A search for scalable evolutionary solutions to the game of MasterMind

Juan-J. Merelo, Antonio M. Mora
and Pedro A. Castillo
Departamento de Arquitectura y Tecnología de Computadores
University of Granada
Email: {jmerelo,amorag,pedro}@geneura.ugr.es

Carlos Cotta
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Email: ccottap@lcc.uma.es

Mario Valdez
Instituto Tecnológico de Tijuana
Tijuana, México
Email: mario@tectijuana.edu.mx

***Abstract*—MasterMind is a puzzle in which a hidden string of symbols must be discovered by producing query strings which are compared with the hidden one; the result of this comparison (in terms of *number* of correct positions and colors) is fed back to the player that is trying to crack the code (codebreaker). Methods for solving this puzzle are usually compared in terms of the number of query strings (guesses) made and the total time needed to produce those strings. In this paper we focus on the latter by trying to find a combination of parameters that is, first, uniform and independent of the problem size, and second, adequate to find a fast solution that is, at the same time, good enough. The key to this combination of parameters will be the *consistent set size*, that is, the maximum number of combinations that are sought before being scored and played as a guess. Having found in previous papers that the consistent set size has an influence on speed, we will concentrate on small sizes and test them through two different scoring methods from literature: most parts and entropy to find out the influence of that parameter on the outcome and which method scales better. With this we try to find out which method and size yield the best results an are effectively able to? find solutions for sizes not approached so far in a reasonable time.**

I. Introduction and state of the art

MasterMind [1], [2] is a puzzle in which one player A hides a combination of κ symbols and length ℓ, while the other player B, called the *codebreaker*, tries to uncover it by playing combinations coded in the same alphabet and with the same length. The answers from player A to every combination include the number of symbols in the combination that are in the correct position (usually white pins) and the number of colors that have been guessed correctly (black pins). Player B then plays new combinations until the hidden one is found. The objective of the game is to play repeatedly minimizing the number of turns needed to find the solution.

Solving a puzzle is a challenge by itself, even if it is a lowly board game such as this one. In fact, it is a game in which almost any computer strategy can beat a human, who uses a very different strategy [3], [4] to play it, namely, trying to first get the colors right and then fix the positions (but more elaborated versions of this one are also possible. But solutions to MasterMind can be applied easily to other games such as MineSweeper [5] or Hangman since the structure is the same: probing a part of the search space and getting a response from whoever holds the secret. We could stop at that and call MasterMind a worthwhile challenge, but it so happens that solving MasterMind can also be used to break ATMs guessing PINs [6] and even for an operation in biotechnology called *selective phenotyping* [7]. Not only that, but since it is a problem which is NP-Hard [8] the issue of finding bounds to the number of guesses needed is still open [9]. Therefore, there is room for making new venues into finding better or faster solutions to the game of MasterMind such as the heuristic solution we present in this paper.

Most solutions so far [10]–[13] use the concept of *eligible*, *possible* or *consistent* combinations: those that, according to responses by player A, could still be the hidden combination or, in other words, those that match the played combinations as indicated by the answer. Exhaustive methods [1], [14] would eliminate all non-consistent solutions and play a consistent one, while non-exhaustive methods would sample the set of consistent solutions and play one of them. Those solutions are guaranteed to reduce the search space at least by one, but obviously different combinations have a different reduction capability. This *capability* is reflected by a score. However, scores are heuristic and there is no rigorous way of scoring combinations. To compute these scores, every combination is compared in turn with the rest of the combinations in the set; the number of combinations that get every response (there is a limited amount of possible responses) is noted. Eventually this results in a series of *partitions* in which the set of consistent combinations is divided by its *distance* (in terms of common positions and colors) to every other. Since the goal of the *codebreaker* is to find the hidden combination in as few steps as possible, she is interested in obtaining as small a partition as possible. Therefore, it makes sense to focus on the size of these partitions (which one will be the remaining feasible partition is obviously not known in advance, so some heuristic reasoning is here required).

As an example, let us suppose the hidden combination considered is AABB (we are representing here colors by letters; in practice there is no difference as long as they are different symbols); then there will be 256 combinations whose

TABLE I. SCHEME OF PARTITIONS AFTER TWO COMBINATIONS HAVE BEEN PLAYED IN AN IMAGINARY GAME; COLORS ARE HERE REPRESENTED WITH CAPITAL LETTERS. EVERY COMBINATION IS MATCHED WITH ALL THE REST OF THE CONSISTENT SET (THAT IS, THE SET OF CONSISTENT COMBINATIONS). IN BOLDFACE, THE COMBINATIONS WHICH HAVE THE MINIMAL WORST SET SCORE (WHICH HAPPENS TO BE IN THE 1B-1W COLUMN, BUT IT COULD BE ANY ONE); IN THIS CASE, EQUAL TO TEN. A STRATEGY THAT TRIES TO MINIMIZE WORST CASE WOULD PLAY ONE OF THOSE COMBINATIONS. ON THE OTHER HAND, THE ENTROPY AND MOST PARTS SCORES USED IN THIS PAPER ARE IN THE LAST COLUMN; COMBINATIONS WITH THE MAXIMUM SCORES ARE *italiziced*. I THIS CASE, THE STRINGS WITH MAX SCORE ARE THE SAME. THE COLUMN 0B-1W WITH ALL VALUES EQUAL TO 0 HAS BEEN SUPPRESSED; COLUMN FOR COMBINATION 3B-1W, BEING IMPOSSIBLE, IS NOT SHOWN EITHER. SOME ROWS HAVE ALSO BEEN ELIMINATED TO SAVE SPACE.

