Evolutionary Genetic Algorithms in a Constraint Satisfaction Problem: Puzzle Eternity II

Jorge Muñoz, German Gutierrez, and Araceli Sanchis

University Carlos III of Madrid
Avda. de la Universidad 30, 28911 Leganés, Spain
{jmfuente,ggutierr,masm}@inf.uc3m.es

Abstract. This paper evaluates a genetic algorithm and a multiobjective evolutionary algorithm in a constraint satisfaction problem (CSP). The problem that has been chosen is the Eternity II puzzle (E2), an edge-matching puzzle. The objective is to analyze the results and the convergence of both algorithms in a problem that is not purely multiobjective but that can be split into multiple related objectives. For the genetic algorithm two different fitness functions will be used, the first one as the score of the puzzle and the second one as a combination of the multiobjective algorithm objectives.

1 Introduction

The Eternity II puzzle [1] is a constraint satisfaction problem created as a challenge, with a prize, to know if someone is able to solve it. The difficulty lies on the huge search space of the problem, there is not any algorithm that can find a solution in a reasonable time. It is an edge-matching puzzle with 256 tiles that are differents one to each other. The tiles have a color pattern in each of its edges and there are not two tiles with the same color pattern in all their edges. All the tiles must be placed in a 16×16 board, and can be rotated 90^o, 180^o or 270^o before being placed. There are tiles with a grey pattern in their edges that must be placed in the corners and the borders of the board. These grey edges must be placed next to the border. There is also known the cell and position of one of the tiles but we will omited this information because it is irrelevant for the objectives of this paper.

If we use the information about the tiles that have to be placed into the corners and the border of the board, then the search space of this problem is approximately $4^{196} \cdot 196! \simeq 10^{485}$. A more accurate approximation can be calculated with the Equation 1 where n is the number of tiles.

$$\begin{aligned} S &= 4! + (4 \cdot (\sqrt{n} - 2))! + 4^{(\sqrt{n}-2)^2} \cdot ((\sqrt{n} - 2)^2)! \\ &\simeq 4^{(\sqrt{n}-2)^2} \cdot ((\sqrt{n} - 2)^2)! \end{aligned} \tag{1}$$

Our objective is not to solve the puzzle, but satisfy the greatest number of constraints with evolutionary algorithms in the minimum possible time. A

constraint is the join of two adjacent edges of tiles, if the adjacent edges match then the constraint is satisfied. We do not bear in mind the edges that must be placed in the border of the board, only the inner ones among tiles. We will called the number of constraints satisfied the score. In a squared board of $m \times m$ cells the maximum score that can be reached is obtained by Equation 2. For the Eternity II the maximum score is 480.

$$\text{score} = m \cdot (m-1) + (m-1) \cdot m = 2 \cdot m \cdot (m-1) \tag{2}$$

The edge-matching puzzles are NP-complete problems [2], specifically they are constraint satisfaction problems (CSPs) [3]. When the problem is computationally intractable and we want to satisfy the maximum number of constraints we call it a MAXSAT [4] problem and here is where an evolutionary algorithm can be used [5].

In the rest of the document it will be shown some related work of evolutionary algorithms in combinatorial problems (*Section 2*), a description of the search algorithms used (*Section 3*), the experimental results of the evolutionary algorithms with different parameters (*Section 4*) and, finally, the conclusions and future works (*Section 5*).

2 Related Work

Constraint satisfaction problems are solved through two different techniques: inference and search; although sometimes a combination of both is used [6,3]. The inference modifies the problem to create another one that is easier to solve, normally this change is made with domain information. The main algorithms in inference are the consistency algorithms, also known as constraint propagation [3]. In search the most known algorithm is backtracking [6], this algorithm essentially performs a depth-first search of the space of potential solutions. The lack of the search algorithms is that they do not use the domain information to improve the search and they repeat the same mistakes, that is, they repeat lot of times searches that do not lead to a solution and could be avoided with some domain information.

The exhaustive search is the unique algorithm that grants to find a solution if it exists, but in problems with big search space this algorithm is useless because the time required to finish. So in this sort of problems, like some MAXSAT, a local search is performed instead of an exhaustive search. The local search algorithms starts with a set of assigned variables and tries to allocate the rest of variables. When it is reached a situation where a constraint is violated the algorithm modify the values of the variables that break the constraint. The best known results in the Eternity II were obtained with an algorithm of these features that uses an hibridization of constraint propagation and very large neighborhood stochastic local search [7], it satisfies 458 of the 480 constraints.

