An Educational Software Tool to Teach Hyper-Heuristics to Engineering Students Based on the Bubble Breaker Puzzle

S. SALCEDO-SANZ, S. JIMÉNEZ-FERNÁNDEZ, J. M. MATÍAS-ROMÁN, J. A. PORTILLA-FIGUERAS

Department of Signal Processing, Communications, Universidad de Alcalá. Escuela Politécnica Superior, 28871, Alcalá de Henares, Madrid, Spain

Received 26 September 2013; accepted 24 December 2013

ABSTRACT: This paper presents an educational software tool to teach Artificial Intelligence (AI) techniques, specifically Hyper-heuristics, to Engineering students. This tool is based on the "Bubble Breaker" puzzle, an addictive game consisting in an M × M matrix of colored bubbles. These balls, when forming sets of two or more same colored balls, can be popped and cleared out. Thus, this puzzle can be solved by setting many different low-level heuristics and applying a global search procedure (i.e., evolutionary algorithm) that conforms a robust hyper-heuristic technique. The hyper-heuristic decides what low-level heuristics are the best, and the sequential way in which they have to be applied to gain the highest score. This approach has proven an interesting method to teach AI techniques, since simple heuristics, evolutionary algorithms, and its combination are studied in an increasing manner. © 2014 Wiley Periodicals, Inc. Comput Appl Eng Educ 23:277–285, 2015; View this article online at wileyonlinelibrary.com/journal/cae; DOI 10.1002/cae.21597

Keywords: computer science education; engineering education; software tools; hyper-heuristics; evolutionary algorithms; introduction

INTRODUCTION

Puzzles and computer games have been long considered in different fields of expertise, such as Artificial Intelligence (AI), Computer Science, or Mathematics, as they constitute difficult problems to test the performance of different algorithms or approaches [1–7]. Kendall et al. [8] performed a survey on different NP-complete puzzles that have been tackled with different approaches related to Soft-Computing, mainly heuristics and meta-heuristics. Moura et al. [9] described a teaching experience used, although not related to puzzles, to introduce a Soft-Computing technique: Particle Swarm Optimization. Other Computational Intelligence or Soft-Computing techniques have also been tested with different puzzles: Sudoku [10], Go [11], Nonograms [12], Mastermind [13], or different Solitaire games [14–16].

In other works, puzzles are used to teach these branches of knowledge in an appealing way [5,17], and show interesting contributions as well in the field of Education [18–20].

In this paper we discuss the use of a puzzle to teach a Computational Intelligence technique: the hyper-heuristics algorithms (HHs). Although the origin of hyper-heuristics goes back to the 1960s, it was in the first years of 2000 when the foundations of these methodologies were set, and they revealed as powerful approaches to solve optimization problems [21]. Hyper-heuristics are a new class of searching methodologies [21,22]. The underlying idea is to develop algorithms that are more generally applicable than many of the current implementations of search methodologies. The main goal is to design generic methods based on a set of easy-to-implement low-level heuristics. It is not necessary that the heuristics on their own are very effective, although it is interesting that they differ from each other, and their number is large enough to generate a large search space [23]. The HH is the high-level methodology that automatically produces an adequate combination of the low-level heuristics to effectively

Contract grant sponsor: Universidad de Alcalá; Contract grant number: UAH/EV523

Correspondence to S. Salcedo-Sanz (E-mail: sancho.salcedo@uah.es).

solve the given problem. HHs can be used to create new heuristics from existing ones. In this case, they are called HHs for heuristics generation.

HHs are currently considered state of the art algorithms in Computational Intelligence, and have been successfully applied to different problems as channel assignment in telecommunication networks [24,25], scheduling [26,27], timetabling [28,29], data mining [30,31], etc. In spite of this, they have deserved little attention in educational journals, and there are very few publications on the educational methods for teaching HHs, only the contributions in [32,33], where a mixture of several simple heuristics are optimized by special methods, and the educational aspects of this approach are presented.

In this paper a method to teach hyper-heuristics is proposed. A simple and famous game is used: the Bubble breaker game. A case of application of the proposed teaching method in a Spanish University is presented and discussed in the paper.

The rest of the paper is structured as follows. Next section gives a description of the Bubble breaker puzzle and its rules, together with the state of the art in algorithms applied to solve it. Proposed Hyper-Heuristic Approach for the Bubble Breaker Puzzle section presents a brief introduction to Hyper-heuristics, and the proposed approach to solve the puzzle under study, and Software Tool section shows the software tool developed to teach the hyper-heuristics. Teaching Method and Assessment section explains the proposed teaching method and its application in a Spanish University. Finally, Conclusions section closes the paper giving some conclusions to this work.

THE BUBBLE BREAKER PUZZLE

Bubble breaker puzzle (also known as Jawbreaker, SameGame, or Bubblets) is a one-player game that initially presents an $M \times M$ matrix (the game board) filled with different-colored bubbles (the total number of colors in the puzzle is called α). The bubbles must be sequentially cleared off by clicking on groups of two or more orthogonally adjacent same-colored bubble' blocks (Fig. 1a). The bubbles on top of the removed group fall down over the existing ones (Fig. 1b). In the standard mode, no new bubbles are added as others are cleared off. If all the bubbles in one column, i, are removed, the columns to the right ($i+1$ to M) are shifted one position to the left to prevent holes (before the move: Fig. 1c; after the move: Fig. 1d). That is, the new matrix has $M-1$ columns. The game ends when the player has no more sets of same-colored bubbles to clear off.