Combination	Number of combinations in the partition with response										Entropy	Most Parts
	0b-2w	0b-3w	0b-4w	1b-1w	1b-2w	1b-3w	2b-0w	2b-1w	2b-2w	3b-0w	Score	Score
AABA	0	0	0	14	8	0	13	1	0	3	1.350	5
AACC	8	0	0	10	5	0	8	4	1	3	1.787	7
AACD	6	2	0	11	6	1	4	5	1	3	1.967	9
AACE	6	2	0	11	6	1	4	5	1	3	1.967	9
AACF	6	2	0	11	6	1	4	5	1	3	1.967	9
ABAB	8	0	0	10	5	0	8	4	1	3	1.787	7
ABAD	6	2	0	11	6	1	4	5	1	3	1.967	9
ABAE	6	2	0	11	6	1	4	5	1	3	1.967	9
ABAF	6	2	0	11	6	1	4	5	1	3	1.967	9
ABBC	4	4	0	10	8	0	8	1	1	3	1.851	8
ABDC	3	4	1	11	9	1	4	2	1	3	1.991	10
ABEC	3	4	1	11	9	1	4	2	1	3	1.991	10
ABFC	3	4	1	11	9	1	4	2	1	3	1.991	10
ACAA	0	0	0	14	8	0	13	1	0	3	1.350	5
ACCB	4	4	0	10	8	0	8	1	1	3	1.851	8
ACDA	0	0	0	16	10	2	5	3	0	3	1.525	6
BBAA	8	0	0	10	5	0	8	4	1	3	1.787	7
BCCA	4	4	0	10	8	0	8	1	1	3	1.851	8
BDCA	3	4	1	11	9	1	4	2	1	3	1.991	10
BECA	3	4	1	11	9	1	4	2	1	3	1.991	10
BFCA	3	4	1	11	9	1	4	2	1	3	1.991	10
CACA	8	0	0	10	5	0	8	4	1	3	1.787	7
CBBA	4	4	0	10	8	0	8	1	1	3	1.851	8
CBDA	3	4	1	11	9	1	4	2	1	3	1.991	10
CBEA	3	4	1	11	9	1	4	2	1	3	1.991	10
CBFA	3	4	1	11	9	1	4	2	1	3	1.991	10
DACA	6	2	0	11	6	1	4	5	1	3	1.967	9
EBAA	6	2	0	11	6	1	4	5	1	3	1.967	9
FACA	6	2	0	11	6	1	4	5	1	3	1.967	9
FBAA	6	2	0	11	6	1	4	5	1	3	1.967	9

response will be 0b, 0w (those with other colors), 256 with 0b, 1w (those with either an A or a B), and so on. Some partitions may also be empty, or contain a single element (4b, 0w will contain just AABB, obviously). For a more exhaustive explanation see [15]; the whole partition set for an advanced stage of the game is shown in Table I. Each combination is thus characterized by the features of these partitions: the number of non-empty ones, the average number of combinations in them, the maximum, and other characteristics related to the reduction of the size of the search space that might distinguish them. For instance, in Table I, the combination ABDC (and others) would have a maximum of non-zero partitions (none of them has zero elements), while AACC (and others, shown in boldface) have a minimum worst-case partition size (equal to ten).

Please note that these are heuristic methods based on the expected outcome. On average, they will reduce the search space size but even combinations with the same score will obtain a different result depending, among other things, on which is the actual secret combination. Finding the best score for a particular size, or even a particular set of combinations, is an open issue which leaves room for the research performed in this paper.

These partitions are used to assign a score to every combination; several scores have been proposed:

- *Worst case* was introduced by Knuth [1] as the first rigorous strategy to play MasterMind; combinations are scored according to the size of the biggest partition; the bigger, the worse. The combination to play is extracted from the set of those with the smallest biggest combination. In the case of Knuth, ties were broken by using the first in lexicographical order, but other methods might break them randomly, for instance.
- *Most parts*, proposed in [15], takes into account only the number of non-zero partitions. This is the one of the scores used in this paper and is shown in the last column in Table I.
- *Entropy* which has been used in [16]–[18], computes the entropy of partitions, and tries to maximize it. This score is illustrated in Table I in the next-to-last column.
- *Expected size* used by [12], [19] tries to minimize the expected size.

Experiments so far show that the best strategies are Most parts and Entropy, with no distinct advantage, for all problem

sizes, of any one of them over the others [13], although Entropy seems to have a slight edge, at the cost of a slightly slower performance [20]. In fact, combinations of them are possible, according to [14], but have not, so far, been proved in problems where a sample of the consistent set is used or, for that matter, in evolutionary algorithms. Most strategies, however, concentrate on finding the right size for minimization of the number of turns. The complexity of the solving algorithm, however, increases quadratically with the set size, since it involves comparing every combination with the rest, which implies that the time needed to find the solution and also number of evaluations increase too fast with the search space size, resulting in a bad scaling behavior.

Currently, the state of the art in average number of guesses and, to a point, in speed, was established by Berghman et al. in [12]. They obtained a system that is able to find the solution in an average number of moves that is, for all sizes tested, better than previously published. The number of evaluations was not published, but time was. In both cases, their solutions were quite good. However, there were many parameters that had to be set for each size, starting with the first guess and the size of the consistent set, as well as population size and other evolutionary algorithm parameters. In this paper we will try to adapt our previously published Evo method [21] by reducing the number of parameters without compromising too much on algorithm performance. However, some researchers have tried to break the barrier of spaces with a high dimension by searching for fast solutions: Khalifa and Yampolsky [22] were able to solve the length 8, 12 colors ($\ell = 8, \kappa = 12$) problem in just 8 seconds, but with an average number of guesses equal to 25 (vs. 20.571 seconds and 8.366 average number of guesses attained by Berghman [12]). Even if the result from the pure game-playing point of view is less than optimal, it open the venue for a new research path, setting the parameters of the algorithm so that faster solutions are found without sacrificing the quality of the guesses played.