Another way to find solutions in a MAXSAT problem is through the use of evolutionary algorithms [8]. Some of the algorithms in the literature are SAWEA [9,10], RFEA [11,12], FlipGa [13], ASAP [14] and genetic algorithm [5], although a clasic algorithm as FC-CBJ [15] outperforms these kind of evolutionary algorithms [16].

3 Search Algorithms

Besides the genetic algorithm and the multiobjective evolutionary algorithm, we have used an exhaustive search algorithm and random search. The exhaustive search is a backtracking algorithm that places the tiles in the board but with an improvement. When the maximal deep of search is reached, instead of do the backtracking in that moment, the algorithm continues browsing the board and placing tiles in the right way with the cells where any tile can be placed left in blank. When it finishes the backtracking is performed.

Next, it will be seen a brief description of the evolutionary algorithms and the two fitness functions used in the genetic algorithms.

3.1 Genetic Algorithm (GA)

In the experiments we have used a simple genetic algorithm with elitism and tournament selection. Codification, crossover, mutation and fitness function will be described below.

The representation of a solution is a bidimensional matrix where each cell contains a tile and its orientation. This representations keeps the spacial relations among the adjacent tiles between generations, so specific crossover operator and mutation have to be used.

The crossover operator for the coding is based on the regions exchange. Two parents are selected, then a region of random size in a random point is chosen. Then the outer cells of the region are copied into the offspring from one of the parents, and the inner from the other one. In this operation it has to bear in mind that there can not be duplicated or missed tiles, this must be avoided. An example of the crossover operator is shown in Fig. 1.

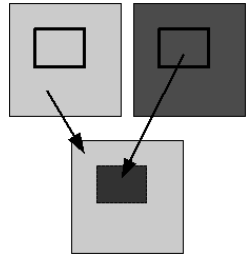
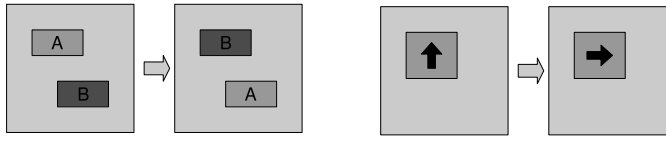


Fig. 1. A simple example of crossover

The mutation operators are two, when a mutation is going to be applied only one of the operators is chosen randomly. One mutation operator is the region exchange (Fig. 2(a)), this operator exchanges two regions of the same size that are not overlap. The second mutation operator is the region rotation (Fig. 2(b)), this operator rotates a squared region.



(a) Exchange (b) Rotation

Fig. 2. Examples of mutations

For crossover and mutation operators, the height and width of the regions are chosen randomly between two bounds. The minimum size of the region for crossover is 2 and the maximum 8, and for mutation the bounds are 1 and 10.

In this algorithm two fitness functions have been used, both of them will try to be minimized. The maximum value of this fitness functions is 1 and the minimum is 0, where 0 also means that the puzzle is solved. The first one, called the *normal fitness*, it is one minus the normalized score (see Equation 3). The second fitness is a combination of the three objectives in the multiobjevite algorithm, so it is called the *combined fitness*. The equation to calculate this *combined fitness* is shown in Equation 4 where k is the number of objectives, $objective_i$ is the value of the objective i and $max\ objective_i$ is the maximum value of the objective i (in Section 3.2 the objectives will be described).

$$\text{normal fitness} = 1 - \frac{\text{score}}{480} \tag{3}$$

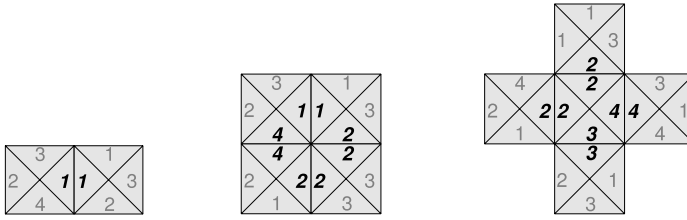
$$\text{combined fitness} = 1 - \left(\frac{1}{k} \cdot \sum_{i=1}^{k} \frac{\text{objective}_i}{\text{max objective}_i} \right) \tag{4}$$

3.2 MultiObjective Evolutionary Algorithm (MOEA)

The multiobjective evolutionary algorithm has been developed from the concepts used in the NSGA-II algorithm [17]. But three modifications have been done to improve the performance in the problem. The chromosome, crossover operator and mutation operator are the same as in the genetic algorithms we saw before.

The first improvement is how the dominance is calculated. Instead of use fronts of dominance, the dominance of one individual is calculated by counting the number of individuals that dominate that individual. This is faster than calculate the fronts of dominance.

