Using the Santa Fe Ant Trail Benchmarking Problem to Optimize Genetic Algorithms

Nimisha Gupta
Yorktown High School, 2727 Crompond Road, Yorktown Heights, NY 10598, United States
Nimisha.gupta@yorktown.org

ABSTRACT: Genetic algorithms have recently gained popularity for solving complex problems through their evolutionary nature. They implement biological mechanisms such as crossovers and mutations, yet it is not known how each variable independently effects the algorithm's ability to work. If found, it would improve the optimization time in genetic algorithms by decreasing the amount of trials the program has to run. To test the benchmarking system, three trials were conducted in which the probability of mutating and mating were changed independently to understand their effects. As the base case, the probability of mating and mutating were set to default values from previous testing to efficiently complete the benchmarking device used (Santa Fe Ant Trail). In trial 2, the probability of mating was independently tested while trial 3 tested the probability of mutating. The increase in crossover rates had a superior efficiency in completing the task given while the increase in mutation rates decreased overall efficiency of the program. In the future, these results can be used in genetic algorithms to increase their efficiency in completing complex problem sets.

KEYWORDS: Evolutionary Computation; Optimization; Genetic Algorithm; Benchmarking; Fitness.

Introduction. As society evolves, certain complex problems arise that are virtually impossible for a human to solve. One example is creating a satellite antenna for broadcasting messages and data transmission. These antennas must be asymmetrical for maximum efficiency; it is both time and material consuming for a human to create because of the amount of trial and error required[1]. The field of evolutionary computation was created at the intersection of biology and computer science to solve such complex problems. Genetic algorithms are used in scenarios for optimizing behavior of artificial agents in achieving some goal. The algorithms use evolutionary tactics, such as mating and mutating, to "evolve" its programs/solutions until a solution is found. As the popularity of these algorithms grows the efficiency of genetic algorithms must be tested. Problems like the Santa Fe Ant Trail have been developed to decipher how factors liked mating and mutating effect the efficiency of the algorithm[1]. This knowledge will lead to the optimization of genetic algorithms.

Review of Literature. Evolutionary computation techniques can produce highly precise and quick solutions to a wide range of problem settings.[1] It has applications in computer science and can be implemented in algorithms inspired by evolutionary biology because of its similarity to real-world problems.

Evolutionary computation encompasses genetic programs that implement mechanisms inspired by biological evolution such as reproduction and mutation. Each genetic program starts by generating a large amount of possible solutions, or evaluations. Through different strategies the less efficient solutions are removed. As a result, the solution population will gradually "evolve" to increase fitness, or efficiency[2]. This repeats until the most effective solution is outputted, shown in Figure 1. The greater the fitness, the greater the optimization of the solution. These algorithms have characteristics which eliminate the need for detail. They take random parameters from various subsets and combine them to create new solutions. For example, when simulating a race car, the focus is on aerodynamics and speed. Taking into account factors such as height and weight, the simulator combines several values of each respective value and tests them to provide the best outcome. Thus, there is no need for exact measurements. Despite this, there has been no research into how a change in mating/reproduction rate and mutation rate can affect an algorithm's efficiency.

Certain benchmarking problems are used to test the efficiency of a genetic algorithm. One

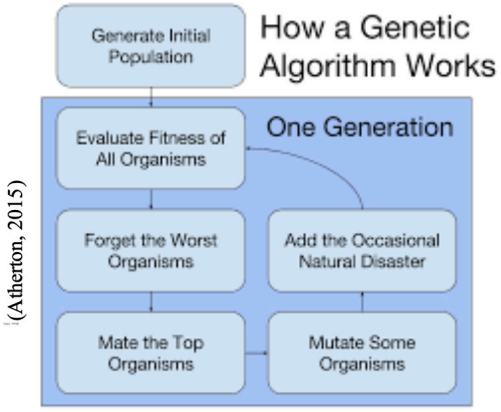
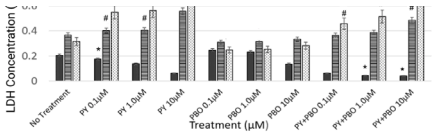


Figure [1]. The genetic algorithm process to optimize solutions for difficult problem sets.

DOI: 10.36838/v2i2.4

benchmarking system, the Even-Parity problem, focuses on finding how many prime numbers less than a given integer have an even number of prime factors3. Another benchmarking problem is called the Artificial Ant problem. In this, a program is supposed to follow a food path and over time holes start appearing on the trail and growing in size. Due to its correlation with ants and food trails the programs are called ants.