However, even as good solutions can be found using only a sample of the consistent set size as proved in [12], [13], different set sizes do have an influence on the outcome. When size is reduced to the minimum it is bound to have an influence on the result, in terms of turns needed to win and number of evaluations needed to do it. The effect of the reduction of this sample size will decrease the probability of finding, and thus playing, the hidden combination and also the probability of finding the combination that maximally reduces the search space size when played, not to mention the fact that scoring will be biased by the absence of some combinations (which, in fact, might lead to having non-optimal combinations with the best score). However, in this paper we will prove that good solutions can be found by using a small sample size and, what is more, a common consistent set size across all MasterMind problem sizes. Using this common size means that we no longer need to worry about setting this parameter to find an optimal result, as we had done in previous papers [23] resulting in a great improvement on the number of experiments needed to find an acceptable solution. In order to do that, we will extend the study published in [24] to bigger problem sizes (while keeping the same consistent set size, cs, fixed to 10) and also include a study of how these small cs values impact the Entropy scoring method by testing two values: $cs = 8, 16$. By doing this we will try to find out not only whose scoring method scales better, but also which value is able to yield the best solutions in general.

The rest of the paper is organized as follows: the next section (II) presents the state of the resolution of the MasterMind puzzle using evolutionary algorithms. Experiments performed and results obtained with this method will be presented in Section III, and we will finish the paper with the conclusions that derive from them (commented in Section IV).

II. Playing MasterMind using evolutionary algorithms

This paper uses the method called, simply, *Evo*[1] [18], [21], [23], [25], [26] to find the hidden combination. This method, which has been released as open source code at CPAN (http://search.cpan.org/dist/Algorithm-MasterMind/), is an evolutionary algorithm that has been optimized for speed and to obtain a solution to the game in the minimal number of evaluations possible. This evolutionary algorithm codifies possible solutions directly (as strings) and scores them according to its closeness to the optimal solution (called *fitness*); an optimal solution in this case is a consistent solution.

Evo, which is explained extensively in [23] is a diversity-enhanced evolutionary algorithm that looks for consistent combinations until a prefixed amount of them has been found; this parameter is called *consistent set size* or simply cs. It uses Most Parts to score consistent combinations, and the *distance to consistency* for non-consistent ones, so that the fitness directs search towards finding consistent solutions and to find solutions with better score. When the algorithm finds cs solutions or when the number of solutions found does not vary for a number of generations (set to three throughout all experiments), the one with the best score is played. If several are tied for best score, a random one is played.

Evo++, as opposed to the plain Evo, incorporates a series of methods to decrease the amount of evaluations needed to find the solution, including *endgames* [23] which makes the evolutionary algorithm revert to exhaustive search in the case the search space has been well characterized (for instance, when we know that the solution is a permutation of one of the combinations played or when we have discarded some colors, reverting to a problem of smaller size).

The results are quite promising, but the main problem is that the number of evaluations needed to find the solution increases rapidly with problem size (fortunately, not as fast as the problem size itself or this solution would not be convenient) and a new parameter is introduced: the optimal size of the set of consistent combinations, that is, the number of combinations that the algorithm tries to find out before it plays one.

In this paper we also test another algorithm which we will call EFE. The main difference between EFE and Evo is that the former has been profiled (implementation *always* matters [27]) and bottlenecks eliminated [2] and uses Entropy instead of Most Parts for scoring. In general, it has been proved extensively in the literature (for instance, in [20], that compares Most

[1]The full name would be, following Perl naming conventions, `Algorithm::MasterMind::Evo`, but in this context we will use just the suffix since its full meaning is clear

[2]The details of this profiling are explained in another paper under review

TABLE II. VALUES FOR THE EVO PARAMETERS THAT OBTAIN THE BEST RESULT. PERMUTATION, CROSSOVER AND MUTATION ARE *priorities*; THEY ARE NORMALIZED TO ONE TO CONVERT THEM TO APPLICATION RATES. IN PRACTICE, CROSSOVER WILL BE APPLIED TO 80% AND MUTATION AND PERMUTATION TO 10% OF THE NEWLY GENERATED COMBINATIONS EACH.

Parameter	Value
Crossover priority	8
Mutation priority	1
Permutation priority	1
Replacement rate	0.75
Tournament size	7

Parts and Entropy) that Entropy should offer better solutions . However, the sampling size of the consistent set will play differently in the two scoring methods, that is why we are interested in finding out which one of them yields the best solutions and for which cs.

What we do in this paper is to test an *one size fits all* approach by making the size of the consistent set unchanged for any problem size. This reduces the algorithm parameter set by one, but since this parameter set has, a priori, a big influence on result and there is no method to set it other than experimentation, it reduces greatly the amount of generations needed to obtain a solution.