In the Bubble breaker, the larger the cleared groups are, the higher the score assigned. This score depends on the number of same-colored bubbles that form the cleared group, and it can be expressed as in Equation (1).

$$\text{Score} = X(X-1) \tag{1}$$

where X stands for the number of bubbles in the removed set.

Moreover, the more different-colored bubbles and the higher the square gets, the more difficult the game is.

Bubble breaker is an NP-complete optimization problem [34,35]. The difficulty to tackle this puzzle with algorithms is that the board continuously changes, as groups of bubbles are cleared. For this reason, it has been not until recently, that a work solving this puzzle has been published [35]. Schadd et al. propose the use of a variant of the Monte-Carlo Tree Search (MCTS), the Single-Player MCTS (SP-MCTS). SP-MCTS builds a search tree employing Monte-Carlo evaluations at the leaf nodes, where each node at the tree represents a board position and stores the average score found in the corresponding subtree and the number of visits. Several modifications are added to the selection, simulation, expansion, and back-propagation strategies used in MCTS to deal with the fact that Bubble breaker is a single-player game. In Ref. [36] an alternative Monte-Carlo tree search algorithm was proposed for the bubble breaker and related problems. Other classical heuristic approaches such as A* or IDA* are not suitable for the Bubble breaker puzzle, as shown also in Ref. [35], due to the fact that it is not easy to obtain an admissible evaluation function to provide accurate estimation.

To our knowledge, there are no other works dealing with pure meta-heuristic approaches to solve the Bubble breaker. However, hyper-heuristics algorithms perfectly fit to the Bubble breaker as it has been previously proven in Ref. [37], since they try to optimize a sequence of existing heuristics that produces the best possible solution to the puzzle.

PROPOSED HYPER-HEURISTIC APPROACH FOR THE BUBBLE BREAKER PUZZLE

The process to optimize the sequence of existing heuristics (HH for heuristics selection) has two important sub-processes. First, a set of low-level (basic) heuristics has to be defined following the considerations made in previous sections. Second, a high-level global search has to be selected. The aim of this global algorithm (whether it is an heuristic or not, and no matter the nature of the algorithm chosen) is to obtain the best set of low-level heuristics to apply and the best sequence in which they should be considered. In many cases, the high-level algorithm is a meta-heuristic such as an evolutionary algorithm, a particle swarm optimization algorithm, an ants algorithm, etc. The outline of a HH approach, similar to the problem considered in this paper, is shown in Figure 2.

The HH described in this paper is based on an evolutionary algorithm (EA) that implements the high-level global search, and a pool of 19 low-level heuristics that form the search space. This basic heuristics represent moves a human player would play during a game. To define them, let us denote as Θ the $M \times M$ matrix containing the different-colored bubbles, and Θ' the resulting matrix after a given group of bubbles is cleared (once a low-level heuristic is applied).

The evolutionary algorithm will later determine the sequence of low-level heuristics to be applied.

Low-Level Heuristics

We propose an integer encoding, numbering the basic heuristics from 1 to 19. Note that it is possible to find different-sized solutions (each with a different number of heuristics to be applied), so a stop criterion has been defined and encoded. In this work we use an extra symbol, and encode it as the low-level heuristic 0, to serve this purpose.

Following the given notation, the low-level heuristics defined are:

H_{01}: Clears the smallest set ($n \geq 2$) in Θ.

H_{02}: Clears the largest set ($n \geq 2$) in Θ.

H_{03}: Clears the first column-shape set (found from left to right, and from top to bottom) in Θ. This heuristic has no effect if there is no group that matches this characteristic.