Santa Fe Ant Trail. A version of the Artificial Ant problem is the Santa Fe Ant Trail. It comes with a layout of 32x32 spatial elements and incorporates a predefined food trail the ant is supposed to follow. Figure 2 shows the food trail consisting of 144 cells with 89 cells containing food. It has the following irregularities which makes it difficult for the ant to follow the trail easily: single gaps, double gaps, single gaps at corners, double gaps at corners, and triple gaps at corners[4]. The ant starts at the top left corner facing east and continues down the trail. The ant can turn left, right, or move forward one step and sense whether there is food ahead of it. With each step the ant loses energy, regaining energy by ingesting food. The goal is for the ant to eat all of the food pellets. Therefore, the more food pellets the ant eats, the more efficient the genetic algorithm is.



2. Cytotoxicity of PY and PBO in a Time Dependent Manner. Human neuroblastoma with various concentrations of PY and PBO. Statistical analysis was performed in compa ınt groups. * = p<0.05; # = p<0.005. Data is presented as mean ± SEM from eight samp

Figure [2]: The Santa Fe Ant Trail. The dark squares contain food and the light squares are holes in the trail.

Through various tests, researchers have called this problem highly deceptive. The Santa Fe Ant Trail is difficult to solve efficiently because of the large, hole-filled landscape and the deceptive nature of the search space limited by a fixed amount of energy[5,6]. Additionally, the fitness space associated with the Santa Fe Trail has a great deal of randomness associated with it, creating difficulties[7].

Problems. It is unknown how mating, also known as crossover, and/or mutating directly effects a genetic algorithm's output when benchmarked. By calculating this, one can increase the optimization of the efficiency of genetic algorithms.

Goals. The goals of this project are

1. To test the effect of mating on a previously created genetic algorithm and understand how it affects evaluation population, or the amount of possible solutions, and the maximum amount of food an "ant" could eat.
2. To test the effect of mutation on a genetic algorithm and understand how it affects the evaluation population and the maximum amount of food an "ant" could eat.

Hypotheses. It is hypothesized that

1. Increasing crossovers in the genetic algorithm will cause an increase in the amount of evaluations and the maximum amount of food eaten. There will be more evolution, causing "adaptable traits" or those that allow the "ant" to eat more food to become more common throughout
2. Increased mutations in the genetic algorithm will cause a decrease in evaluations and increase in food eaten. Positive mutations will lead to adaptive traits being developed quicker.

Results. Three trials using the Santa Fe Ant Trail were run to test the effect of the mating and mutation rates in genetic algorithms and find an optimal solution to the trail problem.

During the first trial, the algorithm was set so the probability of mating and mutating between generations was 0.5 and 0.2, respectively. This trial was used as the control group. At the program's start, the number of evaluations (population) immediately decreased to 152. As shown in Figure 3, the number of evaluations remained around 150 for the remainder of the program. Throughout the 40 generations tested, the average amount of food eaten showed a positive correlation with the amount of generations, shown in Table 1. Therefore, the efficiency of the ant's capability to eat pellets on the trail increased overall. By generation 23, a configuration of the original algorithm had completed the trail (eating all 89 pellets) and provided a solution.

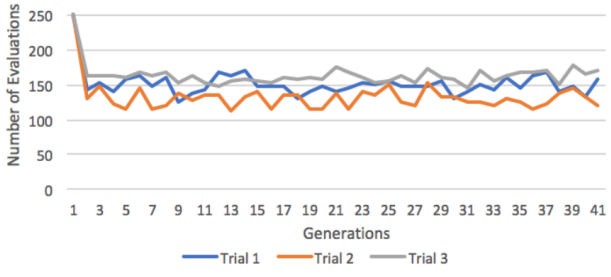


Figure 3. The number of evaluations for each generation in trials 1-3

Figure [3]. The number of evaluations for each generation in trials [1–3].

DOI: 10.36838/v2i2.4

Table 1. The average and standard deviation of food pellets eaten for generations for trial 1.