III. EXPERIMENTS AND RESULTS

The experiments presented in this paper extend those published previously, mainly in [23] which was a general introduction to the Evo method and [24] where we did some initial experiments in using a small consistent set size, limited to Evo using Most Parts scoring. In this paper we will set the consistent set size cs to a minimal value that is common for all sizes: 10 (that is why we will denominate the method tested *Evo10*) and 8 and 16 for the Entropy score. This value has been chosen to be small enough to be convenient, but not so small that the scoring methods are rendered meaningless. This will reduce the parameters needed by one, leaving only the population size to be set, once, of course, the rest of the evolutionary algorithm parameters have been fixed by experimentation; these parameters are shown in Table II have been found to be the best in a previous study by systematically testing different population sizes and operator rates.

For every problem size, a fixed set of 5000 combinations were generated randomly. There is at most a single repetition in the smallest size, and no repetition in the rest. The sets can be downloaded from http://goo.gl/6yu16; these are the same that we have considered in previous papers. A single game is played for every combination. The results for this fixed parameter setting are shown in Tables III(a),III(b), III(c) and III(d).

These tables compare previous results (with its corresponding consistent set size, published in [23]) and the results presented in this paper, labeled Evo10.

As it can be seen in Tables III(a) and III(b), which represent the average number of moves needed to find the solution, results are quite similar. The average for Evo10 is consistently higher (more games are needed to find the solution) but in half the cases the difference is not statistically significant using Wilcoxon paired test. There is a significant difference for the two smaller sizes ($\ell = 4, \kappa = 8$ and $\ell = 5, \kappa = 8$), but not for the larger sizes $\ell = 5, \kappa = 9$ and $\ell = 6, 7$. This is probably due to the fact that, with increasing search space size, the difference among 10 and other sample size, even if they are in different orders of magnitude, become negligible; the difference between 10% and 1% of the actual sample size is significant, but the difference 0.001% and 0.0001% is not.

However, the difference in the number of evaluations (shown in Tables III(c) and III(d)), that is, the total population evaluated to find the solution is quite significant, going from a bit less than half to a third of the total evaluations for the larger size. This means that the time needed scales roughly in the same way, but it is even more interesting to note that it scales better for a fixed size than for the best consistent set size. Besides, in all cases the algorithm does not examine the full set of combinations, while previously the number of combinations evaluated, 6412, was almost 50% bigger than the search space size for that problem. The same argument can be applied to the comparison with Berghman's results (when they are available); Evo++ was able to find solutions which were quite similar to them, but Evo10 obtains an average number of turns that is slightly worse; since we don't have the complete set of results, and besides they have been made on a different set of combinations, it is not possible to compare, but at any rate it would be reasonable to think that this result is significant.

As it can be seen in Figure 1 the number of evaluations needed to find the solution (which is a bit worse for Evo10) falls much faster than for Evo++ so that, in fact, we can approach larger sizes than could be done using the previous methods, as the Figure 2 shows. This figure plots the average number of evaluations needed to find the solution vs. solution size. The most interesting feature of this plot is the observation that the $\kappa = 9, \ell = 5$ needs, on average, as many evaluations as the largest size evaluated here, $\kappa = 10, \ell = 7$; however, search space size is 20 times bigger for this one, which implies that reducing the size of the consistent set to just ten multiplies by 20 the size of the search space approachable using this type of evolutionary algorithms.

One of the keys of managing to find the solution in less evaluations is reducing the population size. As a rule of thumb, we have used in most cases a population that was slightly smaller than the one we used in Evo++. In evolutionary algorithms, if the population is kept at a size large enough to maintain diversity, decreasing its size usually leads to more efficient algorithms. That is why looking at one of the manageable sizes, $\ell = 6, \kappa = 9$, we have tested several population sizes to check its influence on the solutions and the number of evaluations. These have been plotted in Figure 3, which shows a monotonically increasing number of evaluations with population size. In fact, the average number of guesses does not follow a clear trend. It decreases for population=1500, but it increases for population=2000 to be approximately the same than for population=800 (in fact, statistically indistinguishable). This could imply that while population has a strong influence in the number of evaluations needed to find the solution, the consistent set size has an influence mainly on the average number of guesses needed to find the solution. However, a more precise statistical characterization will need more systematic experimentation. For the time being, the results obtained show that, in general, Evo10 needs less or the

TABLE III. COMPARISON AMONG THIS APPROACH (*Evo10*) AND PREVIOUS RESULTS PUBLISHED BY THE AUTHORS (EVO++) IN [23] AND BERGHMAN ET AL. [12].

(a) Mean number of guesses and the standard error of the mean for $\ell = 4, 5$, the quantities in parentheses indicate population and consistent set size (in the case of the previous results).

	$\ell = 4$	$\ell = 5$	
	$\kappa = 8$	$\kappa = 8$	$\kappa = 9$
Berghman et al.		5.618	
EVO++	(400,30) 5.15 ± 0.013	(600,40) 5.62 ± 0.012	(800,80) 5.94 ± 0.012
EVO10	(200) 5.209 ± 0.012	(600) 5.652 ± 0.011	(800) 6.013 ± 0.012
EFE8	(200) 5.247 ± 0.013	(600) 5.689 ± 0.012	(800) 6.005 ± 0.012
EFE16	(200) 5.222 ± 0.013	(600) 5.624 ± 0.012	(800) 5.9832 ± 0.012

(b) Mean number of guesses and the standard error of the mean for $\ell = 6, 7$, the quantities in parentheses indicate population and consistent set size (in the case of the previous results).

	$\ell = 6$		$\ell = 7$
	$\kappa = 9$	$\kappa = 10$	$\kappa = 10$
Berghman et al.	6.475		
EVO++	(1000,100) 6.479 ± 0.012		
EVO10	(800) 6.504 ± 0.012	(1000) 6.877 ± 0.013	(1500) 7.425 ± 0.013
EFE8	(800) 6.537 ± 0.012	(1000) 6.891 ± 0.013	(1500) 7.422 ± 0.013
EFE16	(800) 6.488 ± 0.012	(1000) 6.853 ± 0.013	(1500) 7.372 ± 0.013

(c) Mean number of evaluations and its standard deviation $\ell = 4, 5$.

