Code Cracker : Mastermind Solution Space Explored by Genetic Algorithm and Wisdom of the Artificial Crowd Modification

Maggie Zirnhelt
zirnh008@umn.edu

May 11, 2020

1 Abstract

The game called Mastermind Codebreaker will be the focus of this project. A description of the game can be found at this link : Mastermind gameplay. The goal of my project is to employ a genetic algorithm with Wisdom of the Crowd selection at the end of every 10 generations, with the exception of the first move, to solve the game in a small number of moves. I chose the game Mastermind because I enjoy playing it and it is interesting to see what moves a reasonable algorithm might make in comparison with the moves that I might make. I chose to use a genetic algorithm because this game is one where each move evolves from the last. The Wisdom of the Crowd modification was added because each round of the game only plays one move so, to keep the decision tree at a small size, the "best" choice is elected for each move instead of several different moves playing out individually. Genetic algorithms with a Wisdom of the Crowd modification were successful in the literature.

2 Introduction

Constraint Satisfaction Problems (CSPs) compose a field of study that is broad but well organized. Mastermind, a two player game that involves a *codemaker* and a *codebreaker*, is a CSP with constraints that change throughout the duration of the game. In literature surrounding this topic, gameplay is described as "naive" and "uninformed." These two words are synonymous in this context and simply mean to say that the codebreaker knows only the information suggested by the codemaker's responses. A narrower scope than the general problem of CSPs is that of Mastermind's solution by a Genetic Algorithm (GA). Wisdom of the Artificial Crowd (WoAC) is used as a modification to GAs in several articles and will also be considered a part of the narrower scope of interest. This topic is important because besides indicating refinement in search methods over time, [8] indicates that the particular problem of Mastermind can have real world implications in terms of determining bank account PINs. Some trends in the field are a use of GAs, about 4 guesses for a standard game with $N = 6$ colors and $P = 4$ pieces, and exploration of significantly large search spaces. On the topic of uninformed CSP optimization problems, the literature covers enumeration, tactical, and meta-heuristic style techniques that can be stepwise or strategically optimized for slightly different goals.

The problem of solving Constraint Satisfaction Problems efficiently is an interesting one because CSPs are a format in which many problems may be portrayed. The infamous Traveling Salesman problem and everyday issues such as finding the cheapest available flight or job scheduling problems can be all be framed in this way. Since so many problems come in this form, it is of high interest to efficiently solve these kinds of problems. On a more intellectually stimulating note, it is interesting to find solutions to problems that are naive because a solution can be found even from the most sparse information. Solving difficult problems, sometimes that belong to the N^P hard solution space, is a way to measure how human problem solving has evolved in efficiency since humans have begun to track such a thing.

The particular problem of Mastermind is challenging because of how quickly the solution space grows. A standard game is played with $N = 6$ colors and $P = 4$ pieces as was mentioned earlier where the solution space is $6^4 = 1296$. With even the addition of one more piece, the solution space leaps to 7776. Multiple existing articles describe methods to search spaces as large as $N = P = 16$. The first approaches to the standard problem came in the form of full enumeration. While this worked for the relatively small search space, it would simply not do for the $N = P = 16$ search space. Enumeration is not a common approach anymore and it is the naive approach of the approaches considered in the literature. As soon as the search space is any larger than the original, the naive approach fails to keep up with tactical and meta-heuristic based approaches in terms of time and space complexity.

Key components of the approach described in this article are the use of a GA where different generation sizes and different numbers of generations were tested per turn, the use of WoAC to determine which of the candidate solutions to play after a series of generations

have passed, and the composition of a fitness function that can consisely determine which members of a population are the most expert. Genetic algorithms allow for several variable factors which can be adjusted to maximize both time and space complexity for a given machine. Also, the occasional mutation that a genetic algorithm allows will ensure that local minima are not traps for a changing population. Wisdom of the Artificial Crowd in this particular approach collects a number of experts from each generation into a multi-generational pool. After all of the generations for a turn have passed, the expert of experts is chosen to be played as the current guess. The expert of experts is the expert candidate that has the most in common with the other expert candidates. WoAC allows for good decisions to be rewarded and bad decisions to be disregarded. Finally, the description of the fitness function used in this approach is more detailed than most descriptions of fitness functions found in the literature. The fitness function is of interest because in order to judge the value of a candidate, each candidate must be compared against several rules. This could be time and space consuming. The fitness function described in this approach ensures an efficient assignment of fitness.

