Memetic Algorithms for Optimizing Battleship Search Heuristics

Ken DeVoe – October 2021

Abstract

By writing the search heuristic parameters as a genetic sequence it was possible to improve an agents play for the game of battleship with memetic algorithms. Moves to win improved from 96 with random cell selection as a baseline to 75 with population seeding and finally down to 62 using gene mutations. There is still room for improvement compared to deep reinforcement learning results of 52 and mathematical spatial modeling of 42. However memetic algorithms were shown to be a viable optimization strategy for searching through a large state space.

Introduction

Memetic Algorithms

Memetic algorithms are used to perform search and optimization over potential state spaces [1]. If an algorithms search parameters can be represented as a genetic sequence it is possible to modify and optimize these parameters using biologically inspired mechanisms such as population seeding and selection, mutation and gene crossover [1]. Memetic algorithms saw successful implementation in the 1960's for applications such as shape optimization in wind tunnels and water nozzle design [2]. In general this algorithm is able to perform local search for an optimum solution in complex state-spaces [1] [2]. In this paper the use of memetic algorithms will be explored towards the game of battleship. In addition a comparison will be made of performance against other methods such as neural networks and mathematical spatial modeling.

Motivation for Using Battleship

The game of battleship provides a unique environment for testing artificial intelligence agents that perform search. Battleship is a 10 x 10 grid game where each player places 5 ships randomly on their grid without their opponent observing. The goal of each player is to shoot at the opponents grid to eventually hit all the grid squares with ships and sink each ship. After each shot the opponent informs the player the result from that square on the grid (hit a ship, missed, or sunk a ship). Complete rules are provided on Wikipedia [3]. In this paper the agent is built to search for ships that have been randomly placed by an opponent.

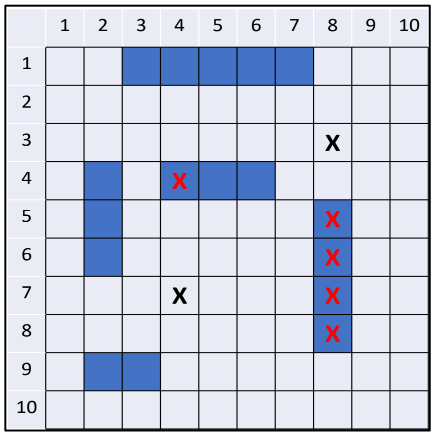


Figure 1 Example of battleship gameboard. Blue squares represent ships, with X's marking guessed locations.

Battleship has complexity in that the board is partially observable to the agent, ship placement is stochastic, and information revealed by guesses is sequential. However at the same time the combination of single agent play, discrete moves, small 10 x 10 grid board size, and a well-defined performance metric (total guesses to win) makes this environment simple to train with and evaluate effects of modifications to an algorithm. Several other groups and individuals have used Battleship to train algorithms such as deep reinforcement learning [4] and mathematical spatial models [5].

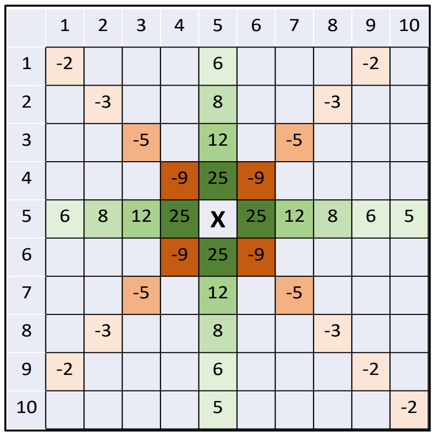
Methods

Performance Metric

As a performance metric the total number of guesses to sink all ships on the board was used. A lower number of guesses to complete the game is considered better performance. As the grid size is 10 x 10 and guesses are not repeated this gives an upper bound of 100. The number of squares for ships = 5 + 4 + 3 + 3 + 2 = 17 which is the lower bound for a score.

Search Heuristic

A search heuristic was built to calculate the probability of a ship being located within a specific cell on the board. The known value of a cell is used to calculate the probability of neighboring cells to have a ship. For example, if cell (5, 5) is known to be a miss, then that miss value contributes to the calculation of cells in the same column, row or diagonal space. Figure 2 shows how these values are propagated through the board.



