Appl Intell (2014) 40:404–414
DOI 10.1007/s10489-013-0470-4

An evolutionary-based hyper-heuristic approach for the Jawbreaker puzzle

S. Salcedo-Sanz · J.M. Matías-Román · S. Jiménez-Fernández · A. Portilla-Figueras · L. Cuadra

Published online: 13 September 2013

Abstract In this paper a hyper-heuristic algorithm is designed and developed for its application to the Jawbreaker puzzle. Jawbreaker is an addictive game consisting in a matrix of colored balls, that must be cleared by popping sets of balls of the same color. This puzzle is perfect to be solved by applying hyper-heuristics algorithms, since many different low-level heuristics are available, and they can be applied in a sequential fashion to solve the puzzle. We detail a set of low-level heuristics and a global search procedure (evolutionary algorithm) that conforms to a robust hyper-heuristic, able to solve very difficult instances of the Jawbreaker puzzle. We test the proposed hyper-heuristic approach in Jawbreaker puzzles of different size and difficulty, with excellent results.

Keywords Jawbreaker puzzle · Hyper-heuristics · Evolutionary algorithms

1 Introduction

Puzzles have been tackled for years in fields such as Computer Science, Mathematics or Artificial Intelligence, as difficult problems for testing algorithms [1–3], or teaching them to students in an appealing way [4]. In the last few years, many different puzzles have been tackled or studied using different approaches. The famous Rubik's cube (and related puzzles known as Rubik-type puzzles), for example, have been comprehensively studied within the frame of groups of symmetry, in different works [5–7]. Other puzzles have been tackled with approaches related to Soft-Computing or Computational Intelligence (mainly heuristic and meta-heuristics approaches) [8], such as Sudoku [9, 10], Mastermind [11, 12], Go [13, 14], Nonograms [15, 16] or different kind of Solitaire games and puzzles [17–19], etc. These works on solving puzzles with heuristics have contributed to the development of different algorithms [20], or have shown interesting contributions in the field of Education [21, 22], and they are the approaches we are interested in.

Specifically, this paper discusses a Computational Intelligence approach for the *Jawbreaker* puzzle. The Jawbreaker is a highly addictive puzzle game developed for mobile devices and PDAs in the early 2000. The game is presented as a set of bubbles of different colors, initially forming a square, that must be cleared off the screen by clicking on groups of three or more of the same color bubble. Point bonuses are given for clearing off larger groups. Jawbreaker can be also found under other names such as *SameGame*, *Bubblets* or *Bubble Breaker*. Jawbreaker is a difficult puzzle to be tackled with algorithms, since the game is continuously changing as groups of bubbles are cleared (the game matrix is continuously reconfigured while bubbles are being cleared). Jawbreaker can be seen as an NP-complete optimization problem, as has been shown before in the literature [8, 25]. Different games strategies for the Jawbreaker are given in manuals [23], and recently there have been a reduced number of works dealing with Monte-Carlo simulation to solve the SameGame puzzle, a variant of the Jawbreaker [24, 25]. Other classic heuristic approaches such as A^* or IDA^* are not suitable in the Jawbreaker puzzle, as has been shown in [25]. Basically, these algorithms are first-search approaches based on a tree-graph, where all nodes must be stored in a list. This list is then sorted by means of a

S. Salcedo-Sanz (✉) · J.M. Matías-Román · S. Jiménez-Fernández · A. Portilla-Figueras · L. Cuadra
Department of Signal Theory and Communications, Escuela Politécnica Superior, Universidad de Alcalá, 28871 Alcalá de Henares, Madrid, Spain
e-mail: sancho.salcedo@uah.es

given evaluation function, and, at iteration of the algorithm, the first element of the list is removed and its children are added to the sorted list. This procedure continues until the final state arrives at the top of the list. The *IDA** is an iterative deepening variant of this algorithm. Following [25], these classic approaches fail in Jawbreaker and related puzzles, mainly because it is not easy to obtain an admissible function which provides an accurate estimation. Different options are shown in [25] to make this estimation function, though the final conclusion is that the application of these kind of heuristics is rather difficult and they do not provide good results in this puzzle.

In general, to our knowledge, there are not works dealing with pure meta-heuristics approaches to solve the Jawbreaker. This indicates that Jawbreaker is a difficult puzzle to be tackled with meta-heuristics algorithms due to the intrinsic rule that makes the game change continuously when groups of bubbles are cleared. There is however, a recently proposed family of algorithms that perfectly fits the Jawbreaker puzzle the Hyper-heuristics (HHs) approaches. HHs are a new class of searching methodologies that have gained importance in Artificial Intelligence in the last few years [26, 27], with applications in different problems related to Operations Research mainly [28–33], but also in other optimization problems, including puzzles and games solutions [34, 35]. In this paper we specifically develop a HH algorithm to solve the Jawbreaker and show that this approach shows an excellent performance in this puzzle.

The HH developed for the Jawbreaker puzzle is based on a global search performed by an evolutionary algorithm, working on a search space formed by low-level (basic) heuristics related to possible actions in the Jawbreaker. We have predefined 19 basic heuristics that, applied in the optimal sequential fashion, conform an excellent methodology to solve the Jawbreaker. The evolutionary algorithm must form the sequence of basic heuristics that produces a best solution to the puzzle. We propose an integer encoding (numbered the basic heuristics from 1 to 19), and we use an extra symbol to represent the final of the basic heuristic application sequence. Thus, we are able to easily manage solutions with different number of basic heuristics to be applied. In the paper we discuss the performance of our approach by solving Jawbreaker instances of different size, and with different number of colors involved, and we also tackle several standardized instances of a version of the Jawbreaker (SameGame puzzle), where we successfully compare the HH performance with that of existing techniques for this puzzle.

The rest of the paper is structured as follows: next section presents the Jawbreaker puzzle, detailing its rules and characteristics. Section 3 gives a brief introduction of hyper-heuristics and their applications in the last years. Section 4 describes the specific HH developed in this work for the Jawbreaker. Section 5 describes the experiments carried out to test the performance of the proposed approach. Section 6 closes the paper with some final conclusions and remarks on the research carried out.

