Evolutionary techniques in a constraint satisfaction problem: Puzzle Eternity II

Jorge Muñoz, German Gutierrez, Araceli Sanchis

***Abstract*— This work evaluates three evolutionary algorithms in a constraint satisfaction problem. Specifically, the problem is the Eternity II, a edge-matching puzzle with 256 unique square tiles that have to be placed on a square board of 16×16 cells. The aim is not to completely solve the problem but satisfy as many constraints as possible. The three evolutionary algorithms are: genetic algorithm, an own implementation of a technique based on immune system concepts and a multiobjective evolutionary algorithm developed from the genetic algorithm. In addition to comparing the results obtained by applying these evolutionary algorithms, they also will be compared with an exhaustive search algorithm (backtracking) and random search. For the evolutionary algorithms 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. We also used two ways to create the initial population, one randomly and the other with some domain information.**

I. Introduction

The Eternity II [1] (also known as E2) puzzle is an edge-matching puzzle created by Christopher Monckton for the game editor Tomy (TM). This puzzle consists in 256 unique square tiles that have to be placed in a 16×16 board. Each tile has a different color pattern in each of its four sides. The tiles are different to each other, there are not two tiles with the same color pattern in all their edges, and each tile can be rotated 90º, 180º or 270º before putting in the board. Two adjacent tiles must match in their color pattern for considering that they are placed correctly in the board. There are special tiles for the corners and the border of the board. These tiles have a grey color in the side that must be in the border of the board. Also, it is known the position and orientation of one of the tiles in the puzzle, but we will omit this information in our evolutionary algorithms.

The quality of the puzzles is measured with a score. Each time two adjacent tiles match in one of their edges the score is increased one unit. The grey edges that must be placed in the border of the board do not count for this score. In a $n \times n$ board, the maximum score is given by equation 1. In Eternity II, with a 16×16 board, the maximum score that can be reached is 480.

$$\begin{aligned} \text{max score} &= n \cdot (n-1) + (n-1) \cdot n \\ &= 2 \cdot n \cdot (n-1) \end{aligned} \quad (1)$$

The size of the search space, S, of edge-matching puzzles with a squared board of $n \times n$ cells and with unique tiles can be calculated with equation 2, where n^2 is the total number of tiles.

$$S = (n^2)! \cdot 4^{n^2} \quad (2)$$

If the problem is divided in three subproblems: the tiles that have to be placed in the corners, that tiles of the border of the board and the interior tiles; then the search space is reduced as equation 3 shows.

$$\begin{aligned} S &= 4! + (4 \cdot (n-2))! + 4^{(n-2)^2} \cdot (n-2)^2! \\ &\simeq 4^{(n-2)^2} \cdot (n-2)^2! \end{aligned} \quad (3)$$

In an edge-matching puzzle as the Eternity II with 256 tiles (16×16 board) the search space is approximately $4^{196} \cdot 196! \simeq 10^{485}$.

The edge-matching puzzles belong to the category of NP-complete problems [2], specifically they are constraint satisfiability problems (CSPs) [3]. Despite the evolutionary algorithms are not the best ones to find a solution in this sort of problems, an enormous search space as it is in this problem means that this problem is computationally intractable, there is not efficient algorithm that can solve it in an admissible time. Thus, the goal is not to find a solution but to find the solution that the more constraints satisfies (the more number of matching edges). These kind of problems are know as MAXSAT [4] and here is where an evolutionary algorithm can be useful [5].

We use three different evolutionary algorithms for this problem:

- *Genetic algorithm (GA)*: an evolutionary algorithm used for NP-complete problems [6][7][8].
- *Artificial immune evolutionary algorithm (AIEA)*: an algorithm developed from some concepts of the artificial immune systems [9].
- *Multiobjective evolutionary algorithm (MOEA)*: a multiobjective algorithm [10] developed from the genetic algorithm which have been added several objectives, to know whether split the problem in objectives is useful.

In the rest of the document we present some related work of evolutionary algorithms in combinatorial problems (*Section II*), a description of the search algorithms used (the exhaustive search algorithm and the three the evolutionary algorithms, *Section III*), the experimental results of the evolutionary algorithms with different parameters (*Section IV*) and, finally, the conclusions and future works (*Section V*).