	1	2	3	4	5	6	7	8	9	10
1	-2				6				-2	
2		-3			8			-3		
3			-5		12		-5			
4				-9	25	-9				
5	6	8	12	25	**X**	25	12	8	6	5
6				-9	25	-9				
7			-5		12		-5			
8		-3			8			-3		
9	-2				6				-2	
10					5					-2

Figure 2 Representation of how the value from a cell is propagated to the rest of the board cells.

How probability values are propagated is controlled by an inverse polynomial function as shown in Figure 3.

$$\frac{Initial\ Value}{Distance^{\left(\frac{fade}{5}\right)}}$$

Figure 3 Formula for calculation of propagating probability values across the board. Initial value and fade are input parameters, distance corresponds the number of cells away from the initial cell.

The initial value along with the decay value (degree of this function) form the basis of calculation for the degree of influence. Finally known cell values are differentiated by type as either open (unchecked), miss, hit (ship found but not sunk yet) or sunk (whole ship is hit).

Gene Structure

By representing each search parameter as a string of digits it is possible to form a genetic sequence that represents a unique search algorithm. Figure 4 shows this genetic sequence of 16 digits total.

Open				Miss				Hit				Sunk			
Horz / Vert		Diagonal		Horz / Vert		Diagonal		Horz / Vert		Diagonal		Horz / Vert		Diagonal	
Init	Dec	Init	Dec	Init	Dec	Init	Dec	Init	Dec	Init	Dec	Init	Dec	Init	Dec
8	-6	-10	2	-6	-9	9	-5	-6	10	-9	-5	0	2	2	8

Figure 4 Genetic structure with example values for 16 markers.

First the genetic sequence is broken into 4 distinct parts, one each for the possible state of a known cell: open, miss, hit or sunk. Each of these parts is subdivided into two directional groups: horizontal/vertical direction or diagonal direction. Finally, these subgroups contain two parameters, the first to set the initial value and the second to set the "decay" value. Each of these digits is allowed to take integer values from -10 to 10 (including 0). This gave an overall possible state space of 21^{16} valid genes or 1.4×10^{21} states. The state space was intentionally set to be large to evaluate the performance of a memetic algorithm for finding an optimal state large to infinite search space.

Genetic Variation

To compare how similar a set of genes was a formula to measure variation within genes was created.

$$\frac{\sum_{i=1}^{16} Stdev(marker\ i)}{Num\ of\ Markers * \ Marker\ Range} * 2$$

Figure 5 Formula for calculation of genetic variation.

For a set of genes the average of standard deviations for each marker was calculated. Then this value was divided by the total range of the marker values (Max – Min + 1). Finally because two genetically opposite genes provide a variation value of 0.5 the final value was multiplied by 2 to normalize to 1.0 for maximum variation. This variation value is used to determine how much similar genetic values are within a set of genes, with lower numbers indicating more similar genes.

Memetic Algorithm for Optimization

In this paper the application of a memetic algorithm included three parts. These parts were initializing the genetic sequences through random seeding and modifying them through genetic mutation.

Random Seeding

To initialize search parameters a random set of valid genes was formed. Each of these random genetic sequences was tested against ten randomly generated boards which served as the training set. For each genetic sequence the average score from each of the ten boards in the training set was calculated as the performance of each gene. Initial population sizes of n=100 to n=32000 were evaluated, with down selection to the top 100 genes from each population.

Gene Mutation

Genetic sequences were mutated by selecting one random marker and either increasing or decreasing that marker by a specified amount. Marker values ranged from -10 to 10 in this trial and change amounts tested were 1, 3, 6 and 10 which corresponded to approximately 5-50% of the markers potential range. After a random change to a single marker the new genetic sequence is tested and kept if it performs better than before the change, or not used if it does not improve performance.
If the random change is outside the markers range (for example if a marker is set at 10 and proposed change is + 6) then the change direction was reversed and the amount of change was reduced by half.

