DTD 5 ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Applied Soft Computing xxx (2005) xxx–xxx

Applied Soft Computing

www.elsevier.com/locate/asoc

Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary MasterMind

J.J. Merelo-Guervós [a,*], P. Castillo [a], V.M. Rivas [b]

[a] *GeNeura Team, Depto. Arquitectura y Tecnología de Computadores, Escuela Técnica Superior de Ingeniería Informática, Universidad de Granada, Granada, Spain*

[b] *Depto. Informdtica, Universidad de Jaen, Spain*

Received 7 July 2003; received in revised form 3 August 2004; accepted 10 September 2004

Abstract

In this paper we present a new version of an evolutionary algorithm that finds the hidden combination in the game of MasterMind by using hints on how close is a combination played to it. The evolutionary algorithm finds the hidden combination in an optimal number of guesses, is efficient in terms of memory and CPU, and examines only a minimal part of the search space. The algorithm is fast, and indeed previous versions can be played in real time on the world wide web. This new version of the algorithm is presented and compared with theoretical bounds and other algorithms. We also examine how the algorithm scales with search space size, and its performance for different values of the EA parameters.
© 2005 Published by Elsevier B.V.

Keywords: Evolutionary algorithm; MasterMind

1. Introduction: oracle problems and the MasterMind game

MasterMind is a commercial version[1] of an old game called ‘‘bulls and cows’’, in which two players, the *codemaker* and the *codebreaker* match their wits. The *codemaker* lays out a secret combination of colored pegs, which is hidden from the sight of the codebreaker. The codebreaker, drawing also pegs from a set, makes a guess at what the hidden combination is, and is answered by the codemaker with a black pin (which was traditionally called ‘‘bull’’) for each colored peg that is in the correct position and with the right color, and with a white pin (‘‘cow’’) for each peg with the correct color, but in a different position.[2]

* Corresponding author. Tel.: +34 958 243162; fax: +34 958 248993.
E-mail addresses: ji@merelo.net, tutti@geneura.ugr.es (J.J. Merelo-Guervós), tutti@geneura.ugr.es (P. Castillo).

[1] MasterMind was originally commercialized by Augusta Plastics, and its trade-mark is currently owned by Pressman Toy Corporation.

[2] Actually, the old British game ‘‘bulls and cows’’ was somewhat different from MasterMind. It was played on paper, not on a board, all the secret digits had to be different, and digits from 0 to 9 were allowed [1].

1568-4946/$ – see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.asoc.2004.09.003

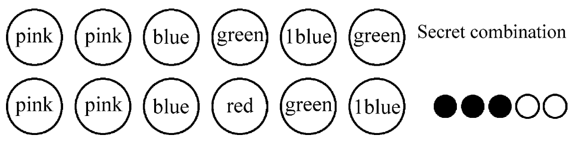


Fig. 1. In the top row, a length-6 *secret combination* using six colors (classical MasterMind uses length-4 combinations). On the bottom row, an example of a guess for that same combination, and the answer given by the codemaker: three black pins for the two pink pegs and the red one which are correctly placed, and the white pins for the green and light blue which are misplaced.

These black and white pins are *hints* that constrain the space of possible solutions and lead the codebreaker to the correct one. An example of a secret combination, and another that matches it partially is shown in Fig. 1.

The codebreaker is allowed to make a maximum number of guesses (15, for instance, which corresponds to the physical size of the board), after which the codemaker has won; if the codebreaker can find the combination before that maximal number, he scores as many points as guesses. In a tournament, the winner is the one that has accumulated the least guesses or points.

The generalized MasterMind problem can be formulated as a search for a hidden code which uses hints provided by a black box or oracle. Secret code and guesses are length L strings extracted from an alphabet with cardinality N; thus, search space size is N^L. The commercial board game [2] has two variants: "classic", with $N = 6$, $L = 4$ and "super MasterMind" with $N = 8$, $L = 5$. In the first case, search space size is 1296 and in the second 32 768.

Each "hint" h in the shape of black and white pins indicates how different the played combination c_{played} is from the secret code, c_{secret}. Thus,

$$h(c_{\text{secret}}, c_{\text{played}}) = (n_{\text{b}}, n_{\text{w}}) \tag{1}$$

that is, a number of black (n_{b}) and white (n_{w}) pins. The optimal hint would be

$$h(c_{\text{secret}}, c_{\text{played}}) = (n_{\text{b}}, n_{\text{w}}) = (L, 0) \tag{2}$$

that is, as many black pins as the length of the combination, and 0 white pins. Since the search space is multi-dimensional, it is not straightforward to compute the subspace that has been ruled out by the hint; usually, the simplest way of specifying it is usually via enumeration.