J. Muñoz, G. Gutierrez, A. Sanchis are with the Computer Science Department, University Carlos III of Madrid, Avda. de la Universidad 30, 28911 Leganés, Spain (emails: { jmfuente, ggutierr, masm }@inf.uc3m.es).

978-1-4244-2959-2/09/$25.00 ©

II. Related Work

There are different techniques that can be used to try to solve a CSP which can be classified into two categories: inference and search, although a mixed of both techniques have been used [11][3]. The inference tries to modify the problem into another that it is easier to solve. This is performed through the domain information. The main algorithms in inference are the consistency algorithms, also known as constraint propagation [3]. For search techniques the main algorithm is the backtracking algorithm [11], this algorithm essentially performs a depth-first search of the space of potential solutions. The lack of search algorithms is that they do not use any kind of domain information and repeats the same mistakes, that is, they repeat a lot of times searches that do not lead to a solution and could be avoided with some domain information.

Due to the exhaustive search is the unique algorithm that guarantees to find a solution if it exists, a mixed algorithm between search and inference is commonly used. The exhaustive search is often enhanced with look-ahead and look-back techniques as constraint propagation. In some problems as the Eternity II puzzle or other with a enormous search space, this sort of search is useless because there is not time enough to browse all the possible solutions. So, when the goal is to satisfy the higher number of restrictions (MAXSAT problem) a local search is performed instead of an exhaustive search.

Local search algorithms start 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. These algorithms apply a heuristic to guide the search. The best known results in the Eternity II were obtained with an algorithm of these features that uses a hibridization of constraint propagation and very large neighborhood stochastic local search [12], it satisfies 458 of the 480 constraints.

Another way to find a good solution in a MAXSAT problem is through the use of evolutionary algorithms [7]. Some of the algorithms in the literature are SAWEA [13][8], RFEA [14][15], FlipGa [16] and ASAP [17], although a genetic algorithm can be used too [5].

III. Search Algorithms Applied

We have used a random search algorithm, an exhaustive search algorithm and three evolutionary algorithms for the Eternity II puzzle. Random search only places the tiles on the board randomly and then checks is a solution has been found.

Below we describe the exhaustive search algorithm and the main features of the evolutionary algorithms. At the end of the section it will be explained how is the fitness function in all the evolutionary algorithms and how is the initial population created. It has been used two kinds of fitness functions and two ways to initialize the initial population in the evolutionary algorithms.

A. Exhaustive Search Algorithm (ES)

The exhaustive search algorithm is just a backtracking algorithm. The algorithm starts with the known tile that is placed in the middle of the board, and then tries to put the rest of the tiles. The board is filled like a spiral, from the inner to the outer.

In order to improve the results of this algorithm when the maximal deep of search is reached, instead of do the backtracking, the algorithm continues browsing the board and placing tiles in the right way. The cells where any tile can be placed are left in blank. This improvement of the algorithm is the same as we do in one of the initializations of the population that will be described in *Section III-F*.

B. Genetic Algorithm (GA)

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

As the domain of the problem is an edge-matching puzzle, we have decided to use a new representation of the chromosome instead of a bit stream or floating point one [7]. This new representation is a bidimensional matrix where each cell contains a tile and its orientation. With this coding of the individuals there is a direct correspondence between the problem domain and the chromosome, and the spacial relations among the adjacent tiles are also kept. Due to our objective is to keep the spatial relations among the tiles in the puzzle between generations we need to apply new operators for crossover and mutation. These new operators use an area or region of the puzzle to realize their operations. With regions the spatial relations within and outside the regions are preserved both vertically and horizontally.

The specific crossover operator for the new coding is based on the regions exchange. From two parents only one offspring is generated. After the parents are selected, a region with random size in a random point is chosen. Then, the outer cells of the region are copied to the offspring from one parent and the inner ones from the other parent. It must be born in mind that one of the constraints of the problem is that each tile is unique and the board must have all the tiles, so there can not be repeated tiles and can not miss anyone. Thus, some extra operations must be taken to avoid this problem when a region is copied into the offspring. See Algorithm 1 and Fig. 1 for details.

There are two types of the mutation operator: the region exchange and the region rotation. In the first one, region exchange, two regions of the same size are selected in the individual and all the cells of the regions are exchanged (Fig. 2). The only requirement here is that the two regions must not be overlapped. In the second mutation operation, the region rotation, a region is selected and rotated (Fig. 3).