	$\ell = 4$	$\ell = 5$	
	$\kappa = 8$	$\kappa = 8$	$\kappa = 9$
EVO++	6949 ± 48	14911 ± 6120	25323 ± 9972
EVO10	2551 ± 20	7981 ± 50	8953 ± 3982
EFE8	2475 ± 56	7596 ± 49	10412 ± 65
EFE16	2896 ± 21	8947 ± 72	12261 ± 74

(d) Mean number of evaluations and its standard deviation $\ell = 6, 7$.

	$\ell = 6$		$\ell = 7$
	$\kappa = 9$	$\kappa = 10$	$\kappa = 10$
EVO++	46483 ± 17031		
EVO10	17562 ± 1914	21804 ± 960	40205 ± 926
EFE8	28875 ± 14155	21256 ± 1873	315437 ± 264054
EFE16	16993 ± 105	22819 ± 206	322319 ± 278938

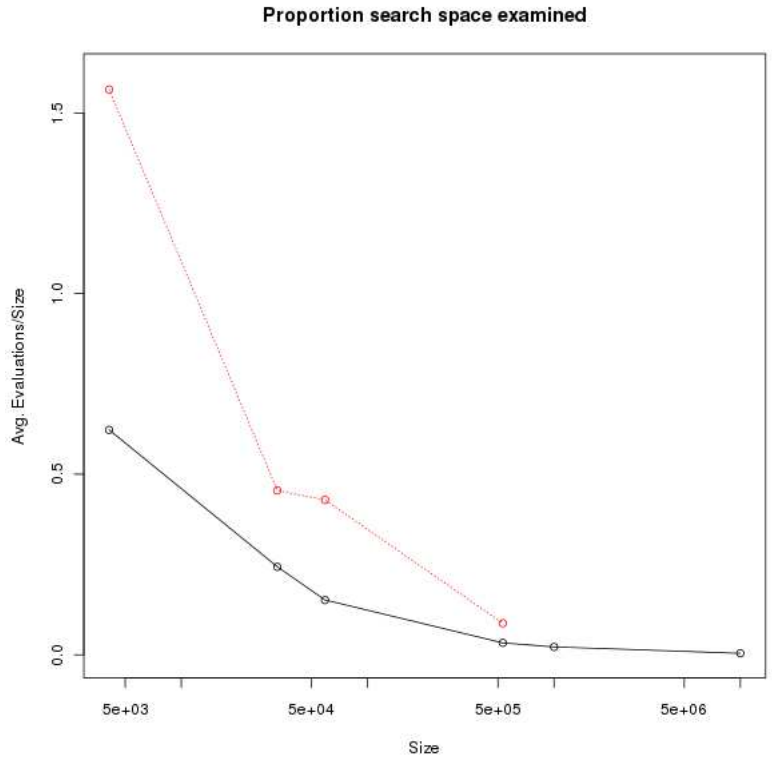

Fig. 1. Average proportion of search space (that is, number of evaluations divided by search space size) vs. search space size for the original Evo++ method (dotted, red or light color) and the current Evo10 (solid, black). Please note that the x axis is logarithmic.

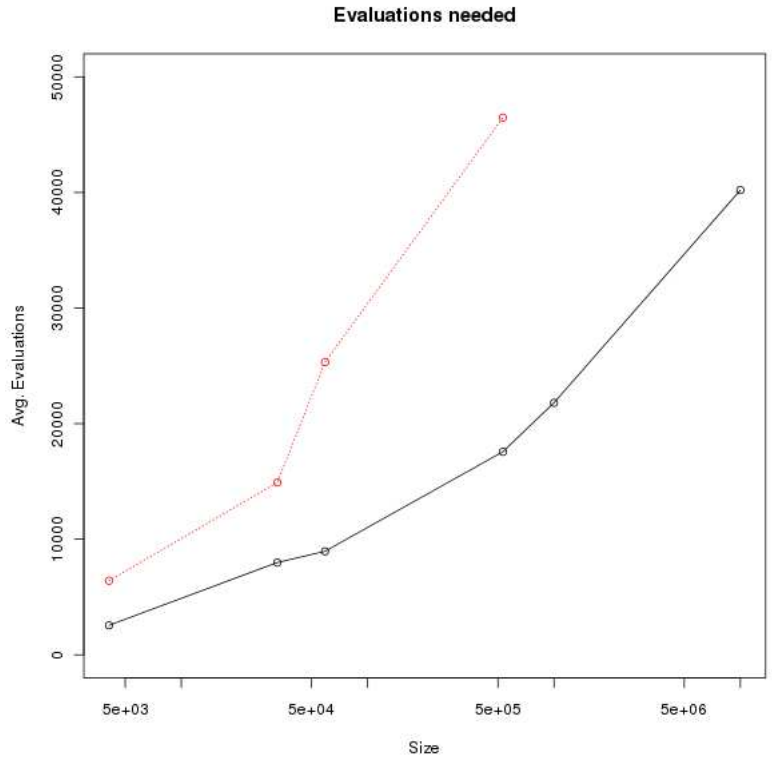

Fig. 2. Average number of evaluations vs. search space size for the original Evo++ method (dashed, light color) and the current Evo10 (solid, black). Please note that the x axis is logarithmic.