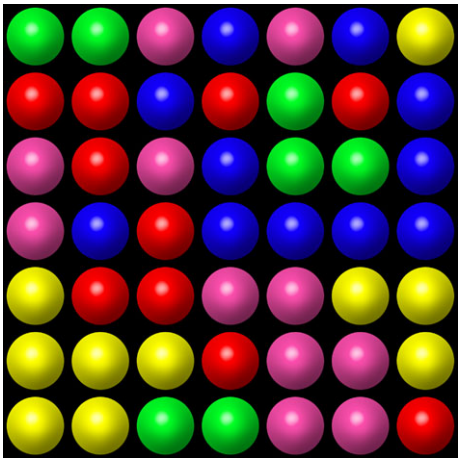


Fig. 1 Example of a Jawbreaker puzzle (initial matrix, 7×7, 5 colors)

2 The Jawbreaker puzzle

The Jawbreaker puzzle is a one-player game consisting of a screen of differently-colored balls arranged in a rectangular $(N \times M)$ matrix Ξ. The number of colors of the game (α) is a variable to be fixed by the game designer. The player then clicks on any two or more connecting similarly-colored balls to eliminate them from the matrix, earning a given number of points in the process. The more balls are cleared at once, the more the points are added to the player's score. In the original version of the game, this scoring is obtained by a simple formula:

$$y = n(n-1) \tag{1}$$

where y is the score obtained for clearing a set of similarly-colored balls, and n is the number of balls in the set.

Figure 1 shows an example of a 7×7 Jawbreaker Ξ. After clearing a set of balls, Ξ is reconfigured in such a way that the remaining balls above the ones cleared occupy the previous position of the cleared ones. If, in the clearing of group of balls, a given column j is completely cleared out, the reconfiguration of Ξ is done in such a way that all columns $j+1$ to M are shifted one position to the left, so matrix Ξ would have now $M-1$ columns. This process avoids the possibility of having isolated sets of balls in Ξ. Figure 2 shows some examples of matrix Ξ reconfiguration after different sets of balls clearing.

In its standard mode, the game ends when the player has no more possible moves, i.e., when there are no more sets of at least two similarly-colored balls in Ξ.

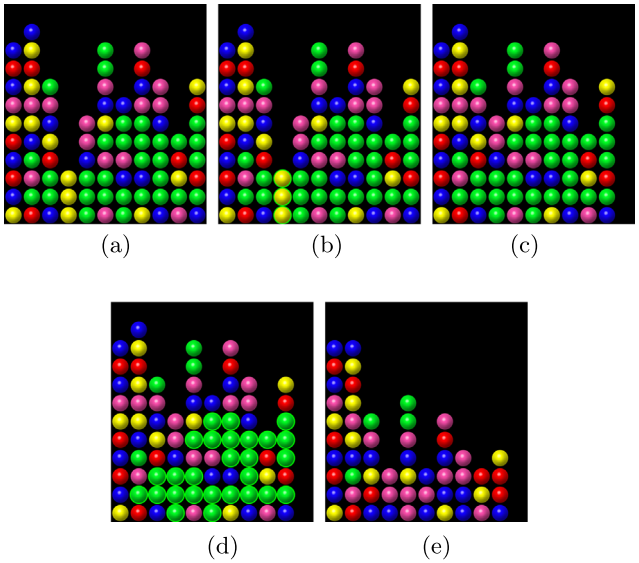


Fig. 2 Example of clearing sets of color balls in Jawbreaker; (**a**), (**b**) and (**c**) example a complete column clearing and its effect in the game matrix; (**d**) and (**e**) example of a large set of balls clearing (a score of 702 points if obtained by clearing this set of 27 *green* balls)

3 A brief introduction to hyper-heuristics

HHs are a new class of searching methodologies that have gained importance in Artificial Intelligence in the last few years. HHs have been defined as "search methods or learning procedures for selecting or generating heuristics in a given optimization problem" [27]. According to [27] the origin of HHs is back to the 60's, and different similar-ideas algorithms are developed in the 80's and 90's. However, is in the first years of 2000 when the foundations of these methodologies are set, and they are reveled as powerful approaches for solving hard optimization problems. Since then, the importance of HHs has been continuously growing, and they are currently considered state of the art algorithms in Computational Intelligence, with application in many different problems such as timetabling and rostering [36, 37], scheduling [38, 39], packing [40, 41], assignment and allocation [42, 43], channel assignment in telecommunication networks [45, 46], or data mining [47, 48], etc.

The idea behind HHs is simple, but brilliant: we can use a collection of basic (low-level) heuristics, which on their own do not produce good solutions to a given problem, to come up with a much better solution, by combining them or by generating new heuristics from them using a high-level algorithm. In fact, according to [49], HHs can be classified in different types or categories attending to the nature of the heuristic search space. Thus, HHs are classified into *HHs for heuristic selection*, that comprises those methodologies focussed on choosing or selecting existing heuristics, and *HHs for heuristics generation*, that includes those methodologies focussed on creating new heuristics from existing ones.[1] There is also a classification of HHs based on how the high-level approach receives feedback during the algorithm. Thus, the HHs can operate in offline or online learning. Online learning implies that the high-level algorithm learns while solving a given instance of a problem, whereas offline operation consists of learning from a set of training instances, that will hopefully generalize to solving unseen instances (similar to the learning method in neural computation).[2]

Focused on heuristic selection HHs approaches, the process to construct the complete algorithm is quite simple: first, it is necessary to come up with a set of good low-level (basic) heuristics to the problem at hand. It is not necessary that the heuristics on their own are very effective in solving the problem, but it is interesting that these basic heuristics are different among them, and their number should be large enough to generate a large enough search space [44]. Then, a high-level approach (heuristic or not) must be selected in order to obtain the best set of low-level heuristics and how to apply them to solve the problem. In many cases the high-level algorithm is a meta-heuristic (evolutionary algorithm, ants algorithm, particle swarm, etc.), which needs a proper encoding of the low-level encoding to perform the search. This encoding completely depends on the problem being solved, and the algorithm's performance strongly depends also on this election. Figure 3 shows a schema of a HH algorithm similar to the one described in this paper. A pool of low-level heuristics is selected, and its sequential application to the resolution of a given problem provides the fitness function of the evolutionary algorithm. An encoding with a specific stop symbol (number 0 in this case) is selected to represent the application of the heuristics to the problem.

