Solving the Eternity II Puzzle using Evolutionary Computing Techniques

Papa Ousmane Niang

A Thesis

In

The Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

December 2010

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Papa Ousmane Niang

Entitled: Solving the Eternity II Puzzle using Evolutionary Computing Techniques

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Dongyu Qiu_____Chair

Dr. Abdelwahab Hamou-Lhadj__ Examiner

Dr. Peter Grogono_____ Examiner

Dr. Nawwaf Kharma_____ Supervisor

Approved by Dr. William E. Lynch_____
Chair of Department or Graduate Program Director

January 20, 2011 Robin A. L. Drew_____
Dean of Faculty

ABSTRACT

Solving the Eternity II Puzzle Using Evolutionary Techniques

Papa Ousmane Niang

The work presented in this thesis describes the application of genetic algorithms to solve an edge-matching puzzle known as Eternity II (E2). E2 is a hard combinatorial puzzle that is commercially available and for which a solution has not yet been found (December 2010). There are thousands of ways that E2 can be solved to win the prize of $US 2.000.000 promised by the company to the first person who solves the puzzle. The puzzle consists of 256 square pieces that are bordered by colored patterns which must be aligned across the whole puzzle. E2 is an NP-complete and multi-constrained combinatorial problem that has received a lot of attention worldwide.

This thesis proposes a framework and a new approach for solving complex combinatorial optimization problems, such as edge-matching puzzles like the E2.

The proposed solution uses a hybrid method composed of Evolutionary Programming (EP) and Constraint Satisfaction Problem (CSP) techniques.

We draw comparisons between the state-of-the-art methods that have been used so far to try to solve the E2 puzzle and our proposed method, and show the advantages of using genetic algorithms to solve edge-matching puzzles in general.

ACKNOWLEDMENTS

I want to express my deeply-felt thanks to my thesis supervisor, Dr. Nawwaf KHARMA, for his warm encouragement and thoughtful guidance.

I would like to thank Dr. Peter Grogono for his inspiration in the fields of Programming Languages and Artificial Intelligence.

I also would like to thank the administration staff of the Electrical and Computer Engineering department, particularly Mrs. Diane Moffat.

Finally, I would like to thank my family for their support during this academic journey.

DEDICATION

To my parents

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

E2 – Eternity II

GA – Genetic Algorithm

TSP – Traveling Salesman Problem

SP – Scheduling Problem

CVRP – Capacitated Vehicle Routing Problem

SAT – Satisfiability Problem

CSP – Constraint Satisfaction Problem

PLA – Partial Look Ahead

MAC – Maintaining-Arc Consistency

VNS – Variable Neighborhood Search

NP-C – NP-Complete

NP – Non-deterministic Polynomial

1. Introduction

The Eternity II (E2) puzzle is a commercial edge-matching puzzle that was created by two mathematicians at Oxford University, namely Alex Selby and Oliver Riordan [Toulis, Tomy]. It is an extremely difficult puzzle with a reward of US$2.000.000 for the first person to submit a solution by December 31, 2010. As of this writing, a solution had not been found.

The publication of the puzzle and the prize reward generated a lot of attention worldwide.

An edge-matching puzzle is a type of tiling puzzle similar to a Jigsaw puzzle that first appeared in the 1890s [Haubrich]. The edges of the tiles are colored or filled with different patterns. In order to solve the puzzle, all tiles must be placed in such a way that all edges of adjacent tiles match. In Jigsaw puzzles, only one solution is expected. All tiles must fit exactly in one place in order to constitute the final image. Edge-matching puzzles are harder and more challenging than Jigsaw puzzles because they don't have a guiding image. The final tile placement is known only once the puzzle has been solved, because a tile can fit in many ways. Additionally, the complexity is increased with the number of patterns and the size of the tiles.

The E2 puzzle is an edge-matching puzzle that is made of 256 unique tiles. Twenty-three (23) patterns are used to decorate the tiles. Each tile has a specific combination of four (4) patterns. The tiles that must be placed on the borders have the edges touching the border of the grid colored in grey. The patterns used to decorate the tiles are shown in figure 1.

Figure 1 - Patterns used in Eternity II

The problem is defined as follows:

Place all 256 tiles on the grid, such that all tiles match along their edges. The tiles can be rotated (90°, 180° and 270°) before being placed on the board.

A quick analysis of the puzzle shows that it is an extremely difficult combinatorial problem.

The effective branching factor for a problem of this size is about 382, which means that the A* search method heuristic would have to consider 382 children nodes for each node it visited [Toulis]. To give the reader an idea of the size and complexity of the problem, the game of chess has an effective branching factor of 100.

On an n x n board, the number of edges that need to be matched is given by the following formula:

$$N = 2n(n+1)$$

Therefore, on a 16x16 board, which is the size of the E2 puzzle, 544 edges will have to be matched to solve the puzzle. Given that the border and corner tiles are easily identifiable, that number can be reduced to 480 (N - 64)[1].

To give the reader a rough idea of the search space, there are 4! ways to place a corner tile, 56! ways to place a border and $196! * 4^{196}$ ways to place internal tiles.

[1] In a 16x16 configuration, there are 16 tiles on each border. The total number of border tiles is therefore 4x16 = 64.

Therefore the size of the search space is approximately $196! * 4^{196}$. If we're taking into consideration the hint piece[2] we have $195! * 4^{195}$ possible combinations.

Solving the E2 puzzle is clearly hard and computationally challenging. It will involve research in the areas of algorithms, parallel computing, software engineering, image processing, pattern matching, etc… We believe that a lot of practical applications will benefit from the results of this experiment. Figure 2 illustrates a solved 5x5 grid using E2 patterns.

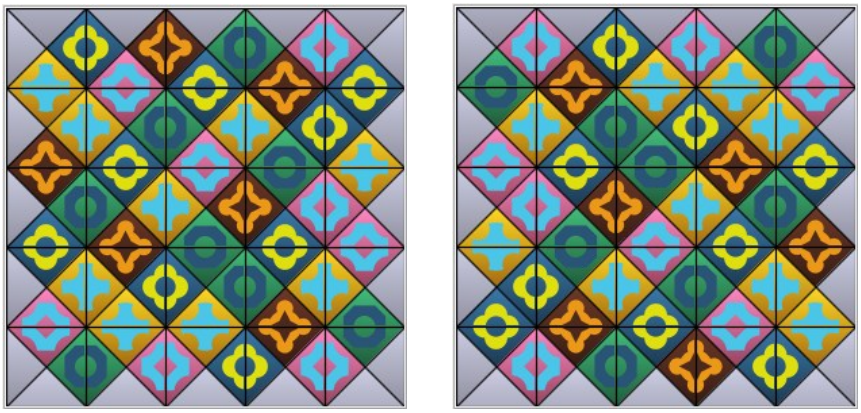

Figure 2 – Two different images of a 5x5 board with E2 patterns generated by the Eternity II Editor

Edge-matching puzzles have been studied widely. They are hard combinatorial optimization problems that are classified as NP-Complete and, in general, there is no efficient algorithm that can solve them [Demaine].

Several researchers (in and outside academia) have attempted to solve the E2 puzzle using different meta-heuristics and techniques. Most of the empirical results that were obtained did not meet the requirements. The number of edges matched ranged between 396 and 459 out of 480, which is the total number of edges that must be matched in order to have a solution to the puzzle.

---

[2] The hint piece is tile number 139, which actually divides the board in 4 regions.

For instance, [Anso] applied state-of-the-art Satisfiability Problem (SAT) and Constraint Satisfaction Problem (CSP) techniques to the problem. They used competitive SAT solvers, Partial-Look-Ahead (PLA) and Maintaining-Arc Consistency (MAC) algorithms, which are known to be very efficient search heuristics and have been used for years to solve highly constrained problems. The benefit of MAC and PLA is that they can increase the depth of the search, but this is done at a high cost [Toulis]. However, their solution could not solve an 8x8 puzzle.

The results obtained are understandable because the solvers will often run into situations similar to the one depicted in figure 3. The solvers will have to backtrack and restart. SAT solvers were also used by [Heule] without noticeable improvements.

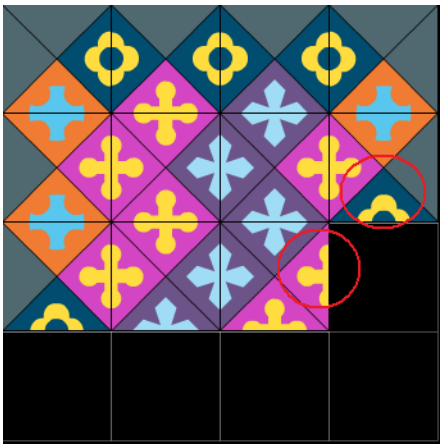

Figure 3 - Dead end reached (Picture is from [Toulis])

Several local search algorithms were applied separately or combined by [Toulis] without any major improvements. [Toulis] was however able to get good results (455/480) with hybrid algorithms, namely a meta-heuristic called Variable Neighborhood Search (VNS) implemented using a fitness function, swaps and rotation sequences. This method was also used by [Coelho] and did not generate better results.

Other hybrid variations of local search meta-heuristics were also used without generating results that were close to goal of 480 matches [Wang].

We found only one implementation that used evolutionary techniques to solve E2. Without any additional heuristics, [Munoz] was able to obtain a score of 396 out 480, which is still lower than most of the scores obtained.

The best results so far have been achieved by [Schaus] and [Vancro]. [Schaus] used a combination of tabu search and very large neighborhood search and was able to obtain 458 out of 480. [Vancro] used hyper-heuristics which are recent trend in heuristic algorithms and obtained a score of 459 out 480. His solution included a DFS (Depth First Search), Tournament Selection and lower level heuristics, such tile rotation, tile exchange, and very large neighborhood search.

We can clearly see that the solutions that produced the best results were a combination of hyper and meta-heuristics. However, they have failed to find a solution to the E2 puzzle.

There appears to be no explanation as to why these methods did not succeed in finding a solution. The number of solutions estimated by [Anso, Toulis, Anonymous] is quite large (around $15x10^6$).

We believe that most these approaches failed because their search was performed on localized areas ignoring completely the cohesiveness of the final appearance of the board. We also noticed that the hint peace divides the board in four regions depicted in figure 4. This could turn out to be an interesting observation for our research.

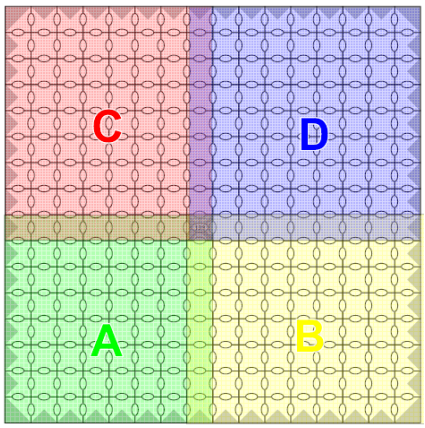

Figure 4 - E2 board with the hint piece divided in 4 regions

Given the enormous space that has to be considered in this problem, we believe that, the application of evolutionary techniques and other meta-heuristics can improve the results obtained by the other researchers. Furthermore, [Demaine] proved that a Jigsaw puzzle and an edge-matching puzzle are computationally the same. Jigsaw puzzles have been solved successfully using evolutionary techniques [Toyama, Gindre]; therefore a correct representation of the problem will increase our chances to find a solution to the E2 puzzle.

In this thesis, we plan to present a new approach to solve the E2 puzzle using evolutionary techniques. The study will provide a model for improving the results presented in [Munoz] and expand the options available to others researchers.

The main contributions of this thesis are:

- An intelligent crossover operator for exchanging genetic material without disrupting too much the structure of the new off-springs.
- Several new and adapted mutation operators that can be combined to achieve good results.
- A software design with an optimal structure for implementing computer programs to run the experiments.