Finding the hidden combination over the N, L search space is thus a combinatorial optimization problem, which bears resemblance with others:

- *Circuit testing* [3]: Each manufactured circuit includes a module for self-test; the problem of generating patterns in such a way that it covers most of the faults with a minimal amount of patterns is similar to an *inverted* MasterMind, that is, finding the set of combinations that would be able to find a given secret code. Indeed, this problem has already been approached using GAs in [4].
- *Differential cryptanalysis* [5] is similar to MasterMind: combinations of letters in an alphabet is submitted to a "black box" encrypting device, and the encrypted output is analyzed to crack the key; once again, the problem is to compute a set of combinations that allow to extract maximal information from the "black box". In fact, some of the posts that deal with MasterMind in the sci.math Usenet forum are also cross-posted to sci.crypt, the forum about cryptography [6].
- *Additive search problem* is like a MasterMind where the oracle tells you which pegs are in the correct position. This problem has been solved using genetic algorithms in [7,8]. In that paper, the similarity to the "royal road" function [9] of both problems is also mentioned.
- In [10], MasterMind is mentioned as a variant of the *on-line model with equivalent queries*; in this model, when an incorrect hypothesis is proposed, the oracle responds with a counterexample to the hypothesis. Then, as in the case of MasterMind, the algorithm proceeds to form the next hypothesis (or combination) using the information obtained from the incorrect queries.
- The strategy used in MasterMind has been also leveraged for finding optimal queries in Master of Orion (multi user online games) [11], which have a much larger search space.

Any conclusion obtained for MasterMind may then be applied, although probably not directly, to any of these problems, as well as any other combinatorial optimization problems, such as job shop scheduling [12]. Asking about a solution to the game of MasterMind is so popular that it has become a FAQ (frequently asked question) for the sci.math Usenet forum, since at least 1994 [13].[3]

In fact, since the first version of the algorithm has been working on the Internet, back in 1995, it has received tens of thousands of visits, been mentioned on AI articles and textbooks [8,15,16] and the genetic algorithms FAQ [17]. It has been also quoted as an example of board games using artificial intelligence in the AI games web site [18]; solving MasterMind using GAs (and other AI techniques, such as simulated annealing) is also a popular assignment in basic-level artificial intelligence courses. Several graduate students from all the continents have written to the authors requesting information on MasterMind to solve it for their master thesis. The web site itself (the original one, http://geneura.ugr.es/mastermind, and the improved at http://geneura.ugr.es/∼jmerelo/GenMM is a popular demo of genetic algorithms at work. But the algorithm that runs those sites leaves some room for improvement, and that is what we present in this paper.

The rest of the paper is organized as follows: Section 2 presents the state of the art in solving MasterMind by algorithmic means; then the evolutionary algorithm used is presented in Section 3, and the results obtained with this new algorithm, comparing it with the old one and others in Section 4. Finally we discuss results and propose new lines of research in Section 5.

2. State of the art

Before presenting the state of the art, a few definitions are needed. We will use the tern *hint h* for the answer of the oracle to a played combination, c_{played}. A hint together with a combination (c_{played}, h) will be called a rule r. A combination c is *consistent* with or *meets a rule r* iff

$$h(c_{\text{played}}, c_{\text{secret}}) = h(c, c_{\text{played}}) \tag{3}$$

that is, if the combination has as many black and white pins with respect to the played combination as the played combination with respect to the secret combination. A combination c is consistent iff

$$\forall\, r_i = (c_i, h_i), \quad i = 1, \ldots, n, \;\; h_i = h(c, c_i) \tag{4}$$

where n is the number of combinations played so far, that is, c is consistent with all rules. A combination that is consistent can possibly be a solution, since it meets all constraints (in the shape of hints) made so far in the game. The concept of consistent combination will be important for characterizing different approaches to the game of MasterMind.

After the first paper presenting a solution to the game of MasterMind by Knuth [1] was published in 1977, there have been many others. They usually fall into one of the following categories:

- *Stepwise optimal*. Each new guess is made so that it is consistent as defined above. In this case, there are no bounds for the number of guesses that will be needed, other than it will be finite: since each guess and its answer rule out a set of combinations that is bigger than or equal to 1,[4] and each new combination must be extracted from that set; the number of possible combinations is reduced each step, and then, finally, only one is possible. A *suboptimal* approach will play a consistent combination most of the times, but, in some cases, will play another combination that is known not to be optimal.
- *Strategically optimal*. In this case, combinations are selected with the intent of extracting information about the shape of the hidden combination, or further reduce the size of remaining search space. A game of this kind could go like this: the first combination would be composed of only one color; the second of the next possible color, and so on. Or else, play a combination with all possible colors in the first place, then shift it left, until a cycle has been

[3] This introduction tries, in part, to answer the question posed by an anonymous referee that reviewed our first paper on solving MasterMind using genetic algorithms [14]: “Why would anyone want to solve the game of MasterMind?”.

[4] For some rules, there are possibly several combinations that, when played, do not decrease the space of compatible combinations except by the combination that has been played. For instance, if 123 is the secret combination, and the codebreaker plays 231, being answered 3 white pins by the codemaker, there is at least another combination, 312, that will obtain the same result (three whites with respect to the played combination and the secret combination), and will shrink the resulting search space minimally by 1. However, the number of combinations that decreases the search space just by one is finite, although its size increases with the combination length and the number of colors.

completed. After a group of ‘‘non-optimal’’ combinations has been played, the correct solution is deduced analytically. The advantage of this approach is that it is guaranteed to find the solution in a fixed amount of guesses; the problem is that usually that number is higher than the optimal one, and thus, as has been mentioned when explaining the game in Section 1, player programs following this strategy would be worse in general. This kind of approach was used originally in Knuth's paper, and is often mentioned in Usenet posts, such as [19].