In the genetic algorithm when an individual has to be mutated it is used only one mutation operator: the region exchange or the region rotation. The probability to choose one operator or the other one is the same.

Both crossover and mutation operators, the height and width of the regions are chosen randomly between two

Algorithm 1 Crossover

Require: $parent_A$, $parent_B$, $offspring$

1: select a random region
2: copy the cells from $parent_A$ to $offspring$ (clone $parent_A$)
3: remove the tiles from $offspring$ that are in the inside the region in $parent_B$
4: add the remaining tiles inside the region in $offspring$ to a $list$
5: copy the cells inside the region from $parent_B$ to $offspring$
6: fill the empty cells in $offspring$ randomly with the tiles in the $list$

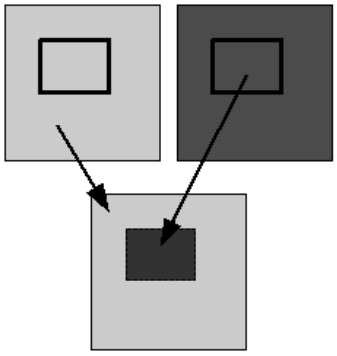


Fig. 1. A simple example of crossover

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 crossover and region exchange mutation the region can be a rectangle but in region rotation mutation the region must be a square.

C. Artificial immune Evolutionary Algorithm (AIEA)

The artificial immune evolutionary algorithm (AIEA) has been developed from some concepts of the artificial immune systems [9][18], specifically the clonal selection principle, also called clonal selection theory. This principle is the process used by the immune system to protect our body to an antigenic stimulus. From this theory we have used two ideas for our algorithm, the clonal selection and the somatic hypermutation. The new cells that the body needs to elaborate a response against the antigen are produced by the clonal selection. This means that only the best cells against the antigen are cloned and increased their number. But when there are not good cells to respond against an antigen it is necessary an adaptation. The body needs to generate new sort of cells quickly against that antigen, so it is necessary a mutation of the cells. This last process is called hypermutation because the cells suffer a rapid accumulation of mutations for a fast maturation.

The algorithm that we propose has a population with a fixed number of individuals. From this population an individual is selected through a selection operator. In the

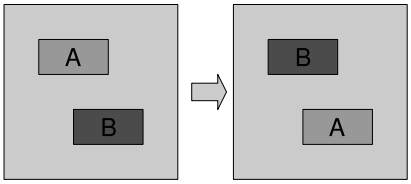


Fig. 2. An example of region exchange mutation

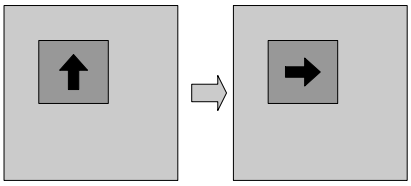


Fig. 3. An example of region rotation mutation

experiments it has been used a tournament selection. The selected individual is cloned, and the new cloned individual is mutated proportionally to its fitness. If the fitness is good compared with the rest of the population the cloned individual suffers a few mutations, the operator of mutation is applied few times. If the fitness is bad it suffers a lot of mutations, the operator of mutation is applied a lot of times. In our case the worst individuals have a great fitness and the best individuals have a lower fitness. Then other individual is selected from the population with another selection operator but in this case the selection operator returns a bad individual. We have also used for this a tournament operator that returns the worst individual of the tournament. The last selected individual is removed from the population and the cloned and mutated individual is added. The process is repeated until a stop criterion is reached. The AIEA algorithm is shown below, in algorithm 2.

In AIEA the mutation operator that has been used is the same that in the genetic algorithm: region exchange and region rotation. In each mutation the probability to choose one or other operator is the same.

The main difference between this algorithm and a genetic algorithm is that our immune algorithm does not use a crossover operator. Also when an individual is mutated it normally suffers more than one mutation. This has a lack that no information from more than one individual is used to create a new one, but this way increases the diversity of the population.

Other algorithms from artificial immune systems like ClonalG [9], negative selection [19], immune networks [9],... and in general all the ideas taken from immune systems are normally applied to pattern recognition systems [20]. But in AIEA that ideas from immune systems are applied into an evolutionary algorithm.