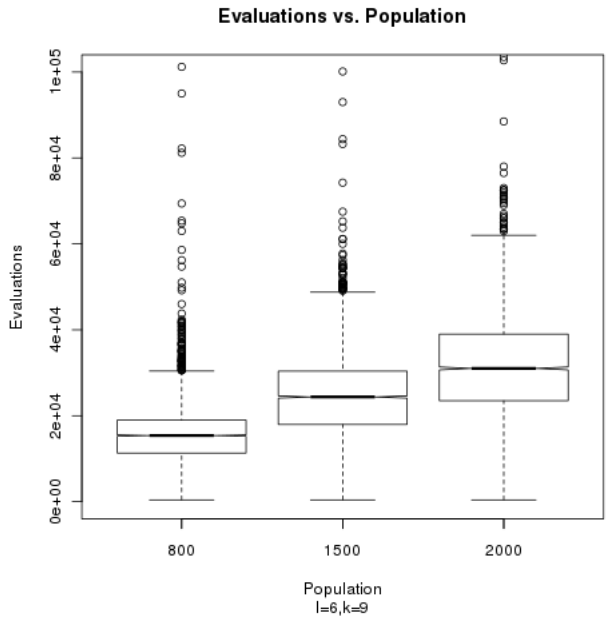

Fig. 3. Boxplot of the average number of evaluations vs. population size for Evo10.

same population than Evo++ to obtain similar results using less evaluations and a very similar number of guesses (at least for the larger search space sizes).

The experiments performed with the EFE entropy-based method prove that using a small set size does, in fact, yield good results independently of the size. In general, it shows also that the smaller size tested ($cs = 8$) is never better than Evo10; at most, it is statistically indistinguishable. However, EFE16 is always better than Evo10, in some cases statistically so (using Wilcoxon test). This is interesting since, as indicated in the introduction, every method will need different cs to obtain good results. In particular, computing entropy of a set with a small size might produce meaningless results (while Most Parts will always produce a meaningful result, number of non-zero partitions); when the size is increased, the results will be increasingly accurate. In fact, these better results come at the cost of more evaluations (as should only be expected, since the evolutionary algorithm must run for a few generations before issuing a new solution) but, since this implementation is faster than Evo10, it is in fact noticeably faster: the $\ell = 6$ experiments take on average a single second (while it was from five to 15 seconds for the non-optimized Evo++ method, depending on cs, as shown in the experiment ChangeLog published below). As usual, raw data, processing scripts and computed results are available from our GitHub repository http://github.com/JJ/algorithm-mastermind/experiments/CEC13.

IV. Discussion, conclusions and future work

This paper has shown that using a small and fixed consistent set size when playing mastermind using evolutionary algorithms does not imply a deterioration of results, while

cutting in half the number of evaluations needed to find them. This makes the configuration of the algorithm shown quite suitable for real-time games such as mobile apps or web games; the actual time varies from less than one second for the smallest configuration to a few seconds for the whole game in the biggest configuration shown; the time being roughly proportional to the number of evaluations, this is at least an improvement of that order; that is, the time is reduced by 2/3 for the $\kappa = 6, \ell = 9$ problem, attaining an average of 4.7 seconds, almost one fourth of the time it can take when we try to achieve the minimum number of turns, 18.6 seconds. This number is closer to the one published by [12] for this size, 1.284s, although without running it under the same conditions we cannot be sure. It is in the same ballpark, anyways. EFE8 and EFE16, since they have been optimized for speed too, reduce that time even further to an median of 0.74 and 0.92 seconds, respectively (averages are 1.7 and 0.9; in the first case it is much higher due to the presence of an execution that took 4000 seconds), so that, indeed, we can obtain solutions faster than the state of the art established by Berghman, and with an average number of guesses of 6.488 (vs. 6.475) statistically indistinguishable (p-value = 0.9908); in fact, in this case it is also statistically indistinguishable from the best value obtained by Evo++ using $cs = 100$. The time needed to find the solution has a strong component in the number of evaluations, but it also depends on the consistent set size, that is why the relation between the time needed (1/4) is smaller than the relation between number of evaluations (roughly 1/3, see Table III(d)) . This allows also to extend the range of feasible sizes, and yields a robust configuration that can be used throughout any MasterMind problem.

The explanation for the results obtained in this paper and the small difference between them and those obtained using a bigger cs (those computed using Evo++) lies in the fact that the parameter that sets the size of the consistent set constitutes just a bound for the actual size the consistent sets have. This size varies as the turns proceed, as shown in [14] and go from a considerable fraction of the total search space to just a few combinations after several turns have been played. This implies that, in fact, there are just a few turns or even a single one (which would be the second one, since a fixed combination is always played on the first turn) in which the evolutionary algorithm is able to actually find a number of combinations over the set limit. At the beginning of the game there are naturally many combinations, and in fact the evolutionary algorithm only kicks in when the number of available combinations falls below the set size. However, once that happens the total size of the consistent set is smaller than cs, which implies that, being impossible to attain that size, the algorithm stops when the number of consistent combinations does not increase for three consecutive turns. The fact that the evolutionary algorithm kicks in after several turns leads also to a reduction in the number of evaluations needed: only when the cs falls below 10 new individuals will be generated; meanwhile, it will be the initial population that will be culled for consistent combinations after each turn. The number of turns where the evolutionary algorithm is really working will vary with the problem size, but it could be as low as one (even 0, in some cases where the solution is already in the original population and is played).