2. Literature Review

The research conducted for this thesis revolved around three major areas: evolutionary techniques for solving complex combinatorial problems, genetic algorithms for solving NP-Complete problems and effectively solving edge-matching puzzles using computers. The literature dealt mostly with complex combinatorial problems and evolutionary techniques. We will provide short summaries of our findings in the following paragraphs.

Several of the books and articles that we reviewed discussed at length the use of evolutionary techniques to solve complex combinatorial problems. The information that we collected was very important to our research since it revealed that evolutionary techniques could be considered to solve the E2 puzzle [Munoz]. We also found that the use of other meta-heuristics could lead us to a global optimal solution [Affen]. The traditional methods of search only find local optima [Jourdan].

Most of the articles that we reviewed considered genetic algorithms (GA) to be among the most powerful optimization heuristics for solving complex combinatorial optimization problems [Eiben, Affen, Toyama].

A combinatorial optimization problem of the size of the E2 puzzle requires the robustness of a genetic algorithm and its ability to perform efficiently on large complex search spaces. Furthermore, GAs are also known for their ability to find global optima in large search spaces [Toyama].

The E2 puzzle belongs to the class of NP-Complete problems [Demaine]. We realized while reviewing the literature that it was also important to understand how GAs performed, as a generic method, on problems that were computationally intractable – NP-Complete.

[DeJong] concluded in a published paper titled "Using Genetic Algorithms to Solve NP-Complete Problems," that genetic algorithms were a robust and effective search heuristic for NP-Complete problems.

All authors agreed that NP-Complete problems were very challenging and hard to solve. In general, there are no efficient algorithms that can solve these problems [Demaine].

The puzzle problem that we studied in our research is a complex combinatorial problem with an extremely large search space that falls into the category of edge-matching puzzles.

[Demaine] states in their work that edge-matching puzzles and jigsaw puzzles are computationally equal. Even though, edge-matching are much harder than jigsaw puzzles, we reviewed the work of several researchers who attempted to solve jigsaw puzzles using genetic algorithms [Gindre, Toyama].

Much of the literature that discussed the E2 puzzle proposed problem-specific algorithms as an effective way of attempting to solve the puzzle. None of the algorithms that were applied to the puzzle found a solution [Toulis]. As of this writing, there are no known solutions to the problem that were submitted.

We also observed that most authors considered defining the E2 puzzle as a Constraint Satisfaction Problem (CSP), which is a natural choice for an edge-matching puzzle given that combinatorial optimization problems can be modeled as constrained optimization problems [Toulis]. Therefore, a GA can combine its search abilities and the use of constraints definitions to find optimal solutions. We found that this method was poorly addressed in the literature, but were encouraged by the results obtained by [Munoz]. [Munoz] transformed the constraints into optimization objectives and used the

optimization power of genetic algorithm to achieve those objectives. We believe that [Munoz] could have achieved better results by additionally using other meta-heuristics and/or biased operators.

In most of the literature discussing the use of evolutionary techniques to solve constrained problems, the following were determining factors in constructing a successful and efficient GA: the problem definition, the representation of solution candidates and the crossover operator.

The literature also revealed that the recent publication of the E2 puzzle generated a lot of interest in academic circles [Anso]. It is an opportunity for researchers to investigate new solutions to old problems and to revisit existing ones.

The literature provided sufficient information on the use of evolutionary techniques to solve complex combinatorial problems, but did not elaborate on the techniques that could be used to solve edge-matching puzzles using genetic algorithms.

3. Problem Statement

The Eternity II (E2) puzzle contest generated a lot of interest in the scientific community [Anso]. The E2 puzzle can be classified as an edge-matching puzzle which is considered NP-Complete (NP-C) [Demaine]. Problems that are categorized as NP-C are known to be the hardest to solve. Therefore, a simple search algorithm will not produce a solution.

The objective of the game is to place 256 unique patterned tiles on a board, such that all touching pairs of edges match (fig. 5 and fig. 6). The tiles with the grey pattern must be placed around the border. Many attempts have been made where local search meta-heuristics such as Tabu search, Simulated Annealing, Hill Climbing and hyper-heuristics [Anso, Toulis, Heule, Schaus] were used, without finding a solution. As of today's writing the company that produced the game has not reported a winner.

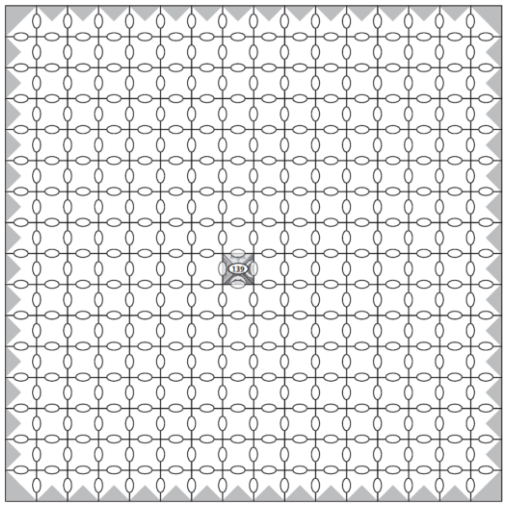

Figure 5 - E2 puzzle solution sheet

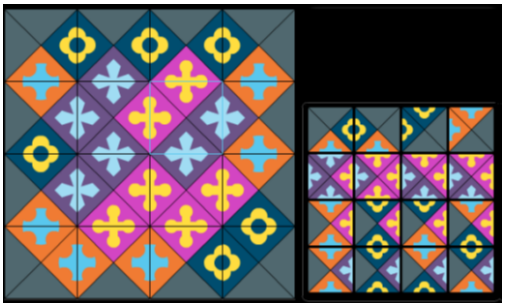

Figure 6 - E2 Online demo puzzle solved [Tomy]

Hill Climbing is an old, simple and fast local search method. It is a good iterative algorithm for finding local optima. The search starts with an arbitrary solution and tries to find a better solution. The search stops when all neighboring candidate solutions are worse than the current solution. The algorithm is show below:

1. Construct a sub-optimal solution that meets the constraints of the problem
2. Take the solution and make an improvement upon it
3. Repeatedly improve the solution until no more improvements are necessary/possible

Algorithm 1 - Hill Climbing Algorithm

With Tabu Search [Glover] the local or neighborhood search procedure looks for a potential solution and once it finds it tags it as "tabu" to prevent the procedure from revisiting it. It enhances the performance of local search meta-heuristics by using memory structures to store the candidate solutions already visited. The algorithm of the Tabu search is shown below:

```
k := 1.
generate initial solution
WHILE the stopping condition is not met DO
   Identify N(s). (Neighbourhood set)
   Identify T(s,k). (Tabu set)
   Identify A(s,k). (Aspirant set)
   Choose the best s' Î  N(s,k) = {N(s) - T(s,k)}+A(s,k).
   Memorize s' if it improves the previous best known solution
   s := s'.
   k := k+1.
END WHILE
```

Algorithm 2 - Basic Tabu Search Algorithm

Simulated Annealing [Kirkpatrick] is a generic probabilistic meta-heuristic that searches for a good solution, rather than the best possible solution. It tries to locate a solution close to the global optimum. The algorithm used in simulated annealing is described below:

```
s ← s0; e ← E(s)                          // Initial state, energy.
sbest ← s; ebest ← e                      // Initial "best" solution
k ← 0                                     // Energy evaluation count.
while k < kmax and e < emax               // While time left & not good enough:
    snew ← neighbour(s)                   // Pick some neighbour.
    enew ← E(snew)                        // Compute its energy.
    if P(e, enew, temp(k/kmax)) > random() then   // Should we move to it?
    s ← snew; e ← enew                    // Yes, change state.
    if enew < ebest then                  // Is this a new best?
      sbest ← snew; ebest ← enew          // Save 'new neighbour' to 'best found'.
  k ← k + 1                               // One more evaluation done
return sbest
```

Algorithm 3 - Simulated Annealing Algorithm

Most of the local search meta-heuristics work in a very small search area and habitually only find local minima and rarely global optima. Finding global optima comes at a high cost, reducing the ability of such algorithms to perform well in large search spaces.

In this thesis we will focus particularly on Evolutionary Algorithms, specifically on Genetic Algorithms and demonstrate that population based algorithms have a better chance to succeed with this type of problem.

Genetic algorithms (GA) have been successfully used to solve a great number of complex combinatorial optimization problems where the search space can be very large [Oliveto]. They were formally introduced in the United States in the 1970s by John Holland at University of Michigan.

A genetic algorithm is a stochastic search technique that takes its inspiration from Darwin's theory of evolution. The fittest individuals in a population survive and reproduce, passing on their traits to future generations. This phenomenon is known as the survival of the fittest.

A genetic algorithm follows an iterative process and usually operates on a population of constant size. Solution candidates or chromosomes are added to a population randomly or using heuristics. Solution candidates are then evaluated and given a fitness value. This evaluation occurs for every generation. In each generation, a new population is created by selecting individuals according to their fitness value and then allowing them to exchange genetic material to create new off-springs, who will become the next generation of parents. In order to produce new chromosomes, GAs use two genetic operators, namely crossover and mutation [Affen].

Crossover is the most important genetic operator. It combines the parts of two parents to create off-springs.

Mutation is an important operator that is essentially used to modify the genetic composition of an individual to encourage exploration and avoid premature convergence.

The general and basic form of the algorithm is given below. It is straightforward to apply and has performed very well on complex problems.

Initialize population with random candidate solutions
Evaluate each candidate

```
while termination criterion has not been reached
{
        Selection;
        Reproduction;
        Crossover;
        Mutation;
        Evaluation;
}
```

Algorithm 4 - Basic form of the Genetic Algorithm

What makes GA unique compared to neighborhood-based search heuristics is that during crossover solutions candidates can inherit properties that may be located in different areas of the search space. This unique feature of GAs makes them much robust in avoiding stagnation in local optima.

There are a number of constraints that must be satisfied in order solve the E2 puzzle (see introduction). Satisfying these constraints manually or using local search heuristics can be a daunting and even an impossible task. In E2, the size of the search space is approximately $195! * 4^{195}$, which represents a huge search space. With the use of genetic algorithms and constraint handling techniques as defined in Constraint Satisfaction Problems (CSP), we are able to explore many solutions and determine the ones that are valid and optimal.

The problem is defined as follows:

1. Each tile has a North (N), East (E), South (S) and West (W) edge.
2. Each edge is decorated with a pattern.
3. Each tile can be rotated 90, 180 and 270 degrees.
4. If a tile is located on the frame, the edges touching the frame are colored in grey.

5. Given a tile t, an internal tile; t_n, the tile located north of t; t_e, the tile located east of t; t_s, the tile located south of t; t_w, the tile located west of t; the following must be true:
    a. The North edge of t must match the South edge of t_n.
    b. The East edge of t must match the West edge of t_e.
    c. The South edge of t must match the North edge of t_s.
    d. The West edge of t must match the East edge of t_w.

Figure 7, 8, 9 and 10 illustrate the rules described above.

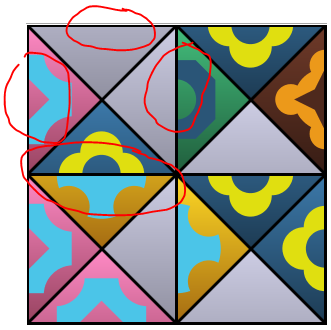

Figure 7 – Non-matching edges

Figure 8 - 2x2 Area not matching

Figure 9 - Matching edges

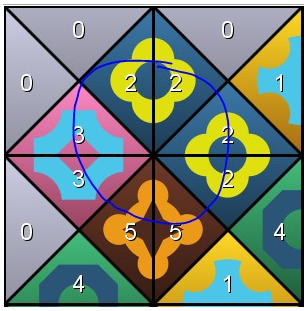

Figure 10 - Matching 2x2 Area

4. Methodology

The heuristics and algorithms used to solve the problem described in the previous section are explained in detail.

Given the constrained nature of the Eternity II (E2) puzzle, we decided to use a hybrid solution that uses a mixture of Evolutionary Programming (EP) and Constraint Satisfaction Problem techniques [Craenen, Gindre, Gottlieb].

