mc Ginley et al. propose a novel adaptive genetic algorithm to apply crossover, mutation, and selection operators to prevent the GA from falling into a local optimum. (Mc Ginley et al., 2011) Specifically, the ACROMUSE algorithm they used measures the standard population diversity (SPD) and healthy population diversity (HPD), and adaptively adjusts crossover, mutation, and selection parameters to maintain a high enough population diversity in order to prevent premature convergence. Similarly, Hassanat et al. proposed another dynamic approach to choose mutation and crossover ratios to maintain a high population diversity between generations. (Hassanat et al., 2019) They developed mechanisms to dynamically decrease the mutation ratio and increase the crossover ratio linearly during the search progress, and found it especially effective in preventing premature convergence for small population sizes. Other than adaptive GA, other scholars have offered methods using other approaches. Harik et al. discuss a parameter-less GA that maintains a moderate value of Selection Pressure (SP) to maintain the

diversity of the population. (Harik et al., 1999) Malik et al. offer the method of DGCA and elitist technique to avoid falling into a premature convergence situation. (Malik et al., 2014) From a different angle, Fuertes et al. incorporate chaos theory and entropy to generate an initial population using chaotic maps in order to prevent premature convergence. (Fuertes et al., 2019) Our work attempts to provide a comparative analysis that previous works have not systematically conducted and visualize the result for easier interpretation for undergraduate students.

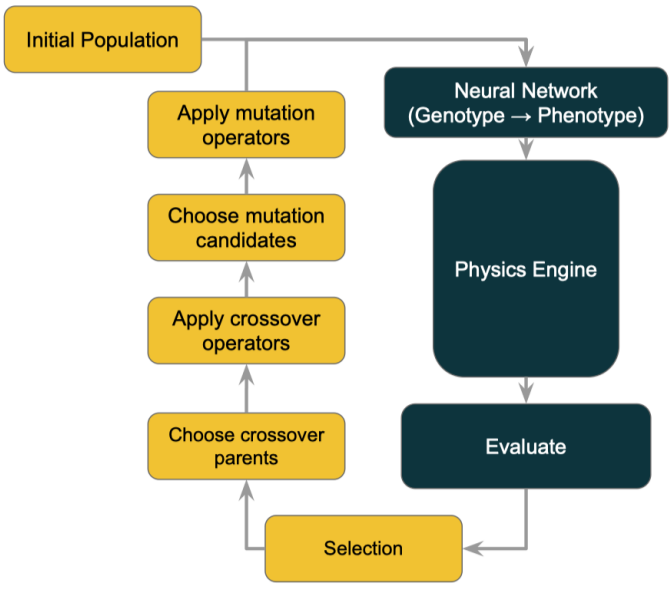
Problem Statement

In our project, we build a gameplaying environment and train the playing agent using a genetic algorithm. Specifically, we apply four different methods to prevent the occurrence of premature convergence in the stages of choosing mutation and crossover candidates: weighted crossover, tournament crossover, adaptive mutation probability, and dynamic mutation rate. We measure each of the four approaches' performance using the overall score of the generations over time, as well as the diversity of each population over generations.

We define premature convergence, according to Malik et al., as a problem "which occurs when genes of high rated individuals quickly attain to dominate the population, constraining it to converge to a local optimum". (Malik et al., 2014) To demonstrate the premature convergence problem's effect in our game, the mechanism of the game has to be briefly discussed. Our gameplaying environment is a 2D basketball game built using Unity. The goal of the game is to move the player and shoot the basketball into the basket, and the evaluation function gives a higher score to the players whose shot ended up closer to the basket (the details of the evaluation function will be discussed in the following section). Thus, the premature convergence problem occurs in our game as players converge to a shooting habit of attempting to make the shot at harder angles

or bouncing the ball at various objects in the environment before scoring – it achieves the same goal with a smaller margin of error. Statistically, we would notice the occurrence of premature convergence in our population as the aggregated score converges to a value much lower than the perfect score of every player making the shot.