Results

Random Seeding

For the first part of the memetic algorithm genetic sequences were initiated randomly. This was considered as a random search through the state space of possible genetic sequences. When

100 random genetic sequences were created and tested the average of the scores was 93.3 ± 3.3. It was noted that this initial score showed some improvement over the strategy of randomly guessing squares, which was 96 moves per game [5].
As the overall pool size increased the scores from the top 100 performers also increased as expected shown in Figure 6.

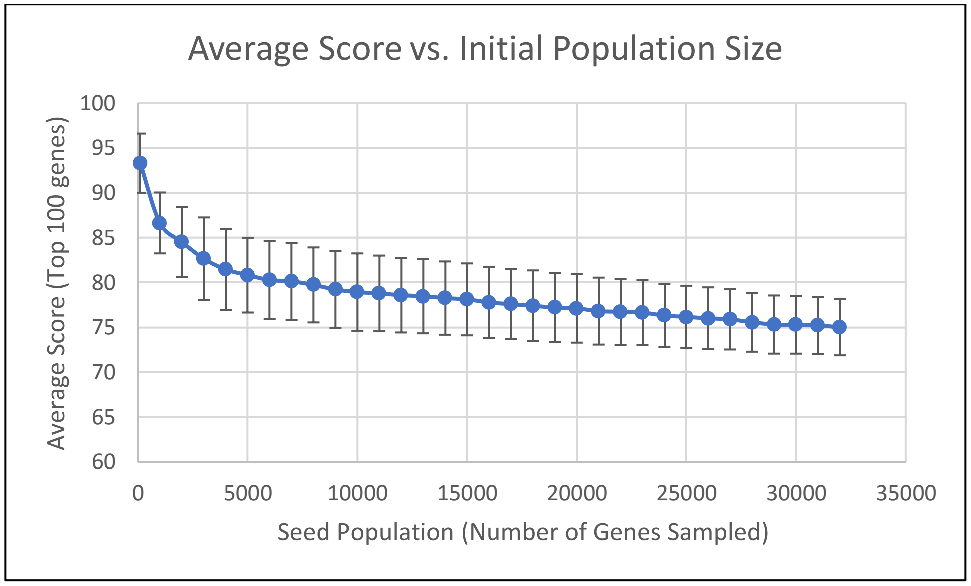


Figure 6 Average score of the top 100 genes selected from various initial population sizes.

However this strategy did show diminishing returns, with an inflection point around a population of 5000. Shown in Figure 7 the performance improvement appears to be a base two logarithmic function of the sample size n. Doubling the population size only results in a linear improvement in the top 100 scores. Extending this trend out to the maximum number of states of 1.4 x 10^{21} the log base 2 value is ~70 which gives a predicted average score of -44 if all states were used as a seed population. Since this is lower than the best possible score of 17 this indicates the overall rate of improvement must slow down beyond the log base 2 trend at higher population sizes.