MA Plurality	0.36868±0.00195	13.3740±0.01269	20.0535±0.01536
Honest Range	1.40410±0.00452	0.94987±0.00210	0.73014±0.00152
Honest Approval	2.33777±0.00726	3.21130±0.00483	2.51405±0.00349
MA Approval	0.03051±0.00027	0.23412±0.00107	0.24150±0.00097
Honest Borda	0.70203±0.00320	0.45114±0.00130	0.30524±0.00084
MA Borda	0.09226±0.00258	231.883±0.10201	289.837±0.09411
Honest Copeland	0.03050±0.00027	0.04243±0.00021	0.04338±0.00019
MA Copeland	0.09070±0.00255	1.35911±0.01584	2.19883±0.02155

In trial 2, the probability of mating, or crossover, was set to 0.4 and the probability of mutating remained at 0.2. This was done to understand the independent effect of mating. Once again, the number of evaluations immediately dropped, this time to 129, shown in Figure 3. Continuing from generation 2, the system's variance stayed around the 130-evaluation mark. The average amount of food showed a positive correlation with the amount of generations, representing that the ant's efficiency in terms of eating food was increasing as the program evolved, shown in Table 2. However, the maximum amount of food eaten by generation 40 was only 60, demonstrating the set variables did not successfully create a solution.

Table 2. The average and standard deviation of food pellets eaten for generations for trial 2.

	Flower insertion to first picking	40	Chili & garlic	effective
	Harvesting	50	Chili & garlic	effective
2	Plant establishment	10	From milk	effective
	Vegetative	30	From milk	effective
	Flower	40	From milk	effective
	Harvesting	46	From milk	effective
3	Plant establishment	10	Decayed vegetables and fruits	effective
	Vegetative	30	Decayed vegetables and fruits	effective
	Flower insertion to first pick up	38	Decayed vegetables and fruits	effective

Table 3. The average and standard deviation of food pellets eaten for generations for trial 3.

	Flower insertion to first picking	7	Chili & garlic	effective
	Harvesting	9	Chili & garlic	effective
2	Plant establishment	3	From milk	effective
	Vegetative	6	From milk	effective
	Flower	8	From milk	effective
	Harvesting	10	From milk	effective
3	Plant establishment	2	Decayed vegetables and fruits	effective
	Vegetative	4	Decayed vegetables and fruits	effective
	Flower insertion to first pick up	5	Decayed vegetables and fruits	effective

In trial 3, the crossover probability returned to the control probability of 0.5 and the = mutation probability was set to 0.3. The number of evaluations dropped from 250 to 163 in between generation 1 and 2. From there, the average amount of evaluations was 161.2. As the generation number grew, there was an increase in the average amount of food eaten, shown in Table 3. However, the maximum amount of food eaten by generation 40 was 62, demonstrating the problem setting did not create a solution. generation 40 was 62, demonstrating the problem setting did not create a solution.

Discussion. Effect of Mating. Three trials were run to understand the effect of crossover and mutation in genetic algorithms. Through the results, the independent effects of mating and mutating are clear. For trials 1 and 2, they both began at 250 evaluations but drastically decreased in generation 2. However, trial 2, which tested crossover/mating, experienced a more drastic decrease in number of evaluations. This could be due to the fact that more programs were combining and evolving, which created an increase in the population counter. This outcome supports the results from previous research[4]. Along with the population count, the maximum amount of food eaten decreased between trial 1 and 2 as there was a 39-pellet-decrease from trial 1. This could also be caused by the increase in mating. Because more programs were crossing over, they evolved faster. Data from trials 1 and 2 support the first hypothesis that an increase in the probability of crossover will increase the amount of evaluations and the maximum amount of food pellets eaten.

Effect of Mutating. Comparing trials 1 and 3, the probability of mutating was changed from 0.2 to 0.3. Both trials began at 250 evaluations but experienced a significant drop. The decrease in trial 1 was greater than trial 3, with populations of 150 and 160 respectively. This drop could be due to the greater mutation rate slightly increasing the chances of crossover occurring. Because more programs were being mutated, more mating occurred and took on the new programs' features. Continuing, with the increase in mutating probability there was a decline in the maximum amount of food eaten. A possible cause for this is there were more negative mutations than positive mutations, or more traits that caused the ant to fall for the holes in the trail were being generated. Therefore, the mutations set the programs back on their goal to venture the trail. This falsifies the second hypothesis because as the mutation rate increased, the number of evaluations increased and the maximum amount of food eaten decreased. This shows that although it increased the algorithm's efficiency, higher mutations caused fewer possible solutions to be created.

Trends Found in Evaluations and Average Amount of Food Eaten. By the 5th generation, most data points regarding the number of evaluations and the maximum amount of food eaten had linearized. In trial 1, it linearized around 150, trial 2 linearized around 130, and trial 3 linearized around 160. This indicates that, over time, the effectiveness of evaluation diminishes. Similarly, the average food eaten constantly grew throughout the generations and trials, slowing down around generation 20. The standard deviation grew until about generation 15 and then began to decline slowly. This could be because most of the programs reached their peak by generation 20 and did not reproduce to create more efficient program.