D. MultiObjective Evolutionary Algorithm (MOEA)

The last of the three evolutionary algorithms is a multiobjective evolutionary algorithm (MOEA). Although our problem is not multiobjective we think that splits the problem

Algorithm 2 Artificial Immune Evolutionary Algorithm

Require: population size, tournament size, stop criterion

1: generate the initial $population$
2: **for** each $individual$ in $population$ **do**
3: calculate fitness of the $individual$
4: **end for**
5: $maxfit \Leftarrow$ maximum fitness in the $population$
6: $minfit \Leftarrow$ minimum fitness in the $population$
7: **repeat**
8: $individual\_to\_clone \Leftarrow$ select the best of a tournament
9: $ind \Leftarrow$ clone $individual\_to\_clone$
10: $fit \Leftarrow ind \rightarrow fitness$
11: $mutation\_range \Leftarrow \frac{fit-minfit}{maxfit-minfit}$
12: mutate ind proportional to $mutationRange$
13: $individual\_to\_remove \Leftarrow$ select the worst of a tournament
14: remove $individual\_to\_remove$ from $population$
15: add ind to $population$
16: $maxfit \Leftarrow$ maximum fitness in the $population$
17: $minfit \Leftarrow$ minimum fitness in the $population$
18: **until** stop criterion

into related subproblems and try to solve all of then at once could improve the results. We used the same concepts explained in the NSGA-II algorithm [10] but with some modifications to improve the performance in the problem. The chromosome, crossover operator and mutation operator are the same as in the other algorithms we saw before.

There are three major changes in our MOEA compared to NSGA-II. Two of them are improvements related to the dominance and the distance among individuals to increase the performance of the algorithm. The last modification is about the number of individuals of the pareto front that are copied from one generation to the next one.

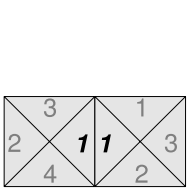
The first improvement is how the dominance is calculated. Instead of using fronts of dominance, the dominance of one individual is calculated by counting the number of individuals that dominate that individual. This is faster than calculate the fronts of dominance.

A second improvement has been needed to calculate the distance among individuals. The MOEA only calculates the distances among the individuals in the pareto front. The distance value of an individual is the sum of the distances of that individual with the others individuals in the pareto front. The distance between two individuals is obtained through the diversity in the values of the objectives. If the same objective in both individuals has different values the distance between those individuals is increased one unit, thus the maximum distance between two individuals is the number of objectives and the minimum is 0.

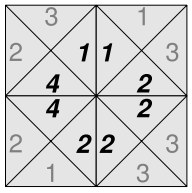
For the elitism, instead of copy all individuals of the pareto front from one generation to the next one, MOEA have a maximum number of individuals that can be copied to the next generation. The MOEA only copies the individuals in the pareto front and it starts coping the individuals with the higher value in their distances. If there are more individuals in the pareto front than the maximum number of individuals that can be copied then starts copying the individuals with the higher value in their distances keeping the diversity in the population between generations. The more distance value in a individual means that individual is more different to others individuals with less distance value.

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

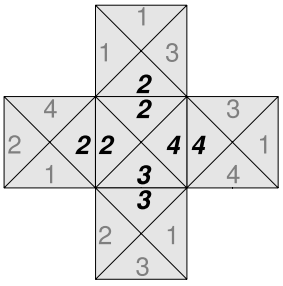
- *Objective 1*: The number of adjacent edges that match, excluding the grey sides of the tiles that must be placed in the border of the board, this is the score of the puzzle we said in *Section I* (Fig. 4(a)). This objective will be used to compare the results with the other two evolutionary algorithms. 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 matches in their adjacent inner sides (Fig 4(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. 4(c)). The maximum value is 256 and the minimum is 0.



(a) Objective 1: edge-matching



(b) Objective 2: a 2×2 region



(c) Objective 3: matched tile

Fig. 4. The three objectives of the MOEA

E. Fitness Function

For the experiments two types of fitness function have been used in the genetic algorithm and in the artificial immune evolutionary algorithm. In both algorithms the objective is to minimize the value returned by the fitness function, so that an individual with fitness equals to 0 is a puzzle solved.