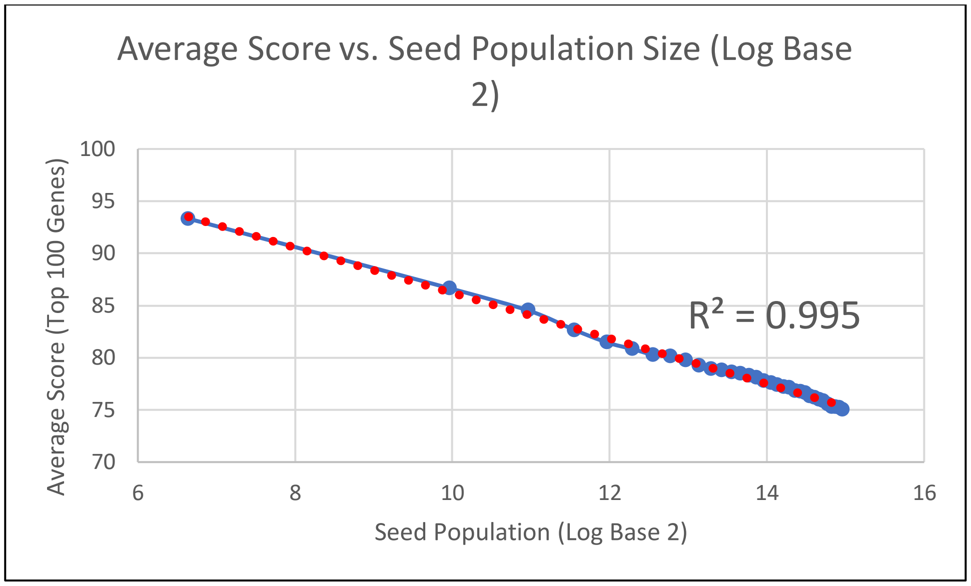


Figure 7 Average score vs initial seed population on a log base 2 scale.

Finally the genetic variation of the top 100 sequences as a function of seed population size is shown in Figure 8. Initial variation values start around 0.6 for a purely random set of sequences, and drops to around 0.45 overall. This trend is similar to the performance improvement from Figure 7 indicating that genetic values start to converge with selection of top performing sequences.

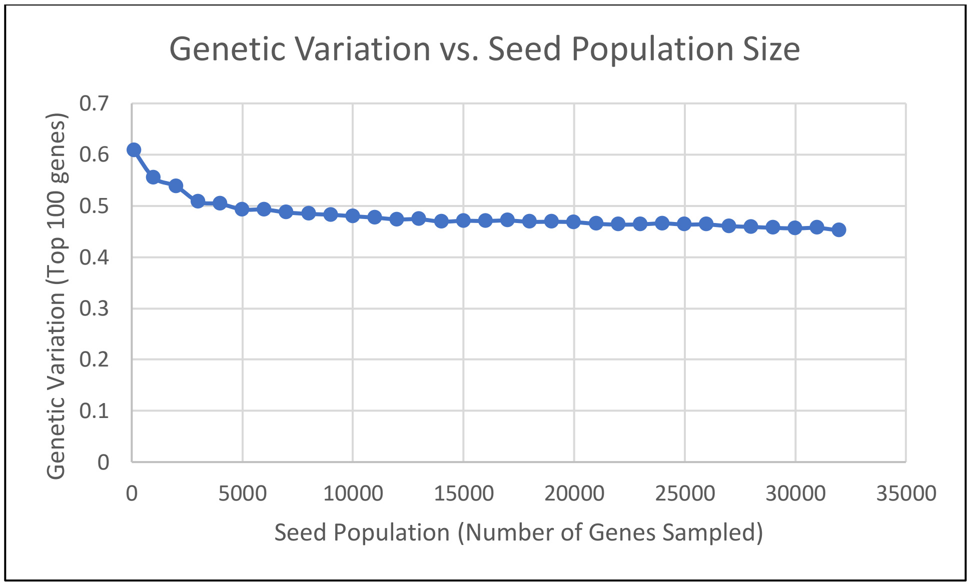


Figure 8 Genetic variation as a function of seed population size for the top 100 genes.

Gene Mutation

After random seeding was performed, the top 100 genes from a random pool of 5000 was then mutated for up to 100 generations with results shown in Figure 9. For each set of mutations the amount a single marker could change at a time was varied between 1, 3, 6 and 10. The rate of improvement generally increased with an increase in change size up to 6. The change size of 10 also showed improved performance up to generation 20, however at higher generation sizes the lower change value of 6 shows better scores. This crossover is believed to be due to the trade-off of being able to move quickly through the state space (large changes) and the ability to fine tune and progress towards minima for scores (small changes).

