Entropy-Driven Evolutionary Approaches to the Mastermind Problem

Carlos Cotta[1], Juan J. Merelo[2], Antonio Mora[2], and Thomas Philip Runarsson[3]

[1] ETSI Informática, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain, ccottap@lcc.uma.es
[2] Dept. of Architecture and Computer Technology, ETSIIT, University of Granada, jj@merelo.net, amorag@geneura.ugr.es
[3] School of Engineering and Natural Sciences, University of Iceland, tpr@hi.is

Abstract. Mastermind is a well-known board game in which one player must discover a hidden color combination set up by an opponent, using the hints the latter provides (the number of places –or pegs– correctly guessed, and the number of colors rightly guessed but out of place in each move). This game has attracted much theoretical attention, since it constitutes a very interesting example of dynamically-constrained combinatorial problem, in which the set of feasible solutions changes with each combination played. We present an evolutionary approach to this problem whose main features are the seeded initialization of the population using feasible solutions discovered in the previous move, and the use of an entropy-based criterion to discern among feasible solutions. This criterion is aimed at maximizing the information that will be returned by the opponent upon playing a combination. Three variants of this approach, respectively based on the use of a single population and two cooperating or competing subpopulations are considered. It is shown that these variants achieve the playing level of previous state-of-the-art evolutionary approaches using much lower computational effort (as measured by the number of evaluations required).

1 Introduction

Mastermind is a board game that has enjoyed world-wide popularity since the 70s, when its current design was put forward (antecedents can be traced back to traditional puzzles such as *bulls and cows* or AB [1] though). Roughly speaking, Mastermind is a two-player code-breaking game, or in some sense a puzzle, since one of the players –the *codemaker* (CM)– has no other role in the game than providing a hidden combination, and automatically providing hints on how close the other player –the *codebreaker* (CB)– has come to guess this combination. More precisely, the functioning of the game is as follows:

– The CM sets a length ℓ combination of κ symbols. Therefore, the CB is faced with κ^ℓ candidates to be the hidden combination. This combination is typically represented by an array of pegs of different colors. We will use uppercase letters to denote these colors.

– The CB tries to guess this secret code by producing a combination with the same length and using the same set of symbols as the secret code. As a response to this move, the CM provides information on the number of symbols guessed in the right position (black pegs in the physical board game), and the number of symbols in an incorrect position (white pegs).
– The CB uses this information to produce a new combination, that is assessed in the same way. If he correctly guesses the hidden combination in at most N attempts, the CB wins. Otherwise, the CM takes the game.

There exist other variants of the game in which more information is provided (i.e., which –rather than how many– symbols are correctly guessed and/or are out of place), additional constraints are enforced on the hidden combination (e.g., not allowing repeated symbols in the secret code, as in *bulls and cows*, thus reducing the search space), or even allowing the CM to change the code during the game in a way that is compatible with previous moves (this variant is aimed to exploit any bias the CB may have when selecting the next combination played, such as preferring certain symbols over others).

As mentioned before, the game is asymmetrical in the sense that the CM has no freedom of move after setting up the hidden combination (at least in static variants of the game in which the hidden combination does not change), and therefore the CB does not have to outsmart the CM, but he has to put his own analytical skills at play to determine the course of action given the information available at each step. The resulting combinatorial problem is enormously interesting, as it relates to other generally called *oracle* problems such as circuit and program testing, differential cryptanalysis, uniquely identifying a person from queries to a genetic database [2] and other puzzle-like games and problems – check also [3]. It is also a complex problem, which has been shown to be NP-complete under different formulations [4, 5], for which several issues remain open, e.g., what is the lowest average number of guesses needed to solve the problem for any given κ and ℓ. Associated to this, there arises the issue of coming up with an efficient mechanism for finding these guesses in any particular case. To this end, several attempts have been made to use evolutionary algorithms (EAs) for exploring the space of feasible (meaning here compatible with the available information) combinations, e.g., [6–8].