The first fitness function is maximum score possible in a puzzle (480 for Eternity II) minus the score of the puzzle (equation 4). The score value is the same as the objective 1 in

the multiobjective algorithm: the number of adjacent sides of tiles that match. This fitness will be called the *normal fitness* in the rest of the document.

$$normal\_fitness = 480 - score \quad (4)$$

The second fitness is a combination of the three objectives in the multiobjective algorithm, so we called it the *combined fitness*. The idea of this fitness is to increase the domain information as in the multiobjective algorithm to make a comparison with this last algorithm. The combined fitness is the average of the normalized values of the objectives in the multiobjective algorithm. The equation to calculate this combined fitness is shown in equation 5 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.

$$combined\_fitness = 1 - \left(\frac{1}{k} \cdot \sum_{i=1}^{k} \frac{objective_i}{max\_objective_i} \right) \quad (5)$$

The time cost to calculate the combined fitness is three times higher than the normal fitness because the board have to be browsed three times instead of one, one per each objective.

F. Initial Population

As the problem has a big search space it was decided to create the initial population in two different ways. One randomly from scratch which we called *random initialization* and another with some domain information which we called *initialization with knowledge*. In the random initialization the bidimensional matrix is filled placing the tiles randomly with a random orientation, while in the initialization with knowledge the cells of the matrix are browsed trying to put the tiles in the right way. A more detailed description of the initialization with knowledge algorithm can be shown in algorithm 3.

The initialization with knowledge algorithm has a complexity of $O(M) = (M^2 + M)$ per individual where M is the number of tiles (256) while random initialization has a complexity of $O(M) = M$.

The individuals generated with domain knowledge have an average score of 301.932 with a standard deviation of 0.016, 352 of maximum and 250 of minimum. The individuals generated with random initialization have an average score of 18.654 with a standard deviation of 0.042, 40 of maximum and 5 of minimum. These results were generated over 10^5 experiments.

The initialization with knowledge is the improvement that has been applied to the exhaustive search explained in *Section III-A*.

Algorithm 3 Initialization With Knowledge

Require: list of $tiles$, a $path$ to browse the bidimensional matrix

```
1: for each individual in the population do
2:   tile_list ⇐ copy tiles
3:   randomize tile_list
4:   for each cell in path do
5:     tile ⇐ find the first tile that matches in the cell
       from tile_list
6:     if no tile fits then
7:       cell is empty
8:     else
9:       remove tile from tile_list
10:    end if
11:  end for
12:  for each empty cell in the path do
13:    fill the cell with a tile form tile_list
14:    remove the tile form tile_list
15:  end for
16: end for
```

IV. Experimental Results

In this section can be shown the setup of the experiments realized with the three evolutionary algorithms: genetic algorithm (GA), artificial immune evolutionary algorithm (AIEA) and multiobjective evolutionary algorithm (MOEA). Then the results of these algorithms will be shown and compared with an exhaustive search (ES) and random search (RAND) algorithms. The evolutionary algorithms have been executed with the two types of fitness functions and initialization of the initial population that have been explained in *Sections III-E* and *III-F*.

For the experiments the evolutionary algorithms were executed with different parameters. These parameters can be shown in table I. First column (Experiment) shows the experiment name. Second column (Alg.) is the algorithm used in that experiment where GA means genetic algorithm, AIEA means artificial immune evolutionary algorithm and MOEA means multiobjective evolutionary algorithm. Third column (Pop.) is the population size. Next three columns are the number of individuals that are generated for the next generation by crossover (Cross.), mutation (Mut.) and elitism (Elit.) respectively. Last column (Tour.) is the tournament size. An empty cell ('-') in the table means that parameter is not used in that algorithm.

TABLE I
ALGORITHM PARAMETERS

Experiment	Alg.	Pop.	Cross.	Mut.	Elit.	Tour.
GA_1	GA	10000	9000	999	1	3
GA_2	GA	10000	5000	4999	1	3
GA_3	GA	10000	7500	2500	0	3
AIEA_1	AIEA	10000	-	-	-	3
AIEA_2	AIEA	10000	-	-	-	20
AIEA_3	AIEA	1000	-	-	-	3
MOEA_1	MOEA	10000	9000	990	10	3
MOEA_2	MOEA	10000	5000	4990	10	3

For genetic algorithm there are three experiments. First

one has commonly used values for the crossover, mutation, tournament and elitism. Second experiment has the same ratio of mutation and crossover. And the last experiment has intermediate values between the other two experiments but without elitism.