When applying the approaches listed above to prevent premature convergence, we chose the methods that take place in the steps of choosing mutation and crossover candidates. Other stages of the GA are also popular choices for scholars to apply premature convergence prevention techniques, including initial population generation, mutation and crossover operators, selection process, and evaluation function. However, our particular setting aims at visualizing the result rather than a wholistic analysis that enumerates premature convergence methods that take place at all stages, and we chose to focus on the two stages of choosing crossover parents and choosing mutation candidates. The figure demonstrates the control flow graph of our environment, and we will next elaborate on each step and discuss the methods applied in each.



Methods and Results

The above figure lays out the structure of our process, and in this section, we discuss each step in detail.

Physics engine & Evaluation

Our environment in which the GA performs is a 2D basketball game built using Unity in C#, in which the single player attempts to move around and shoot the basketball. We formulate our game as the following.

Game state: The game state is described by three values. Float *playerX* denotes player's X-coordinate, which tells its distance from the basket. Float *velocity* tells the speed at which the player is moving. Float *time means the amount of time passed since the start of the generation*. The unit of these values is normalized to the range of [-1, 1].

Inputs: The game allows four inputs. Boolean *shootOrNot* decides whether to shoot the basketball. Float *shootDirection* dictates the shooting direction. Float *shootForce* decides the shooting force. Float *movementX* tells the player which direction to move and how much. Boolean *jump* decides whether the player jumps or not.

Feature / Restrictions: There exist several special rules to make the game environment more realistic. First of all, each generation is given a 5-second lifespan to perform, after which they are evaluated and go in to the selection process. Secondly, when the player moves and holds the ball, the player *dribbles*, and shooting accuracy decreases if the player shoots when dribbling without stopping to mimic a jump shot. Moreover, the shooting accuracy also decreases when the player is further away from the basket. The accuracy is controlled by a random value added to the game input *shootForce* – when the accuracy decreases, the random addition results in a larger range.

Evaluation: The goal of the evaluation function is to provide a metric for each individual in the population to be measured against to obtain its fitness. The objective is to encourage shots that ended up close to the basket and award the shots that actually made the basket. The evaluation is implemented by a linear mapping that measures the X-distance between the ball and the basket on the plane of the basket, and the score ranges from [1, 100], and the highest score of 100 is only awarded to the balls that made the basket.

Selection & Crossover

Our population size is 250, and the evaluation score comes from a sum of three trials. The bottom half of the population gets dropped, and the top 50% participate in the following crossover and mutation steps according to each method. For the crossover steps, we perform two different premature convergence prevention techniques to choose crossover parents, after which we perform the same crossover operator on the chosen parents.

Weighted crossover: The method of weighted crossover attempts to ensure the children inherit more of the better-performing parents. It achieves the goal by assigning each parent with a different probability to perform the crossover, and the parents with a higher fitness receive a higher crossover likelihood than the worse performing parents. We implement the weighted crossover by choosing the two parents according to their performance in the previous round – the weights in the random choosing function are the scores associated with each of them.

Tournament crossover: The tournament crossover approach aims at selecting better-performing crossover parents while maintaining the population diversity. It achieves the goal by holding multiple "tournaments" within the population – in each tournament, a small group of genes is selected at random, and the two best performing genes perform a crossover. This is repeated until we generate the desired amount of children. The key in tournament crossover is that it avoids

sticking to the best-performing parents to perform crossover and thus potentially falling into a local optimum; by randomly choosing each tournament group across the entire population, both the performance and diversity of the parents are guaranteed.

In either the tournament crossover or the weighted crossover method, all parents participate in the crossover selection pool, and the only differentiating factor is the probability that they are selected from the crossover pool and generate children. After applying either technique to choose the crossover parents, we applied the same crossover operator to each pair of parents: we swapped each layer of weights between the neural networks (discussed later) of the two parents at a given probability of 0.3. By swapping the weights on the same layer entirely rather than at a finer granularity, we make sure children's phenotypes are reflective of the parents' phenotypes.

Mutation

After the crossover step, we perform mutation on a selected subset of genes. Here, we apply two approaches to explore their performance in maintaining population diversity. Similar to the crossover step, the control variable is the mutation operator itself, and the differentiating factor is how we choose the genes to participate in mutation.