There are several possible stepwise optimal strategies, depending on how the next combination to play is chosen:

- *Exhaustive search* considers one by one all possible combinations, and rules out those that are not consistent. There are many ways of running over the search space, depending on where you start and how you follow, but one possible way is playing a random combination to start, and then consider the next number-wise; that is, if the played combination has been 1112, the next would be 1113, up to 1116, and then 1121, and so on. This can be further improved by using some heuristics to rule out combinations: for instance, if a combination got neither white nor blacks as a hint, combinations with those colors will immediately be ruled out. The great advantage of this approach is that it is guaranteed to find the solution in finite time, and it runs over the search space only once, which means its complexity is O(LN). On the other hand, all search space must be stored, which will take as many bytes as possible combinations, not a problem for the 4×6 game (4 pegs, 6 colors), which only uses 1296, but impossible for the generalized game; for instance, a 10×8 game would need exactly 1 terabyte (1024 gigabytes) of storage. This approach has been taken, for instance, by Koyama and Lai [20].
- *Random search* randomly generates combinations, and plays the first that is found consistent. Once again, some combinations can be ruled out before being checked, but that would produce only a marginal improvement. The expected value of the number of combinations examined is half the total size of the search space multiplied by the number of combinations played (since no memory if combinations already evaluated is used). On the other hand, memory used is almost negligible, since only one combination and the set of played guesses need to be in memory at once. This approach has been taken, for instance, by [21], who built a calculator using a micro-controller to play MasterMind.

Knuth [1] used a strategically-optimal approach. Each guess was made so that it minimized the maximum number of remaining possibilities; if several guesses met this condition, and one of them was a ‘‘valid’’ pattern (what we call consistent in this paper) that was the one used. This allowed to solve the game in at most five guesses, with an expected number of guesses of 4.478. This result was further improved by Irving [22] and Neuwirth [23] to 4.364. None of this strategies have been proved to be optimal, so there was further room for improvement, and indeed, Koyama and Lai [20], using an exhaustive computer search, lowered that bound to 4.340 if five guesses were allowed at most, and 4.341 if six guesses were allowed. Chen and Amd Steven Homer [10] generalize these results to the general case of n colors and m positions, obtaining upper and lower bounds for the number of queries that should be made to find the hidden code; they also give an algorithm that uses binary search. Chen's paper improves previous bounds found by Chvatal [24]; which were also attacked by Roche [25], by using answers to early questions in the formulation of the later questions (or combinations played). Roche proves that the number of guesses needed is $O(L(\log(\log L)))$, where L is the length of the combination; however, it does not give numerical values or a strategy to achieve that bound.

Other approaches of the same kind have also been tried: Bestavros and Belal [26] use information theory to solve the game: each guess is made in such a way that the answer maximizes information on the hidden code, be it on the average or in the worst case. With that algorithm, they find the hidden code in 3.9 ± 0.5 or 3.8 ± 0.6 average combinations in the standard game (4 positions, 6 colors). This result is consistent with Knuth's, but it is a clear improvement over it. Variants of the MasterMind game have also been researched by Flood [27] and Ko and Teng [28]. Other approaches have also been reviewed in one of our previous papers [29].

In fact, the game of MasterMind has been approached several times using evolutionary algorithms. The authors used genetic algorithms and simulated annealing (SA) [14,29] to solve the generalized MasterMind problem. The first evolutionary algorithm and the SA algorithm in the second paper followed non-optimal strategies: non-consistent guesses were made if no consistent combination was found in a fixed time; the GA in the second paper played only consistent guesses and achieved a higher score (less average guesses), at the cost of a higher average time to find the solution. In both cases, the fitness of each combination in the GA were computed directly from the number of rules that the combination met, with more importance given to combinations with more black pegs. A suboptimal approach was also used in a paper by Bento et al. [30]: each generation, the best combination was played. Each combination had a fitness computed as the difference between the number of black and white pegs the last played combination was awarded and the number of black and white pegs of the combination with respect to this last played combination. Since this algorithm does not play optimally, it obtains an average of only 6.866 (no standard deviation is quoted) for the standard game.

Recently, another paper has been published by Kalisker and Camens [31], that combines a simple heuristic with an evolutionary algorithm; the approach is very similar to the one presented in this paper (and in other papers by us), but the fitness function is different: it takes into account the black and white pegs, but it also gives a score of 0 if the combination has been previously played. They quote good results, time-wise, up to 10 colors and length-7 combinations; our results will be compared against them in Section 3. In general it has been proved than combining heuristic methods with evolutionary algorithms, the way it is done by Chang et al. [32] for film scheduling problems, yields better results than any of them separately.