The second improvement is how the distance among individuals is calculated. Only the distances among the individuals of the pareto front are calculated. This distance is through the diversity in the values of the objectives of the individuals. If the same objective in both individuals has different values the distance between that individuals is increased one unit, thus the maximum distance between two individuals is the number of objectives and the minimum is 0.



(a) Objective 1 (b) Objective 2 (c) Objective 3

Fig. 3. The three objectives of the MOEA

The third improvement is the number of elitism individuals that are copied from one generation to the next one. Only a fixed number of individuals in the pareto front are copied. The first ones that are copied are those ones which have more distance with the rest of individuals.

The objectives that have to be maximized for the problem are three:

- *Objective 1*: The score of the puzzle explained in *Section 1* (Fig. 3(a)). The maximum value of this objective is 480 and the minimum is 0.
- *Objective 2*: The number of square regions of four tiles (2×2 regions) that match in their adjacent inner sides (Fig 3(b)). The maximum value is 225 and the minimum is 0.
- *Objective 3*: The number of tiles that have their four sides matched with the adjacent tiles (Fig. 3(c)). The maximum value is 256 and the minimum is 0.

4 Experimental Results

In the experimentation it has been launched a random algorithm (RAND), the exhaustive search (ES) explained in the beginning of Section 3, a genetic

Table 1. Results

Experiment	Algorithm	Cross.	Mut.	Elit.	Fitness	Max.	Min.	Mean	Std. Dev.
RAND	RAND	-	-	-	-	49	45	46.5	± 1.02
ES	ES	-	-	-	-	369	357	363.7	± 3.66
GA1	GA	9000	999	1	norm	316	286	295.9	± 7.76
GA2	GA	5000	4999	1	norm	160	151	157.3	± 2.53
GA3	GA	7500	2500	0	norm	130	75	106.7	± 18.48
GA1C	GA	9000	999	1	comb	355	337	347.9	± 6.5
GA2C	GA	5000	4999	1	comb	357	146	210.4	± 90.03
GA3C	GA	7500	2500	0	comb	366	348	357.5	± 5.48
MOEA1	MOEA	9000	990	10	-	367	345	353.7	± 6.56
MOEA2	MOEA	5000	4990	10	-	371	350	361.7	± 5.1

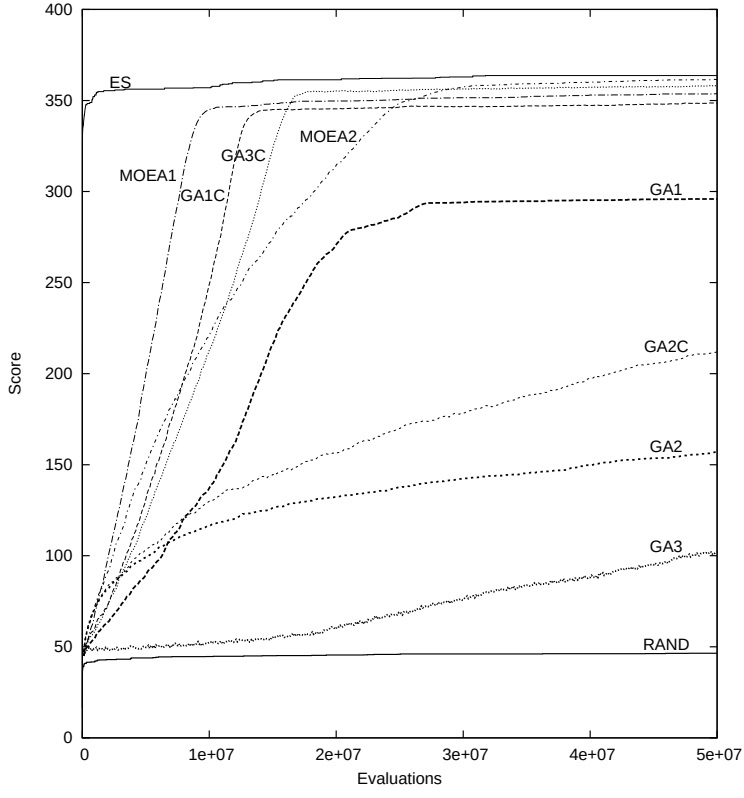


Fig. 4. Evolution of the score in the experiments

algorithm (GA) and a multiobjective evolutionary algorithm (MOEA). Both evolutionary algorithms have been executed with different values for crossover, mutation and elitism, but the population has been 10000 individuals for all experiments and the tournament is always of size 3. The results of the experiments are shown in Table 1. First column (Experiment) shows the experiment name. Second column (Alg.) is the algorithm used in that experiment. Next three columns are the number of individuals that are generated for the next generation by crossover (Cross.), mutation (Mut.) and elitism (Elit.) respectively. The next column (Fitness) is the sort of fitness that have been used in the genetic algorithm: normal fitness (norm) or combined fitness (comb). In the next three columns are shown the minimum (Min.), maximum (Max.) and mean (Mean) value of the bests scores of the puzzles in the experiments. For these columns the higher value is the better is, with a maximum value of 480. The last column (Std. Dev.) is the standard deviation of the experiments. Empty cells ('-') mean that parameter is not used in the algorithm. Each experiment was executed 10

times and each time it was running until $5 \cdot 10^7$ evaluations, a evaluation means each time the fitness of an individual is obtained.