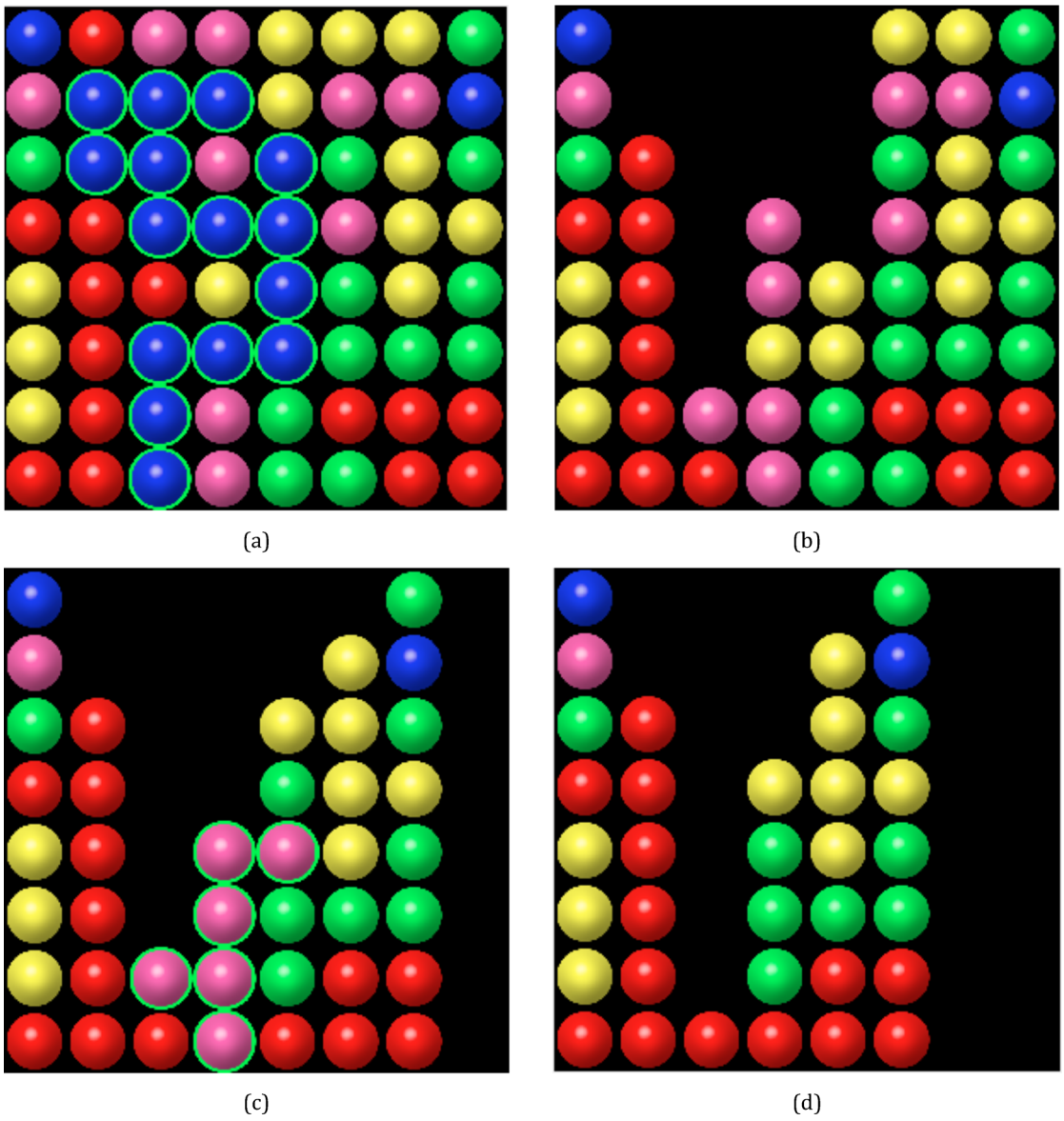

Figure 1 Example of a Bubble breaker puzzle (matrix 8 × 8, five colors) (a) Group selected to be cleared (b) Matrix after the group is cleared. (c) A group is selected to be cleared and a column is going to disappear. (d) A whole column is cleared and the remaining rightmost columns are shifted left. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

H_{04}: Same as H_{03} but search is performed from left to right, and from bottom to top.

H_{05}: Same as H_{03} but search is performed from right to left, and from top to bottom.

H_{06}: Same as H_{03} but search is performed from right to left, and from bottom to top.

H_{07}: Clears the first row-shape set (found from left to right, and from top to bottom) in Θ. This heuristic has no effect if there is no group that matches this characteristic.

H_{08}: Same as H_{07} but search is performed from left to right, and from bottom to top.

H_{09}: Same as H_{07} but search is performed from right to left, and from top to bottom.

H_{10}: Same as H_{07} but search is performed from right to left, and from bottom to top.

H_{11}: Clears the largest set ($n \geq 2$) in Θ, provided that clearing any other set does not result in any group in Θ' larger than the largest in Θ (the one we are clearing).

H_{12}: Clears the set in Θ that maximizes the score obtained by clearing the sets in Θ' (one by one, not considering that clearing one set modifies the resulting matrix).

H_{13}: Clears the set that minimizes the number of sets present in the resulting Θ'.

H_{14}: Clears the set that minimizes the number of minimum-order sets present in the resulting Θ'.

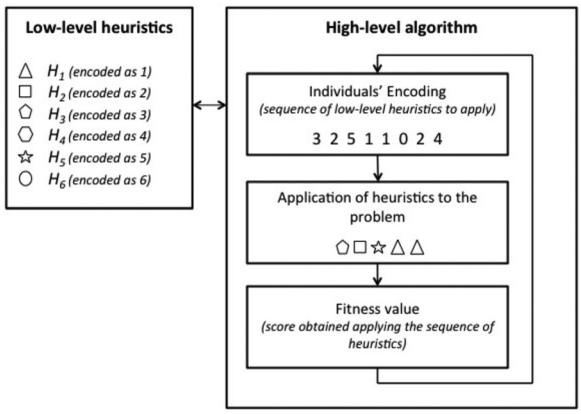

Figure 2 Outline of the HH algorithm.

H_{15}: Clears the set that ensures that the number of minimum-order sets present in the resulting Θ' is equal or less than the number of those sets in Θ.

H_{16}: Clears a set from the less abundant color in Θ.

H_{17}: Clears the largest set in Θ, namely s_i, provided that clearing any other set would result in a smaller s_i in Θ'.

H_{18}: Clears the set in Θ that results in a Θ' containing a larger set than those in Θ.

H_{19}: Clears a set if it contains all the same-colored elements remaining in Θ.