In this paper, we try to solve the general MasterMind case using genetic algorithms, by playing a stepwise optimal strategy, and by using a fitness function which is different from that used in previous papers [14,29] and similar to the one used by Bento et al. [30].

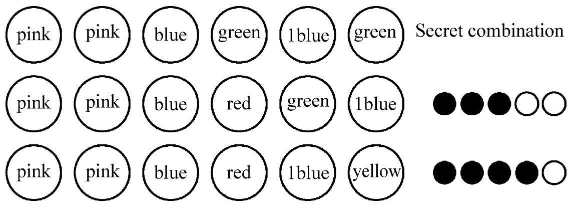


Fig. 2. Example of how combinations can be scored. The top row shows the hidden combination, the middle row one of the combinations the codebreaker has played and the answers. The bottom combination has to be scored against the middle row. In principle, it would not be "consistent" with the combination played: while this one matches three places and two colors with the hidden combination, the combination to score coincides in four places and one color with it, so it would not meet that rule. However, only two pegs would have to be changed to meet the rule (the rule that it should have, at most, three pegs in common and two with a common color with the played combination) for instance, swap the red for the yellow peg, so that would the "distance" to a correct combination; the score would then be -2.

3. Algorithm

The algorithm used to solve MasterMind is basically a stepwise optimal algorithm, which tries to minimize the distance to a consistent combination, and plays when one is found. Each combination is scored by computing how close it is to meeting all hints [30]. For instance, in the situation shown in Fig. 2, the combination in the bottom row would have a score of -2, because that is the distance that separates it from being consistent with the guess made in the middle row. A consistent combination would have 0 distance to all combinations played.

In general, if $h_i = n_{bi}$, n_{wi} and c_i form rule r_i, and

$$d(c, c_i) = d(h(c, c_i), h_i) = \text{abs}(n_b - n_{bi}) + \text{abs}(n_w - n_{wi}) \quad (5)$$

then fitness f for combination c will be

$$f(c) = \sum_{i=1,\dots,n} -d(c, c_i) \quad (6)$$

it is negative if the rule is not consistent, and becomes 0 for a consistent rule.

This genetic MasterMind evolves combinations directly, without representing them in any way; each combination is a string of valid colors, and all genetic operators give way to other valid combinations. Actually, genetic MasterMind uses EO (evolving | evolutionary objects) [33,34], which can evolve any object, provided you can assign it a fitness and change and/or combine with others of the same class.

Several operators that change diversity are used: *transpose*, which applies a permutation of the combination with priority 2; *circular mutation*, which substitutes one of the colors by the next, circling back to the first one if it is the last color, with priority 1; and *crossover*, which interchanges the central segment of two combinations, applied with priority 2. These priorities are normalized to one, which means that, every new generation, transposition is applied to 20% of the new combinations, and crossover and mutation to 40% each. Optionally, a *zapping* operator which substitutes a color for a different and random one could also be applied; however, it was not used in this case because the circular mutation searches in a more systematic way, and results showed no improvement over using only the two previous operators.

A rank and elite based selection scheme is used. Each generation, the 50% worst combinations are eliminated and substituted by the offspring resulting from the application of the genetic operators to the remaining members of the population. The score of each combination as shown in Section 1 was used as fitness; the basic idea was to score each combination by how far they were from meeting all rules. As will be shown in the results, this makes the fitness landscape smoother, as compared to one created by the fitness function used before in [29] (the number of rules met); and thus easier to search over. On the other hand, this fitness function actually makes the landscape dynamic: as soon as one combination is played, its fitness becomes negative, it will have ‘‘all blacks’’ with respect to itself, and thus its score will be $-L + n_b - n_w$. On the other hand, this needs not be bad by itself: it happens to all elements in the population, increasing diversity, and thus a reinitialization of the population is seldom necessary. A crowding scheme [35] was also used: each new combination substituted the most similar to it within the pool of combinations.

The default population was 400, which is also an improvement over our previous work, where 500 and even 10 000 individuals were used. The new fitness function proposed in this paper made possible to have many different fitness values (as opposed to just a few in [14,29], corresponding to the number of rules met), so that the population was more diverse in fitness space. Besides, if a certain number of generations, set to 15, passed without finding the a consistent combination, the population was all killed and new individuals were generated; this step is called *hypermutation*, and is evidently a drastic measure, but it was a valid way of increasing diversity; besides, it was rarely used in actual runs. The evolution of maximum and minimum fitness, together with average and online fitness are shown in Fig. 3.

In general, the main problem of this algorithm is still to maintain diversity: if many generations pass without finding a consistent combination, the population pool stagnates, and is likely to need a reset. The problem was solved in part with the introduction of the new fitness function, and might be further attacked by increasing the mutation rate; however, this would have made the algorithm more akin to random search. A hypermutation step only when needed was used as the lesser evil.

Two kinds of initial combinations were tested: the first contained all possible colors (that is, colors from 1 to L, N being the number of colors), or half the available colors (as proposed by Knuth [1]). The justification for the first combination was intuitive: the answer would give, at least, the number of colors present in the secret code; the justification for the second is theoretical: it will give, at least, as much information as the former. Results will be shown in the next section.