We use the canonical Genetic Algorithm (GA) definition to guide the search for a solution.

```
Initialize population with random candidate solutions
Evaluate each candidate
while termination criterion has not been reached
{
        Selection;
        Reproduction;
        Crossover;
        Mutation;
        Evaluation;
}
```

Algorithm 5 - The Canonical GA Algorithm

Our objective is to demonstrate that the E2 puzzle can be solved using evolutionary techniques. In order to achieve that, we designed and implemented an efficient computer program based on our genetic algorithm. The program was optimized to run for hours with a steady use of computer resources. The problem description and the discussion of our GA coding standards follow.

4.1 Description of the problem as a CSP

E2 can be naturally classified as a Constraint Satisfaction Problem and formulated as follows:

Problem = {T, P, N, E, S, W, and O}, where:

T_{ij} = {1, t2 …, n^2} is a set of tiles,

P = {0, p1…, p} is a set of patterns,

N_{ij} = {1, c2 …p} is a set of constraints belonging to variables,

E_{ij} = {1, o1…p} is a set of constraints belonging to variables,

S_{ij} = {1, q1 …p} is a set of constraints belonging to variables,

W_{ij} = {w1, w2 …p} is a set of constraints belonging to variables,

O_{ij} = {0…3} is set representing the orientation.

c = {1, 2…m} set of constraints associated with edges

C = {1, 2…x} set of constraints associated with 2x2 squares

Where the edge-matching constraints are expressed as:

$i, j \in NxN$, $N_{i,j} = S_{i-1,j}$, $E_{i,j} = W_{i,j+1}$, $S_{i,j} = N_{i+1,j}$, $W_{i,j} = E_{i,j-1}$

And the penalties as:

$$f_1(\bar{s}) = \sum_{1}^{m} w\ x\ \varphi(\bar{s}, c_i),\ where\ \varphi(\bar{s}, c_i) = \begin{cases} 1\ if\ \bar{s}\ violates\ c_i \\ 0\ Otherwise \end{cases}$$

respectively,

$$f_2(\bar{s}) = \sum_{1}^{x} w\ x\ \varphi(\bar{s}, C_i),\ where\ \varphi(\bar{s}, C_i) = \begin{cases} 1\ if\ \bar{s}\ violates\ C_i \\ 0\ Otherwise \end{cases}$$

For each $\bar{s}\ \in S, \varphi(\bar{s}) = true, only\ if\ f_1(\bar{s}) = 0\ and\ f_2(\bar{s}) = 0$

4.2 Individuals and Population

A population is collection of individuals which are defined as n x n boards. Each board is filled using n x n 4-patterned tiles.

4.3 Representation (Encoding)

A board is a 2-dimensional structure that contains N x N tiles. Each tile is represented as a vector containing five (5) integers. The first four (4) integers represent the four edges

of the tile and the last one is used to maintain the information about the orientation of the tile.

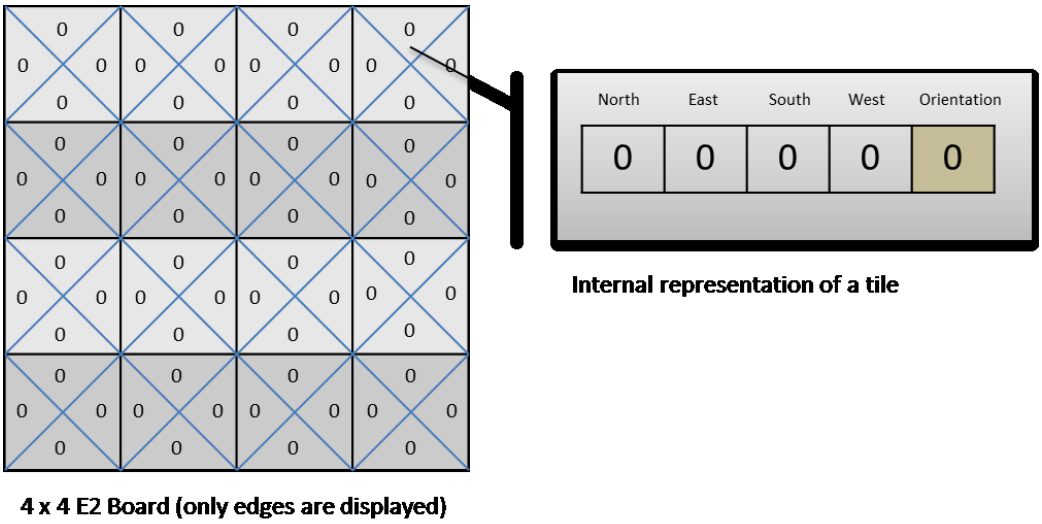

Figure 11 Internal representation of a tile

For example, 1 – 4 – 5 – 6 – 0 represents the following tile:

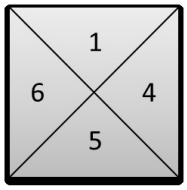

Figure 12 Tile before rotation

And 1 – 4 – 5 – 6 – 1 represents the following rotated tile:

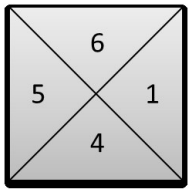

Figure 13 Rotated tile

4.4 Selection operators

4.4.1 Fitness Proportional Selection (Roulette Wheel)

This is the classical selection method that has been proposed by [Holland]. This method is also known as the Roulette Wheel. Each candidate is represented on the wheel according to its probability to be selected. If a candidate i, has a fitnessfi, then its selection probability is $\frac{fi}{\sum_{i=0}^{n} fi}$, where n is the size of the population.

This type of selection favors individual with the best fitness, even though they can be eliminated. When the fitness values are close to each other, there is almost no selection pressure. It is as if we were selecting individuals randomly [Eiben].

We decided not to use the method of selection, because of the poor results. The candidate solutions need to retain the same number of tiles without duplicate. Under this constraint most candidates will have fitness values that close together.

4.4.2 Linear Ranking Selection

This selection method is known to keep the selection pressure constant. It tries to correct some of the drawbacks observed with Fitness Proportional Selection (high selection pressure) [Eiben].

In Linear-Ranking Selection, the individuals of the population are sorted based on their fitness and assigned a selection probability calculated using the following formula for the linear ranking scheme:

$$P_{lr}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}, where\ 1.0 < s \leq 2.0\ and\ \mu = \lambda$$

The selection is then based on their rank rather than their fitness values, with the intent of reducing the dominating effect of individuals with the best fitness values [Affen].

4.4.3 Tournament Selection

The Tournament Selection is one of the most used selection scheme in modern applications of GAs. It is easy to implement and apply and does not rely on global knowledge of the population, which could be very costly to obtain [Eiben]. *k* individuals selected from the population compete based on their fitness values and the best candidate is selected and added to the mating pool. The advantage of this selection method is that the selection pressure can be scaled easily by altering *k* [Affen].
We observed great improvement with this method and decided to use it as part of our selection heuristics.

The following algorithm was implemented as part of the experiments.

```
Begin
    Set current_member = 1;
    While (current_member < μ)
        Pick k individuals randomly, with or without replacement;
        Select the best of these k individuals comparing their fitness values;
        Denote this individual as i;
        Set mating_pool[current_member] = i;
        Set current_member = current_member + 1;
    End While
End
```

Algorithm 6 – Tournament Selection Algorithm

4.4.4 Elitism

The Elitism replacement scheme is very popular with combinatorial optimization problems. The best candidates of the last generation are kept in the newly created population by replacing the individual with the worst fitness value.

We experimented mostly by keeping the best individual of the previous generation (n-elitism, with n=1). This is also referred to as the "golden cage model".

4.5 Mutation Operators

We experimented with several mutation schemes, either by using them individually or by combining them. We describe the most effective ones below.

4.5.1 Rotate Mutation

This rotation mutation operator simply rotates the tile clockwise.

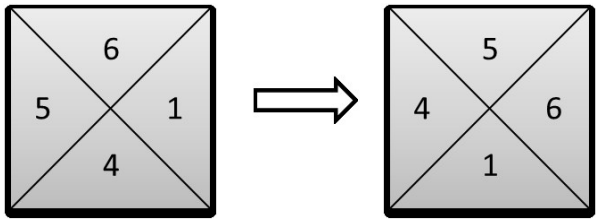

Figure 14 - Tile appearance after a Rotate Mutation

4.5.2 Swap Mutation

The Swap Mutation is one of the most natural methods for combinatorial problems. Two (2) tiles are randomly selected from the board and exchanged. This basic operator did not contribute to change the fitness value of the individual and we did not notice and improvement in the overall fitness of the population. The operation is illustrated in figure 15, where tile #1 and tile #11 have been swapped.

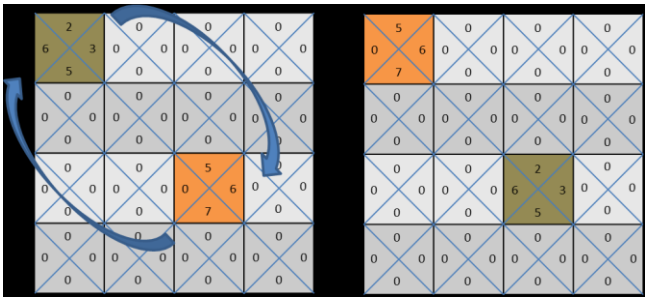

Figure 15 - Board appearance after a Swap Mutation

4.5.3 Swap & Rotate Mutation

This is a combination of the Rotate and Swap mutation operators described above. Two (2) tiles are randomly selected from the board, rotated and swapped. This operation is illustrated in figure 16.

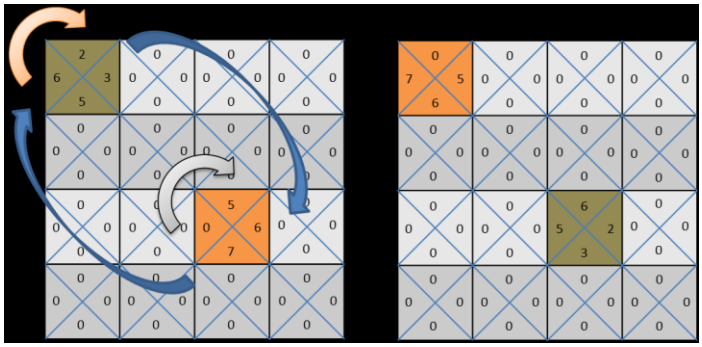

Figure 16 - Board appearance after a Swap & Rotate Mutation

4.5.4 Rotate Region Mutation

The Rotate Region Mutation schema is described in [Munoz]. A square region with a minimum width of two (2) is selected on the board and rotated. This mutation operator outperformed all other mutation operators that were used in the experiments. At the board assembles and collates matching pieces, a mutation operator should not disturb that order. The region rotation allows the board to try a different combination and keep the sub-region consistency. The operation is illustrated in figure 17.

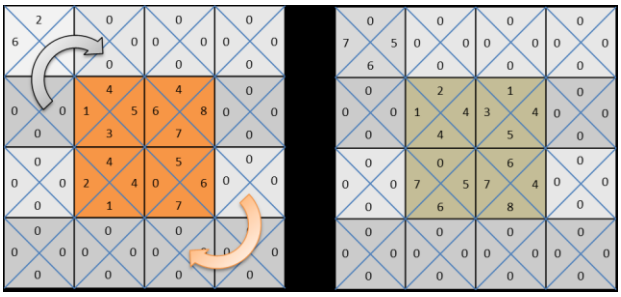

Figure 17 - Board appearance after a Region Rotation Mutation

4.5.5 Swap Region Mutation

The Swap Region Mutation schema is described in [Munoz]. A slight modification was applied to our implementation. We used vertical and horizontal swapping. The board is divided in two halves vertically or horizontally, and a region of the same size is selected in each half and swapped. The operation has the advantage of preserving blocks of matching tiles. It is described in figure 18.