Evolutionary Algorithm

The high-level evolutionary algorithm (EA) evolves the sequence of the above-defined heuristics. Typically, an EA starts from an initial set (*population*) of random solutions (*individuals*). The evolution of the population takes place using, mainly, selection, crossover, and mutation operators [38]. Individuals are typically selected according to the quality of the solution they represent (*fitness function*). Hence, the better the fitness of an individual, the higher possibilities it has to be chosen for reproduction purposes (and its genetic material is passed on to the next generations). The selected individuals are reproduced by means of crossover and mutation operators. In our work we use a tournament selection operator that chooses the γ parents of the next generation (among the total population of the current one), and the process is repeated until a maximum number of generations is met [38]. Additionally, a classical two-points crossover operator is used [38]. Two mutation mechanisms are applied to add diversity to the newly generated population: (1) a classical mutation that randomly selects elements of an individual and substitutes them by a new value; (2) a mutation that swaps two randomly chosen elements of an individual. Both mutation operators are applied with a low probability of 1% [38].

For this process, each individual is encoded as a vector of integer numbers $\mathbf{x} \in \{0, 1, 2, 3, \ldots, 19\}^k$. Fix-length, k, vectors are used. For each individual, the application of the heuristics is performed in a loop fashion. The first (leftmost) element in the vector is applied, then the second one, and so on until no more sets of two or more bubbles remain in Θ. Fitness function is computed by means of the total score obtained as ζ sets of bubbles are cleared off (Eq. 2).

$$f(\mathbf{x}) = \sum_{i=1}^{\zeta} \text{Score}_i \tag{2}$$

Note that each solution may not include all heuristics, or may contain a given heuristic several times. Also note that the number of total heuristics to be applied (the number of heuristics before the stop criterion is found in the sequence) may be different from one individual to another. Although the heuristics after the stop symbol (0) are not used, it is important to leave them in the sequence, as they are used to generate new individuals when crossover and mutation operators are applied.

Furthermore, to cope with premature convergence of the algorithm (which would result in the algorithm sticking in a local maximum instead of evolving to the optimum solution) a *hyper-mutation* operator is included in this work. Hyper-mutation is a technique previously applied to game-solving [11] that adds diversity to the population. Whenever a generation ends, if the mean fitness value of all individuals is equal to the best fitness value among them (diversity is null), hyper-mutation takes place. We have implemented it by keeping the best three individuals. The rest of the population is randomly initialized or obtained by mutating the three kept individuals.

In order to find a good solution, a good set of parameters for the EA has to be set. Thus, population size, encoding size (k), number of generations, and mutation probability have to be determined. Encoding size has to ensure that, at least, all heuristics plus the stop symbol are included in each individual ($k \geq 20$). Population size, based on experimentation on different-sized and different number of color puzzle, is expressed in Equation (3).

$$\text{Population\_size} = \lceil 2M + 2\alpha + 20 \rceil \tag{3}$$

where M stands for the number of rows/columns in Θ, and α for the number of different-colored bubbles. Finally, the number of generations used, based also on experimentation, is expressed in Equation (4).

$$\text{Number\_of\_generations} = \lceil 20M + 20\alpha \rceil \tag{4}$$

SOFTWARE TOOL

A software tool is provided to the students to get to know the Bubble breaker, test it, and learn how HHs work (steps followed, EA, low-level heuristics, etc.). To achieve this, different options are offered, such as choosing the difficulty of the game, game saving, automated play or user play, etc. Therefore, in a first stage, the student is shown the best solution found by the EA (low-level heuristics to be applied, order to be followed and, if present, stop symbol). Afterwards, a label is shown explaining the move (low-level heuristic) that has to take place in order to optimize the score. This process takes place until the board is fully cleared or no moves are left.

The software tool presented in this work has been developed in Matlab. The user interface has been designed to let the student choose the number of rows and columns, the number of colors, and the difficulty (1 easy–10 hard). Once these parameters are chosen, a board is randomly generated, and the student can preview it, generate a new one or save it. If there are previously saved boards, the user can also load them and play them (see Fig. 3a).

The newly generated or loaded board is shown in the center of the interface. At this point, the student may start playing on his/her own, or calculate the best solution using the HHs (Fig. 3b). The solved hyper-heuristic (solution containing the low-level

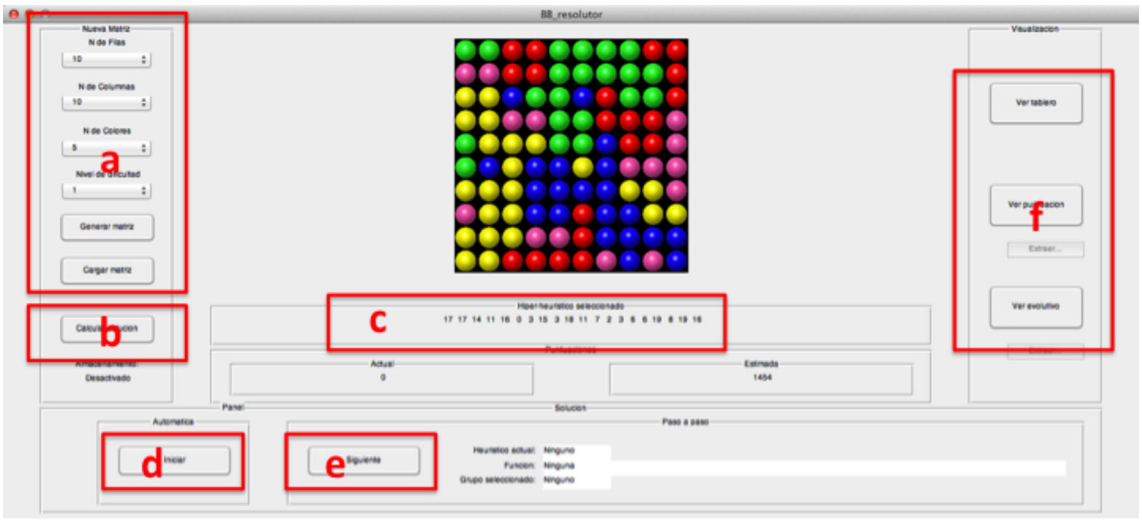

Figure 3 Bubble breaker main interface. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

heuristics) is then presented (Fig. 3c), and the sequence applied can be played in an automatic way (Fig. 3d), or manually, step-by-step (Fig. 3e).