Application. The results presented above can be used to optimize genetic algorithms by decreasing the amount of trial-and-error necessary. By using semi-random techniques to generate a population of programs and narrow its search to the solution, the efficiency of the algorithms is increased and allows the users to get their solutions faster.

Genetic algorithms are mainly applied for designing parts with specific tasks. For example, genetic algorithms are used to create race car parts to increase speed and aerodynamics. Another application is gene expression profiling. This method measures the activity of thousands of genes at once to determine a pattern the genes express. Genetic algorithms make this analysis more efficient.

If the results from this experiment are implemented, it may further increase the speed of such programs, making it possible for researchers to focus on individual patients' unique gene expression profiles.

Future research. The code from trial 1 that was able to obtain all 89 food pellets in the trail at the end of generation 40, can be compared to Koza's solution[9]. This may demonstrate this experiment's algorithm is more effective than the previous solution. Through comparison, a more efficient algorithm can be found and/or created. This could then be applied to genetic benchmark problems outside of the Santa Fe Ant Trail. Along with comparing the generated program with the previous solution, the code can be applied to an Arduino bot to understand if algorithms can be successfully transitioned from a virtual world to the real world. This could help decrease the amount of time it takes to test certain programs because testing in the real world is less efficient than in the virtual world.

Conclusion. This project aimed to find the effect of crossovers and mutations on a genetic algorithm using the Santa Fe Ant Trail. Crossover refers to when two programs in the population combine their traits or features to create a new program. Mutation refers to when a random feature is generated and introduced to the population of solutions. When the probability of mating increased the amount of the evaluation population increased and the maximum amount of food pellets increased. When the probability of mutating decreased the number of evaluations increased but the maximum amount of food eaten decreased. This data can optimize the process of creating efficient genetic algorithms and change the way engineers implement algorithms into the real world.

Methods.

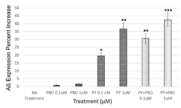


Figure 4. ELISA of PY and PBO on Aß protein concentration. * = p< 0.05; ** = p<.005; *** = p<0.001. Data is presented as mean ± SEM from four repeated experiments.

Figure [4]. An example of how the virtual world appears when the program is run. The blue dot is the ant, the green dots are the food, and the white dots are the ant's trail

Mentor Role. Throughout the research period of 18 weeks encompassing the summer and fall of 2018, my mentor and I collaborated on various aspects of the experiment. My mentor provided advice and background on genetic algorithms in evolutionary computation, their implementation, and how to benchmark them. My mentor also provided a pseudo-code to give a visual representation of any inputted genetic algorithm. Following the pseudo-code and my mentor's guidance, I created a genetic algorithm benchmarking test that mirrored the Santa Fe Ant Trail and implemented it using several different settings.

Virtual World. A virtual world, or computer-based stimulated environment, was created and implemented to display graphics of the implemented genetic algorithm. It was designed to display the Santa Fe Ant Trail and the program carrying out its path, shown in Figure 4. The programming language Python was used to create the virtual world because of its incorporated wide range of modules that allow for code to be created in a short and efficient manner[11]. The compiler/tester Eclipse was used to implement Python. The cloud/interpreter Anaconda, which comes preinstalled with popular Python packages and environments, was used to import packages, or groups of modules, for the program including NumPy and DEAP (Figure 5). NumPy allows the user to use high-level mathematical functions and DEAP allows the user to use genetic algorithm functions.

Figure [5]. Examples of packages in the Anaconda cloud. The package name is on the left and a brief summary of its functions is on the right.

Santa Fe Ant Trail. The Santa Fe Ant Trail was implemented to provide a benchmark for the genetic algorithm. One necessary component to the Trail problem is the ant. The ant was designed to implement a genetic algorithm and eat the food. the algorithm would be tested based on the efficiency of the algorithm's ability to maneuver through the complex trail.

DOI:10.36838/v2i2.4

Figure 6. The pseudo code for Koza's 1992 solution reading from top to bottom and left to right[9]. "PROGN" means to carry out all of its following branches.

Algorithms. The algorithms tested in the Santa Fe Ant Trail included a random case to provide a base, a previously tested solution to provide a benchmark, and genetic programs to understand different methods to approach the problem. The solution algorithm was taken from Koza[9]. As shown in Figure 6, the old solution uses a combination of "if" statements and programs.