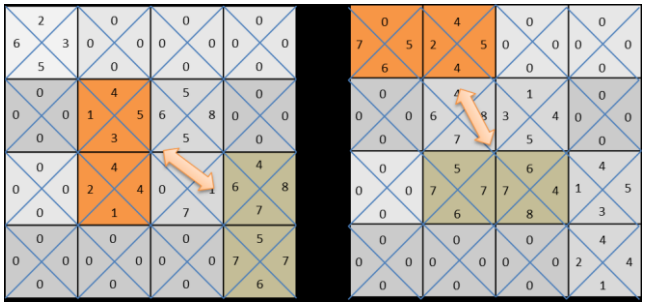

Figure 18 - Swap Region Mutation

4.5.6 Region Inversion Mutation

The Region Inversion operator follows the principle used in the well know Inversion Mutation scheme. All tiles collected from a randomly selected region on the board are copied back to the same region starting with the last tile in the block. The first tile is always at the top left corner of the region and the tiles are copied for left to right. This operator's functioning is depicted in figure 19.

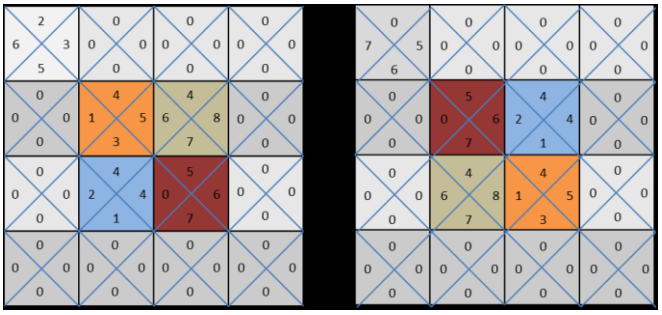

Figure 19 - Region Inversion Mutation

4.5.7 Row and Column Inversion Mutation

The Column and Row mutations schemes are similar to the Inversion Mutation operator defined in [Eiben]. In order to selectively apply mutations to rows and columns, we decided to create two separate operators. The operators are described in figure 20 and 21.

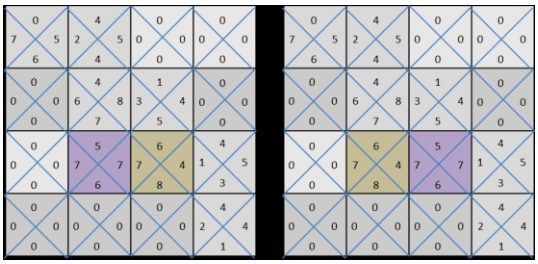

Figure 20 - Row Inversion

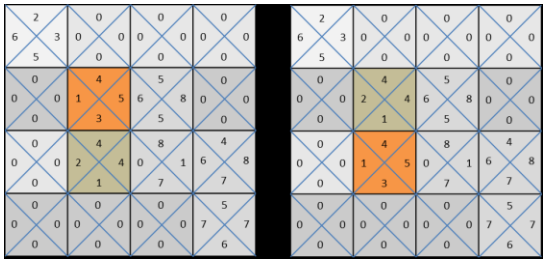

Figure 21 - Column Inversion

4.5.8 Scramble Mutation

The Scramble mutation operator maintains a corner tile in the left corner and scrambles the rest of the tiles, in order to force the board to re-assemble. It is a very effective mutation operator when the board is completely stuck.

4.6 Crossover Operators

Combinatorial optimization problems are complex problems that require crossover operators to be designed very carefully [Affen]. In order to test our solution we considered several crossover operators that were used successfully to solved complex combinatorial problems such as the traveling salesman problem (TSP), the capacitated

vehicle routing problem (CVRP), the scheduling problem (SP), but after a careful analysis we decided not retain any of them. These crossover operators are built to preserve order which is not an essential requirement for our problems.

In order to assemble properties of candidate located in different regions of the search space, the region exchange crossover defined by [Munoz] produced the best results. We also ran some experiments with the Uniform Crossover operator. The Region Exchange and Uniform crossover operators are described below.

4.6.1 Region Exchange Crossover

The region exchange crossover was proposed by [Munoz]. We implemented a slightly different version to obtain better results.

The operator works very intelligently. Two regions randomly selected are exchanged between two parents and two off-springs are produced. The off-springs will acquire some new genetic material but at the same time preserve the overall genetic composition of the parents [Munoz]. Keeping the relationship between adjacent tiles is essential but not critical, as a tile can fit many other tiles. Further details are provided in Algorithm 1 and Figure 22.

1. *Select a random region [select two points, a random length and width]*
2. *Clone the two parents [parent A, parent B]*
3. *Remove from parent A all tiles that are inside the region in parent B.*
4. *Remove from parent B all tiles that are inside the region in parent A.*
5. *Add the tiles remaining in both regions to two separate lists: list and list B*
6. *Copy to parent A's region all tiles that in parent B's region.*
7. *Copy to parent B's region all tiles that in parent A's region.*

8. Fill the empty places in child A and child B using the tiles from list A and list B, respectively.

Algorithm 7 - Region Exchange Crossover

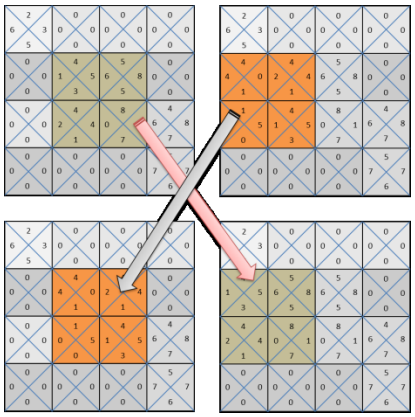

Figure 22 - Region Exchange Crossover

4.6.2 Uniform Crossover

The Uniform crossover follows the same principle as the one defined in [Eiben]. A template is created and randomly filled with values of 1 and 2. Then, the two parents are selected and two off-springs are created by choosing from parent 1 or parent 2, depending on the value read from the template. The procedure makes sure that the tiles are not duplicated. We obtained some good results with this crossover, but the region exchange crossover was the preferred method.

4.7 Fitness Function

The fitness function was designed with the minimum number of constraints to avoid excluding "bad" individuals. Those individual might have some genetic material needed by the individuals with the best fitness. A fit individual has a fitness value of 1.

It was designed with a minimum of two objectives:

1. A penalty is incurred if the edges of two adjacent tiles do not match. (fig. 7)
2. A penalty is incurred if the interior edges of 2x2 area do not match (fig. 8)

$$P(\bar{s}) = 1 - \frac{1}{k} \sum_{1}^{k} \frac{total_i - score_i}{total_i}$$

4.8 Selection Heuristics

The selection heuristic that we developed for this experiment checks the best individuals to see if they are deadlocked, when the repair threshold is reached. All individuals who seem to be deadlocked are tagged and the threshold is lowered by a percentage. The next time the threshold is reached, if the tagged individuals are still unable to improve the repair heuristic corresponding to the symptoms is executed.

4.9 Repair Heuristics

Two repair heuristics were developed to cope with two different symptoms. One repair function was developed to remove the deadlocks and one to rearrange the 2x2 areas and another to rearrange the border tiles.

5. Software Application

An experimental software application was developed C#.Net to test the solution. It is console application with no input capabilities. The information displayed is very informative and allows the user to observe the progress of the transformation. Figure 23 shows the execution of a 5x5 board.

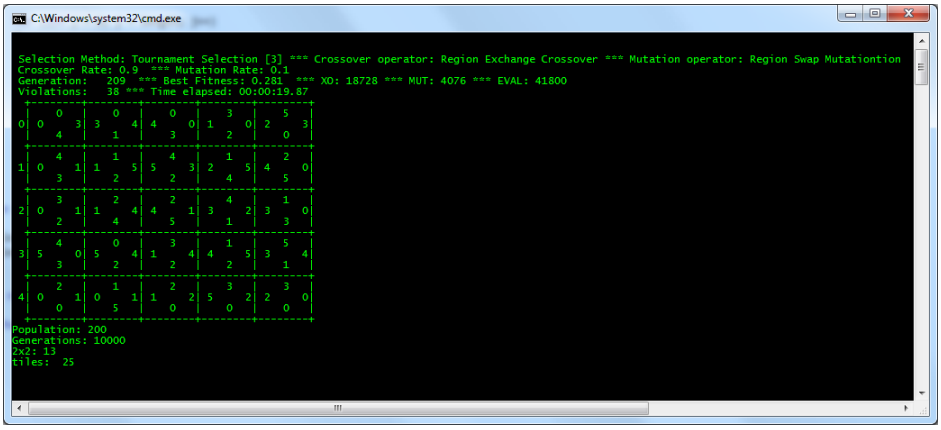

Figure 23 - Execution of a 5x5 E2 board

The following screen (fig. 24) shows the successful completion of a run. The output is formatted in such a way that it can be copied to the Eternity II Editor to check the results.

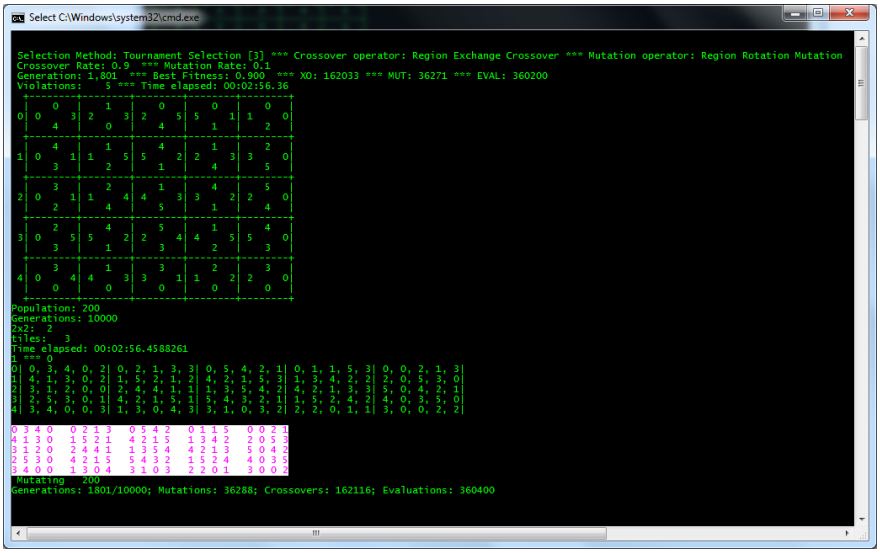

Figure 24 - Successful completion of a run

The results can be copied to the Eternity II Editor and visualized.