A number of good tutorials and reviews on HHs can be found in the literature for the interested reader [27, 50, 51]. To our knowledge, [27] is the most recent one. A summary of the main applications of HHs is also given in that reference, which is really useful for practitioners. Different web sites maintain updated bibliography on HHs [52, 53], where the last published references on HHs can be found.

4 A hyper-heuristic approach for the Jawbreaker puzzle

Following the process stated above to construct a HH for the Jawbreaker puzzle, we first set the pool of low-level heuristics to this puzzle. We have chosen 19 low-level heuristics

[1] Note that in this paper we consider the first type of HHs approach, i.e., HHs for heuristic selection, since we try to optimize the sequence of basic (existing) heuristic that produces the best solution the Jawbreaker.

[2] In this paper we consider, of course, online learning to solve the Jawbreaker puzzle.

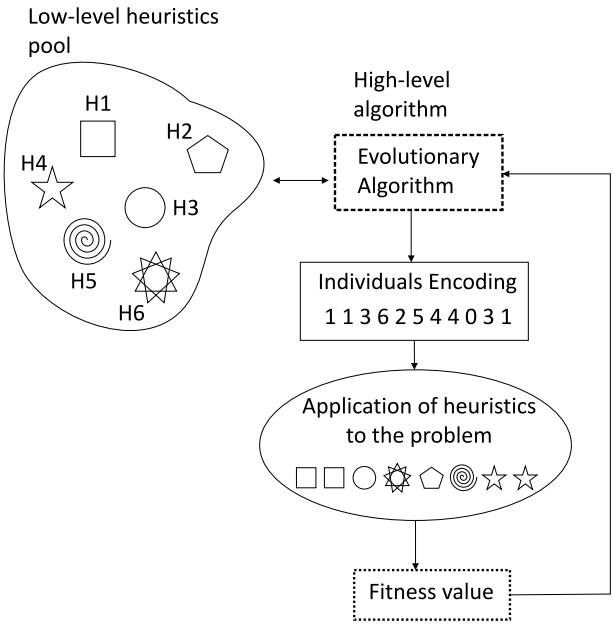


Fig. 3 Schema of the construction of a HH algorithm

which modify the matrix Ξ in different ways. These basic heuristics have been chosen by considering the typical actions that an expert human player would do during a game. Following the notation given in Sect. 2, we denote the matrix of the game at a given point as Ξ, and when needed, we denote Ξ' to the resulting matrix after a group has been cleared.

- H1: Clears the smallest set ($n \leq 2$) in Ξ.
- H2: Clears the largest set ($n \leq 2$) in Ξ.
- H3: Clears the largest set ($n \leq 2$) in Ξ, only if clearing any other group makes that all the sets remaining in Ξ' are smaller than the largest in Ξ. This heuristic has no effect in case that the condition for the clearing is not fulfilled.
- H4: Clears the first set in form of column, found from left to right and from the top to the bottom of the matrix Ξ. The heuristic has no effect if there is not a group with this characteristics.
- H5: Same as H4, but scans Ξ from left to right and from the bottom to the top.
- H6: Same as H4, but scans Ξ from right to left and from the top to the bottom.
- H7: Same as H4, but scans Ξ from right to left and from the bottom to the top.
- H8: Clears the first set in form of row, found from left to right and from the top to the bottom of the matrix Ξ. The heuristic has no effect if there is not a group with this characteristics.
- H9: Same as H8, but scans Ξ from left to right and from the bottom to the top.
- H10: Same as H8, but scans Ξ from right to left and from the top to the bottom.
- H11: Same as H8, but scans Ξ from right to left and from the bottom to the top.
- H12: Clears the set in Ξ which maximizes the remaining points (sum of the points provided by all the remaining sets) in the resulting Ξ'.
- H13: Clears the set such that minimizes the number of different sets in Ξ'.
- H14: This heuristic searches for the smallest size among the groups in Ξ that fulfill $n \leq 2$. Let us call it n^*. Then, it clears the group of size n^* that generates a matrix Ξ' with less sets of that size. The heuristic is not applied if this condition is not fulfilled.
- H15: Similar to H14, but it clears the set of size n^* that generates a matrix Ξ' with less or equal sets of size n^*.
- H16: Clears a set ($n \leq 2$) from the less abundant color in Ξ.
- H17: Clears a set in Ξ if in the resulting matrix Ξ' appears a set larger than the largest set in Ξ. This heuristic has no effect in the case that the condition is not fulfilled.
- H18: Clears the largest set in Ξ, if clearing any other set does not produce a larger group in Ξ'.
- H19: Clears a set if it contains all the elements of the same color remaining in Ξ.

We consider an evolutionary algorithm as high-level approach to evolve the sequential application of the above-defined heuristics. We describe next the proposed encoding and evolutionary operators that define the high-level search.