4. Results

To check if this algorithm was really an improvement over the old one, we compared it with computed bounds previous approaches (including our own [29], which was also stepwise optimal). There cannot be a lot of difference among optimal approaches: all will play more or less in the same number of average guesses, but the number of evaluated combinations and the standard deviation for both will yield the best approach. Several runs were made for each N, L combination in some cases, up to 1000, in other cases, only 100.

The first test would be a reality check, that is, how the algorithm scores in the standard $N = 6$, $L = 4$ and $N = 8$, $L = 5$ (also called super MasterMind) case. Results are shown in Tables 1 and 2.

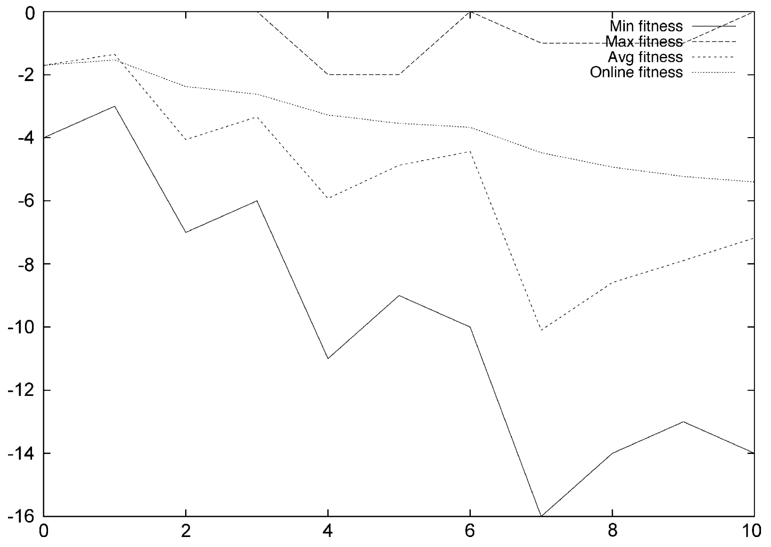


Fig. 3. Evolution of fitness for a $N = 8$, $L = 5$ run. y-Axis represents fitness, x-axis the number of generations. Maximum (dashed line) and minimum (continuous line) fitness are plotted along with average fitness and online fitness (average fitness for all generated combinations so far; represented with dotted line). When the maximum fitness reaches 0, it means that a consistent combination has been found and thus it is played; that happens at generations 0, 1, 2, 3, 6 and 10. After each guess, all quantities go down, since fitness is reevaluated for each combination, and there will be another ‘‘rule’’ to take into account. Minimum fitness also goes down, and in fact the possible span of fitness, and that accounts for the decrement of average and online fitness. However, between guesses (for instance, between generation 4 and 6, and 7 and 10) online and average fitness increase, which indicates the evolutionary algorithm is advancing towards a decision.

These results show that the genetic MasterMind algorithm presented in this paper obtains results that are somewhat better than previous bounds, or compatible with them (in the case of results published by Bestavros and Belal [26] and Rosu [36]), and it evaluates only a part of the search space to arrive at them. In comparisons with other genetic MasterMinds [30,31], GenMM outperforms it, due to the fact that they do not play an optimal strategy (but that is not clear in Kalisker and Camens' case).

Once it has been proved that GenMM performs correctly, some other tests were made to check the

Table 1
Results in the standard case, six colors and four positions

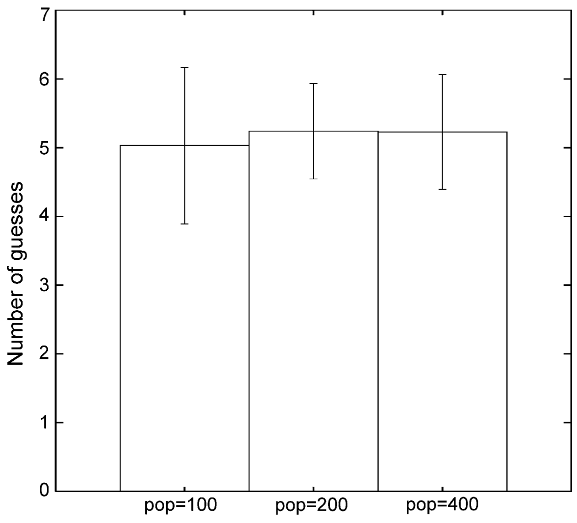
Author	Average guesses	Average combinations
Knuth [1]	4.478	All
Koyama and Lai [20]	4.340	All
Bestavros and Belal [26]	3.8 ± 0.5	All
Rosu [36]	4.66	Random search
Kalisker and Camens [31]	4.75	Unknown
GenMM (1)	4.132 ± 1.00	279 ± 188
GenMM (2)	4.312 ± 0.989	320 ± 268

Our results (labeled GenMM, for genetic MasterMind) are consistent with the best results found, and averages are lower than in Koyama's and Knuth's case. It must be taken into account that these strategies are not stepwise optimal. It is also consistent with Bestavros' result. Our algorithm obtained better results by playing as first guess 1122, as suggested by Knuth [1] (GenMM (1)); results when playing 1234 first were not as good, either from the point of view of number of guesses or the number of evaluated combinations. GenMM was run 1000 times, with a population of 100.