The fitness function and the score evolution with the number of generations are shown as well by clicking "View fitness function evolution" and "View score evolution" (Fig. 3f).

In Figure 4 we can see how the interface guides the student through the game solving. First, the first low-level heuristic found in the best solution is applied. On the lower part of the screen we can see that low-level heuristic number 12 is being applied. Moreover, information explaining that this heuristic "clears the set in Θ that maximizes the score obtained by clearing the sets in Θ' "is provided. Figure 5 shows the score obtained by this move. At the end of the move, it will be added to the current score. Finally, a total score to be achieved during the game is also presented by the interface.

At any given point, the student can click on "View score evolution" to check how the score has evolved as the low-level heuristics were applied (see Fig. 6), "View fitness function evolution" (see Fig. 7), or just go "Back to the board" to keep playing. In Figure 7 the best fitness value is shown in blue, while mean fitness value is represented in red. This helps the student to

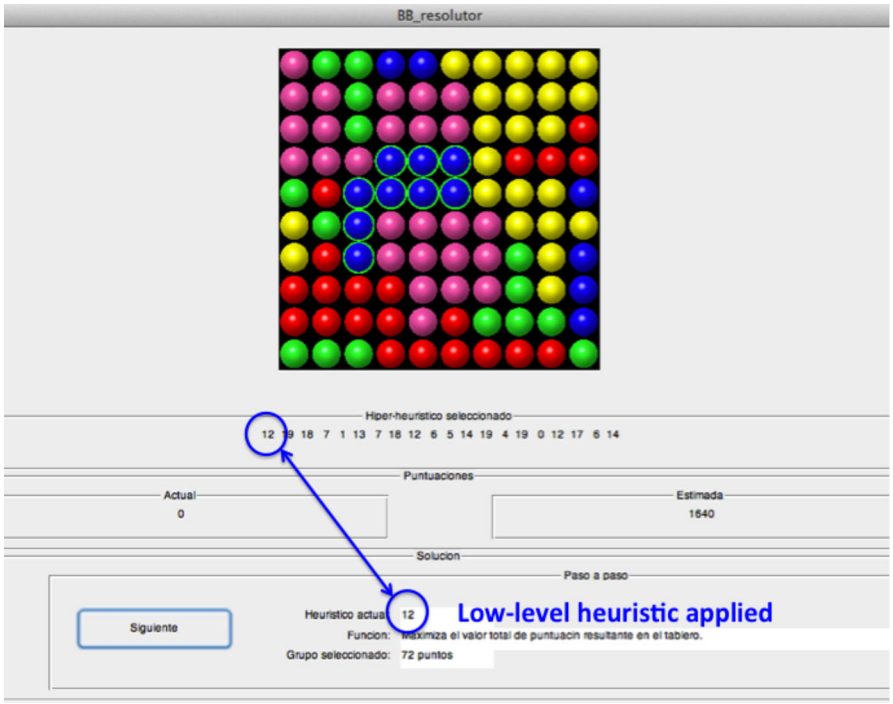

Figure 4 Playing a game. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

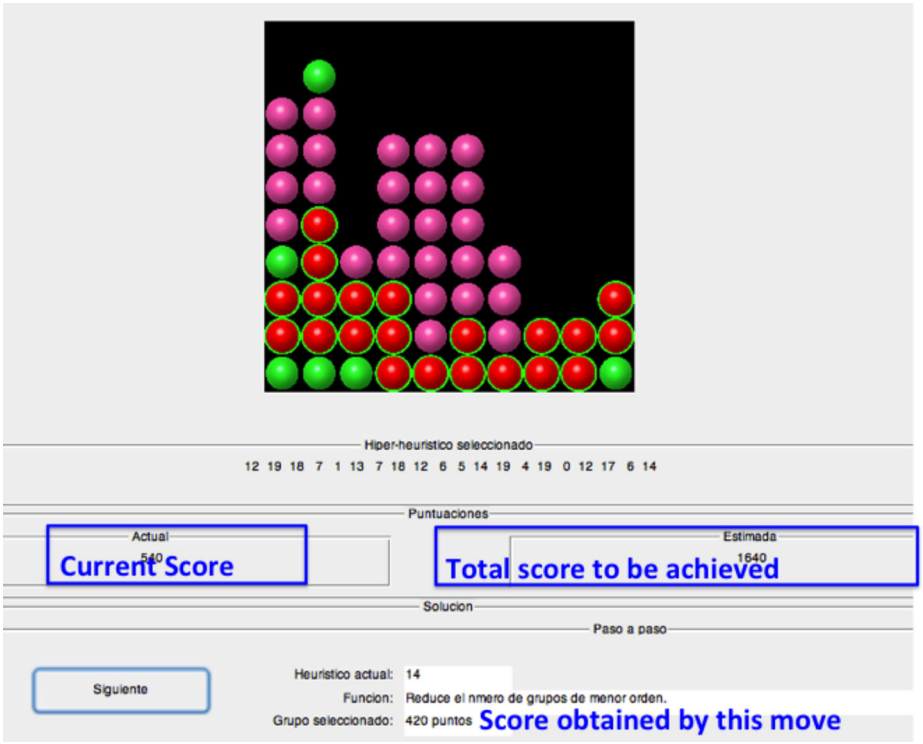

Figure 5 Game's scoring. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