4.1 Evolutionary encoding and fitness function

Each individual in the evolutionary algorithm encodes a given application of the heuristics defined above to the objective matrix Ξ. An intuitive encoding consists of a vector of integer numbers $\mathbf{x} \in \{0, 1, 2, \ldots, 19\}^k$. We use fixed-length vectors (length k), and the stop symbol (0) is used to set the maximum number of heuristics used in an individual (not all heuristics must be included in a solution). Thus, the application of the heuristics to the game is applied in a loop fashion, from the leftmost element in the vector, to the first stop-symbol in the individual, until the game is finished (no more sets of two or more balls remain in Ξ). As an example, consider the jawbreaker puzzle given by matrix Ξ of Fig. 1. Let us set $k = 20$, then, an example of individual for this problem is the following:

$$\mathbf{x} = 19\ 17\ 18\ 16\ 12\ 12\ 0\ 8\ 2\ 19\ 12\ 17\ 6\ 19\ 2\ 15\ 15\ 12\ 13\ 8$$

It is important to note that only 5 low-level heuristics will be used to generate a solution for the puzzle in this individual (heuristics 12, 16, 17, 18 and 19). The rest of the solution is not considered in this case, but it may be important in crossover and mutation operators to generate new (good-quality) individuals. Note that a constraint on the position of

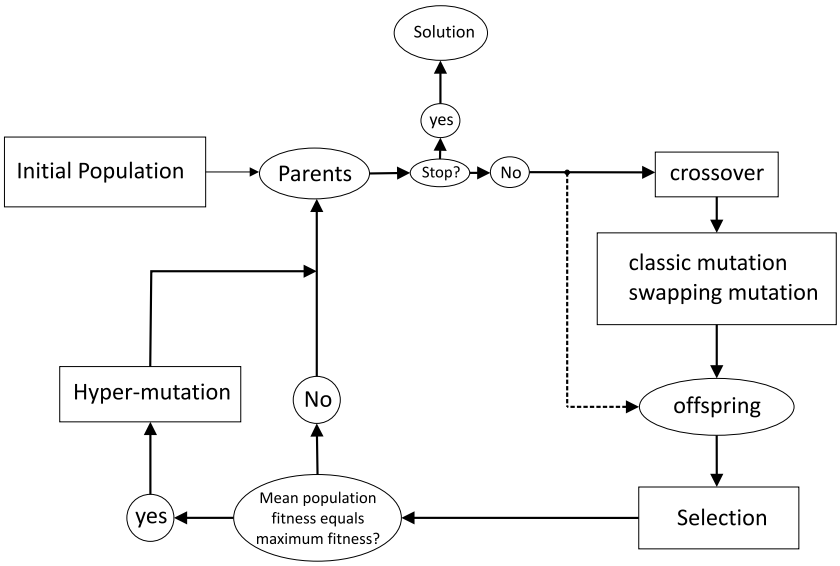


Fig. 4 Outline of the evolutionary algorithm considered as high-level global search procedure in the proposed HH

the stopping symbol must be fulfilled by all individuals in the population: stopping symbol (0) cannot be positioned in the first element of the string, in order to avoid individuals that do not produce any solution.

Regarding this stopping symbol, note that in the early stages of the algorithm's development, we conceived it as a variable length GA. The main problem with this is that we must adapt the evolutionary operators, whereas with the fixed-length encoding with stopping symbol, the inclusion of standard crossover and mutation is much easier. We did not obtain a significant improvement with the variable length encoding in some preliminary test carried out, so we finally focused on the fixed-length encoding with stopping symbol as the encoding proposed.

Regarding the fitness function, the sum of the scores obtained when clearing all the sets during the game is used, i.e., we use the following expression:

$$f(\mathbf{x}) = \sum_{i=1}^{\xi} y_i \tag{2}$$

where ξ stands for the total number of sets cleared in a given game (different number for every matrix Ξ), and y_i stands for the score obtained after clearing each set in Ξ, according to (1).

4.2 Evolution and operators

The evolution process proposed is shown in Fig. 4. Basically, after the initialization of a population of μ individuals at random, the different operators are applied in a loop fashion: first, an offspring population of the same size that the initial one is generated by means of the crossover operator, and two mutation mechanisms are then applied to generate

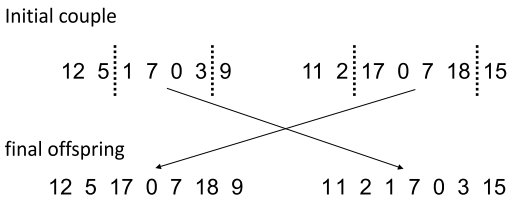
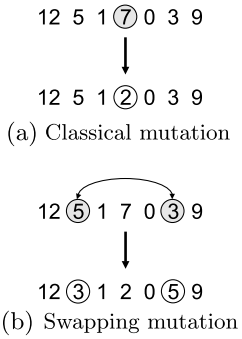


Fig. 5 Example of the crossover operator used in the HH

diversity. A tournament selection operator [54] selects the μ parents of the next generation among the complete population of parents and offspring of the current one, and the process continues until a maximum number of generations is completed.

A classical two-points crossover operator is used in this paper. First, we form couples of parents at random, and then the operator is applied as shown in Fig. 5, to form two new offspring individuals. As mentioned, we apply two different mutations for adding diversity to the offspring: First, a classical mutation that works by randomly selecting elements of individuals and substituting them by values in $\{0, 1, 2, \ldots, 19\}$ is applied (Fig. 6(a)). We consider a second mutation operator that works by swapping elements of a given individual, as shown in Fig. 6(b). Both mutations are applied with a low probability of 1 % to the individuals of the offspring population.

In order to improve the performance of the algorithm in the Jawbreaker puzzle, it is important to include a mechanism to increment the diversity in the population. Therefore, we have implemented an operator that restarts part of the population when its diversity is very low. This technique has been previously applied in the context of game solutions, specifically in solving the Mastermind game [12]. The name given to this operator is *hyper-mutation*, and it is implemented by keeping the best 3 individuals in current pop-



(a) Classical mutation

(b) Swapping mutation

Fig. 6 Example of the mutations operators applied

ulation, and re-initialize the rest, some of then completely at random, and others by mutations of the 3 kept individuals.

4.3 Evolutionary algorithm parameters

There are different parameters to be set in the evolutionary algorithm, basically the encoding length (k), population size (μ) and number of generations (g_{max}). The probability of mutation is set to 0.01 (1 % of the individuals in the offspring are mutated), whereas the crossover are used to generate the offspring, so no probability is associated to crossover in this case (this operator is always applied to randomly selected couples until the offspring population of size μ is completed). Parameter k is set in such a way that, at least, all the considered low-level heuristics can have the possibility of appearing in a solution. Thus, $k = 20$ ensures this, also including the stopping symbol (0). This value is enough for obtaining good results in the usual Jawbreaker puzzles sizes considered. Regarding the population size of the algorithm μ, we set a value depending on the matrix Ξ size, and the number of colors in the puzzle (α). The formula, obtained by experimentation in many different Jawbreaker puzzles, is the following:

$$\mu = \lceil N + M + 2\alpha + 20 \rceil \tag{3}$$

where N stands for the number of rows of Ξ, M stands for the number of columns of Ξ and α stands for the number of colors of the puzzle.