The code was created to implement new genetic algorithms in every trial. There were four inputs for the code including the initial population, the probability of mating, the probability of mutating, and the amount of generations. Each trial maintained the initial population (250) and the number of generations (40). At the end of each trial, the most efficient set of directions is outputted and recorded. Along with this, the amount of evolutions, average amount of food eaten, and maximum amount of food eaten is outputted for each generation. Once the effects of each variable are calculated, the variables will be manipulated to output the best solution to the Santa Fe Ant Trail.

Acknowledgements. I would like to thank my parents, Ajeet and Sanjita Gupta, and my sister, Amani Gupta. I would also like to thank my teachers, Michael Blueglass and Rachel Koenigstein. Lastly, I would like to thank my mentor, Theodore van Kessel, for aiding me throughout the project.

References.

(1) Dyer, D. W. (2010). Evolutionary computation in Java: A practical guide to the Watchmaker Framework. Uncommons.org. https://watchmaker.uncommons.org/manual/

(2) Mallawaarachchi, V. (2017). Introduction to genetic algorithms – including example code. Medium. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

(3) Even-parity problem. (n.d.). DEAP Project. https://deap.readthedocs.io/en/master/examples/gp_parity.html

(4) Wilson, D., & Kaur, D. (2014). How Santa Fe ants evolve. Neural and Evolutionary Computing. https://arxiv.org/pdf/1312.1858v2.pdf

(5) Langdon, W., & Poli, R. (1998). Why ants are hard (CSRP-98-04). Birmingham, England: University of Birmingham. http://www0.cs.ucl.ac.uk/staff/W.Langdon/antspace_csrp-98-04/

(6) Langdon, W., & Poli, R. (1998). Better trained ants for genetic programming (CSRP-98-12). Birmingham, England: University of Birmingham.

(7) Miller, J. F., & Thomson, P. (2000). Cartesian genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., & Fogarty, T. C. (Eds.), Genetic programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 2000 proceedings (pp. 121-132). Lecture notes in computer science (Vol. 1802). Berlin, Germany: Springer. https://doi.org/10.1007/978-3-540-46239-2_9

(8) Chellapilla, K. (1997). Evolutionary programming with tree mutations: Evolving computer programs without crossover. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., & Riolo, R.L. (Eds.), Genetic programming 1997: Proceedings of the second annual conference (pp.431-438). Stanford, CA: Stanford University.

(9) Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. Cambridge, MA: MIT Press.

(10) Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., & Wang, A. (1991). Evolution as a theme in artificial life: The Genesys/Tracker system. In Langton C. G., Taylor, C., Farmer, J. D., & Rasmussen, S. (Eds.), Artificial Life II: Proceedings of the workshop on artificial life held February 1990 in Santa Fe, New Mexico (Vol. X) (pp. 549-578). Redwood City, CA: Addison-Wesley. https://www.gwern.net/docs/ai/1992-langton-artificiallife-2.pdf

(11) Mindfire Solutions. (2017). Python: 7 important reasons why you should use Python. Medium. https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b

(12) Galván-López, E., McDermott, J., O'Neill, M., & Brabazon, A. (2010). Towards an understanding of locality in genetic programming. In GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation (pp. 901-908). https://doi.org/10.1145/1830483.1830646

(13) McDermott, J., Galván-Lopéz, E., & O'Neill, M. (2010) A fine-grained view of GP locality with binary decision diagrams as ant phenotypes. In Schaefer, R., Coota, C., Kolodziej, J., & Rudolph, G. (Eds.), Parallel problem solving from nature – PPSN XI: 11th international conference, Krakov, Poland, September 2010 proceedings, part 1 (pp. 164-173). Lecture notes in computer science (Vol. 6238). Berlin, Germany: Springer. https://doi.org/10.1007/978-3-642-15844-5_17

(14) 14.O'Neill, M., & and Ryan, C. (1999) Evolving multi-line compliable C programs. In Poli, R., Nordin, P., Langdon, W. B., Fogarty, T. C. (Eds.), Genetic programming: Second European workshop, EuroGP'99, Göteberg, Sweden, May 1999 proceedings (pp. 83-92). Lecture notes in computer science (Vol. 1598). Berlin, Germany: Springer. https://doi.org/10.1007/3-540-48885-5_7

(15) Oplatková, Z., & Zelinka, I. (2009). Investigation on evolutionary synthesis of movement commands. Modelling and Simulation in Engineering, 2009(845080), 1-12. https://doi.org/10.1155/2009/845080

Author. Nimisha Gupta is a rising senior at Yorktown High School. She loves computer science and is fascinated by algorithms and AI. She plans to double major in computer science and some form of engineering in college.