Adaptive mutation probability: The goal of adaptive mutation probability is to adaptively adjust the probability each gene participates in mutation according to the population diversity. Since our gameplaying environment has its genotypes as floats, traditional methods to measure Hamming distance between genes is not suitable. Thus, we achieve the same concept by measuring the similarity between two parents when performing crossover. When the crossover takes place, the second child from each crossover receives a probability of mutating, and the probability is determined by its parents' distance to each other. Using this method, a significant portion of the population is mutated at a probability that reflects the last generation's diversity.

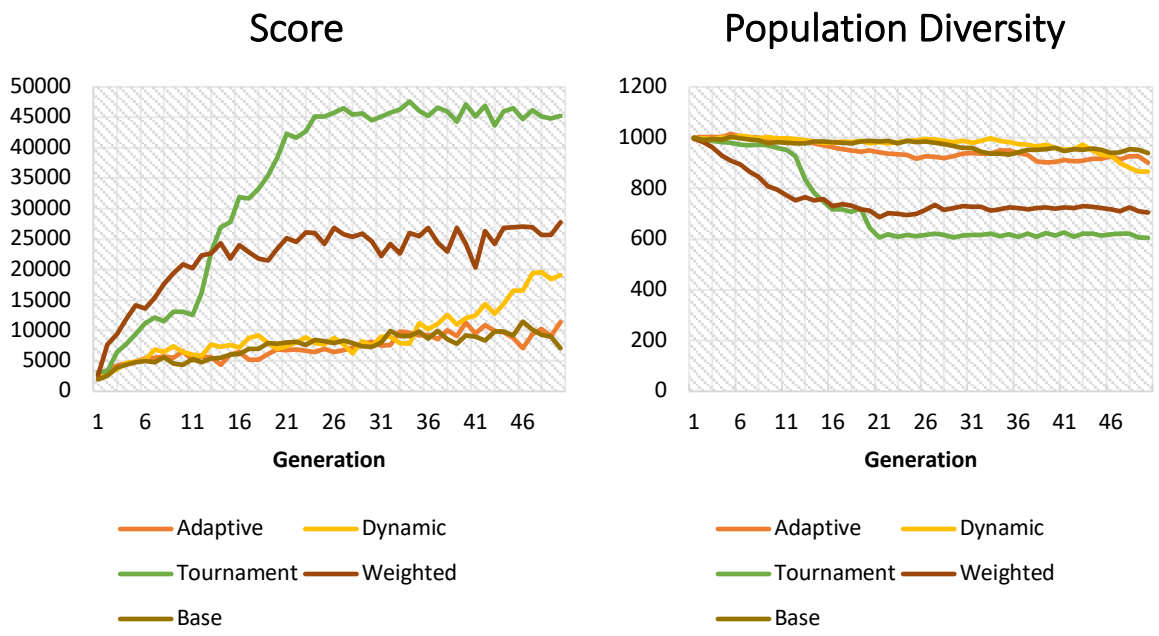
Dynamic mutation rate: This approach assigns an identical mutation probability to each gene in the generation, and decreases this mutation rate as the generations evolve. It attempts to mutate the population more at the beginning to allow a wide range of exploration and slows down the mutation rate as the generations become more refined. It achieves the goal of maintaining population diversity by allowing the population to explore a wide range of options at first, and refining each approach later on. We implement this approach by decreasing the mutation rate by 4% each generation, starting from 60%.

After each mutation method, we apply the mutation operator itself to the selected genes that should participate in mutation. Specifically, we mutate each float within the gene by a randomly generated magnitude within a constant range of [-0.2, 0.2].

Neural Network

Our environment differentiates itself from traditional TSP, as its genotype and phenotype are not identical. Specifically, the genotype needs to denote the "decision making" parameters of the player, and the phenotype should determine the input into the game engine. We need to perform the translation from a given game state to the player's action. We have applied a neural network to achieve this translation using only its feed-forward functionality. The neural network has 1 input layer of 3 nodes, 2 hidden layers each with 4 nodes, and 1output layer with 4 nodes, in which the 3 nodes in the input layer correspond to the 3 values defining a game state, and the fou4r output nodes correspond to the 4 game inputs. The 44 edges in the neural network have 44 weights, and they compose one gene in the population. With this setup, we are able to use the weights of the neural network as genes to denote the "decision making", and perform the generation of game inputs given a game state.