Most evolutionary approaches presented for this problem are based on providing the CB with a set of potential combinations among which he has to select his next move. This decision-making process is very important, since although all potential candidates may be compatible with the information available, the outcome of the move can be very different, ranging from minimal reductions in the set of potential solutions, to a major pruning of the search space. Several metrics have been defined for this purpose. We consider here the use of an entropy-based criterion, which is further introduced in the fitness function to guide the search, and provide a variable-size, high-quality set of potential candidates. We show that this approach, combined with the seeded initialization of the population can result in a large reduction in the computational effort of the EA, with respect to other evolutionary approaches described in the literature.

2 Background

This section provides a brief overview of the problem, presents its classical formulation, and discusses how it has been tackled in the literature.

2.1 Formulation

As mentioned in Sect. 1, a Mastermind problem instance is characterized by two parameters, namely the number κ of colors and the number ℓ of pegs. Let $\mathbb{N}_\kappa = \{1, 2, \cdots \kappa\}$ be the set of symbols used to denote the colors. Subsequently, any combination, either the hidden one or one played by the CB, is a string $c \in \mathbb{N}_\kappa^\ell$. Whenever the CB plays a combination c_p, a *response* $h(c_p, c_h) \in \mathbb{N}^2$ is obtained from the CM, where c_h is the hidden combination. A response $\langle b, w \rangle$ indicates that the c_p matches c_h in b positions, and there exist other w symbols in c_p present in c_h but in different positions.

A central notion in the context of the game is that of consistency. A combination c is *consistent* with a played combination c_p if, and only if, $h(c, c_p) = h(c_p, c_h)$, i.e., if c has as many black and white pegs with respect to the c_p as c_p has with respect to the hidden combination. Intuitively, this captures the fact that c might be a potential candidate to be the hidden combination in light of the outcome of playing c_p. We can easily extend this notion and denote a combination c as consistent (or feasible) if, and only if, it is consistent with all combinations played so far, i.e.,

$$h(c, c_p^i) = h(c_p^i, c_h)\ 1 \leqslant i \leqslant n \tag{1}$$

where n is the number of combinations played so far, and c_p^i is the $i-$th combination played. Any consistent combination is a candidate solution. It is straightforward to see that the number of feasible solutions decreases with each guess made by the CB (provided he always plays feasible solutions). By the same token, the feasibility of a candidate solution is a transient property that can be irreversibly lost upon obtaining further information from the CM. This turns out to be a central feature in the strategies devised to play Mastermind, as shown next.

2.2 Game Strategy

Most naive approaches to Mastermind play a consistent combination as soon as one is found. An example of such an approach within an evolutionary context was proposed by Merelo *et al.* [3]. While this can constitute a simple and fast strategy early in the game (when there are many feasible combinations left), and is ensured to find eventually the hidden combination, it is a poorly performing strategy in terms on the number of attempts the CB needs to win the game. Indeed, unless some bias is introduced in the way solutions are searched, this strategy reduces to random approach, as solutions found (and played) are a random sample of the space of consistent guesses. It is also highly inefficient in the last stages of the game (when there are very few –maybe just one– feasible

solutions), since a large part of the feasible space has to be examined to find a feasible solution. We will return to this latter issue later on.

One of the reasons behind the poor performance of these naive strategies was anticipated in Sect. 2.1: the feasibility of a solution can vanish upon receiving feedback from the CM. Thus, a sensible strategy would try to take this into account when selecting the next move. More precisely, the CB would like to play a feasible combination that –were it not the hidden one– left him in the best position in the next move. This leads to a generic framework for defining Mastermind strategies in which (1) a procedure for finding a large set (even a complete set) Φ of feasible combinations is firstly used, and (2) a decision-making procedure to select which combination $c \in \Phi$ will be played is then used.