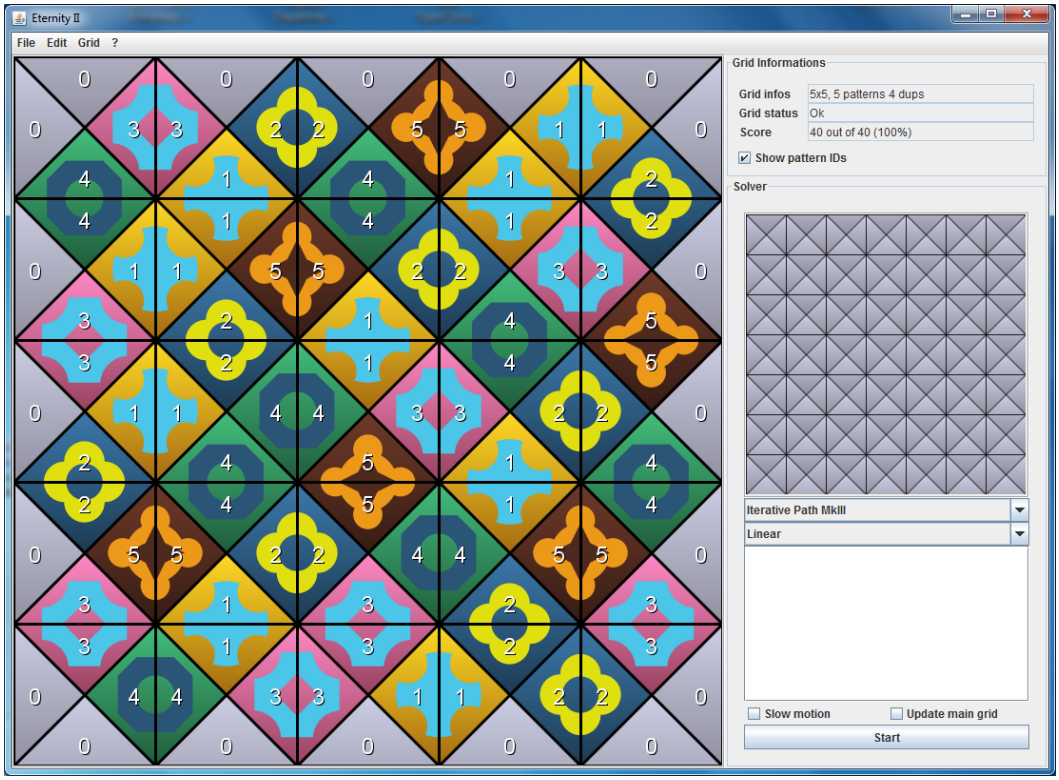

Figure 25 - 5x5 Eternity II board displayed by the Eternity II Editor

The program has been optimized to run for hours without running out memory. It uses the basic C# construct to store the information about the tiles. We need O (1) to access the tile structure if we want the application to run efficiently and fast.

We constructed a main Board class that includes all the method necessary to manipulate the tile patterns and rotation. There are also very useful methods for searching for tiles.

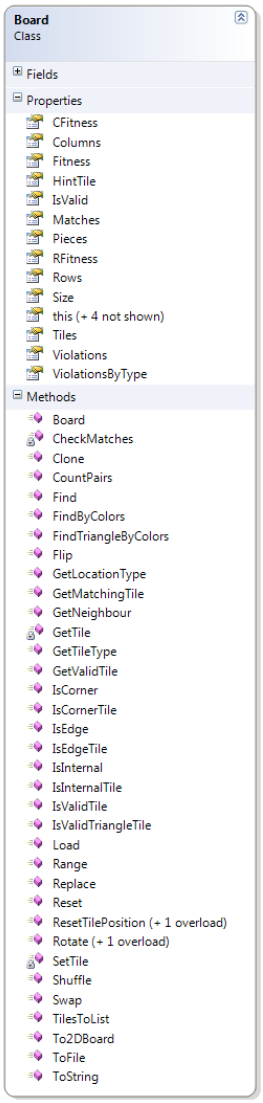

Figure 26 - Class Board

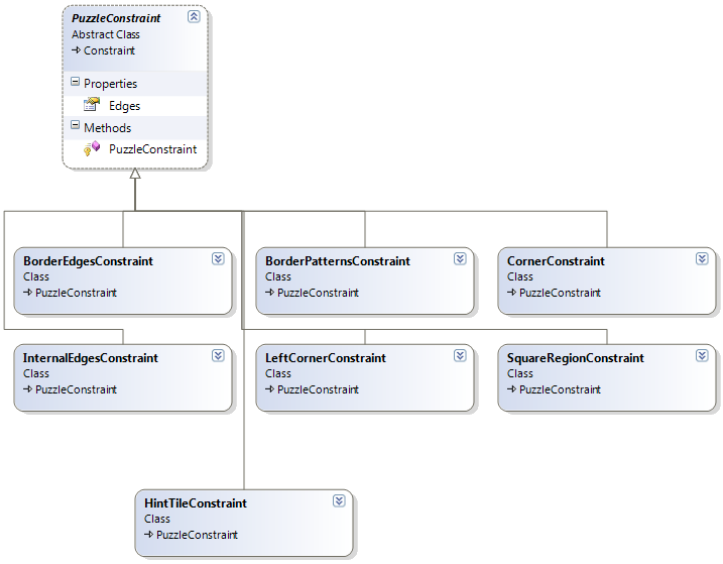

Figure 27 - Constraint structure

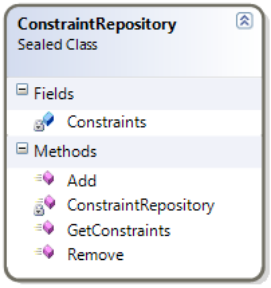

Figure 28 - Constraint Repository class

The class Board uses jagged-arrays to store the tile information. We store five piece of information: the 4 colors and the rotation direction. Access to the colors and rotation is very fast.

```
namespace Puzzle.Solver
{
   /// <summary>
   /// Represents a puzzle board/grid. Each slot contains a pattern.
   /// </summary>
   public class Board
   {
      #region Fields
```

```
int _columns;
int _rows;
private readonly int[] _hintTile;
int[][][] _tiles;

public const int Edges = 5;
int _gridSize = 2;
private readonly int[][] _matches;
private readonly int _pieces;
public static readonly Random RandomGenerator = new Random();

public static int DirNorth = 0;
public static int DirEast = 1;
public static int DirSouth = 2;
public static int DirWest = 3;

public static readonly int[] NegativeTile = new[] { -1, -1, -1, -1, -1 };

public Guid Id = Guid.NewGuid();

public bool Tagged = false;

#endregion

#region Properties

public int[] HintTile
{
    get { return _hintTile; }
}

public int Size
{
    get { return _gridSize; }
}

public int[][][] Tiles
{
    get { return _tiles; }
}

public int Rows
{
    get { return _rows; }
}

public int Columns
{
    get { return _columns; }
}
```

```
        public int[] this[int tileIndex]
        {
            get
            {
                var row = (tileIndex - 1) / _gridSize;
                var column = (tileIndex - 1) % _gridSize;
                return this[row, column];
            }
            set
            {
                var row = (tileIndex - 1) / _gridSize;
                var column = (tileIndex - 1) % _gridSize;
                this[row, column] = value;
            }
        }
            }
}
```

Figure 29 – class Board

The fitness is computed dynamically by iterating through the constraints. Every constraint is defined separately and the compute method is executed when the fitness is evaluated.

```
namespace Puzzle.Solver
{
    public class PenaltyFunction : IFitness
    {
        private int _violations;

        public Double Evaluate(Board board)
        {

            foreach (var key in ConstraintRepository.GetConstraints().Keys)
            {
                try
                {
                    var constraint = ConstraintRepository.GetConstraints()[key];
                    var total = constraint.Compute(board);
                    var violations = constraint.Violations;

                }
                catch (IOException e)
                {
                    Console.WriteLine(e.StackTrace);
                }
            }

            return fitness;
```

```
        }

        public int GetViolations()
        {
            return _violations;
        }
    }
}
```

Figure 30 - Fitness function

6. Results and Analysis

We present here the results obtained during the experiments. We developed a computer program in C#.Net to execute the algorithms. Most of the experiments were run on a 2.4GHz dual-processor machine running a 64-bit version of Windows 7.

The main objective of the experiments was to see if we were getting constant improvements with time and that the best solution obtained was within the range of expectations.

The parameters for the first runs are listed below in Table 1.

Table 1 - GA Parameters (01)

Parameters for the GA	
Generations	100
Population Size	200
Elitism Rate	0
Mutation Rate	0.1
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange

We observed the behavior of the fitness with respect to the following factors: the mutation rate, the crossover rate, elitism, the size of the population. We also looked at the fitness values when selection and repair heuristics were applied. All fitness values are plotted against the number of iterations (generations).

The following graphs show the evolution of the fitness depending on whether elitism and/or repair functions. In figure 31, we observe that the fitness values are not improving after the 40^{th} iteration and they stagnate and stay within a range. There appear to be no convergence towards the final solution.

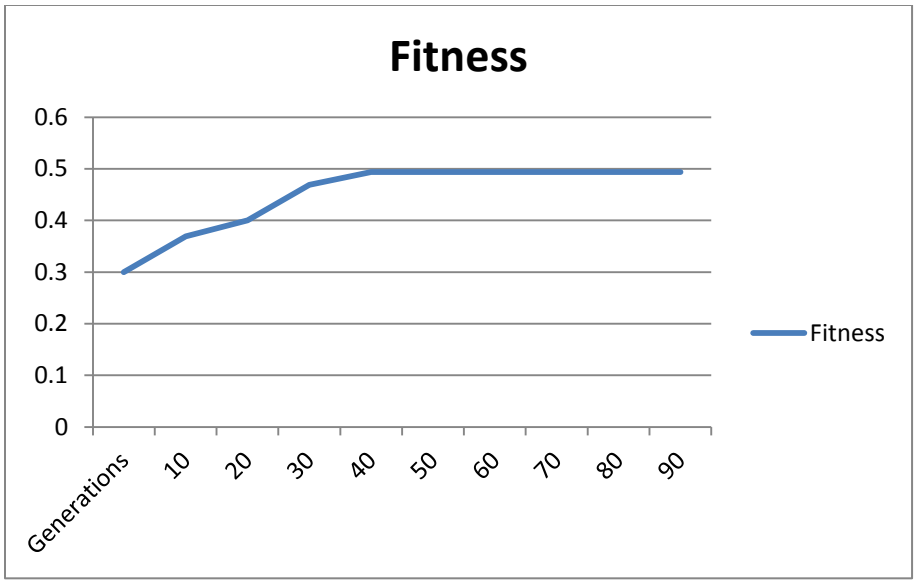

Figure 31 - Fitness evolution without elitism and no repair Heuristic

The following graph (fig. 32) shows an improvement of the fitness when elitism and repair heuristics are applied. We can clearly see that the fitness is converging towards the final solution.

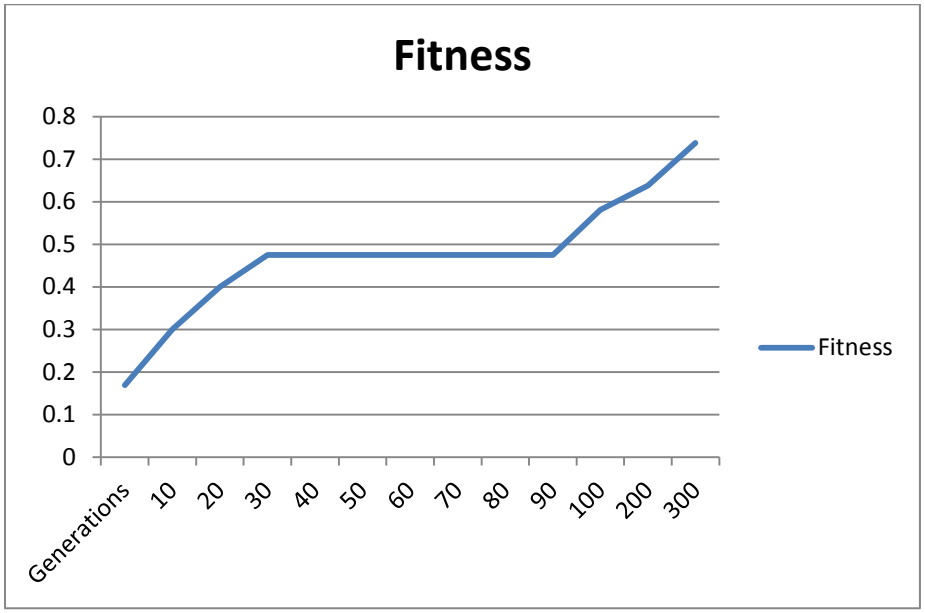

Figure 32 - Fitness evolution with elitism and repair heuristic

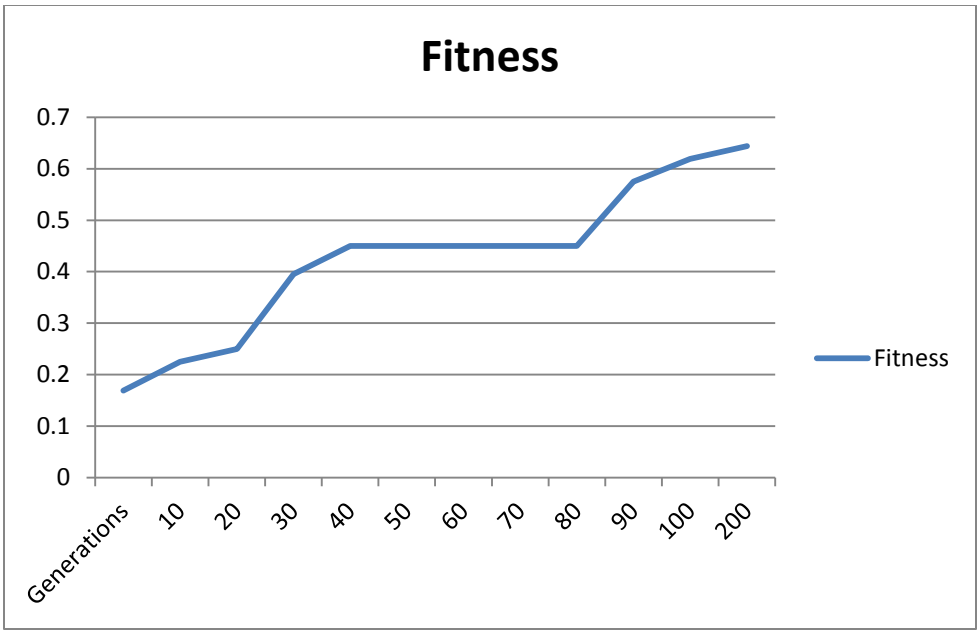

Figure 33 - Fitness evolution with elitism and repair heuristic

The following graph (fig. 34) seems to indicate that the repair heuristic should be run after a certain number of iterations.