We have also obtained a formula to set the number of generations of the evolutionary algorithm, also in an experimental way, as follows:

$$\mu = \lceil 10N + 10M + 20\alpha \rceil \tag{4}$$

and we have checked out that the results obtained are good with this number of generations, that depends on the current puzzle (matrix Ξ) being solved.

Note that we have obtained these formulas by experimentation. These formulas provide a number of generations large enough to obtain good results in all the instances considered in the experimental part of the paper. In order to obtain the formulas, we have carried out some prior test for relating the number of generations to the puzzle size, and we have found that the proposed formulas worked quite well. The same applies with the population size, in which we have tried to adapt it to the puzzle size.

Finally, we have launched the hyper-mutation operator when the diversity is null in the population, i.e. we launch the hyper-mutation if the mean fitness of the population is equal to the best fitness in it.

5 Experimental part

In order to show the performance of our HH approach for the Jawbreaker puzzle, we have carried out a set of experiments, consisting in solving a number of Jawbreaker instances, of different size and difficulty. Eighteen instances have been randomly generated, including small, medium and large Jawbreaker puzzles (from 5×5 instances to 10×10, with 3, 4 and 5 colors). Thirty runs of the proposed HH with and without hyper-mutation operator have been run, and the results in the Jawbreaker instances considered are shown in Table 1. We have collected in this table the best solution found, the mean of the 30 runs, standard deviation and the average computation time taken. It is easy to see how the HH proposed is a good approach to solve the Jawbreaker, obtaining excellent best and average scores in all the puzzles taken. Note also that the hyper-mutation operator improves the performance of the HH, and its convergence to the best score solution. In the largest instances, the differences between the HH with and without hyper-mutation are more accused. Also, the convergence to the best score found is less frequent than in small and medium size instances. It is interesting to check out that, given a matrix Ξ size ($M \times N$), the score obtained is larger for a smaller number of colors considered in the game. The reason is simple: the size of the sets of balls that can be constructed in this case is larger than considering more colors in Ξ.

Though the Jawbreaker puzzle does not include a time restriction, we have computed the time taken by the proposed HH in solving all the instances considered, just to give an idea of its performance in this aspect. Table 1 includes computation time for the HH with and without hyper-mutation. Note that it varies between 30 seconds in the smallest instances, to 15 minutes in the largest instances, with very few differences between the two versions of the algorithm compared. It is intuitive that the differences in computation time appear depending on the size of the matrix Ξ, and the number of colors considered, since the population size and number of generations depends on these parameters ((3) and (4), respectively).

In order to complete the analysis of the HH in the Jawbreaker puzzle, we discuss a complete solution given to the instance 7×7, with 5 colors. Figure 1, is the initial matrix Ξ

Table 1 Results obtained with the proposed hyper-heuristic (over 30 runs) in Jawbreaker puzzle instances of different sizes, and with different number of colors (α) involved

Instance	HH, EV with hyper-mutation				HH, EV without hyper-mutation			
	Max	Mean	Std.	Time (s)	Max	Mean	Std.	Time (s)
5 × 5 (3)	112	112.0	0.0	33.4	112	112.00	0.0	33.1
5 × 5 (4)	108	108.0	0.0	25.2	108	105.96	2.87	25.7
5 × 5 (5)	98	98.0	0.0	44.9	98	98.00	0.0	44.3
6 × 6 (3)	310	310.0	0.0	43.3	310	308.4	7.41	43.4
6 × 6 (4)	186	186.0	0.0	65.0	186	185.4	2.40	64.2
6 × 6 (5)	120	112.48	2.93	85.98	112	108.36	3.78	85.40
7 × 7 (3)	650	650.0	0.0	98.0	650	644.72	7.11	96.3
7 × 7 (4)	256	256.0	0.0	132.3	256	253.40	2.22	130.3
7 × 7 (5)	280	280.0	0.0	154.7	280	274.92	6.83	152.7
8 × 8 (3)	942	942.0	0.0	186.0	942	938.0	10.87	182.59
8 × 8 (4)	690	690.0	0.0	175.59	690	687.08	8.01	174.80
8 × 8 (5)	438	436.88	1.94	346.5	438	431.96	5.49	342.4
9 × 9 (3)	1522	1522.0	0.0	177.4	1522	1492.96	37.71	177.0
9 × 9 (4)	974	967.6	27.71	445.1	974	951.1	20.55	489.74
9 × 9 (5)	454	410.9	21.9	709.4	454	396.36	22.82	705.61
10 × 10 (3)	2032	2005.4	19.32	344.3	2032	1913.08	59.93	342.2
10 × 10 (4)	1186	1143.92	21.40	755.2	1186	1125.16	39.54	752.36
10 × 10 (5)	1066	1030.52	13.13	1303.3	1066	1008.60	28.32	1298.1

of the puzzle. The proposed HH has provided the following best solution:

9 15 12 7 12 0 7 5 3 15 9 6 8 2 12 2 12 7 13 17

This solution solves the puzzle by the sequential application of just 4 low-level heuristics (7, 9, 12 and 15), in the order given by the solution, and with the repetition of heuristic 12 in the sequence. Figure 7 shows the complete sequence of each heuristics's application to solve the puzzle (only moves where score are shown). It is interesting to note how the HH applies low-level heuristics which tend to group large sets of balls during the game (blue ones and then pink ones in this case), which is the natural winner strategy in this game.