understand both convergence and to see how hyper-mutation takes place. Whenever a catastrophic effect takes place (hyper-mutation), mean fitness value decays significantly and takes several rounds to recover from it.

TEACHING METHOD AND ASSESSMENT

In this work we present how HHs are taught as part of the course *Heuristic Methods for Optimization Problems in Engineering* in the Doctoral Program *Computer Architecture and Signal Processing Techniques in Telecommunication*, at the Universidad de Alcalá. In this course, the students are guided through different heuristic methods for optimization problems, starting with basic heuristics, and increasing the difficulty to evolutionary algorithms. Their first assignment is to get to know the Bubble breaker puzzle by using the software tool presented in Software Tool section. The second assignment is to implement by themselves the low-level heuristics proposed in Low-level Heuristics section. The third

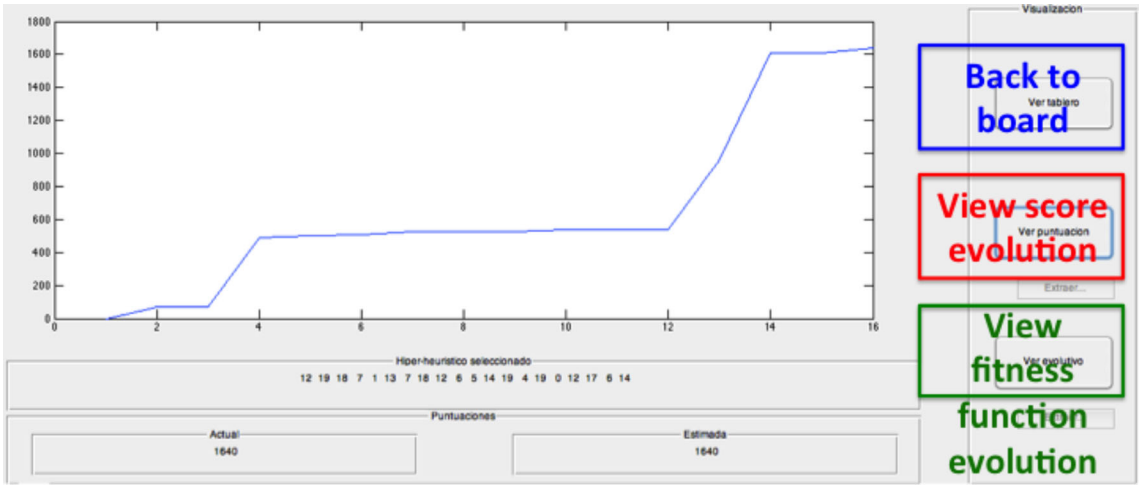

Figure 6 Score evolution. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

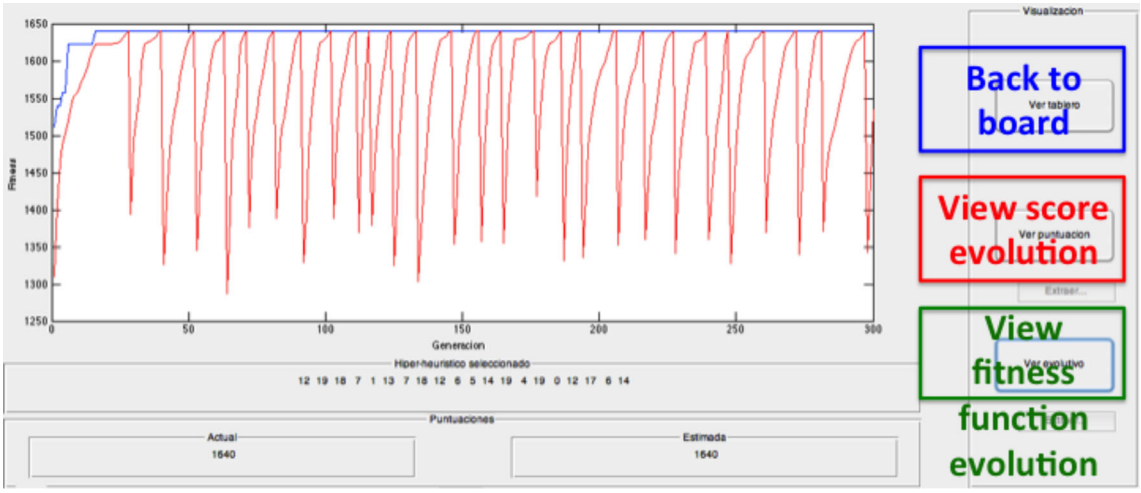

Figure 7 Fitness function evolution. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

assignment is to implement the EA explained in Evolutionary Algorithm section (fitness function, selection, crossover, and mutation operators, etc.). The implementation of the operators described in Evolutionary Algorithm section is proposed, although the students can also explore other possibilities explained in class as optional assignments.