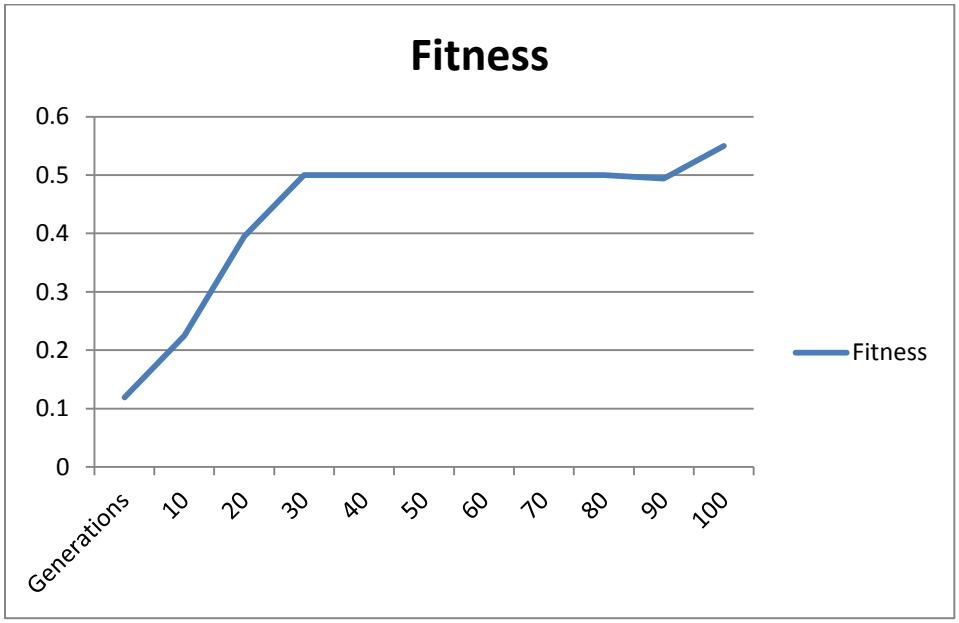

Figure 34 - Fitness evolution with elitism and no repair

In the following experiment, we look at the impact of a high mutation rate.

Table 2 – GA Parameters (02)

Parameters for the GA	
Generations	100
Population Size	200
Elitism Rate	1
Mutation Rate	0.8
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange

The following graph (fig. 35) shows that a high mutation rate can disturb the genetic composition of a candidate solution. Starting at the 60th generation, the quality of the best individuals is decreased.

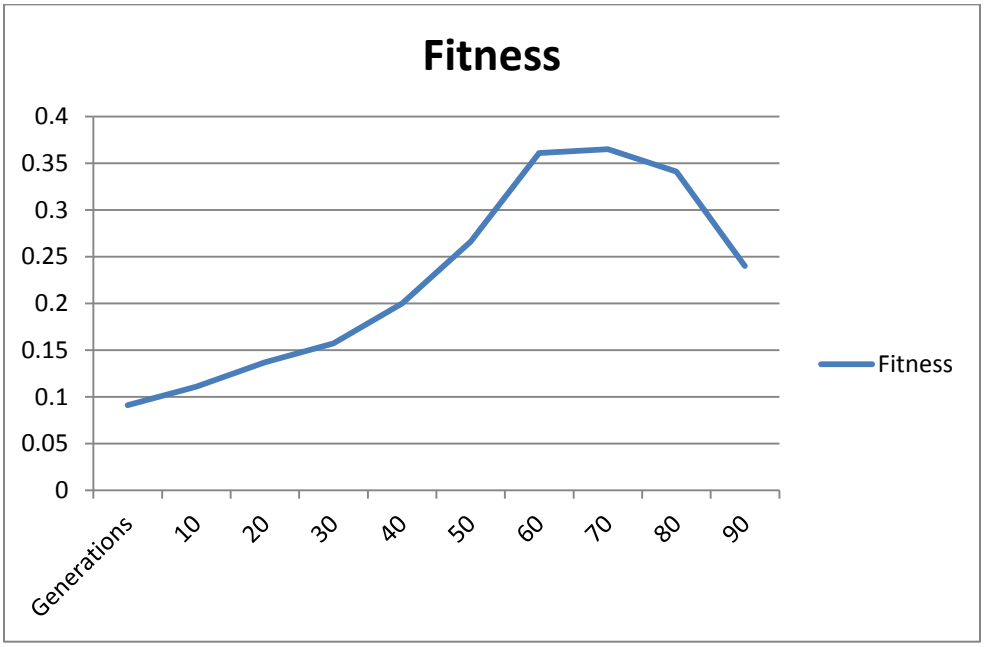

Figure 35 - Evolution of the fitness with high mutation rate (0.8)

In the following experiment, we study the impact of the absence of a mutation operator.

Table 3 - GA Parameters (03)

Parameters for the GA	
Generations	100
Population Size	200
Elitism Rate	1
Mutation Rate	0
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange

We notice in figure 36 that the individuals stop improving after a while. This is probably an indication that mutation plays an important role in producing a successful candidate.

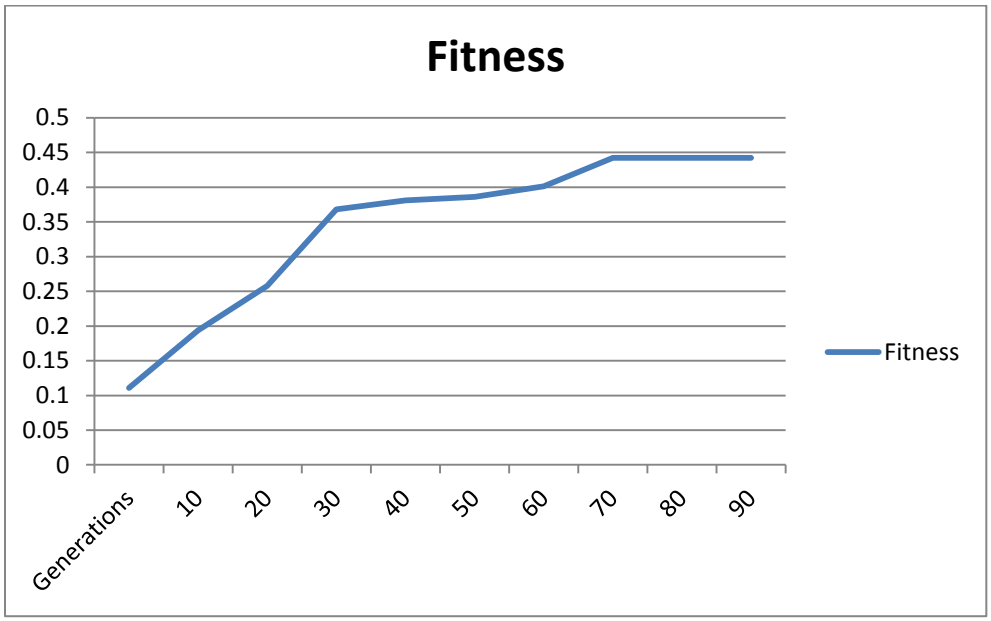

Figure 36 - Evolution of the fitness without mutation (0%)

In the following experiment, we look the impact of the absence of a crossover operator.

Table 4 - GA Parameters (04)

Parameters for the GA	
Generations	1000
Population Size	200
Elitism Rate	1
Mutation Rate	1.0
Crossover Rate	0
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange

In figure 37, the graph shows that the fitness of the individuals is lower compared to when there is crossover.

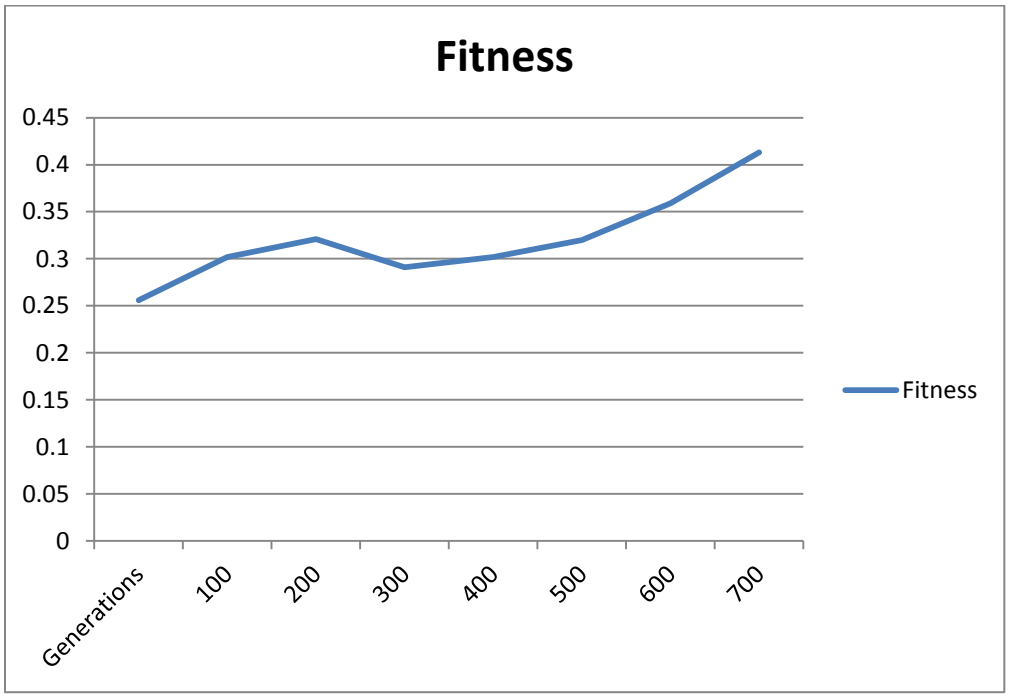

Figure 37 - Evolution of fitness with mutation and no crossover

In the following experiment, we look the impact of repair heuristics. They are applied at calculated intervals that depend on the rigidness of the individual to increase its fitness.

Table 5 - GA Parameters (05)

Parameters for the GA	
Generations	1000
Population Size	1000
Elitism Rate	1
Mutation Rate	0.1
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange
Repair	Selection

We observe in figure 38 that repairing the individuals can help them increase their fitness values. However, we notice that immediately after a repair, individuals have lower fitness values.