For artificial immune evolutionary algorithm there are also three experiments. The first one has the same population and tournament size that the genetic algorithm experiments. In the other two experiments the selective pressure has been increased, in the second experiment the tournament size is higher and in the third experiment the population is lesser. With a higher tournament size the probability to choose a good individual is higher and with less population the probability to choose any individual is also higher, so the selective pressure is increased.

The two experiments of the multiobjective evolutionary algorithm have the same setup as the first two experiments of the genetic algorithm.

This experimental setup has been chosen to cover different situations, we want to know an initial results of the algorithms in this problem. Instead of prove more parameters, we prefer using bigger populations and more number of evaluations for two reasons: avoid the problem of fast convergence problem with small populations and give enough time to the algorithms to converge.

Each experiment was executed 10 times and each time it were running until $5 \cdot 10^7$ evaluations, where evaluation means each time the fitness of an individual is obtained. The results of all experiments can be shown in table II. First column (Experiment) shows the name of the experiments, furthermore the experiments showed in table I there are two more experiments: a random search algorithm (RAND) and a exhaustive search algorithm (ES). The second column (Init.) shows the sort of initialization used to create the initial population: random (rand) or initialization with knowledge (iwk). Third column shows the fitness that has been used: normal fitness (norm) or combined fitness (comb). Empty cells ('-') in these two last columns mean that the algorithm does not use that sort of initialization or fitness. In the next three columns are shown the minimum (Min.), maximum (Max.) and mean (Mean) value of the score 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.

TABLE II
RESULTS

Experiment	Init.	Fitness	Max.	Min.	Mean	Std. Dev.
RAND	-	-	48	44	46.3	1
ES	-	-	371	358	364.4	4.36
GA_1	rand	norm	303	278	292.5	8.56
GA_2	rand	norm	169	152	159.7	4.78
GA_3	rand	norm	105	63	90.2	12.79
GA_1	iwk	norm	394	382	385.9	3.33
GA_2	iwk	norm	358	348	351.3	2.76
GA_3	iwk	norm	343	334	337.9	3.08
GA_1	rand	comb	352	334	345.8	4.92
GA_2	rand	comb	338	144	214.7	78.02
GA_3	rand	comb	364	344	355.9	6.36
GA_1	iwk	comb	387	377	382.3	3.26
GA_2	iwk	comb	391	341	355.3	12.97
GA_3	iwk	comb	374	363	367.4	3.85
AIEA_1	rand	norm	236	224	230.1	3.56
AIEA_2	rand	norm	272	260	266.5	3.64
AIEA_3	rand	norm	285	274	278.3	3.16
AIEA_1	iwk	norm	373	364	369.4	3.14
AIEA_2	iwk	norm	385	374	379.8	3.16
AIEA_3	iwk	norm	379	373	375.9	1.81
AIEA_1	rand	comb	244	218	229.5	8.89
AIEA_2	rand	comb	292	270	279.5	7.49
AIEA_3	rand	comb	307	383	292.5	6.42
AIEA_1	iwk	comb	372	361	366.4	3.44
AIEA_2	iwk	comb	383	372	378.4	3.35
AIEA_3	iwk	comb	376	365	372.3	3.72
MOEA_1	rand	-	365	346	356.7	5.98
MOEA_2	rand	-	364	346	358.3	4.9
MOEA_1	iwk	-	394	382	387.6	2.94
MOEA_2	iwk	-	**396**	**388**	**392.5**	2.66

V. CONCLUSIONS AND FUTURE WORKS

In all the evolutionary algorithms the results are better when a initialization with knowledge is used instead of the random one. With random initialization MOEA and GA with combined fitness are the only ones that have similar results as the exhaustive search. When the initialization with knowledge is used the results are better than the exhaustive search. Furthermore, this kind of initialization reduces the difference among the results of the different experiments with the same algorithm.

