AM207 Final Project
Mastermind

Gioia Dominedò Amy Lee Kendrick Lo Reiner Maat

April 27, 2016

Abstract

1 Introduction

The game **Mastermind** was invented in 1970 by Mordecai Meirowitz, and is similar to an earlier game called Bulls and Cows. There are many variations of the game[1], but they all follow a broadly consistent format. The game involves two players: a code-maker and a code-breaker. The code-maker selects a sequence of length L (usually 4), where each element in the sequence is chosen with replacement from one of C colors (usually 6). At each turn, the code-breaker guesses a sequence and receives feedback in the form of black pegs and white pegs, where the black pegs denote the number of correct guesses in the right position, and the white pegs denote the number of correct guesses in the wrong position. Based on this information, the code-breaker refines her guesses and attempts to crack the code within the maximum number of guesses (usually XX).

For our project, we set out to implement different strategies and algorithms for optimizing the code-breaker's guesses. We then tested their performance based on TBC - performance metrics.

2 Methods

It is helpful at this stage to introduce some mathematical notation to describe the game, which we will use consistently when describing the various methods that we tested. For simplicity, we use one-indexing in the formulas below, despite the Python code being zero-indexed. We define:

- set of possible colors: $C = \{1, ..., C\}$
- set of positions in the code: $L = \{1, ..., L\}$
- hidden code: $H_i \; \forall i \in L$
- guess of the hidden code: $T_i \; \forall i \in L$
- indicator function $\mathbb{1}_{A=B}$, which equals 1 if A=B and equals 0 otherwise

[1] https://en.wikipedia.org/wiki/Mastermind_(board_game)#Variations

Using the above notation, we can denote the responses at each turn as follows:

- correct guesses in the right position: $B = \sum_{i=1}^{L} \mathbb{1}_{T_i=H_i} \; \forall i \in L$
- correct guesses in the wrong position: $W = \sum_{i=1}^{C} \min(\sum_{j=1}^{L} \mathbb{1}_{H_j=i,G_i}, \sum_{j=1}^{L} \mathbb{1}_{T_j=i,G_i}) - B$

2.1 Baseline: Knuth's Five-Guess Algorithm

TODO

2.2 Random Sampling From Posterior

This method is described in the Mastermind literature both as a constrained random search algorithm [1] and in terms of posterior distribution updates [8]. We use the latter approach below.

We start by defining the joint probability distribution over all possible code sequences as $P(H_1, ..., H_L)$. As we have no information, our prior is uniformly distributed.

$$P(H_1 = h_1, ..., H_L = h_l) = \frac{1}{C^L}, \quad \text{for all combinations of } (h_1, ..., h_l)$$

We can denote the evidence that obtain at each step as $e = (B, W)$, where B and W are defined as above, and use this to update the posterior joint distribution over code sequences as follows:

$$P(H_1 = h_1, ..., H_L = h_l | e) = \begin{cases} \frac{1}{|s(e)|}, & \text{if } (h_1, ..., h_l) \text{ is a possible code} \\ 0, & \text{otherwise} \end{cases}$$

where s(e) denotes the set of possible hidden codes, given the evidence, and $|s(e)|$ denotes the cardinality of this set.

We can define the posterior after multiple game steps analogously:

$$P(H_1 = h_1, ..., H_L = h_l | e_1, ..., e_n) = \begin{cases} \frac{1}{|s(e_1) \cap ... \cap s(e_n)|}, & \text{if } (h_1, ..., h_l) \text{ is a possible code} \\ 0, & \text{otherwise} \end{cases}$$

where $s(e_1) \cap ... \cap s(e_n)$ denotes the intersection of the sets of possible hidden codes, given the evidence at each step, and the entire denominator denotes the cardinality of this intersection.

We can use this framework to define the posterior updates at each round of the game, and then choose the next guess randomly from the updated distribution. We note that some of the Mastermind literature uses Shannon entropy to measure the uncertainty associated with codes after each posterior update. We chose not to use this measure, as it will be the same for all remaining valid codes at each update step and therefore does not provide any additional information or ranking.

2.3 Simulated Annealing

TODO

2.4 Genetic Algorithms

TODO

Other possibilities: local entropy? importance sampling? metropolis hastings?

3 Experiments

TODO

4 Conclusion

TODO

A Figures and Tables

TODO

References

[1] José Luis Bernier, C Ilia Herráiz, JJ Merelo, S Olmeda, and Alberto Prieto. Solving mastermind using gas and simulated annealing: a case of dynamic constraint optimization. In *Parallel Problem Solving from Nature?PPSN IV*, pages 553–563. Springer, 1996.

[2] Benjamin Doerr, Reto Spöhel, Henning Thomas, and Carola Winzen. Playing mastermind with many colors. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 695–704. SIAM, 2013.

[3] D.E. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 1976.

[4] Juan J Merelo-Guervós and Thomas Philip Runarsson. Finding better solutions to the mastermind puzzle using evolutionary algorithms. In *Applications of Evolutionary Computation*, pages 121–130. Springer, 2010.

[5] Juan Julián Merelo-Guervós, Pedro Castillo, Antonio M Mora García, and Anna I Esparcia-Alcázar. Improving evolutionary solutions to the game of mastermind using an entropy-based scoring method. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 829–836. ACM, 2013.

[6] Thomas Philip Runarsson and Juan J Merelo-Guervós. Adapting heuristic mastermind strategies to evolutionary algorithms. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 255–267. Springer, 2010.

[7] Angela Snyder. Mastermind by importance sampling and metropolis-hastings.

[8] Jırı Vomlel. Bayesian networks in mastermind. In *Proceedings of the 7th Czech-Japan Seminar*, 2004.