The *actual* consistent size is illustrated in Figure 4, which

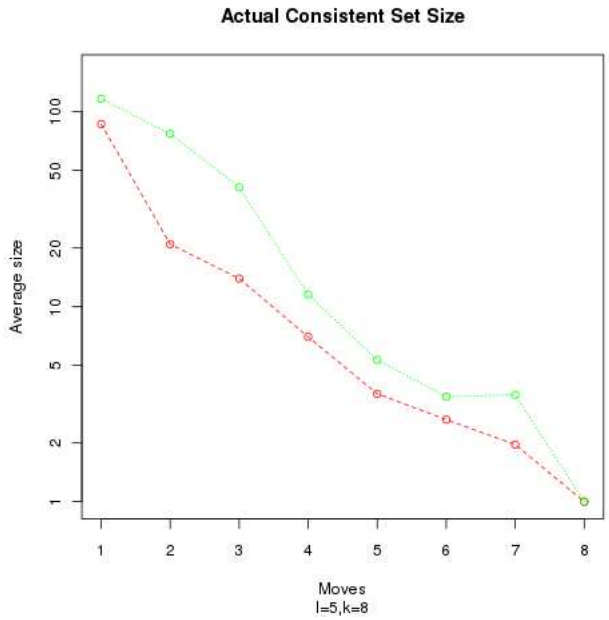

Fig. 4. Average actual size of the set of consistent combinations for the original Evo++ method (solid, light color) and Evo10 (dashed, intense or red) in the $\ell = 5, \kappa = 9$ problem. Please note that the y axis is logarithmic. As shown in Table III(a) the bound to the consistent set size was 60 for Evo++. EFE is not shown in this illustration since our objective is to prove only that actual consistent set size varies with every move, and that changing cs to a smaller value does have an influence on that. A similar scenario should be expected for EFE.

shows how the average (for all 5000 games) size of the consistent set evolves with the turn number. The actual size of the set depends on the bound; the larger the bound, the larger the size. In fact, the actual average size is for all turns, not only for those that are above the limit, which reflects the fact that this is an *average* size and whenever the actual set size is below the bound the algorithm will work for a few generations trying to increase the size, until several generations without improvement have elapsed. This Figure also shows that, for Evo++, there will be normally 3 turns until the evolutionary algorithm starts to work (average consistent set size falling below 60), but Evo10 will produce 4 combinations (including the first one, which is fixed) before start. Even if this is just a single example of all problem sizes (and methods) studied in this paper, it is representative in the sense that the situation will be very similar (Evo10 with smaller consistent set size than Evo++), although the turns in which the evolutionary algorithm starts to actually run will vary.

The fact that the actual consistent set size only falls below cs after a few games (in 4, on average, after the third guess for Evo10 and the first for Evo++) explains the low number of evaluations obtained by this new parametrization of the algorithm. In fact, many games will be played using the combinations present in the initial population, with just a few games needing more than a few generations to find the hidden combination. Please note that, in the case of the $\ell = 6, \kappa = 9$ problem, the average number of combinations played implies that just a few dozens of generations have actually run; this number obviously increases with the problem size.

As future lines of work, we will try to reduce even more this size and try to check whether it offers good results for bigger sizes such as $\ell = 7, \kappa = 11$ or even $\ell = 8, \kappa = 12$ and once again try to beat the state of the art in that size (Berghman et al. were able to obtain a solution in less than 9 seconds). Several consistent set sizes will be systematically evaluated, looking mainly for a reduction in the number of evaluations, and time, needed. Eventually, what we are looking is for a method that is able to resolve problems with moderate size, but this will need to be tackled from different points of view: implementation (it always matters, even more so in evolutionary algorithms [27]), middle-level algorithms used, even the programming language we will be using. We might even have to abandon the paradigm of playing always consistent solutions to settle, sometimes, for non-consistent solutions for the sake of speed. At these sizes, besides, populations reach a level in which a concurrent or parallel solution might be a better option to reach the solution in a reasonable amount of time.

It is also clear than, when increasing the search space size, the size of the consistent set will become negligible with respect to the actual size of the consistent set. This could work both ways: first, by making the results independent of sample size (for this small size, at least) or by making the strategy of extracting a sample of a particular size indistinguishable from finding a single consistent combination and playing it. As we improve the computation speed, it would be interesting to take measurements to prove these hypotheses.

ACKNOWLEDGEMENTS

This work is supported by projects TIN2011-28627-C04-02 and TIN2011-28627-C04-01 (ANYSELF), awarded by the Spanish Ministry of Science and Innovation, P08-TIC-03903 and TIC-6083 (DNEMESIS) awarded by the Andalusian Regional Government, and project 83 (CANUBE) awarded by the CEI-BioTIC UGR (http://biotic.ugr.es).

REFERENCES

[1] D. E. Knuth, "The computer as Master Mind," *J. Recreational Mathematics*, vol. 9, no. 1, pp. 1–6, 1976-77.

[2] G. Montgomery, "Mastermind: Improving the search," *AI Expert*, vol. 7, no. 4, pp. 40–47, 1992.

[3] P. Laughlin, R. Lange, and J. Adamopoulos, "Selection strategies for "Mastermind" problems." *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 8, no. 5, p. 475, 1982.

[4] S. Hubalovsky, "Modeling and computer simulation of real process–solution of Mastermind board game," *International Journal of Mathematics and Computers in Simulation*, pp. 107–118, 2012.

[5] M. Legendre, K. Hollard, O. Buffet, A. Dutech *et al.*, "MineSweeper: Where to probe?" INRIA, Tech. Rep. RR-8041, 2012, http://hal.inria.fr/hal-00723550.