5.1 Experiments in the SameGame problem

In order to further analyze the performance of the proposed hyper-heuristic in the Jawbreaker puzzle, we refer to a specific case of the puzzle, called SameGame. SameGame is a Jawbreaker puzzle played in a 15 × 15 board, using balls of 5 different colors in the matrix Ξ. The scoring formula in the SameGame (number of points when a set of balls is cleared) changes in respect to the standard Jawbreaker. It is the following in this case:

$$y = (n-2)^2 \tag{5}$$

where y is the score obtained for clearing a set of similarly-colored balls, and n is the number of balls in the set.

We compare our HH approach against three alternative methods to solve the SameGame, the Depth-Budgeted Search (DBS) [56], the Single-Player Monte-Carlo tree search (SP-MCTS) described in [24] and the Monte-Carlo with Roulette-Wheel Selection algorithm (MC-RWS), described in [57]. Standard SameGame instances can be freely downloaded from [55]. They consist of 20 standardized SameGame instances in such a way that the results of different algorithms can be compared.

Table 2 shows the results obtained in the SameGame by applying the HH with hyper-mutation in the EV, and the alternative algorithms results in these standardized SameGame instances. In the case of the HH, the best solution after 10 runs of the algorithm is shown. The obtained results show that the proposed HH performs quite well in these instances, outperforming DBS and SP-MCTS approaches. It also improves in 13 out of 20 times the results obtained by the MC-RWS algorithm. In the instances where the HH does not improve the MC-RWS, its results are quite close to the ones obtained by that approach. These experiments show that the proposed HH is a competitive technique able to compete with existing algorithms in solving the Jawbreaking puzzle.

6 Conclusions and future lines of research

This paper presents a hyper-heuristic approach to the Jawbreaker puzzle. In the paper, after introducing the Jawbreaker puzzle, we have detailed the structure, low-level

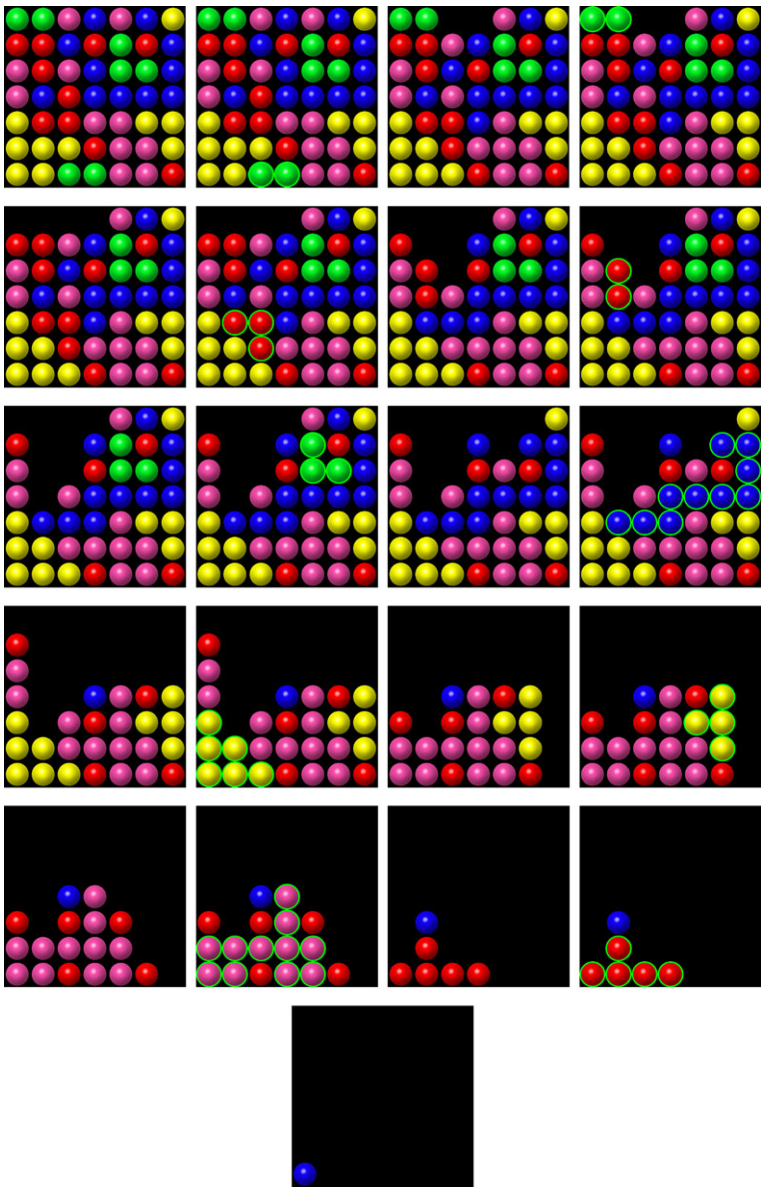


Fig. 7 Complete solution of the 7×7 (5) Jawbreaker puzzle, by sequentially applying the low-level heuristics: 9 15 12 7 12. The fist image is the initial matrix Ξ of the puzzle

heuristics and global search approach that forms the proposed hyper-heuristic. We have then shown the performance of the proposed hyper-heuristic approach in puzzles of different sizes and difficulty, obtaining excellent results in all of them. An analysis including and not including a specific operator of hyper-mutation has also been carried out. The performance of the proposed approach has been also evaluated in several standardized instances of a version of the Jawbreaker (the SameGame), where we successfully compare the HH with alternative existing techniques for this puzzle.

The presented work can be applied in education, where puzzles are often used to illustrate different algorithms or computational paradigms. In this case, the Jawbreaker puzzle is appealing to teach hyper-heuristics at university level, where there are not specific courses on this matter.

The application of hyper-heuristics to solving puzzles is a broad field to be explored. We plan to extend the same hyper-heuristic structure presented in this paper to different, related puzzles, which can be used to evaluate algorithms and for educational purposes. Another line of possible research in this and alternative problems, is to explore the possibility of having parallel populations of hyper-heuristics, evolving different sets of basic heuristics each. The combination of the different populations, and the relationship between them, could improve the performance of hyper-heuristics in the problem.

Acknowledgements This work has been partially supported by Spanish Ministry of Science and Innovation, under project number ECO2010-22065-C03-02.