3 Literature Review

The algorithms discussed in the literature are grouped into enumeration, tactical, and meta-heuristic approach methods. Enumeration is no longer a popular approach because of not only how much time it takes to search all of a standard game's search space ($for N = 6 and P = 4, N^P = 1296$) but for how much time it would take to search any other large search space that is being considered today. The only considered article that used this technique was [2], likely because bank account PIN numbers are only ever $N = 6 and P = 4$. Tactical approaches follow a particular rule in order to find a solution. For example, Knuth in [4] uses an approach where half of the colors are always used in the first guess and according to the kind of response, an action will be executed. Finally, the most popular method of searching Mastermind's solution space is by meta-heuristic. This encompasses Genetic (also known as Evolutionary) Algorithms, their variations, and Particle Swarm Optimization (PSO). The majority of considered papers employ some version of meta-heuristic. Merelo-Guervós et al. [6] and Partynski [7] are special cases of this catagory. In both of these papers, the researchers used a strategy called *hypermutation* when the algorithm stopped producing usable results after a defined number of iterations. In hypermutation, the entire population is replaced. There is another characteristic that makes [7] special, though. It was the only investigated article that discussed a variation of PSO where in one iteration, particles gather at the local minimum of their cluster and then at the global minimum to determine which combination is the most fit. This is reminicient of Wisdom of the Artificial Crowd in that each individual agent makes a decision and helpful decisions are reinforced.

General approaches do not necessarily dictate whether stepwise or strategically optimal techniques will be applied. Stepwise optimization is defined as playing an eligible guess for

each move. This is intuitive and allows the chance for any guess to be correct. Most algorithms in the literature proceed with this strategy including Berghman et al. [1], Partynski [7], and Knuth [4]. Indeed Partynski [7] states that their algorithm consistently found a solution in the largest solution space mentioned in the literature to date. For $N = P = 16$, their algorithm was able to find a solution. The alternative is strategic optimization. Strategic optimization is different from stepwise in that not every guess that the algorithm makes is an eligible solution. Each guess is, however, calculated and more informative than an eligible guess could be for an ineligible move. Strategic optimization also allows for the number of possible guesses evaluated to be decreased and therefore allows the search space to become smaller. This strategy is less intuitive and could be thought to "waste" valuable guesses because some guesses will have no chance of being correct. Depending on a researcher's goals, either strategy can be used with any general approach.

Port and Yampolskiy [8] discuss an approach similar to the one described in this report in that they both use GA and WoAC. Their proposed approach introduces a population of many random guesses, reproduces, and occasionally mutates during the reproduction. New children replace their parents in the population so that the population does not grow. Reproducing, in this approach, can be any combination of parent 1 and parent 2 where for a crossing point r and n colors, a child has $1 - r$ pieces of parent 1 and $(r + 1) - n$ pieces of parent to. The fitness measure in this approach is briefly touched upon, describing that each guess recieves a score that is compared to the scores of previous guesses. Upon deciding which candidate to play, the candidate would ideally be the one that leaves the least remaining other possible candidates. Port and Yampolskiy mention that this takes a long time, however, and being time efficient was prioritized in their method. The expert is played that displays the maximum amount of similarity to the rest of the group.

In another article discussing Wisdom of the Artificial Crowd, Ryan Hughes [3] writes of the problem of Sudoku. This approach, although for a different problem, also uses the approach of a genetic algorithm together with WoAC. Sudoku is a slightly different problem that Mastermind in that it is not an uninformed scope. Everything that is necessary for a solution is immediately available and there is no dynamic element to sudoku where there is in Mastermind. Despite these differences, the proposed algorithm in this method also involves an initial random population, reproduction, and occasional mutation. A point of interest in this report is that two reproduction functions exist; one that randomly chooses one of the two parents to use as the whole child and the other that takes several pieces of the parent at a time to have in the child. WoAC in this method is present again for each generation to create a multi-generational pool but with the top 5% of each generation entering the expert pool. Similarities are again determined between the experts and the most common solution is used as the current guess. When experts do not agree on a choice, a random element is generated and added to the current solution. This method, despite the mutations and diversity, can get stuck in local minima so after many generations pass without a proposed solution, the algorithm will stop. For Hughes' experiments, the GA + WoAC performed only marginally better than the GA alone.