Regarding the decision-making procedure, its purpose must be to minimize the losses of the CB, putting him in the best position for the next move. This is substantiated in reducing the number of feasible solutions as much as possible, and hence that the set of available options in the next step is minimal. However, it is obvious that the reduction in the number of feasible solutions attainable by a certain guess depends on the hidden combination which is unknown. Therefore, any strategy based on this principle must rely on heuristics. To be precise, let us consider the concept of *partitions* (also called Hash Collision Groups, HCG [1]) defined as follows:

1. Let Φ be the set of feasible solutions available.
2. Let Ξ be a $|\Phi| \times (\ell+1) \times (\ell+1)$ all-zero matrix.
3. **for each** $\{c_i, c_j\} \subseteq \Phi$ **do**
   (a) $\langle b, w \rangle \leftarrow h(c_1, c_2)$.
   (b) $\Xi_{ibw} \leftarrow \Xi_{ibw} + 1$
   (c) $\Xi_{jbw} \leftarrow \Xi_{jbw} + 1$

Notice that each combination c_i has a partition matrix $\Xi_{i[\cdot,\cdot]}$, indicating how it relates to the rest of feasible solutions. Notice also that after playing combination c_i and obtaining response $\langle b, w \rangle$, entry Ξ_{ibw} indicates how many solutions in Φ remain feasible. Most approaches to Mastermind use the information in this matrix to select the next combination played, typically using some kind of worst-case or average-case reasoning. This idea was introduced by Knuth [9], whose algorithm tries to minimize the worst case by following the strategy of minimizing the worst expected set size, i.e., let $\eta(c_i, \Xi) = \max\{\Xi_{ibw} \mid b, w \leqslant \ell\}$, then $c_p = \min^{-1}\{\eta(c_i, \Xi) \mid c_i \in \Phi\}$ (ties are broken by lexicographical order). Using a complete minimax search Knuth shows that a maximum of 5 guesses (4.478 on average) are needed to solve the game using this strategy for $\kappa = 6$, $\ell = 4$.

2.3 Related work

The path leading to the most successful heuristic (non-evolutionary) strategies to date include minimization of the *worst case* [9] or *average case* [10], or maximization of *entropy* [11, 12] or *number of partitions* [13]. We defer to next section a more detailed description of the use of entropy, which has been the strategy chosen in this work.

EAs that try to solve this problem have also historically proceeded more or less in the same way. After using naive strategies that played the first combination found [6], using suboptimal strategies with the objective of avoiding the search to be stuck [7], or even playing the best guess each generation in a policy that resulted in a fast and very bad solution to the puzzle [14, 15]. However, it was not until recently when Berghman *et al.* [8] adopted the method of partitions to an EA. The strategy they apply is similar the *expected size* strategy.

The use of the information in Ξ can be considered as a form of look-ahead, which is computationally expensive and requires the availability of set Φ. Notice however that if no look-ahead is used to guide the search, any other way of ranking solutions (i.e., any mechanism that analyze solutions on an individual basis) might find solutions that were slightly better than random, but not more. In any case, it has been shown [16] that in order to get the benefit of using look-ahead methods, Φ need not be the full set of feasible solutions at a certain step: a fraction of around one sixth is enough to find solutions that are statistically indistinguishable from the best solutions found. This was statistically established, and then tested in an EA termed EvoRank [17], where the *most-partitions* strategy was used. Solutions were quite competitive, being significantly better than random search and also similar to the results obtained by Berghman *et al.*, but using an smaller set size and a computationally simpler strategy.

The strategy presented in this paper provides two fundamental steps beyond this previous work. On one hand, the use of an entropy-based fitness function and guided initialization reduces the computational effort required by the algorithm. On the other hand, it is shown that not fixing in advance the size of the set Φ (and thus letting the algorithm use limited computational resources to find a set as large as possible but without any lower bound on its size) does not penalize performance. This indicates the EA can be run on a fixed computational budget, rather than until completing a large enough set Φ.