Table 2 Results obtained in standardized sets of SameGame (specific case of Jawbreaker puzzle)

Instance	HH	DBS	SP-MCTS	MC-RWS
1	2921	2061	2557	2633
2	3801	3513	3749	3755
3	3101	3151	3085	3167
4	3795	3653	3641	3795
5	3969	3093	3653	3943
6	4243	4101	3971	4179
7	2963	2507	2797	2971
8	3923	3819	3715	3935
9	4729	4649	4603	4707
10	3259	3199	3213	3239
11	3321	2911	3047	3327
12	3257	2979	3131	3281
13	3395	3209	3097	3379
14	2793	2685	2859	2697
15	3427	3259	3183	3399
16	4925	4765	4879	4935
17	4901	4447	4609	4737
18	5161	5099	4853	5133
19	4933	4865	4503	4903
20	4863	4851	4853	4649

References

1. Hartmann D, van den Herik HJ, Iida H (eds) (2000) Games in AI research. ICGA J (special issue) 23(2)
2. Laird JE (2001) Using a computer game to develop advanced AI. Computer 34(7):70–75
3. Khoo A, Zubek R (2002) Applying inexpensive AI techniques to computer games. IEEE Intell Syst 17(4):48–53
4. Wallace SA, McCartney R, Russell I (2010) Games and machine learning: a powerful combination in an artificial intelligence course. Comput Sci Educ 20(1):17–36
5. Joyner D (2002) Adventures in group theory: Rubik's cube, Merlin's machine, and other mathematical toys. Johns Hopkins Press, Baltimore
6. Kunkle D, Cooperman G (2009) Harnessing parallel disks to solve Rubik's cube. J Symb Comput 44(7):872–890
7. Ryabogin D (2012) On the continual Rubik's cube. Adv Math 231(6):3429–3444
8. Kendall G, Parkes A, Spoerer K (2008) A survey of NP-complete puzzles. ICGA J 31(1):13–34
9. Mantere T, Koljonen J (2007) Solving, rating and generating Sudoku puzzles with GA. In: Proc of the IEEE congress on evolutionary computation, pp 1382–1389
10. Hereford JM, Gerlach H (2008) Integer-valued particle swarm optimization applied to Sudoku puzzles. In: Proc of the IEEE swarm intelligence symposium, pp 1–7
11. Berghman L, Goossens D, Leus R (2009) Efficient solutions for MasterMind using genetic algorithms. Comput Oper Res 36(6):1880–1885
12. Merelo-Guervós JJ, Castillo P, Rivas V (2006) Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary MasterMind. Appl Soft Comput 6(2):170–179
13. Chen KH (2000) Some practical techniques for global search in go. ICGA J 23(2):67–74
14. Drake P (2009) The last-good-reply policy for Monte-Carlo go. ICGA J 32(4):221–227
15. Tsai JT (2012) Solving Japanese nonograms by Taguchi-based genetic algorithm. Appl Intell 37(3):405–419
16. Batenburg KJ, Kosters WA (2009) Solving nonograms by combining relaxations. Pattern Recognit 42(8):1672–1683
17. Jefferson C, Miguel A, Miguel I, Armagan-Tarim S (2006) Modelling and solving English peg solitaire. Comput Oper Res 33(10):2935–2959
18. Gindre F, Trejo Pizzo DA, Barrera G, Lopez De Luise MD (2010) A criterion-based genetic algorithm solution to the Jigsaw puzzle NP-complete problem. In: Proc of the world congress on engineering and computer science, pp 367–372
19. van Eck NJ, van Wezel M (2008) Application of reinforcement learning to the game of Othello. Comput Oper Res 35(6):1999–2017
20. Lucas SS, Kendall G (2006) Evolutionary computation and games. IEEE Comput Intell Mag 1(1):10–18
21. Salcedo-Sanz S, Portilla-Figueras J, Bellido AP, Ortiz-García E, Yao X (2007) Teaching advanced features of evolutionary algorithms using Japanese puzzles. IEEE Trans Ed 50(2):151–155
22. Tsai JT, Chou PY, Fang JC (2012) Learning intelligent genetic algorithms using Japanese nonograms. IEEE Trans Ed 55(2):164–168
23. Pocket PC Jawbreaker Game, The ultimate guide to PDA games. http://www.pdagameguide.com/jawbreaker-game.html
24. Schadd MP, Winands MH, van den Herik HJ, Chaslot GM, Uiterwijk JW (2008) Single-player Monte-Carlo tree search. In: Proc of the 6th international conference on computers and games, pp 24–26
25. Schadd MP, Winands MH, van den Herik HJ, Chaslot GM, Uiterwijk JW (2012) Single-player Monte-Carlo tree search for SameGame. Knowl-Based Syst 34:3–11
26. Burke EK, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic, Norwell, pp 457–474
27. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc., in press
28. Han L, Cowling PI, Kendall G (2002) An investigation of a hyper-heuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of congress on evolutionary computation (CEC2002), pp 1185–1190
29. Sabar NR, Ayob M, Qu R, Kendall G (2011) A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence
30. Soghier A, Qu R (2013) Adaptive selection of heuristics for assigning time slots and rooms in exam timetables. Applied Intelligence, in press
31. Abuhamdah A, Ayob M, Kendall G, Sabar NR (2013) Population based local search for university course timetabling problems. Appl. Intell. (in press)
32. Hunt R, Neshatian K, Zhang M (2012) A genetic programming approach to hyper-heuristic feature selection. In: Proc of the 9th international conference on simulated evolution and learning (SEAL12). LNCS, vol 7673. Hanoi, Vietnam
33. Shafi K, Bender A, Abbass HA (2012) Multi-objective learning classifier systems based hyperheuristics for modularised fleet mix problem. In: Proc of the 9th international conference on simulated evolution and learning (SEAL12). LNCS, vol 7673/2012. Hanoi, Vietnam
34. Wauters T, Vancrooenburg W, Vanden Berghe G (2010) A two phase hyper-heuristic approach for solving the Eternity II puzzle. In: Proc of the 2nd international conference on metaheuristics and nature inspired computing (META10), Djerba Island, Tunisia