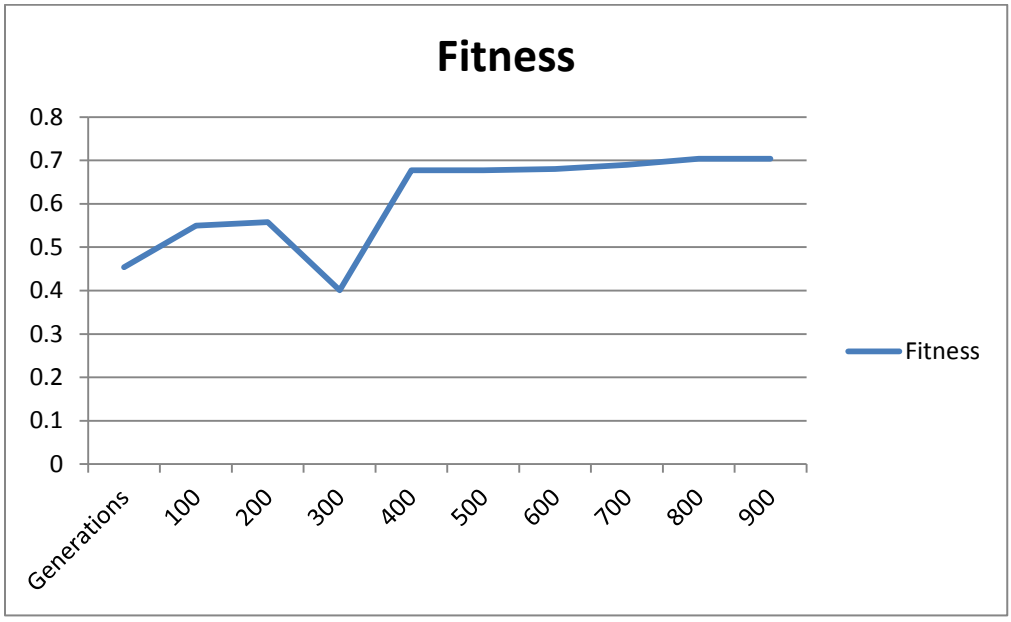

Figure 38 - Fitness evolution using a bigger population (1000) and repair

In the next experiment, we look at the impact of the size of the population.

Table 6 - GA Parameters (06)

Parameters for the GA	
Generations	1000
Population Size	500
Elitism Rate	1
Mutation Rate	0.1
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange
Repair	Selection

In figure 39, we notice higher fitness values compared to figure 38. The size of the population is not necessarily a dominant factor.

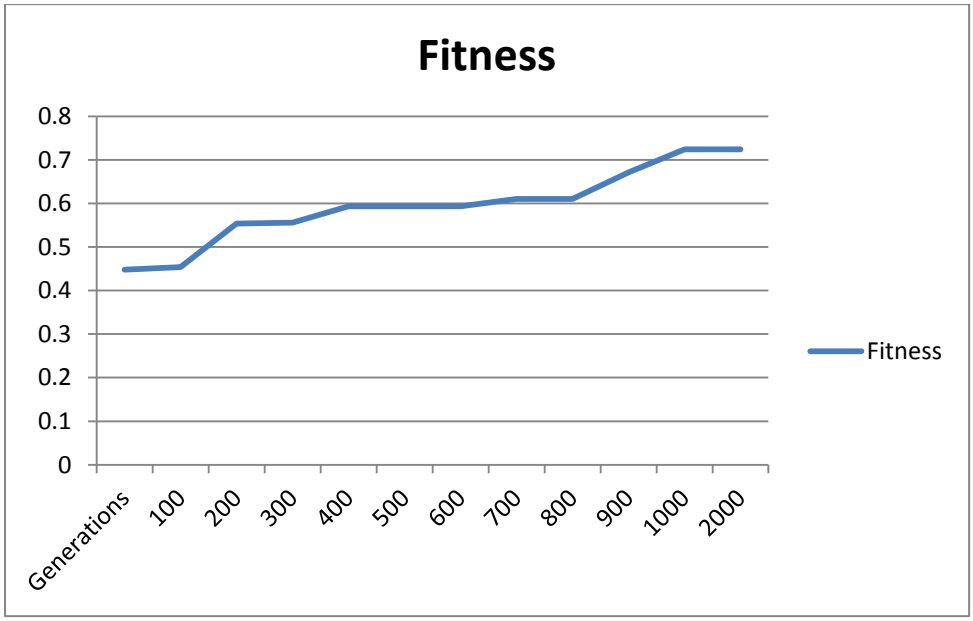

Figure 39 - Fitness evolution with smaller population (500) and repair

In the following, we look at four different experiments that were executed using Uniform crossover. We notice that it took longer to achieve results obtained using the Region Exchange crossover. The crossover operation also takes longer to execute. We could determine the exact cause of the slowdown. One of the four GAs used in this experiment used two constraints (refer to the fitness function definition, sec. 4.21).

Table 7 - GA Parameters (07)

Parameters for the GA	
Generations	1000
Population Size	200
Elitism Rate	1
Mutation Rate	0.1
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Uniform

The graph in figure 40 shows that there is a point at which the fitness of the individuals stops improving. (2c) does not seem to produce individuals with desirable fitness values.

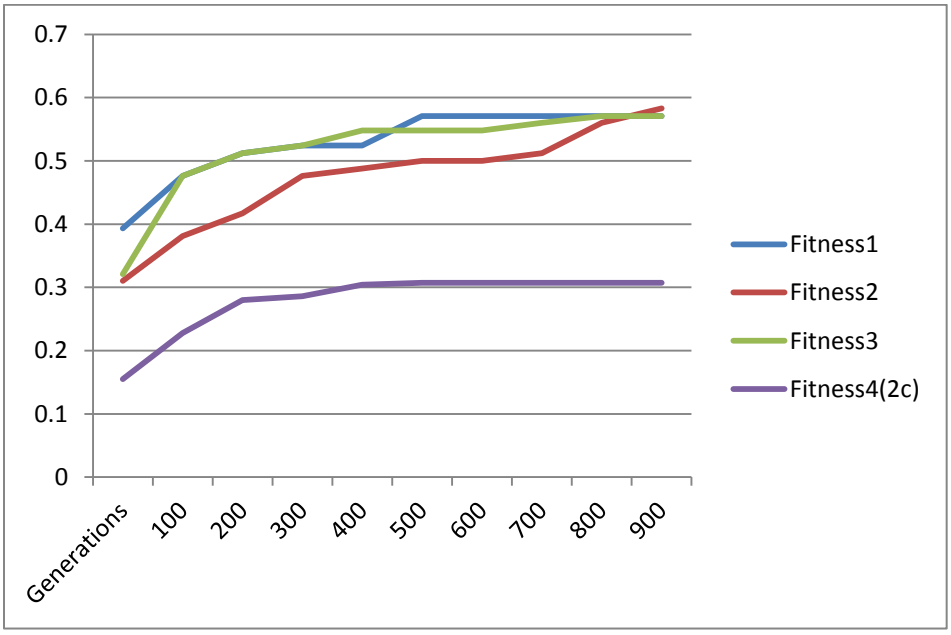

Figure 40 - Fitness evolution for 4 runs (1 run has 2 constraints)

The setup in this experiment is similar to the previous one. The only parameter that was changed was the crossover operator. Instead of the Uniform crossover, we are using the Region Exchange crossover.

Table 8 - GA Parameters (08)

Parameters for the GA	
Generations	1000
Population Size	200
Elitism Rate	1
Mutation Rate	0.1
Crossover Rate	0.9
Selection Operator	Tournament
Mutation Operator	Region Rot/Swap
Crossover Operator	Region Exchange

In figure 41, the graph is almost the same as in figure 40. We have higher fitness values when using Region Exchange crossover.

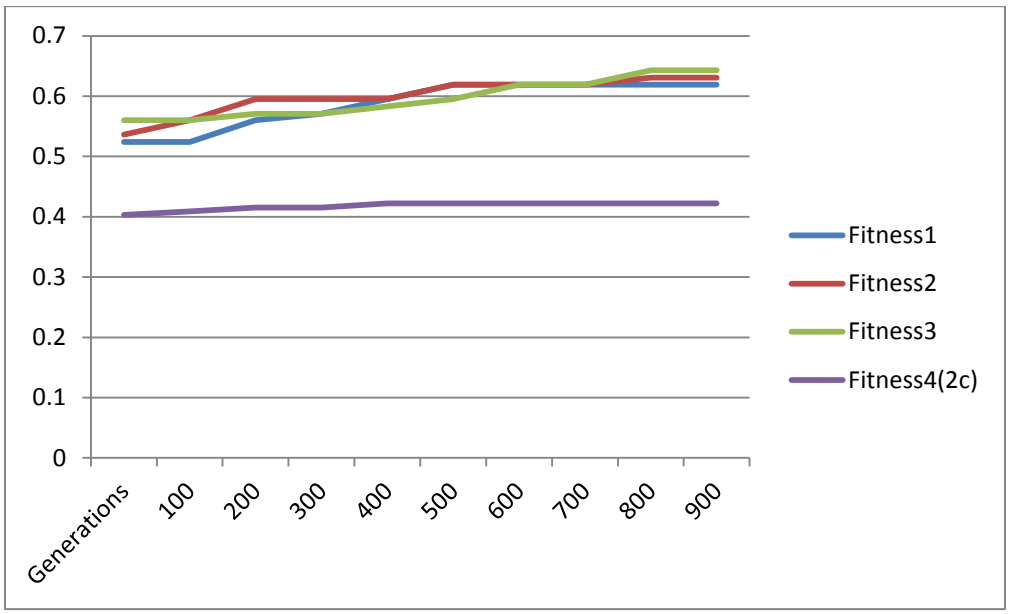

Figure 41 - Fitness evolution for 4 runs (1 run has 2 constraints)

In the following experiment, we applied the algorithm to three boards of sizes, 4x4, 5x5 and 6x6. We executed each configuration ten (10) times and calculated the average time it took before a solution was found. Table 9 shows the summary of 30 runs (10 for each board).

Table 9 - Summary of experimental results: board runs

Board Size (patterns)	Average time	Average Evaluations	Average Mutations	Average Crossovers
4x4 (5)	27s	92,700	9,284	41,480
5x5 (5)	30s	212,140	20,333	95,286
6x6 (7)	195s	267,800	26,805	120,500

It is clear that the algorithm worked well and that it was able to locate solutions in a very short time. We also see that the time it takes to find a solution increases exponentially with the board size. The following graph (fig. 42) shows the average time for each board.

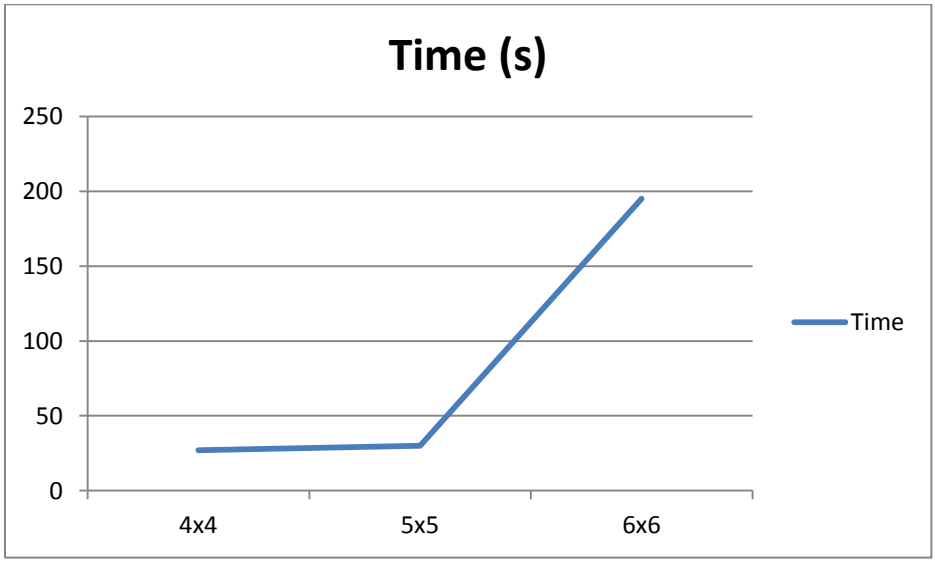

Figure 42 - Time vs. Size of the board

.