Several methods in the reviewed literature used genetic algorithms without adjustments or modifications like WoAC. One such approach is detailed in Sengoku and Yoshihara [9] where the problem at hand was the Traveling Salesman Problem. This problem is similar to Mastermind in that it is dynamic but it is not so naive as Mastermind. In [9], the mutation method defined is perhaps the most interesting part of the approach. "Natural selection" eliminates a percentage of the population each generation and a varying percentage of the population reproduces each generation. A tactic reminicient of relaxation is used on any candidates so that they are made to be as optimal as possible. A subprocess in the proposed algorithm is called Greedy Subtour Crossover and Sengoku and Yoshihara claim that their greedy approach performs better than simulated annealing. Berghman et al. in [1] review some of the literature available in the field and, in their own proposed algorithm, suggest considering candidate solutions that are not only very fit by a fitness standard but that are also promising in predictive power. In a similar way to Port and Yampolskiy [8], Berghman et al. [1] advocate for the usefulness of shrinking the solutions space at every opportunity. Certianly as the solutions spaces get larger and larger, any method by which they can be shrinked will be called upon. In Merelo-Guervos et al. [5] it was explicitly discovered that a smaller generation size worked more efficiently, at least with the proposed algorithm. This approach also expresses the use of hypermutation upon reaching a local minimum where the entire population is nearly started from scratch. Finally, Merelo [6] emphasize a strategically optimal approach as was encouraged in Berghman et al. [1] and Port and Yampolskiy [8]. In this same report, smaller generation sizes were again discovered to be more effective than large ones although a large number of relative generations (500) were run in their experiments.

4 Approach

This approach to find a solution to an instance of the game Mastermind is to use a genetic algorithm with a Wisdom of the Artificial Crowd modification. Specifically, this approach will have 3 steps. The genetic algorithm with Wisdom of the Artificial Crowd modification is described in detail below the holistic approach. Some terminology associated with genetic algorithms include crossover operator, mutation operator, and fitness measure; each of these terms are described below. The agent described below is the codebreaker.

4.1 Software

The code used has been adapted from existing code that did not originally use a genetic algorithm to solve the game. Mastermind. The algorithms have been modified to fit the new functions including reproduce, fitness, and expert choice functions. The structure of the prexisting code exists in some form but has been heavily shifted to fit the new flow of the new algorithm. One file still encapsulates all of the code and it is about the same number of lines as it was before adjustments. C++ was the language of choice for this project due to its combination of speed, structure, and easy functionality.

4.2 General Structure

1. On the first guess, there are no clues so a random combination of colors and placements of pegs may be made within the confines of the rules, of course.

2. The codemaker responds with the number of colors that are correct and the number of placements that are correct. Based on these new conditions of fitness, 10 generations of 50 members of the genetic algorithm will pass and after the 10th generation, the Wisdom of the Crowd algorithm will choose the "best" proposed solution to be the current move.

3. The previous step will be repeated until the codemaker reveals that the codebreaker has successfully found the solution or has used up the alotted turns and has lost the game.

The specifics of the genetic algorithm and Wisdom of the Crowd algorithm that will be employed for this project were inspired by namely by [1]. The crossover operator used in this approach is one where pseudorandom values dictate whether parent 1 or parent 2 will be the donator for each particular spot. The, depending again on the pseudo random values, a spot from the chosen parent is pushed onto the back of the new child until the new child is a whole new candidate. The mutation operator in this method is one where the frequency of mutation needs to be defined manually. This could also be set to a pseudorandom value. The decided frequency of mutation will ensure that that percentage of children produced will be mutated at a random spot to a random appropriate value.