GA and AIEA with random initialization get better results with the combined fitness, although more significant in the GA. However these algorithms with initialization with knowledge have the same results with both fitness functions. We can conclude about the fitness that the combined is only useful in the random initialization, this is especially significant in the GA which results with this fitness are close to the MOEA with random initialization. We also have to mention that the results of GA with random initialization and normal fitness and AIE with random initialization and both fitness functions have worse results than the algorithm that creates the initial population with domain information (initialization with knowledge).

In the GA, we get better results in the first experiment (GA_1) with 90% of crossover 10% of mutation and one individual generated from elitims. But we must remark here that in the third experiment (GA_3) the results suffer a big improvement with the combined fitness. With random initialization and normal fitness the results are near the random search but with the combined fitness the results are close the exhaustive search.

For the AIEA we only can remark that results are better with more selective pressure. The experiments with a higher tournament size or less population have better results than the first experiment (AIEA_1).

In the MOEA the results of the two experiments are

very close. Furthermore, MOEA with initialization with knowledge gets the best results of all algorithms, we satisfied 396 of the 480 constraints.

If we compare our results with those obtained in other papers, we are very far from the 458 over 480 that has been got by P. Schaus [12]. But we must bear in mind that we have used generic evolutionary search algorithms without any kind of heuristic in our crossover and mutation operators. So the future work for this problem will point out in three directions:

- improve the mutation and crossover operators with some kind of heuristic that bear in mind the well formed regions
- include a local search to improve the individuals between generations
- using GA or AIEA with individuals that have an array of tiles as chromosome which order is used to realize an exhaustive search during a fixed time, and the result of that search would be the fitness function
- conduct more experiments with differents setups

ACKNOWLEDGMENT

This work was supported in part by the University Carlos III 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)," *http://www.eternityii.com*, November 2008.
[2] E. Demaine and M. Demaine, "Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity," *Graphs and Combinatorics*, vol. 23, pp. 195–208, 2007.
[3] R. Dechter, *Constraint Processing*. Morgan Kaufmann, 2003.
[4] V. Raman, B. Ravikumar, and S. Rao, "A simplified NP-complete MAXSAT problem," *Information Processing Letters*, vol. 65, no. 1, pp. 1–6, 1998.
[5] S. Rana and D. Whitley, "Genetic algorithm behavior in the MAXSAT domain," *Proc. of PPSN-V*, pp. 785–794, 1998.
[6] M. Sipser, "The history and status of the P versus NP question," *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pp. 603–618, 1992.
[7] J. Gottlieb, E. Marchiori, and C. Rossi, "Evolutionary Algorithms for the Satisfiability Problem," *Evolutionary Computation*, vol. 10, no. 1, pp. 35–50, 2002.
[8] A. Eiben and J. van der Hauw, "Solving 3-SAT with adaptive Genetic Algorithms," *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pp. 81–86, 1997.
[9] L. de Castro and F. Von Zuben, "Artificial Immune Systems: Part I–Basic Theory and Applications," Universidade Estadual de Campinas, Dezembro de, Tech. Rep., 1999.
[10] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.
[11] V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey," *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992.
[12] Y. D. Pierre Schaus, "Hybridization of CP and VLNS for Eternity II," *JFPC*, 2008.
[13] T. Bäck, A. Eiben, and M. Vink, "A Superior Evolutionary Algorithm for 3-SAT," *Proceedings of the 7th International Conference on Evolutionary Programming VII*, pp. 125–136, 1998.
[14] J. Gottlieb and N. Voss, "Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, vol. 1498, pp. 755–764, 1998.
[15] J. Gottlieb and Voss, "Adaptive fitness functions for the satisfiability problem," *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pp. 621–630, 2000.
[16] E. Marchiori and C. Rossi, "A flipping genetic algorithm for hard 3-SAT problems," *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 1, pp. 459–465, 1999.
[17] C. Rossi, E. Marchiori, and J. Kok, "An adaptive evolutionary algorithm for the satisfiability problem," *Proceedings of the 2000 ACM symposium on Applied computing-Volume 1*, pp. 463–469, 2000.
[18] D. DasGupta, *Artficial Immune Systems and Their Applications*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1998.
[19] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, 1994, pp. 202–212.
[20] L. de Castro and F. Von Zuben, "Artificial Immune Systems: Part II A Survey Of Applications," Department of Computer Engineering and Industrial, Tech. Rep., 2000.