MasterMind game

Introduction

MasterMind, a deductive game in which a code master creates a code and a code breaker tries to break it using the feedback given by the code master. A typical mastermind game has four pegs(length) and six colours(numbers). A black hit indicates there is a peg with the right colour at the right position. A white hit indicates there is a peg with the right colour at the wrong position.

In this version of MasterMind, the number of pegs is replaced by the number of digits(length) in the code, and the number of colours is replaced by the number of numbers. The length and numbers are not limited to under 10. In fact, really big sized codes such as (length, num) = (100, 100), (300, 300) could be solved.

With a short code, what I aim for is solving the code in as few attempts as possible. Knuth's algorithm is used to ensure this. While with a long code, the use of Knuth's algorithm is not possible due to the sheer size of the whole set of code that is dealt with. A different approach is taken to solve the code within a reasonable time frame with as few attempts as possible.

Terminology

- An attempt is the code made by the code breaker
- A secret code is the code set by the code master.
- A potential answer is the answer that could be the secret code which gives the same feedback when tested against the attempt (or when the attempt is tested against it) as the feedback given when testing attempt against the secret code.
- A full set is a set that contains all possible code combination of certain length and num.
- A potential set is a set that contains all the potential answers.
- A code is a code within the full set.

Knuth's algorithm

Knuth's algorithm works by comparing the existing attempt against the secret code, then comparing against all potential answers and keeping only the potential answers that give the same feedback as the secret code gives. After each new attempt, the potential answer set size reduces as less answers in the full set satisfy the feedback required when tested against the attempt. So the answer can be found with few attempts.

The limit to Knuth's algorithm is that the computer needs to do comparison between the attempt and every code in the full set, then between the attempt and every code in the potential set, then the next potential set, and so on. So as can be seen, when a code set is of large length and num, the full set size is exponentially large and becomes not viable and worthwhile to compute. In fact, a full set of even (10, 10) codes is of size 10^{10}.

When making an attempt to reduce the size of the potential set, what should the attempt be? Although just a random attempt works quite well, the number of attempts can be further improved by using attempt finding algorithms such as Minimax and Most-parts algorithm.

Minimax

Minimax algorithm chooses an attempt that will at most keep the least potential answers. In the best-case scenario, the potential answer set will have only one potential answer which is the secret code, so we ignore that. In the worst-case scenario, the potential answer set will have some potential answers. To get the best worst-case, an attempt that has the smallest maximum potential answer set is chosen. The same applies for the following steps. There is also one caveat that the attempt is chosen out of the full set instead of just from the potential set. So the calculation is O(num^{length}*num^{length}*number of attempts). The calculations performed increase exponentially when the full set size increases and the full set size increase exponentially when length is increased. So minimax cannot be efficiently implemented for a big set.

My minimax is optimised by using map and set structure. A map is created to have different hit conditions as keys and the frequency they appear as values. So the minimax score can be accessed quickly and also the there is no need to loop through each unused code in the full set(attempts) with each hit condition against each potential answer. With this optimisation, each unused code in the full set is looped through the potential answer only once instead of 13 times with all 13 different hit conditions.

Most-parts

In a card guessing game with players 1 and 2. When picking a card from a deck of playing cards, assuming the questions asked by player 2 will always have a positive answer, it does not matter if player 2's question reduce the number of potential cards into a big partition x1 or a small partition x2. After the second query, the one card wanted will always be found with a probability of 1/52.

$$Px1 = \frac{x1}{52} * \frac{1}{x1} = \frac{1}{52}$$

$$Px2 = \frac{x2}{52} * \frac{1}{x2} = \frac{1}{52}$$

So to increase the probability of finding a card, the number of partitions should be increased. #S denotes size of set S. #A denotes a set of smaller size.

As seen below:

$$Psingle - step = \sum(n, i = 1)Pxi = \frac{x1}{\#S} * \frac{1}{x1} + \frac{x2}{\#S} * \frac{1}{x2} + \cdots + \frac{xn}{\#S} * \frac{1}{xn} = \frac{n}{\#S}$$

$$Pmulti - step = \sum(m, i = 1)Pyi * Pxi = \sum(m, i = 1)\frac{\#A}{\#S} * \frac{ni}{\#A}$$

By mathematical strong induction, the above reasoning holds true for a multistep game such as mastermind. In this case, each partition contains the potential answers with a specific hit condition when compared against the attempt(feedback). So to increase the probability of finding the final secret code. For each attempt, the algorithm should choose the attempt that will split the potential set into the biggest number of partitions. This algorithm has the same O(num^{length}*num^{length}*number of attempts) for its calculation, but better optimisation is possible for the first guess. As soon as the number of partitions(hit conditions) is equal to the max number of partitions, the guess can be chosen without testing against other potential answers and the other attempts. But when the number of partitions possible is not equal to the max number anymore, the whole potential set will have to be traversed and every attempt will have to be tested.

This algorithm guarantees a worst game of (4,6) within 6 steps, but the average number of steps it takes is less than that of minimax.

One by one algorithm

This algorithm uses more attempts for a set size of (4,6), (5,7) ant etc.. But for set sizes (10, 10) and above, it is much faster than Knuth's algorithm. Fundamentally, every digit is tested with every value. Once the number of black hit increases, the digit is correct, the algorithm goes to the next digit and test it with every value. To optimise further, before testing individual digits, the algorithm test all digits with the same value for every value to find out what values are needed and eliminate the values not needed. So when the algorithm runs through individual digits, the number of attempts is decreased. The calculation required is O(num*length).

Permutation algorithm

When a code has short length but large number of different values, the full set size is big. Knuth's algorithm doesn't work well and the one by one algorithm takes a bit too many attempts. This is a useful walkaround. First all digits are tested with the same value for every value to find out what values are included in the code. This requires "num" many attempts. Then the code can be broken by creating a full set that includes the permutation of all the values included in the secret code. After this, Knuth's algorithm is used to minimise the number of attempts. This reduce the full set's size from num^{length} in Knuth's algorithm to length factorial.

Bibliography

http://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf

https://www.researchgate.net/publication/30485793_Yet_another_Mastermind_strategy

		Numbers				
		4	5	6	7	8
	3	A:3.48, t=0.00203 *				
	4			Mini(A:4.632, t:0.474)	A:5.24 t:0.00348 *	
	5		A:4.65, t=0.00447 *	A:5 t:0.0105 *		
	6			A:5.64 t:0.0678 *		
	7					
length	8					A:15.37, t:0.129 *
	9					
	10					
	50					
	100					
	200					
	500		A:505.65, t=0.00111 *			

		Numbers					
	3	9	10	50	100	200	500
	4						
	5				A:104.54 t:0.000205 *		
	6				A:105.56 t:0.000508 *		
	7				A:106.77 t:0.00224 *		
length	8				A:107.74 t:0.0155 *		
	9				A:109 t:0.1333 *		
	10	A:17.34, t:1.18 *					
	50		A:21.34, t:11.552 *				
	100			A:707.03, t:0.011 *			
	200				A:2672.88, t:0.142 *		
	500					A:10240.6, t:2.05 *	
		* denotes 100 attempts					A:62323-66891 t:76.4-81.22

Most parts for 1296 attempts: 4.57, t: 0.494