The fitness measure for this method is as follows. The current candidate is searched for the number of colors in common indicated by each previous guess' reponse from the codemaker. A response from the codemaker consists of "W"s and "B"s. Each "W" indicates a color that the current code shares in common with the answer code and so the number of colors that each previous guess shares with the current guess must be equal to the number of "W"s in that guess' response code. If a code shares the correct number of colors with a previous guess, one point it added to its fitness score. Only if the current code shares colors with a previous guess will it take part in the second phase of evaluation. A current code in the second phase of the fitness function will be judged on whether it has the proper number of spots, determined by the number of "B"s in the previous guess' response code, in common with the previous guess. If the current code share the correct number of spots with a previous guess, the number of spots that it shares will be added to its fitness score. The reason that a single point is added for having the correct colors but several points may be added for sharing the correct spots is that having correct spots means that there are correct colors in correct spots. This is more useful information than simply having the right colors and should be more heavily weighted. This reduces the search space when the expert of experts is to be determined and therefore makes the algorithm more efficient.

4.3 Detailed Algorithm

1. Make a population of many 50 random guesses.
2. Reproduce for every pair in the guess pool. After two parents reproduce, one is removed from the candidate set. Probabilistically mutate with a rate of 30%. After each generation, keep the 7 individuals with the highest fitness scores in the expert set.
3. Execute the previous step 10 times total to produce 10 generations.
4. Determine the most common solution in the multigenerational expert pool. Compare each expert's colors with the others. Use the median-of-medians dynamic programming algorithm. The expert candidate with the highest similarity score is played as the current move.
5. Repeat steps 2 through 4 with an updated fitness standard based on the codemaker's response to the most recent move.
6. Once the proposed solution matches the codemaker's solution, the agent has won. If 10 moves have been made and the proposed solution does not match the codemaker's solution, the agent has lost.

Experiments and Results

This code will be tested on a large number of runs. Values are averaged over 10 runs in [2], suggesting that the authors of that paper found 10 runs to be sufficient. In order to test more thoroughly, I will test over 30 runs. In [1], Table 3 shows that when N is the number of colors and P is the number of pieces, N = 8 and P = 5 for N^P yielded a search space large enough for the genetic algorithm with the Wisdom of the Artificial Crowd modification to improve over just the genetic algorithm alone. I will use these numbers, 8 colors and 5 pieces, to create a solution space of sufficient size. This solution space along with the large number of tests will be able to sufficiently test the algorithm I produce.

In terms of analysis, each run will be associated with the number of moves it took to win that round. Besides just performance, I will measure runtime and memory usage against the initial program that I modified. It is possible that timing will need to be limited but that won't be apparent until testing of the finalized algorithm is underway. Between the 30 runs of the finalized algorithm and the 3 component result analysis (performance, runtime, memory usage), the experiments and analysis will be extensive.

This code was tested over 30 runs. All 30 runs were tested at the standard solution space size of $N = 6 and P = 4$ and 25 of the 30 runs had 10 generations per turn and 50 candidates per generation. Of the 30 runs, 10 were given a mutation frequency of 50% instead of 30%. Another 10 of the 30 runs were given 14 experts per generation instead of the 7 experts that the other tests were given. Finally, the 5 tests that did not have 10 generations and 50 candidates per generation had 5 generations instead and 500 candidates per generation.

4.4 First Group

The first 10 runs had a mutation frequency of 50% instead of 30%, 10 generations per turn, 50 candidates per generation, and 7 experts per generation. The runs took an average of 1.76 seconds and an average of 105k kB to complete. This test was complete in an average 7 turns, ranging between 3 and 9.

4.5 Second Group

The next 10 runs were unique in that there were double the number of experts collected per generation, 14 instead of 7, so that at the end of 10 generations there were 140 experts to determine the similarity of. There were a standard 30% mutation operator, 10 generations, and 50 candidates per generation. The difference in set size made a sigificant difference in time complexity, taking an average of 8.12 seconds per run. An average of 112kB of memory were used over the 10 runs. This test completed in an average of 6.5 turns, ranging between 4 and 8 turns to completion.

4.6 Third Group

There were 5 runs tested with 5 generations of 500 candidates per generation. All of the other variables were controlled in this test with 30% of children mutating and 7 experts per generation. The average run under these circumstances took 1.82 seconds and an average of 104k kB of memory. The solution was found between 3 and 7 guesses with an average of 7.