3 Entropy-Driven Approaches

The general framework used in this work is depicted in Algorithm 1. An EA plays the role of the CB, selecting a combination to be played at each step on the basis of (i) past guesses and their outcomes, and (ii) the set of feasible solutions found in the previous step. In the first step no previous information is available, and all combinations are potentially the hidden one with equal probability. Hence, a fixed guess is made in this case.

The EA handles a population of candidate combinations. This population is initialized using the feasible solutions found in the previous step. More precisely, random sampling with replacement is used to select the initial member of the population from the set of known feasible solutions. Notice that this set is probably not exhaustive, since the EA is not guaranteed to have found all feasible solutions in the previous move. Furthermore, many of these feasible solutions will no longer be feasible in light of the outcome of the last move. In any case, using this set as a seed provides the EA with valuable information to focus the search

Algorithm 1: Outline of the evolutionary Mastermind approach

```
1  typedef Combination: vector[1..ℓ] of ℕ_κ;
2  procedure MASTERMIND (in: c_h: Combination, out: guesses, evals: ℕ);
3  var c: Combination;
4  var b, w, e: ℕ;
5  var P: List[⟨Combination, ℕ²⟩] ;                          // game history
6  var F: List[Combination] ;                          // known feasible solutions
7  evals ← 0; guesses ← 0; P ← [] ;                          // initialize game
8  repeat
9      guesses ← guesses + 1;
10     if guesses = 1 then                                          // initial guess
11         c ← INITIALGUESS(ℓ, κ);
12         F ← [];
13     else
14         RUNEA (↓ P, ↕ F, ↑ c, ↑ e) ;                              // run the EA
15         evals ← evals + e ;   // update cummulative number of evaluations
16     end if
17     ⟨b, w⟩ ← h(c, c_h) ;                          // current guess is evaluated
18     played.ADD(⟨c, ⟨b, w⟩⟩) ;                          // game history is updated
19 until b = ℓ;
```

towards the new feasible region of the search space, which will likely intersect with the known feasible set (and ideally would be a subset of the latter).

Solutions can be manipulated using standard reproductive operators for recombination and mutation. The fitness function is responsible for determining the feasibility of a given combination given the currently available information. Every feasible solution discovered during this run of the EA is kept in a secondary population for post-processing after the run finishes. This post-processing is aimed to select the single combination that will be played in the next move. The procedure used for this purpose is analogous to that used for fitness assignment, so let us firstly describe the latter.

As mentioned in Sect. 2.3, entropy is used as a quality indicator to optimize the status of the game after the guessed combination is played. To be precise, the fitness function firstly scans the population to divide the population in two groups: feasible solutions and infeasible ones. Infeasible solutions are assigned a fitness value (to be maximized) that indicates its closeness to feasibility. This is done by computing the Manhattan distance between the score $h(c, c_p^i)$ the solution obtains against the i-th combination played, and the score $h(c_p^i, c_h)$ the latter obtained against the hidden combination. This distance is summed over all previously played combinations, normalized by dividing by the maximum possible distance, and deducted from 1.0. This way, the closer (from below) the fitness is to 1.0, the closer to feasibility the solution is.

Feasible solutions have thus a fitness equal to 1.0, and receive a bonus based on entropy. To this end, the partition matrix Ξ is computed on the basis of the

feasible solutions contained in the current population. Subsequently, the entropy $H_i(\Xi)$ of each combination is computed as:

$$H_i(\Xi) = -\sum_{0 \leqslant b,w \leqslant \ell} p_i(b,w|\Xi) \log p_i(b,w|\Xi) \tag{2}$$

where $p_i(b,w|\Xi) = \Xi_{ibw} / \sum_{0 \leqslant b',w' \leqslant \ell} \Xi_{ib'w'}$. The underlying the idea of using this entropy measure is to reward feasible solutions tending to produce a uniform partition of the feasible search space, hence minimizing the worst-case scenario. This procedure can be also regarded as a way of maximizing the average information returned by the CM once the combination is played. The actual selection of this combination is done after the EA terminates, using the procedure described above on the set of feasible solutions discovered in the run.