35. Wauters T, Vancrooenburg W, Vanden Berghe G (2012) A guide-and-observe hyper-heuristic approach to the Eternity II puzzle. Journal of Mathematical Modelling and Algorithms 11(3)
36. Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyper-heuristic for timetabling and rostering. J Heuristics 9(6):451–470
37. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyperheuristic for educational timetabling problems. Eur J Oper Res 176:177–192
38. Cowling P, Kendall G, Soubeiga E (2001) A parameter-free hyper-heuristic for scheduling a sales summit. In: Proc of the 4th meta-heuristic international conference, pp 127–131
39. Cowling P, Kendall G, Soubeiga E (2002) Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Proc of EvoWorkshops 2002. Lecture notes in computer science, vol 2279, pp 1–10
40. Burke EK, Hyde MR, Kendall G, Woodward J (2010) A genetic programming hyperheuristic approach for evolving two dimensional strip packing heuristics. IEEE Trans Evol Comput 14(6):942–958
41. Burke EK, Hyde MR, Kendall G, Woodward J (2007) The scalability of evolved on line bin packing heuristics. In: Proc of the IEEE congress on evolutionary computation, pp 2530–2537
42. Bai R, Kendall G (2008) A model for fresh produce shelf-space allocation and inventory management with freshness-condition-dependent demand. INFORMS J Comput 20(1):78–85
43. Bai R, Burke EK, Kendall G (2008) Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. J Oper Res Soc 59:1387–1397
44. Remde S, Cowling P, Dahal K, Colledge N, Selensky E (2011) An empirical study of hyperheuristics for managing very large sets of low level heuristics. J Oper Res Soc 63(3):392–405
45. Kendall G, Mohamad M (2004) Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proc of the IEEE international conference on network, pp 769–773
46. Kendall G, Mohamad M (2004) Channel assignment optimisation using a hyper-heuristic. In: Proc of the IEEE conference on cybernetic and intelligent systems, pp 790–795
47. Li J, Burke EK, Qu R (2011) Integrating neural networks and logistic regression to underpin hyper-heuristic search. Knowl-Based Syst 24(2):322–330
48. Furtuna R, Curteanu S, Leon F (2012) Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic. Appl Soft Comput 12(1):133–144
49. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward J (2009) A classification of hyper-heuristics approaches. In: Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics. International series in operations research and management science. Springer, Berlin
50. Ozcan E, Bilgin B, Korkmaz EE (2008) A comprehensive analysis of hyper-heuristics. Intell Data Anal 12(1):3–23
51. Ross P (2005) In: Burke EK, Kendall G (eds) Hyper-heuristics, search methodologies: introductory tutorials in optimization and decision support techniques. Springer, Berlin, pp 529–556
52. http://allserv.kahosl.be/~mustafa.misir/hh.html
53. http://www.hyper-heuristic.org
54. Eiben AE, Smith JE (2003) Introduction to evolutionary computing, 1st edn. Natural computing series. Springer, Berlin
55. http://www.js-games.de/eng/games/samegame
56. Billings D (2007) Personal communication. University of Alberta, Canada
57. Takes FW, Kosters WA (2009) Solving SameGame and its chessboard variants. In: Proc of the 21st Benelux conference on artificial intelligence, Eindhoven, The Netherlands, pp 249–256

S. Salcedo-Sanz was born in Madrid, Spain, in 1974. He received the B.S. degree in Physics from the Universidad Complutense de Madrid, Spain, in 1998, and the Ph.D. degree in Telecommunications Engineering from the Universidad Carlos III de Madrid, Spain, in 2002. He spent one year in the School of Computer Science, The University of Birmingham, U.K, as postdoctoral Research Fellow. Currently, he is an associate professor at the department of Signal Processing and Communications, Universidad de Alcalá, Spain. He has co-authored more than 170 international journal and conference papers in the field of machine learning and soft-computing. His current interests deal with Soft-computing techniques, hybrid algorithms and neural networks in different applications of Science and Technology.

J.M. Matías-Román was born in Madrid in 1982. He received the B.S. degree in Telecommunication Engineering from Universidad de Alcalá, Madrid, Spain, in 2012, where he was a research fellow in Gheode Research group. He is currently involved in developing specialized software for mobile technologies.

S. Jiménez-Fernández was born in Madrid, Spain, in 1976. She received the B.S. degree (2000), and the Ph.D. degree (2009) in Telecommunications Engineering from Universidad Politécnica de Madrid, Spain. She is currently an Associate Professor at the department of Signal Theory and Communications, Universidad de Alcalá, Spain. She has co-authored more than 20 conference and journal papers in different areas of engineering. Her current interests are related to Soft-computing approaches and its applications.

A. Portilla-Figueras was born in Santander, Spain, in 1976. He received the B.S. degree (1999), and the Ph.D. degree (2004) in Telecommunications Engineering from Universidad de Cantabria, Spain. He is currently an Associate Professor at the department of Signal Theory and Communications, Universidad de Alcalá, Spain. He has coauthored more than 70 conference and journal papers in the area of softcomputing. His current interests are related to Soft-computing approaches to network design and topology discovering, telecommunications and energy applications.

L. Cuadra was born in Madrid, Spain, in 1971. He received the B.Sc. degree in Telecommunication Engineering, the M.Sc. degree in Physics of Semiconductor Devices, the M.A.S. degree in Information and Communication Technology, and the Ph.D. degree (summa cum laude) from the Polytechnic University of Madrid (Universidad Politécnica de Madrid, UPM). He spent 6 months at the Faculty of Engineering, The University of Glasgow, U.K, as doctoral Research Fellow. Dr. Cuadra received the award for the Best Doctoral Thesis from UPM, and the award for the Best Doctoral Thesis in Fundamental Technologies from the Spanish Telecommunication Engineers Association (COIT). He is currently Associate Professor at the University of Alcalá, Spain. He has published more than 80 international papers and conference contributions. His research interests include Soft Computing applied to problems in Science and Engineering and, in particular, those related to energy, the limits of the photovoltaic conversion and the physics of electronic and optical devices.