To test their algorithm, the software tool proposed implements the possibility to load $M \times M$ matrices and execute the HH on it.

The presented method has been tested with 10 postgraduate students. To assess the proposed method, the students were asked to fill a small questionnaire about the teaching methodology used along with their feelings about it. The questionnaire recorded information provided using a five choices scale: "very much" (numerically coded as 1), "much" (numerically coded as 2), "indifferent" (numerically coded as 3), "little" (numerically coded as 4), and "very little" (numerically coded as 5). Questions asked were:

(1) Did you enjoy learning optimization methods?
(2) Did you find that the Bubble breaker tool presented in this course helped you understand the problem better?
(3) Would you rather use specific problems or other real applications instead of puzzle solving as assignments?
(4) Did you find a good learning method to increasingly add up difficulty to finally build a Bubble breaker puzzle solver?
(5) Do you believe that having the challenge to build up a tool that solves a puzzle made you understand better the different techniques explained in this course?
(6) Did you take "Soft-computing applications in Engineering" in grade program? (Yes = 1/No = 0 question).
(7) If you answered Yes to the previous question:
    (a) Do you think the methodology used in Doctoral Program is better than that used in Grade Program (where no tools were provided and you had to hand several assignments based on solving different problems)?

All students agreed that they better understood concepts than studying them over problems or other real applications. Most students took optional assignments and compared the results between the proposed methods and the ones chosen by them.

CONCLUSIONS

In this paper, the Bubble breaker puzzle has been proposed as an appealing teaching methodology to learn hyper-heuristics algorithms. HHs are a new class of searching methodologies that have gained popularity in the last decade. They have been applied to many different problems in Engineering, and they are currently one of the state of the art algorithms in meta-heuristics. In spite of this, they have deserved little attention in educational journals. In this paper we have presented an educational tool specifically focused on teaching HHs by means of the Bubble Breaker Puzzle. We have fully described the application, its use, and how it has been successfully used to introduce this methodology to postgraduate students in a Doctorate course in Universidad de Alcalá, Spain.

ACKNOWLEDGMENT

This work has been partially supported by Universidad de Alcalá (call for Innovation in the Teaching/Learning process), under project number UAH/EV523.

REFERENCES

[1] D. Hartmann, H. J. van den Herik, H. Iida, Games in AI research, ICGA J 23 (2000), 97–99.
[2] J. E. Laird, Using a computer game to develop advanced AI, IEEE Comput 34 (2001), 70–75.
[3] A. Khoo and R. Zubek, Applying inexpensive AI techniques to computer games, IEEE Intell Syst 17 (2002), 48–53.
[4] J. Mockus, Stock exchange game model as an example for graduate level distance studies, Comput Appl Eng Educ 10 (2002), 229–237.
[5] O. Montiel-Ross, R. Sepúlveda, O. Castillo, and P. Melin, Ant Colony test center for planning autonomous mobile navigation, Comput Appl Eng Educ 21 (2013), 214–229.
[6] M. Arevalillo-Herráez, R. Morán-Gómez, and J. M. Claver, Conquer the Net: An educational computer game to learn the basic