Table 2
Results in the ‘‘super’’ case, eight colors and five positions

Author	Average guesses	Average combinations
Rosu [36]	5.88	Random search
Kalisker and Camens [31]	6.39	Unknown
Bento et al. [30]	6.866	1029.9
GenMM	5.904 ± 0.975	2171 ± 1268

Our results (labeled GenMM, for genetic MasterMind) are compatible with the best results found (which are the ones quoted by Rosu, but no standard deviation is mentioned and it is thus difficult to compare). The number of evaluated combinations is also consistent with Bento's, but our algorithm achieves a solution in less moves. GenMM was run 500 times, with a population of 400.



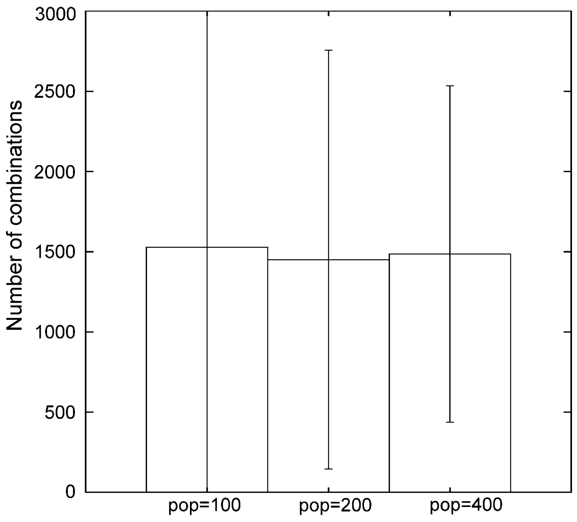


Fig. 4. The top graph shows the average and standard deviation for the number of guesses (top) and combinations evaluated (bottom) made for different population sizes. Averages are substantially similar independently of the population size, however, a bigger population means a lower standard deviation, which at the same time means that the range of possible times the algorithm will take is reduced; that is why, for problems of a certain size, a bigger population is preferred. In all cases, the problem evaluated was $N = 6$, $L = 6$.

influence of the population size on the quality of the solution. Results are shown in Fig. 4.

These results show that, since the main problem of this algorithm is to maintain diversity, bigger populations usually yield slightly better, or at least more robust, results: the number of combinations evaluated, and thus the total time taken to evaluate them, has a smaller standard deviation.

Finally, another test was made to check how the algorithm scaled with problem size; since GenMM is a general solution to the game of MasterMind, a desired feature would be that results do not degrade with problem size. To check that, 100 runs were made for a $L = 6$ and $N = 6, \ldots, 9$ problem. Results of these runs are shown in Fig. 5.

The most interesting result of this experiment is to find that, even as search space size grows exponentially, the number of combinations evaluated grows only linearly; the number of combinations has been fitted to a straight line $f(x) = ax + b$ with $a = 1504.22 \pm 189.1$ and $b = 1519.26 \pm 143.8$; the final sum of squares of residuals was 0.0386008.

5. Conclusions

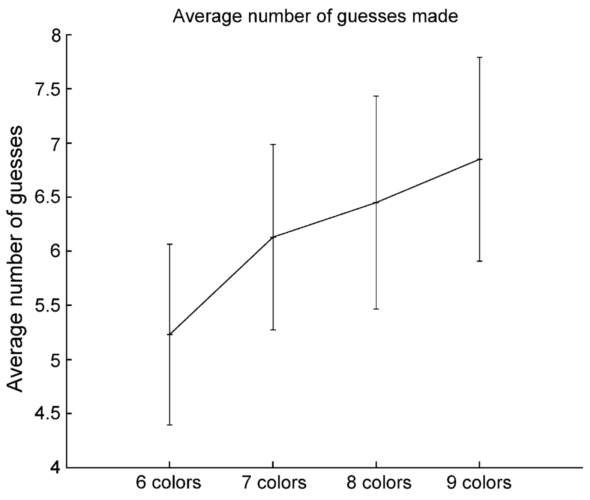
In this paper, we have proved that an evolutionary algorithm is an efficient way of solving oracle-type problems such as the generalized MasterMind game. The genetic algorithm performs as well or better than globally optimal and exhaustive search algorithms and scales well with size, since the number of combinations evaluated (and thus the time taken to reach the solution) grows linearly with it. The algorithm also outperforms other evolutionary algorithms like Bento et al.'s [30], Kalisker and Camens [31] and our own old version of the algorithm [29].

