DePosit, Mass, Tolleson

CS541 Artificial Intelligence, Prof. Rhodes

Winter 2020 Term Project Report

Table of Contents

[Overview 1](#_Toc35429061)

[Old & Busted – Genetic Algorithm 1](#_Toc35429062)

[New Hotness - Constraints Satisfaction 2](#_Toc35429063)

[Conclusion 2](#_Toc35429064)

[Appendix A – TwoNotTouch from the New York Times [2] 3](#_Toc35429065)

[Appendix B – StarBattle [3] 4](#_Toc35429066)

[Works Cited 5](#_Toc35429067)

# Overview

For our final coding project, we chose to create solvers for the puzzle TwoNotTouch (as so named in The New York Times), also more commonly known as StarBattle. We chose to primarily attempt to solve a puzzle of regular difficulty which is a 10x10 board with 10 regions onto which a user is to place 20 stars. Each row, column, and region is to contain exactly 2 stars, and no 2 stars may touch one another.

We attempted two methods of solving this problem; a genetic algorithm and constraints satisfaction. We adapted the genetic algorithmic solver from the 8-Queens programming assignment since the environments are structured similarly. A novel constraint satisfaction algorithm was coded. In this paper we will summarize the logic behind both approaches and compare their results.

# Old & Busted – Genetic Algorithm

For our second programming assignment for this course, we were tasked with solving the 8-Queens problem using a genetic algorithm. While three of us coded very different implementations of the genetic algorithmic solver, the underlying logic for all three are based on the Genetic Algorithm described in our class lecture and the Russell Norvig textbook. [1]

At a very high level, these are the steps the original genetic algorithm took:

1. Create k random specimens for the first generation. Each specimen represents the location of 20 stars on a 10x10 board.
2. Calculate the fitness for each specimen. For this puzzle we calculated the maximum fitness to be 55, from which we subtract the number of pairs of adjoining stars and the number of regions containing less than 2 stars.
3. Create the next generation by simulated evolution:
   1. weighted random selection based on relative fitness for breeding pairs,
   2. crossover of star location segments between pairs, and
   3. random mutation of one specific star location.
4. Calculate fitness for each specimen in new generation.
5. Repeat for N generations.

A close up of a map

Description automatically generated

The graph above demonstrates that the GA gets us into an average generation fitness of approximately 37.5 (55 is the maximum fitness measure) fairly quickly. However, the performance plateaus fairly quickly thereafter.

After getting these unimpressive results, we tweaked some parameters of the algorithm. Here are some of the changes we made in attempt to eke out more performance from the algorithm.

1. Mutation – instead of changing just one star location in a mutation, we reduce the risk of creating illegal star locations by swapping two stars within a random set of stars. Each specimen contains two sets of ten stars. Each set of stars consists of a list of 10 integers. The combination of an index value and a list value is the row/column representation of a star location. By swapping two star column values we ensure that each row/column pair within a star set is unique. This didn’t change the performance in a meaningful way.

A close up of a map

Description automatically generated

1. Crossover – we changed the crossover function from swapping the tail ends of a random star set to swapping entire star sets. This managed to push our plateau up from about 37.5 to about 40.

A close up of a map

Description automatically generated

1. Natural Selection – We modified the probability of a specimen being selected for reproducing a member of the next generation such that if a specimen’s fitness evaluation is below 44, the probability of reproduction is reduced to 0. This produced similar results as previous iterations.

A screenshot of a cell phone

Description automatically generated

1. Finally, we observed that most of the gains from the crossover occurs rather quickly. So we capped the crossovers at 40 turns, after which future generations are modified only by mutation. Since we capped the crossovers, we also increased the mutation rate from 20% to 90%. This seems to get us up to that familiar plateau quicker and decreases our overall processing time.

A screenshot of a cell phone

Description automatically generated

The only thing left to try at this point is to increase the number of generations from 100 to experimentally larger generation caps. Essentially at this point we are just hoping to randomly find a solution. As of the writing of this draft report, so solution has been found in any of the GA experiments.

INSERT CHART OF HUGE POPULATION AND HUGE GENERATIONS HERE

# New Hotness - Constraints Satisfaction

Black magic voodoo stuff goes here.

# Conclusion

Pull it all together here.

# Appendix A – TwoNotTouch from the New York Times [2]

A picture containing text, refrigerator

Description automatically generated

# Appendix B – StarBattle [3]

A picture containing building, drawing

Description automatically generated

# Works Cited

|  |  |
| --- | --- |
| [1] | P. N. Stuart Russell, Artificial Intelligence: A Modern Approach, 3rd Edition, Upper Saddle River: Prentice Hall, 2010, pp. 126-129. |
| [2] | "The New York Times," *The New York Times,* 2019. |
| [3] | Puzzle Star Battle, "Puzzle-Star-Battle," 2020. [Online]. Available: https://www.puzzle-star-battle/com/?size=5. [Accessed 2020]. |