configuration of networking components, Comput Appl Eng Educ 20 (2012), 72–77.
[7] A. A. Deshpande and S. H. Huang, Simulation games in engineering education: A state-of-the-art review, Comput Appl Eng Educ 19 (2011), 399–410.
[8] G. Kendall, A. Parkes, and K. Spoerer, A survey of NP-complete puzzles, ICGA J 31 (2008), 13–34.
[9] P. Moura, D. Oliveira, B. Vrancic, E. J. Cunha, and Pires. Solteiro, Teaching particle swarm optimization through an open-loop system identification project, Comput Appl Eng Educ (2011).
[10] J. M. Hereford and H. Gerlach, Integer-valued particle swarm optimization applied to Sudoku puzzles, IEEE Swarm Intell Symp (2008), 1–7.
[11] K. H. Chen, Some practical techniques for global search in Go, ICGA J 23 (2000), 67–74.
[12] J. T. Tsai, Solving Japanese nonograms by Taguchi-based genetic algorithm, Appl Intell 37 (2012), 405–419.
[13] L. Berghman, D. Goossens, and R. Leus, Efficient solutions for Mastermind using genetic algorithms, Comput Oper Res 36 (2009), 1880–1885.
[14] C. Jefferson, A. Miguel, I. Miguel, and S. Armagan-Tarim, Modelling and solving English Peg Solitaire, Comput Oper Res 33 (2006), 2935–2959.
[15] F. Gindre, D. A. Trejo, G. Pizzo, M. D. Barrera, and De. Luise. Lopez, A criterion-based genetic algorithm solution to the Jigsaw puzzle NP-complete problem, Proc World Congr Eng Comput Sci (2010), 367–372.
[16] N. J. van Eck and M. van Wezel, Application of reinforcement learning to the game of Othello, Comput Oper Res 35 (2008), 1999–2017.
[17] D. Joyner, Adventures in group theory: Rubik's cube, Merlin's machine, and other mathematical toys. The Johns Hopkins University Press, Baltimore, Maryland 2002.
[18] S. Salcedo-Sanz, J. Portilla-Figueras, A. Pérez-Bellido, E. Ortiz-García, and X. Yao, Teaching advanced features of evolutionary algorithms using Japanese puzzles, IEEE Trans Educ 50 (2007), 151–155.
[19] J. T. Tsai, P. Y. Chou, and J. C. Fang, Learning intelligent genetic algorithms using Japanese Nonograms, IEEE Trans Educ 55 (2012), 164–168.
[20] C. C. Liao, Z. H. Chen, and T. W. Chan, Effectiveness of pet-nurturing hand-held Game on the aspects of learner motivation. In: Proceedings of the 16th International Conference on Computers in Education 2008, pp 705–712.
[21] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (Eds.), Hand-book of metaheuristics, Kluwer, New York, USA, 2003, pp 457–474.
[22] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, Hyper-heuristics: A survey of the state of the art, J Oper Res Soc 64 (2013), 1725–1741.
[23] S. Remde, P. Cowling, K. Dahal, N. Colledge, and E. Selensky, An empirical study of hyperheuristics for managing very large sets of low level heuristics, J Oper Res Soc 63 (2011), 392–405.
[24] G. Kendall and M. Mohamad, Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proceedings of the IEEE International Conference on Network 2004, pp 769–773.
[25] G. Kendall and M. Mohamad, Channel assignment optimisation using a hyper-heuristic. In: Proceedings of the IEEE Conference on Cybernetic and Intelligent Systems, 2004, pp 790–795.
[26] P. Cowling, G. Kendall, and E. Soubeiga, A parameter-free hyper-heuristic for scheduling a sales summit. In: Proceedings of the 4th Metaheuristic International Conference, 2001, pp 127–131.
[27] P. Cowling, G. Kendall, and E. Soubeiga, Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation, Applications of Evolutionary Computing: Proc. of Evo Workshops 2002, Lect Notes Comput Sci 2279 (2002), 1–10.
[28] E. K. Burke, G. Kendalland, and E. Soubeiga, A Tabu-search hyper-heuristic for timetabling and rostering, J Heuristics 9 (2003), 451–470.
[29] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, A graph-based hyperheuristic for educational timetabling problems, Eur J Oper Res 176 (2007), 177–192.
[30] R. Furtuna, S. Curteanu, and F. Leon, Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic, Appl Soft Comput 12 (2012), 133–144.
[31] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward, A classification of hyper-heuristics approaches. In: M. Gendreau, J.Y. Potvin (Eds.), Handbook of metaheuristics, international series in operations research & management science, Springer, New York, USA, 2010.
[32] J. Mockus, Investigation of examples of E-education environment for scientific collaboration and distance graduate students. Part 1, Informatica 17 (2006), 259–277.
[33] J. Mockus, Investigation of examples of E-education environment for scientific collaboration and distance graduate students. Part 2, Informatica 19 (2008), 45–61.
[34] T. Mantere and J. Koljonen, Solving, rating and generating Sudoku puzzles with GA. In: Proceedings of the IEEE Congress on Evolutionary Computation, 2007, pp 1382–1389.
[35] M. P. Schadd, M. H. Winands, H. J. van den Herik, G. M. Chaslot, and J. W. Uiterwijk, Single-player Monte-Carlo tree search for Same-Game, Knowl-Based Syst 34 (2012), 3–11.
[36] H. Baier and M. H. Winands, Beam Monte-Carlo tree search. In: Proceedings of the IEEE Conference on Computational Intelligence in Games, 2012, pp 227–233.
[37] S. Salcedo-Sanz, J. M. Matías-Román, S. Jiménez-Fernández, A. Portilla-Figueras, and L. Cuadra, "An evolutionary-based hyper-heuristic approach for the Jawbreaker puzzle, Appl Intell in press.
[38] A. E. Eiben and J. E. Smith, Introduction to evolutionary computing, natural computing series, 1st ed., Springer-Verlag, New York, USA, 2003.

BIOGRAPHIES

Sancho Salcedo-Sanz received his BS degree in Physics from the Universidad Complutense de Madrid, Spain, in 1998, and his PhD in Telecommunications Engineering from the Universidad Carlos III de Madrid, Spain, in 2002. Currently, he is an Associate Professor in the Department of Signal Processing and Communications, Universidad de Alcalá, Spain. He has co-authored more than 150 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 Engineering.

Silvia Jiménez-Fernández received her MS and PhD degrees from Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2000 and 2009, respectively, both in Telecommunication Engineering. She is an Associate Professor in the Signal Processing and Communications Department at the Universidad de Alcalá. She has participated in more than 30 EU-funded projects, national-funded projects, and industrial contracts. Her current interests deal with optimization problems in telecommunications and energy fields, genetic algorithms, evolutionary algorithms and hybrid algorithms.

José Manuel Matías-Román received his BS degree in Telecommunication Engineering in 2012 from Universidad de Alcalá. He is currently working as a software engineer in an IT company.

José Antonio Portilla-Figueras received his BS degree in Telecommunication Engineering in 1999 and his PhD in Telecommunication Engineering in 2004 from the Universidad de Cantabria in Spain. He is an Associate Professor in the Signal Processing and Communications Department at the Universidad de Alcalá since 2004, and since 2010 Vice-dean of the Polytechnic School at the same University. His field of expertise is mainly focused on mobile telecommunications, applying modern heuristics to their design and planning. He has published more than 40 articles in international journals and has participated as an external consultant on several projects with international companies.