Three variants of the EA described above have been considered, each one using a different population management strategy. The first one is an EA that follows the procedure above on a single population. We will denote this algorithm as EGA. Secondly, we have considered a version of the algorithm in which the population is divided in two equal-size tiers A and B; partitions (and subsequently entropy) are computed in a crossed way, i.e., determining how solutions in tier A compare to solutions in tier B and vice versa. The rationale is to preclude (or at least hinder) the appearance of endosymbiotic relationship (the population evolving to maximize entropy internally, but not globally). This co-evolving variant is denoted as EGACO. Finally, a third variant is considered analogously to EGACO, but having combinations in tier B trying to minimize (rather than maximize) the entropy of combinations in tier A. This can be regarded as competitive co-evolution [18], aimed to provide a constant thrust in the search of new feasible solutions. We denote this competitive variant as EGACM.

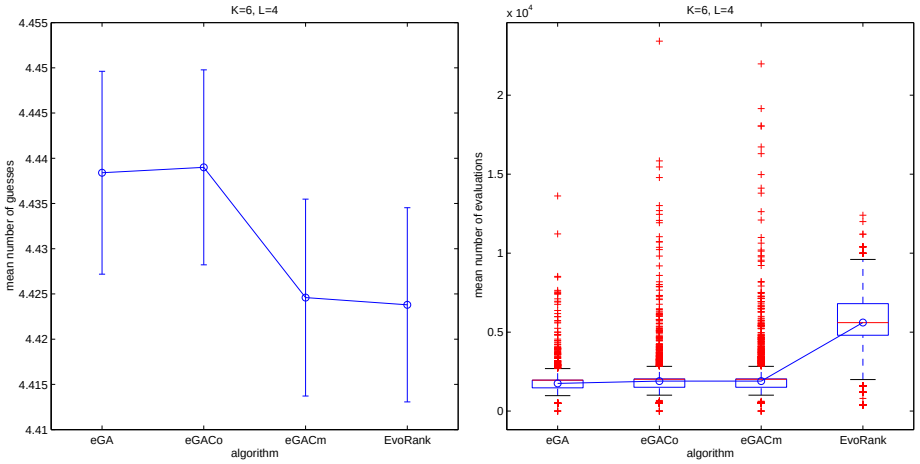
4 Experimental results

The evolutionary approaches considered use a population of 50 solutions (divided in two subpopulations of 25 solutions in the case of EGACO and EGACM), binary tournament selection, one-point crossover ($p_X = .9$), random-substitution mutation ($p_m = 1/\ell$), and an elitist generational replacement policy. The algorithms are run for a minimum number of 500 evaluations in each move. If no feasible solution is in the (sub)population(s) at this point, the algorithm keeps on running until one is found.

The experiments have been performed on two Mastermind problems, namely the classical version defined by $\ell = 4$ pegs and $\kappa = 6$ colors, and a harder version involving the same number of pegs but more colors ($\kappa = 8$). In both cases a problem-generator approach has been considered: 5,000 runs of each algorithm have been carried out, each one of a randomly generated instance (i.e., hidden combination). To maximize the breadth of the benchmark, instances have been generated so that any instance in the test set is used at most once more that any other existing instance.

Table 1. Comparison of the evolutionary approaches with random (R) and seeded (S) initialization. Underlined results are statistically significant.

		eGA		eGACo		eGACm	
		played	evals	played	evals	played	evals
$\kappa = 6$	R	4.489 ± .011	1872 ± 10	4.448 ± .011	2315 ± 24	4.425 ± .011	2252 ± 20
	S	<u>4.438</u> ± .011	<u>1792</u> ± 9	4.439 ± .011	<u>1977</u> ± 14	4.425 ± .011	<u>1982</u> ± 15
$\kappa = 8$	R	5.279 ± .013	2501 ± 20	5.207 ± .013	3456 ± 51	5.207 ± .013	3465 ± 50
	S	<u>5.240</u> ± .013	<u>2348</u> ± 19	5.222 ± .013	<u>2804</u> ± 38	5.207 ± .013	<u>2810</u> ± 37