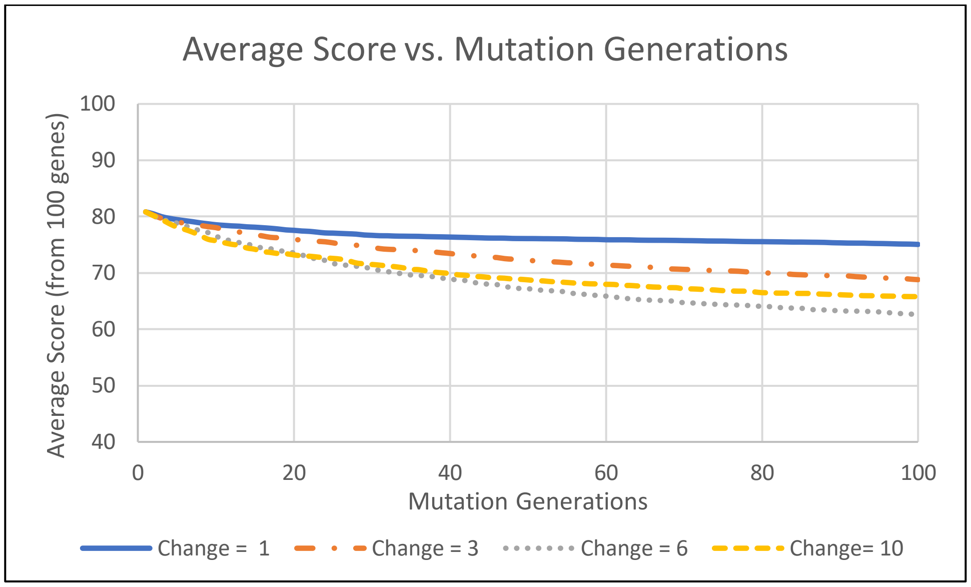


Figure 9 Average score as a function of generation across 100 genes. Potential change in genetic markers varied from 1 to 10.

Figure 10 shows the change in variation of the top 100 genes as a function of mutation generation. The genetic variation dropped faster for larger change sizes, indicating that the larger changes allowed the set of genes to converge faster.

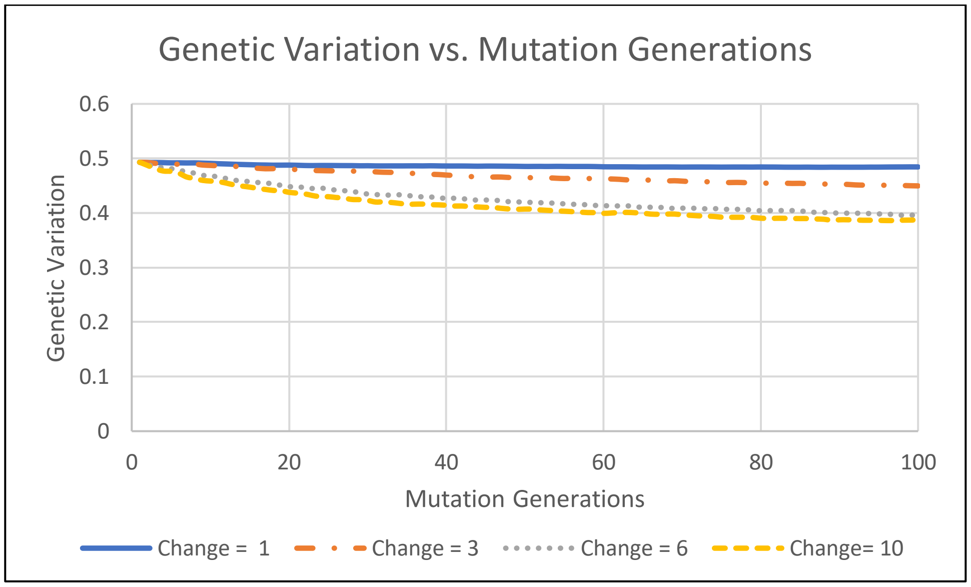


Figure 10 Change in genetic variation with mutation generation.

One final item for mutation that was evaluated was score improvement on an individual gene level. Figure 11 shows an individual breakdown of scores before and after mutation with change equal to 6. The top line shows the performance scores before mutation, with values ranging from 65.1 to 84.8. The lower line shows the final score after mutating for 100 generations. In general there was no strong trend between initial and final score. In extreme cases a gene showed no improvement to the score even after attempting 100 modified values. This was attributed to the gene being located at a local score minima and unable to escape. In contrast gene number 99 showed the biggest improvement in performance, improving from an average score of 84.8 to 51.4.