Fig. 4 shows the evolution of best average score of the experiments. The x-axis is the number of evaluations and the y-axis is the average best score. The name of the experiment is shown next to its line, just top-left of the line.

5 Conclusions and Future Works

The best result was obtained in the *MOEA2* experiment with a score of 371 over 480, but the best average score was obtained with the exhaustive search (*ES*), 363.7 over 480.

The genetic algorithm with the normal fitness got better results with a higher crossover rate and the experiment *GA1* got the best results among then. There also notice that the experiment without elitism (*GA3*) got very bad results, it looks like it was slower to converge. In the genetic algorithm with combined fitness the results were better than with normal fitness and converged faster. But unlike before, here the experiment without elitism (*GA3C*) got the best results and converged almost as fast as the experiment with the higest crossover rate (*GA1C*). The difference between use a normal fitness or combined one without elitism is incredible huge, the algorithm goes from a results near the random search (*RAND*) to results very close the better ones (*ES*). If we compare the experiments *GA2* and *GA2C*, where the only difference is that the second one uses a combined fitness, the results were better in *GA2C* but the standar deviation increased a lot.

Between the two multiobjective experiments, *MOEA1* converged faster than *MOEA2* but this second one got better results, closer to the exhaustive search.

In future works we will compare these results with more evolutionary algorithms like SAWEA or FlipGA and with more specific algorithms for CSPs like FC-CBJ.

Acknowledgment

This work was supported in part by the Carlos III University of Madrid under grant PIF UC3M01-0809 and by the Ministry of Science and Innovation under project TRA2007-67374-C02-02.

References

1. Tomy: Eternity II (official site) (November 2008), `http://www.eternityii.com`
2. Demaine, E., Demaine, M.: Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity. Graphs and Combinatorics 23, 195–208 (2007)
3. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
4. Raman, V., Ravikumar, B., Rao, S.S.: A simplified NP-complete MAXSAT problem. Information Processing Letters 65(1), 1–6 (1998)

5. Rana, S., Whitley, D.: Genetic algorithm behavior in the MAXSAT domain. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 785–794. Springer, Heidelberg (1998)
6. Kumar, V.: Algorithms for Constraint-Satisfaction Problems: A Survey. AI Magazine 13(1), 32–44 (1992)
7. Pierre Schaus, Y.D.: Hybridization of CP and VLNS for Eternity II. JFPC (2008)
8. Gottlieb, J., Marchiori, E., Rossi, C.: Evolutionary Algorithms for the Satisfiability Problem. Evolutionary Computation 10(1), 35–50 (2002)
9. Bäck, T., Eiben, A., Vink, M.: A Superior Evolutionary Algorithm for 3-SAT. In: Porto, V.W., Waagen, D. (eds.) EP 1998. LNCS, vol. 1447, pp. 125–136. Springer, Heidelberg (1998)
10. Eiben, A., van der Hauw, J.: Solving 3-SAT with adaptive Genetic Algorithms. In: Proceedings of the 4th IEEE Conference on Evolutionary Computation, pp. 81–86 (1997)
11. Gottlieb, J., Voss, N.: Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 755–764. Springer, Heidelberg (1998)
12. Gottlieb, J., Voss: Adaptive fitness functions for the satisfiability problem. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 621–630. Springer, Heidelberg (2000)
13. Marchiori, E., Rossi, C.: A flipping genetic algorithm for hard 3-SAT problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, pp. 459–465 (1999)
14. Rossi, C., Marchiori, E., Kok, J.N.: An adaptive evolutionary algorithm for the satisfiability problem. In: Proceedings of the 2000 ACM symposium on Applied computing, vol. 1, pp. 463–469 (2000)
15. Haralick, R.M., Elliott, G.L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. Artificial Intelligence 14(3), 263–313 (1980)
16. Craenen, B., Eiben, A., van Hemert, J.: Comparing evolutionary algorithms on binary constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 7(5), 424–444 (2003)
17. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester (2001)