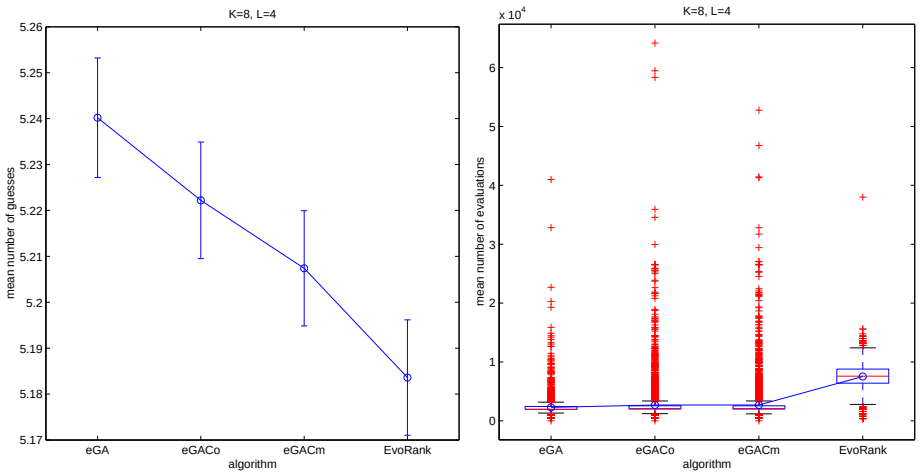


(a) (b)

Fig. 1. Results for $\kappa = 6$ colors and $\ell = 4$ pegs. (a) Mean number of guesses required. The error bars indicate the standard deviation of the mean. (b) Distribution of the number of evaluations required.

The first set of experiments is devoted to test the impact of the seeded initialization of the population. For this purpose, the algorithms are run both using random initial populations and seeded initialization. The results are shown in Table 1. As it can be seen, there is no remarkable difference in the mean number of guesses required, but the computational effort is clearly better (with statistical significance at 0.05 level using a Wilcoxon signed rank test).

Subsequently, the eGA* variants are compared with EvoRank, a state-of-the-art algorithm for this problem, on the same set of instances. The results of the comparison are shown in Figs. 1 and 2. Notice all algorithms perform very similarly in number of guesses for $\kappa = 6$ (statistically not significant results). EvoRank seems to be slightly better for $\kappa = 8$, but note that the difference is only statistically significant for eGA and eGACo, and there is no statistically significant difference between eGACm and EvoRank (p-value=.18). The situation is different in the case of the number of evaluations required: all eGA* variants are much computationally cheaper than EvoRank (with statistical significance).



(a) (b)

Fig. 2. Results for $\kappa = 8$ colors and $\ell = 4$ pegs. (a) Mean number of guesses required. The error bars indicate the standard deviation of the mean. (a) Distribution of the number of evaluations required.

Among EGA* variants, the computational cost of EGA* is significantly lower than that of EGACO and EGACM for $\kappa = 6$ and $\kappa = 8$.

5 Conclusions and Future Work

Mastermind is a prime example of dynamically constrained problem which offer an excellent playground for optimization techniques. We have presented a new evolutionary approach for this problem in which in addition to the hard constraint of consistency with previous guesses, solutions are also evaluated in terms of maximizing the information potentially returned by the CM. Seeded initialization of the population with feasible solutions for the previous move turns out to be an important ingredient for reducing the computational effort of the algorithm. As a result, it can perform at state-of-the-art level, at lower cost than other algorithms. This also indicates that entropy-guided construction of a variable-size feasible set Φ is competitive against other procedures that impose a lower bound on the size of this set. We believe this feature will be increasingly important for larger problem instances, where the search space is vaster and feasible candidate configurations can be sparsely distributed.