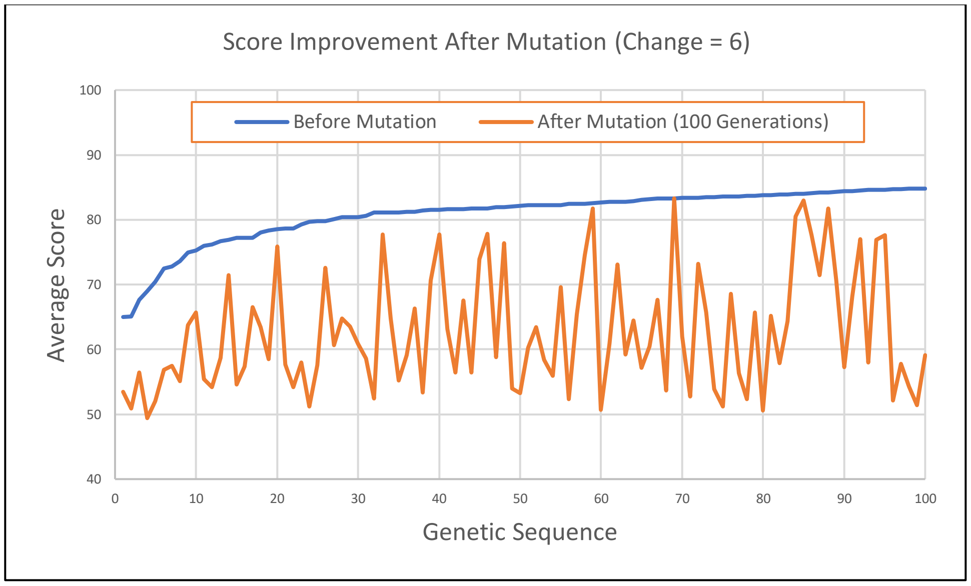


Figure 11 Comparison of individual gene scores before and after mutation. Example from mutations with change size of 6.

Overall the method of gene mutation was effective for greatly improving scores even within the first 100 generations. Using a change rate of 6, it was possible to take the average of top 100 gene scores from 80.8 down to 62.6. Compared to random seeding which only reached a score of 70.0 after 32,000 genes and diminishing returns the mutation method showed a dramatic improvement.

Conclusions

By using a simple search heuristic and evolutionary algorithmic techniques it was possible to significantly improve the play of an agent for the game of battleship. The first technique of random population seeding showed good performance improvements up to around a population of 5000 genetic sequences, however this performance improvement dropped off dramatically as a logarithmic function of sample size. The second technique of gene mutation provided further improvement, with best results from setting the genetic marker change size to

6, or about 30% of the range of marker values. Ultimately the best result of an average 62 moves per game was achieved after performing both initial seeding and mutation together on genetic sequences.

Comparison to Other Methods

Unfortunately while the memetic algorithm explored here did show significant improvement over baseline random play it did not catch up to the performance reported for other algorithms. In the deep reinforcement Q-learning study a best average performance of 52 moves per game was reported, 10 moves better per game than that shown here after running mutations. For the probability algorithm using spatial modeling performance was even better, down to 40 – 45 moves per game.

Future Improvements

There is room for performance improvements in the current memetic algorithm where for example even at 100 generations the mutation algorithm was still showing improvement. By increasing the number of mutation generations or further optimizing the change amount performance better than 62 moves could likely be achieved. Gene-splicing is another technique that could be implemented to further improve gene selection. [1]

Works Cited

[1] S. Russel and P. Norvig, Artificial Intelligence A Modern Approach, New Jersey: Pearson, 2021.

[2] H.-G. Beyer, "Evolution strategies - A comprehensive introduction," *Natural Computing,* vol. I, 2002.

[3] "Wikipedia," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Battleship_(game).

[4] S. He, "Deep Reinforcement Learning-of how to win at Battleship," 25 August 2017. [Online]. Available: https://ccri.com/deep-reinforcement-learning-win-battleship/. [Accessed 30 September 2021].

[5] N. Berry, "Battleship," 3 December 2011. [Online]. Available: https://www.datagenetics.com/blog/december32011/. [Accessed 15 September 2021].