In fact, this algorithm has been working since May 1999 in the genetic MasterMind 2.0 site, at http://geneura.ugr.es/~jmerelo/GenMM. From this site, the full code for GenMM can be downloaded; in fact, it is included as an example for the EOlib (evolving | evolutionary objects) library.[5]

In the future, this algorithm is going to be updated with several improvements; in particular, we would like to evaluate every combination not only by taking into account its distance to consistent combinations, but also by its predictive power. Not all combinations are able to extract the same information from the codebreaker, and we will try to find a formula evaluate it.

The initial population and other genetic algorithm parameters will also have to be evaluated so that they

[5] Available from: http://eodev.sourceforge.net.



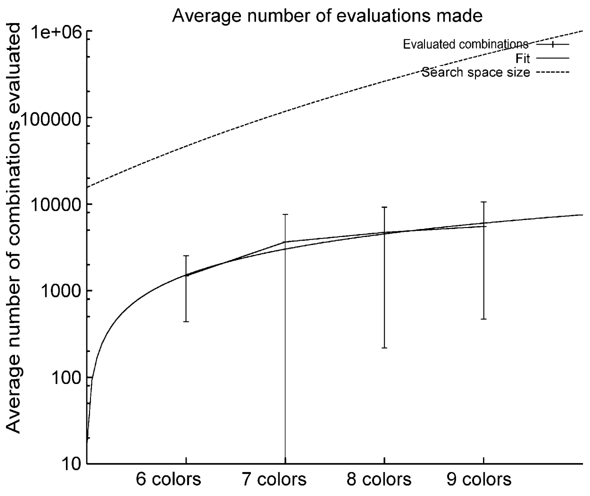


Fig. 5. The top graph shows the average number of guesses made for different number of colors, and the bottom graph the average number of combinations evaluated together with a fitting to a straight line and search space size in a semilogarithmic plot. In all cases, population was 400 and the combination length-6.

adjust themselves automatically to the search space size, eliminating the need for fine-tuning for each size. Possibly, a self-adaptive strategy will be the best in this case.

Finally, we will try to fine-tune the algorithm for speed so that it can be run, in real time (the way it is now) up to bigger combination sizes. An improvement could come from using fast evaluation strategies, such as the one proposed in [37].

Acknowledgements

This paper has been funded by project TIC2003-09481-C04-04, of the Spanish Ministry of Science and Technology. Source code is available from the authors. We are also very grateful to the anonymous reviewers of the first version of this paper, which made many helpful suggestions, and have no undoubtedly helped to improve it.

References

[1] D.E. Knuth, The computer as Master Mind, J. Recreational Math. 9 (1) (1976–1977) 1–6.

[2] Pressman Toys Ltd., Mastermind: the challenging game of logic and deduction for 2 players ages 8 to adult.

[3] E.M. Rudnick, J.H. Patel, G.S. Greenstein, T.M. Niermann, Sequential circuit test generation in a genetic algorithm framework, in: Proceedings of the Design Automation Conference, 1994, pp. 698–704. http://www.citeseer.nj.nec.com/rudnick94-sequential.html.

[4] J. Mira, F. Sandoval (Eds.), From Natural to Artificial Neural Computation, Lecture Notes in Computer Science, vol. 930, Springer-Verlag, 1995.

[5] E. Biham, A. Shamir, Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer (extended abstract), Lecture Notes in Computer Science, vol. 576, 1991 p. 156 in: http://www.citeseer.nj.nec.com/biham91differential.html.

[6] T.Y. Chow, Re: basic mastermind decryption, March 1995, Available from: http://groups.google.es/groups?selm=3kkmcc%24d7p%40senator-bedfellow.MIT.EDU&output=gplain.

[7] E.B. Baum, D. Boneh, C. Garrett, On genetic algorithms, in: Proceedings of the Eighth Annual Conference on Computational Learning Theory, ACM Press, 1995 , pp. 230–239(expanded version: "where genetic algorithms excel").

[8] E.B. Baum, D. Boneh, C. Garrett, Where genetic algorithms excel, Technical Report, NEC Research Institute, 4 Independence Way, Princeton, NJ, 1995. http://citeseer.nj.nec.com/baum95where.html.

[9] M. Mitchell, S. Forrest, J.H. Holland, The royal road for genetic algorithms: fitness landscapes and GA performance, in: F.J. Varela, P. Bourgine (Eds.), Proceedings of the First European Conference on Artificial Life: Toward a Practice of Autonomous Systems, MIT Press, Cambridge, MA, 1991, pp. 245–254.

[10] Z. Chen, C.C. Amd Steven Homer, Finding a hidden code by asking questions, in: Proceedings of the COCOON'96, Computing and Combinatorics, 1996, pp. 50–55.

[11] T. Tanaka, An optimal moo strategy, in: Proceedings of the Game Programming Workshop, Japan, 1996, , pp. 202–209 (in Japanese).

[12] I.T. Tanev, T. Uozumi, Y. Morotome, Hybrid evolutionary algorithm-based real-world flexible job shop scheduling

problem: application service provider approach, Appl. Soft Comput. 5 (1) (2004) 87–100.

[13] sci.math FAQ maintainer, Master mind, available as sci-math-faq/mastermind from the FAQ archives, such as http://www.faqs.org/faqs.