[6] R. Focardi and F. Luccio, "Guessing bank pins by winning a mastermind game," *Theory of Computing Systems*, pp. 1–20, 2011.

[7] J. Gagneur, M. Elze, and A. Tresch, "Selective phenotyping, entropy reduction, and the mastermind game," *BMC bioinformatics*, vol. 12, no. 1, p. 406, 2011, http://www.biomedcentral.com/1471-2105/12/406.

[8] G. Viglietta, "Hardness of Mastermind," *CoRR*, vol. abs/1111.6922, 2011.

[9] B. Doerr, R. Spöhel, H. Thomas, and C. Winzen, "Playing Mastermind with Many Colors," *arXiv preprint arXiv:1207.0773*, 2012.

[10] J. O´Geran, H. Wynn, and A. Zhigljavsky, "Mastermind as a test-bed for search algorithms," *Chance*, vol. 6, pp. 31–37, 1993.

[11] J. Francis, "Strategies for playing MOO, or "Bulls and Cows"," http://www.jfwaf.com/Bulls%20and%20Cows.pdf.

[12] L. Berghman, D. Goossens, and R. Leus, "Efficient solutions for Mastermind using genetic algorithms," *Computers and Operations Research*, vol. 36, no. 6, pp. 1880–1885, 2009.

[13] T. P. Runarsson and J. J. Merelo, "Adapting heuristic Mastermind strategies to evolutionary algorithms," in *NICSO'10 Proceedings*, ser. Studies in Computational Intelligence. Springer-Verlag, 2010, pp. 255–267, also available from ArXiV: http://arxiv.org/abs/0912.2415v1.

[14] J.-J. Merelo-Guervós, A.-M. Mora, C. Cotta, and T.-P. Runarsson, "An experimental study of exhaustive solutions for the Mastermind puzzle," *CoRR*, vol. abs/1207.1315, 2012.

[15] B. Kooi, "Yet another Mastermind strategy," *ICGA Journal*, vol. 28, no. 1, pp. 13–20, 2005.

[16] E. Neuwirth, "Some strategies for Mastermind," *Zeitschrift fur Operations Research. Serie B*, vol. 26, no. 8, pp. B257–B278, 1982.

[17] A. Bestavros and A. Belal, "Mastermind, a game of diagnosis strategies," *Bulletin of the Faculty of Engineering, Alexandria University*, December 1986. [Online]. Available: http://citeseer.ist.psu.edu/bestavros86mastermind.html

[18] C. Cotta, J. J. Merelo Guervós, A. Mora García, and T. Runarsson, "Entropy-driven evolutionary approaches to the Mastermind problem," in *Parallel Problem Solving from Nature PPSN XI*, ser. Lecture Notes in Computer Science, R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds., vol. 6239. Springer Berlin / Heidelberg, 2010, pp. 421–431. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15871-1_43

[19] R. W. Irving, "Towards an optimum Mastermind strategy," *Journal of Recreational Mathematics*, vol. 11, no. 2, pp. 81–87, 1978-79.

[20] J. Maestro-Montojo, J.-J. Merelo-Guervós, and S. Salcedo-Sanz, "Comparing Evolutionary Algorithms to Solve the Game of MasterMind," in *EvoApplications*, ser. Lecture Notes in Computer Science, A. I. Esparcia-Alcázar, Ed., vol. 7835. Springer, 2013, pp. 304–313.

[21] J.-J. Merelo-Guervós, C. Cotta, and A. Mora, "Improving and Scaling Evolutionary Approaches to the MasterMind Problem," in *EvoApplications (1)*, ser. Lecture Notes in Computer Science, C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. J. M. Guervós, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis, Eds., vol. 6624. Springer, 2011, pp. 103–112.

[22] A. Khalifa and R. Yampolskiy, "GA with Wisdom of Artificial Crowds for Solving Mastermind Satisfiability Problem," *International Journal of Intelligent Games & Simulation*, vol. 6, no. 2, p. 6, 2011.

[23] J.-J. Merelo-Guervós, A.-M. Mora, and C. Cotta, "Optimizing worst-case scenario in evolutionary solutions to the MasterMind puzzle," in *IEEE Congress on Evolutionary Computation*. IEEE, 2011, pp. 2669–2676.

[24] J. J. Merelo, A. M. Mora, and C. Cotta, "Finding fast solutions to the game of Mastermind," in *GAME-ON 2012, 13th International Conference on Intelligent Games and Simulation*, C. C. Antonio Fernández-Leiva and R. Lara, Eds. Eurosis, 2012, pp. 25–28.

[25] J.-J. Merelo and T. P. Runarsson, "Finding better solutions to the Mastermind puzzle using evolutionary algorithms," in *Applications of Evolutionary Computing, Part I*, ser. Lecture Notes in Computer Science, C. di Chio et al., Ed., vol. 6024. Istanbul, Turkey: Springer-Verlag, 7 - 9 Apr. 2010, pp. 120–129.

[26] J. J. Merelo, A. M. Mora, T. P. Runarsson, and C. Cotta, "Assessing efficiency of different evolutionary strategies playing mastermind," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, August 2010, pp. 38–45.

[27] J.-J. Merelo-Guervós, G. Romero, M. García-Arenas, P. A. Castillo, A.-M. Mora, and J.-L. Jiménez-Laredo, "Implementation matters: Programming best practices for evolutionary algorithms," in *IWANN (2)*, ser. Lecture Notes in Computer Science, J. Cabestany, I. Rojas, and G. J. Caparrós, Eds., vol. 6692. Springer, 2011, pp. 333–340.