Results



In the base case, none of the four premature convergence methods is used – after selection, two parents are chosen at random with equal probability, then mutation is applied to at each gene with a fixed probability. In this case, the total score of each generation steadily increases at a much lower rate than the cases where different methods are applied. We suspect this majorly due to the random choice of parents not giving the next generation enough useful genetic information. The premature convergence in the base case occurs late towards the 50 generation count, but due to the fact that the crossover and mutation are purely random and evenly distributed across the population, the population diversity does not decrease much forming a premature convergence. This can be interpreted as a tradeoff between performance and population diversity – if the performance is not good enough and the population is scattered across worse-preforming domains, premature convergence, or even any convergence, hardly occurs.

Compared with the base case, applying weighted crossover accelerates the GA's learning by a great amount. In fact, it achieves the similar performance of the base case using only around

1/4 of the time. This is due to the weighted crossover prioritize the selection of better-performing parents to participate in crossover, generating more useful genetic information for the children. However, a better performance comes with a more apparent premature convergence, where the total score hardly grows after 15 generations and remains far from a full score. The population diversity confirms the premature convergence, where it decreases to a steady level at the same time as the performance stagnates.

The tournament crossover method performs extremely well compared with the others. The overall score grows rapidly and reaches a stagnation very close to the theoretical perfect scorer. The population diversity remains relatively high before converging close to the global optima. We suspect the good performance is a result of tournament crossover incorporating both diversity encouragement and useful information preservation in its nature. By choosing the tournament group at random, the crossover parents' diversity is guaranteed across the population; the tournament chooses the top two performing parents to perform crossover, preserving the good genetic information in them.

The two mutation methods perform closely to the base case, where no special treatments are done to the crossover and mutation process. We think it is due to the fact that when applied alone, mutation methods don't have any components encouraging the useful genetic information to be preserved, but rather only relying on the selection process. While mutation gives a way to maintain the population diversity, a high diversity alone with no incentive to preserve useful genetic information yields a low performance, let alone premature convergence.

Discussion & Conclusion

Our result shows that the weighted crossover and tournament crossover have a significant improvement in the performance of the genetic algorithm in our gameplaying environment. The tournament crossover approach is especially effective in preventing the occurrence of a premature convergence, while the weighted crossover method enhances the overall performance. On the other hand, the mutation methods, both the dynamic approach and the adaptive approach have little effect in enhancing the performance or preventing premature convergence when applied alone. In future works, we hope to study the performance of different combinations of these methods in gameplaying environments rather than applying them independently. Moreover, we recognize that we are only utilizing a subset of the effectiveness of the neural network, and it is a valuable direction to explore in the future to allow the neural networks to perform rounds of back-propagation before being selected by the genetic algorithm. To sum up, this project demonstrates a basic platform on which different crossover and mutation methods are applied and compared to combat premature convergence – it provides future works a preliminary example of conducting a more rigorous and quantified analysis of premature convergence prevention methods in gameplaying environments.

Bibliography

Hassanat, Ahmad, Khalid Almohammadi, Esra Alkafaween, Eman Abunawas, Awni Hammouri, and V. B. Prasath. "Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach." *Information* 10, no. 12 (2019): 390.

McGinley, Brian, John Maher, Colm O'Riordan, and Fearghal Morgan. "Maintaining healthy population diversity using adaptive crossover, mutation, and selection." *IEEE Transactions on Evolutionary Computation* 15, no. 5 (2011): 692-714.

Malik, S., and S. Wadhwa. "Preventing premature convergence in genetic algorithm using DGCA and elitist technique." *International Journal of Advanced Research in Computer Science and Software Engineering* 4, no. 6 (2014).

Fuertes, Guillermo, Manuel Vargas, Miguel Alfaro, Rodrigo Soto-Garrido, Jorge Sabattin, and María Alejandra Peralta. "Chaotic genetic algorithm and the effects of entropy in performance optimization." *Chaos: An Interdisciplinary Journal of Nonlinear Science* 29, no. 1 (2019): 013132.

Harik, Georges R., and Fernando G. Lobo. "A parameter-less genetic algorithm." In *GECCO*, vol. 99, pp. 258-267. 1999.