As an avenue for further research, the scalability of the approach will be tested on larger instances. It will be also interesting to test whether heuristics tricks, such as using *endgames*, that is, deterministic or exhaustive search methods when certain situations arise, would enhance the algorithms, and in which way (search effort and average number of combinations played).

Acknowledgements This work is supported by projects NEMESIS (TIN2008-05941), NoHNES (TIN2007-68083), P06-TIC-02025, and P07-TIC-03044.

References

1. Chen, S.T., Lin, S.S., Huang, L.T.: A two-phase optimization algorithm for mastermind. Computer Journal **50**(4) (2007) 435–443
2. Goodrich, M.: On the algorithmic complexity of the Mastermind game with black-peg results. Information Processing Letters **109**(13) (2009) 675–678
3. Merelo-Guervós, J.J., Castillo, P., Rivas, V.: Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary MasterMind. Applied Soft Computing **6**(2) (2006) 170–179
4. Stuckman, J., Zhang, G.Q.: Mastermind is NP-complete. INFOCOMP J. Comput. Sci **5** (2006) 25–28
5. Kendall, G., Parkes, A., Spoerer, K.: A survey of NP-complete puzzles. ICGA Journal **31**(1) (2008) 13–34
6. Merelo, J.J.: Genetic Mastermind, a case of dynamic constraint optimization. GeNeura Technical Report G-96-1, Universidad de Granada (1996)
7. Bernier, J.L., Herriz, C.I., Merelo-Guervós, J.J., Olmeda, S., Prieto, A.: Solving *mastermind* using GAs and simulated annealing: a case of dynamic constraint optimization. In Voigt, H.M., et al., eds.: Parallel Problem Solving from Nature IV. Volume 1141 of Lecture Notes in Computer Science., Springer-Verlag (1996) 553–563
8. Berghman, L., Goossens, D., Leus, R.: Efficient solutions for Mastermind using genetic algorithms. Computers and Operations Research **36**(6) (2009) 1880–1885
9. Knuth, D.E.: The computer as Master Mind. J. Recreational Mathematics **9**(1) (1976-77) 1–6
10. Irving, R.W.: Towards an optimum Mastermind strategy. Journal of Recreational Mathematics **11**(2) (1978-79) 81–87
11. Neuwirth, E.: Some strategies for Mastermind. Zeitschrift fur Operations Research. Serie B **26**(8) (1982) B257–B278
12. Bestavros, A., Belal, A.: Mastermind, a game of diagnosis strategies. Bulletin of the Faculty of Engineering, Alexandria University (December 1986)
13. Kooi, B.: Yet another Mastermind strategy. ICGA Journal **28**(1) (2005) 13–20
14. Bento, L., Pereira, L., Rosa, A.: Mastermind by evolutionary algorithms. In: Procs SAC 99. (1999) 307–311
15. Kalisker, T., Camens, D.: Solving Mastermind using genetic algorithms. In Cant-Paz, E., et al., eds.: Genetic and Evolutionary Computation Conference 2003. Volume 2724 of Lecture Notes in Computer Science., Springer Verlag (2003) 1590–1591
16. Runarsson, T.P., Merelo, J.J.: Adapting heuristic Mastermind strategies to evolutionary algorithms. In González, J., et al., eds.: Nature Inspired Cooperative Strategies for Optimization 2010. Volume 284 of Studies in Computational Intelligence., Granada, Spain, Springer-Verlag (2010) 255–267
17. Merelo, J.J., Runarsson, T.P.: Finding better solutions to the mastermind puzzle using evolutionary algorithms. In Di Chio, C., et al., eds.: Applications of Evolutionary Computing, Part I. Volume 6024 of Lecture Notes in Computer Science., Istanbul, Turkey (2010) 120–129
18. Angeline, P., Pollack, J.: Competitive environments evolve better solutions for complex tasks. In Forrest, S., ed.: Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann (1993) 264–270