4.7 Fourth Group

The control group was run with 10 generations with a 50 candidate population, a mutation frequency of 30%, and 7 experts per generation. On average, these 5 runs took 1.47 seconds and used 101k kB of memory. The number of turns this group took to find a solution was an average of 6.5 ranging between 4 and 8 guesses.

5 Result Analysis

The first set of results to consider are the 10 runs where the mutation frequency is 50% instead of the usual 30%. With all of the other variables controlled, the mutation frequency must allow for the runs to take a longer time than group 4. This may have been because more swapping of elements had to be done in this group of tests than in the tests of group 4. The amount of memory consumed by this process on average is not surpring compared to the memory consumption of the other tests. Perhaps there was slightly more memory used again because of the more frequent movement of values in and out of candidates. Overall, the first

group of tests were not successful enough to be considered the best by any standards. At an average of 7 turns, it is tied with the third group for the highest number of average guesses.

The second group of tests where the number of experts per generation were doubled from the standard 7 to 14, the consumed memory was noticably higher than it was in any of the other tests. Even more noticable was the 8.12 seconds on average that the test took complete, more than 5 seconds more than any of the other groups of tests. This was likely due to the fact that the similarity scores needed to be calculated between many more experts than were available in any other group of tests. Determining the similarity between each of the experts is by far the most expensive process in the algorithm and should be considered for future work. This group did, however, meet the fourth group at 6.5 guesses on average. The time spent considering which experts were the most fit appears to have paid off to some degree in the improvement of average guesses over groups one and three.

In the third group, 5 generations each with 500 members were explored. The other controlled variables made it clear that the slightly longer runtime and the slightly larger amount of consumed memory than the control group were due to generations took longer and used more space. Despite the fact that fewer generations overall were completed for each turn, the tests overall took longer. Since 500 members had to be given a fitness score for each generation, it is indeed unexpected that these tests did not take more time. The larger generation sizes did not contribute in a positive way toward lowering the number of turns to a solution.

The control group of tests produced the best results in terms of time, memory, and performance. Since there was not an overload of work to do for each generation and not a large number of generations, the time and memory values were able to stay low. Certainly even this time and memory usage would be improved with a more efficient similarity function. Nevertheless, the control tests were able to solve each of the five games with an average of 6.5 guesses.

6 Conclusion

The state of the art algorithms for searching uninformed CSP optimization problems and minimizing the number of guesses is, according to the field's literature, GAs with a modification of WoAC. Armed with this knowledge, this report details an approach that uses a GA and WoAC where the fitness function is distinctly efficient. The most efficient versions of this algorithm ran with a small number of generations, a small generation size, a relatively small mutation frequency, and a small number of experts. Future work related to this method could be related to determining similarity between experts or related to conducting more extensive tests with different variables.

References

Berghman, Lotte and Goossens, Dries and Leus, Roel. "Efficient solutions for Mastermind using genetic algorithms". Computers & operations research. Elsevier. 2009.
Focardi, Riccardo and Luccio, Flaminia L. "Cracking bank pins by playing mastermind". International Conference on Fun with Algorithms. Springer. 2010. Hughes, Ryan and Yampolskiy, Roman V. "Solving Sudoku Puzzles with Wisdom of Artificial Crowds.". Int. J. Intell. Games & Simulation. 2012.
Knuth, Donald E. "The computer as master mind". 1976. Merelo-Guervos, JJ and Castillo, P and Rivas, VM. "Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary MasterMind". Elsevier. 2006.
Merelo, Juan J and Cotta, Carlos and Mora, Antonio. "Improving and scaling evolutionary approaches to the mastermind problem". European Conference on the Applications of Evolutionary Computation. Springer. 2011. Partynski, Dan. "Cluster-based Particle Swarm Algorithm for Solving the Mastermind Problem". Proceedings of the International Multi-Conference of Engineers and Computer Scientists. 2014.
Port, Aaron C and Yampolskiy, Roman V. "Using a GA and Wisdom of Artificial Crowds to solve solitaire battleship puzzles". 2012 17th International Conference on Computer Games (CGAMES). IEEE. 2012.
Sengoku, Hiroaki and Yoshihara, Ikuo. "A fast TSP solver using GA on JAVA". Third International Symposium on Artificial Life, and Robotics (AROB III'98). 283–288. 1998.