[14] J.J. Merelo, Genetic Mastermind, a case of dynamic constraint optimization, GeNeura Technical Report No. G-96-1, Universidad de Granada, 1996.

[15] F. Grant, Artificial life begins its own evolution, Scientific Computing World, Available from: http://www.raygirvan.co.uk/apoth/bl9809.htm.

[16] E. Turban, J.E. Aronson, Decision Support Systems and Intelligent Systems, Prentice-Hall, 1999(mentions "J.J. Merelo's mastermind" in Chapter 18, as an "exercise site").

[17] J. Heitkoter, D. Beasley, The hitch-hiker's guide to evolutionary computation (FAQ for comp.ai. genetic), Available from: http://surf.de.uu.net/encore/www/.

[18] "Classic" games and AI, Available from: http://www.gameai.com/clagames.html.

[19] S. Pope, Re: basic mastermind decryption, posting in the sci.math and sci.crypt newsgroups, Available from: http://www.math.niu.edu/ñusin/uses-math/games/mastermind/usenet.posts and from http://www.deja.com.

[20] K. Koyama, T.W. Lai, An optimal mastermind strategy, J. Recreational Math. 25 (4) (1993/1994) 251–256.

[21] J. Strobl, How to build a calculator for master minds, Available from: http://www.mystrobl.de/ws/pic/mm47/.

[22] R.W. Irving, Towards an optimum mastermind strategy, J. Recreational Math. 11 (2) (1978–1979) 81–87.

[23] E. Neuwirth, Some strategies for mastermind, Z. Oper. Res. Ser. B 26 (8) (1982) B257–B278.

[24] V. Chvatal, Mastermind, Combinatorica 3 (3–4) (1983) 325–329.

[25] J. Roche, The value of adaptive questions in generalized mastermind, in: Proceedings of the IEEE International Symposium on Information Theory, 1997, p. 135 in: http://ieeexplore.ieee.org/iel3/4845/13396/00613050.pdf.

[26] A. Bestavros, A. Belal, Mastermind, a game of diagnosis strategies, Bulletin of the Faculty of Engineering, Alexandria University, Available from: http://www.cs.bu.edu/fac/best/res/papers/alybull86.ps, http://citeseer.ist.psu.edu/bestavros86mastermind.html.

[27] M.M. Flood, Mastermind strategy, J. Recreational Math. 18 (3) (1985–1986) 194–202.

[28] K.-I. Ko, S.-C. Teng, On the number of queries necessary to identify a permutation, J. Algorithms 7 (4) (1986) 449–462.

[29] J.L. Bernier, C.-I. Herraiz, J.-J. Merelo-Guervos, S. Olmeda, A. Prieto, Solving mastermind using GAs and simulated annealing: a case of dynamic constraint optimization, in: Proceedings of the Parallel Problem Solving from Nature (PPSN) IV, Lecture Notes in Computer Science, vol. 1141, Springer-Verlag, 1996, pp. 554–563 (citeseer context).

[30] L. Bento, L. Pereira, A. Rosa, Mastermind by evolutionary algorithms, in: Proceedings of the SAC'99, 1999, pp. 307–311. in: http://laseeb.isr.ist.utl.pt/publications/papers/acrosa/sac99/sac99-ps.zip.

[31] T. Kalisker, D. Camens, Solving mastermind using genetic algorithms, in: E. Cantú-Paz, J.A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R.K. Standish, G. Kendall, S.W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K.A. Dowsland, N. Jonoska, J.F. Miller (Eds.), Genetic and Evolutionary Computation Conference, GECCO 2003, Lecture Notes in Computer Science, vol. 2724, Springer-Verlag, 2003, pp. 1590–1591, Available from Springer Link: http://link.springer.de/link/service/series/0558/papers/2724/27241590.pdf.

[32] P.-C. Chang, J.-C. Hsieh, Y.-W. Wang, Genetic algorithms applied in BOPP film scheduling problems: minimizing total absolute deviation and setup times, Appl. Soft Comput. 3 (2) (2003) 139–148.

[33] J.-J. Merelo-Guervos, Evolving objects, Available from: http://geneura.ugr.es/~jmerelo/EOpaper/.

[34] M. Keijzer, J.-J. Merelo-Guervos, G. Romero, M. Schoenauer, Evolving objects: a general purpose evolutionary computation library, in: P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution, Proceedings of the 5th International Conference on Evolution Artificielle, EA 2001, Le Creusot, France, October 29–31, 2001, Lecture Notes in Computer Science, vol. 2310, Springer, 2002, pp. 231–244, CiteSeer Context, Metapress URL.

[35] K. DeJong, An analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, University of Michigan, 1975.

[36] R.T. Rosu, Mastermind, Master's Thesis, NCSU, 1999, Available from: http://www.csc.ncsu.edu/degrees/undergrad/reports/rtrosu/heuristic.htm.

[37] M. Salami, T. Hendtlass, A fast evaluation strategy for evolutionary algorithms, Appl. Soft Comput. 2 (3) (2003) 156–173.