All experiments were conducted using a 7x7 board. Despite the fact that our algorithm tested well on 4x4, 5x5 and 6x6, we were not able to obtain a solution on a 7x7. Apparently, our solution did not scale well.

However, the results of the experiments gave us a lot of insight as to what needs to be improved in order to obtain a solution.

- We need to improve the selection and the repair heuristics.
- The crossover and mutation operators need to be more biased towards creating fitter individuals. We consider testing this option, but we are not convinced that it make an enormous difference given the size of the problem.
- We need to revisit our algorithm and identify potential errors.

7. Conclusion and Future Works

The purpose of my research was to formulate a method using evolutionary techniques that could possibly be a basis for further study in the area edge-matching puzzles.

This thesis was designed to contribute to advance the state-of-the-art in genetic algorithms and to:

- Provide a model for improving GAs in the area of edge matching puzzles
- Expand the options available to GA practitioners for solving E2-type puzzles

The objective was to implement and test a solution using genetic algorithms that could produce better solutions to the E2 puzzle than the ones obtained using other methods, such as hybrid local search meta-heuristics and hyper-heuristics.

The idea of solving the E2 puzzle using genetic algorithms is not naïve. We have expanded the research that was started by [Munoz] and created a solution that can be shared by other to further the work.

In summary, we have designed, implemented and tested a genetic algorithm to solve a very complex problem known to the world. We have identified the limitations of other methods used so far and established a framework for further research in evolutionary algorithms.

Researching problems of this kind has the benefit of accelerating problem solving, developing machine intelligence and advancing the field of artificial intelligence. We must be able to solve complex problems without relying on human expertise.

I hope that the work presented in this thesis will stimulate interested in the subject and encourage the reader to invest in learning evolutionary techniques.

8. References

[Affen] M. Affenzeller, S. Winkler, S. Wagner and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. CRC Press, 2009.

[Alajlan] N. Alajlan. "Solving Square Jigsaw Puzzles Using Dynamic Programming and the Hungarian Procedure," *American Journal of Applied Sciences* (2009): Vol. 6, Issue 11, pp. 1942-1948.

[Anonymous] kubzpa@yahoo.fr. "The number of solutions of the Eternity II puzzle," http://games.groups.yahoo.com/group/eternity_two.

[Anso] C. Ansótegui, R. Béjar, C. Fernández and C. Mateu. "How Hard is a Commercial Puzzle: the Eternity II Challenge," *Proceeding of the 2008 conference on Artificial Intelligence Research and Development* (2008): pp. 99-108.

[Benoist] T. Benoist. "How many edges can be shared by N square tiles on a board?" *e-lab research report* (April, 2008). http://tbenoist.pagesperso-orange.fr.

[Benoist] T. Benoist. "Fast Global Filtering for Eternity II," *Constraint Programming Letters* (2008): Vol. 3, pp. 36-49.

[Craenen] B. G. W. Craenen, A. E. Eiben, E. Marchiori. "Solving Constraint Satisfaction Problems with Heuristics-based Evolutionary Algorithms," *IEEE Congress on Evolutionary Computation, 2000* (2000): Vol. 2, pp. 1571.

[DeJong] K. A De Jong, W. M. Spears. "Using Genetic Algorithms to Solve NP-Complete Problems," *Proceedings of the Int'l Conference on Genetic Algorithms* (1989): pp. 124-132.

[Demaine] E. D. Demaine and M. L. Demaine. "Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity," *Graphs and Combinatorics* (February 2007): Vol. 23, Issue 1, pp. 195-208.

[Eiben] A. E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[Glover] F. Glover. "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research* (1986): Vol. 13, pp. 533-549.

[Gindre] F. Gindre, D. A. Trejo Pizzo, G. Barrera and M. D. Lopez De Luise. "A Criterion-Based Genetic Algorithm Solution to the Jigsaw Puzzle NP-Complete Problem," *Proceedings of the World Congress on Engineering and Computer Science 2010* (San Francisco, October 20-22, 2010): Vol. 1.

[Gottlieb] J. Gottlieb, E. Marchiori and C. Rossi "Evolutionary Algorithms for the Satisfiability Problem," *Evolutionary Computation* (Spring 2002): Vol. 10, No. 1, pp. 35-50.

[Haubrich] J. Haubrich. *Compendium of Card Matching Puzzles*. Self-published, May 1995. Three volumes.

[Heule] M. J. H. Heule. "Solving Edge-Matching Problems with Satisfiability Solvers," *In Proceedings of the Second International Workshop on Logic and Search* (LaSh 2008): pp. 88-102. University of Leuven.

[Holand] J. H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, 1975.

[Kumar] R. Kumar. "Simulation Optimization for Manufacturing System Design," *Master's Thesis* (2003): pp. 17-18

[Kendall] G. Kendall, K. Spoerer. "Scripting the Game of Lemmings with a Genetic Algorithm," *Congress on Evolutionary Computation, CEC2004* (2004): Vol. 1, pp. 117-124.

[Kirkpatrick] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. "Optimization by Simulated Annealing," *American Association for the Advancement of Science. New Series* (May 13, 1983): Vol. 220, No. 4598, pp. 671-680.

[Munoz] J. Munoz, G. Gutierrez and A. Sanchis. "Evolutionary Techniques in a Constrained Satisfaction Problem: Puzzle Eternity II." *IEEE Congress on Evolutionary Computation, 2009* (2009): pp. 2985-2991.

[Oliveto] P. S. Oliveto, J. He and X. Yao. "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results," *International Journal of Automation and Computing* (2007): vol. 04, no. 3, pp. 281-293.

[Schaus] P. Schaus and Y. Deville. "Hybridization of CP and VLNS for Eternity II," *Journées Francophones de Programmation par Contraintes JFPC'08* (2008).

[Tomy] Tomy. "Eternity II (official site)," http://us.eternityii.com

[Wang] W. Wang and T. Chiang. "Solving Eternity-II Puzzles With a Tabu Search Algorithm," *META 2010* (2010).

[Toyama] F. Toyama**,** K. Shoji and J. Miyamichi. "Assembly of Puzzles Using a Genetic Algorithm," *16th International Conference on Pattern Recognition, 2002* (2002): 389-392.

[Zaritsky] A. Zaritsky and M Sipper. "The preservation of favored building blocks in the struggle for fitness: the puzzle algorithm," *IEEE Transactions on Evolutionary Computation 2004* (October 2004): Vol. 8, Issue 5, pp. 443.

Appendix A: Screenshots of the results obtained using different board configurations

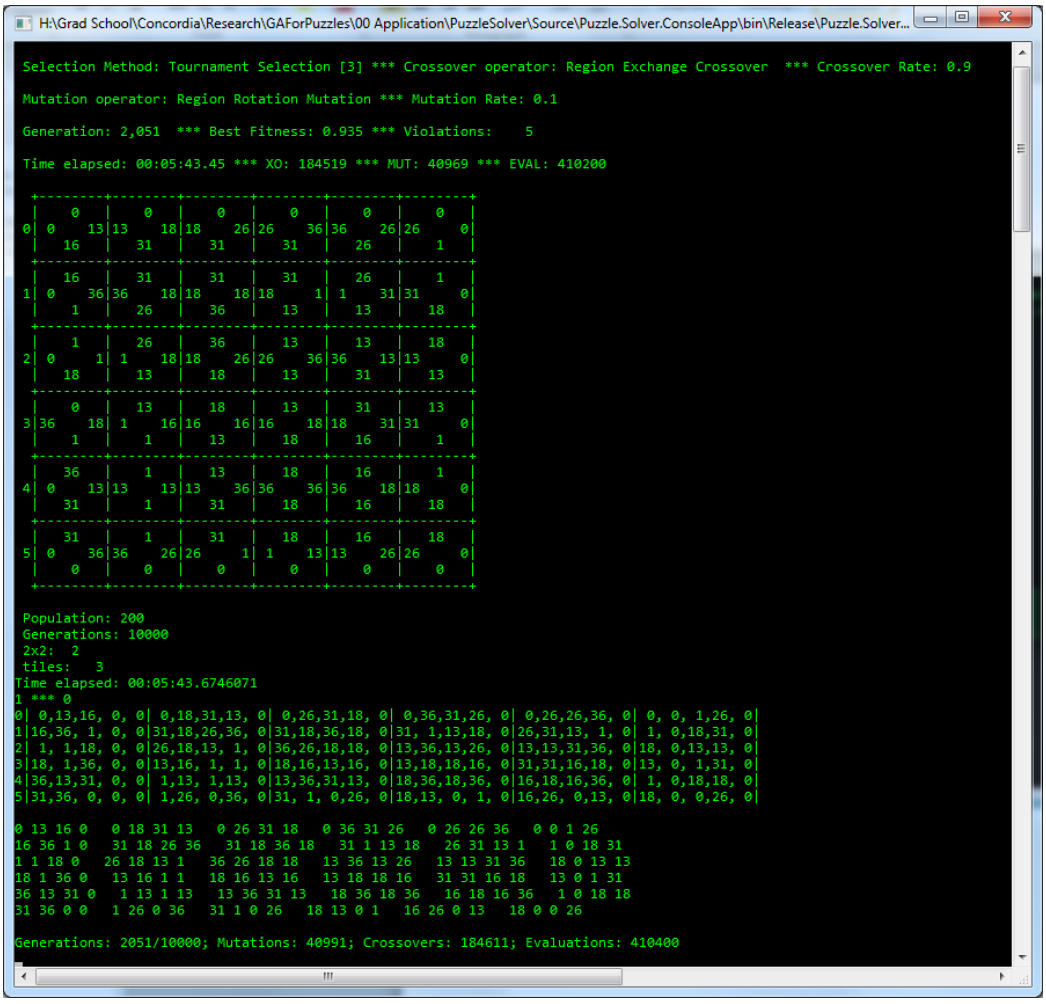

Figure 43 - Results of a 6x6 run

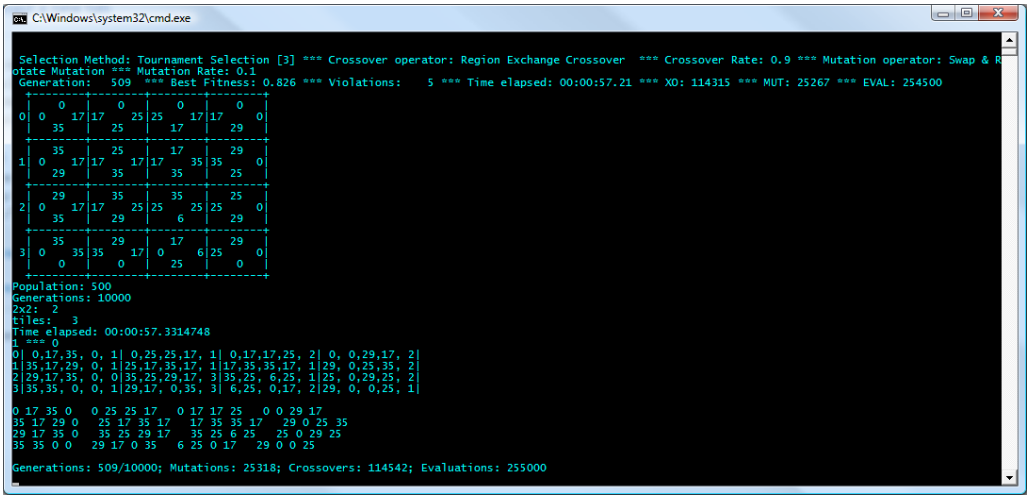

Figure 44 - Results of a 4x4 run

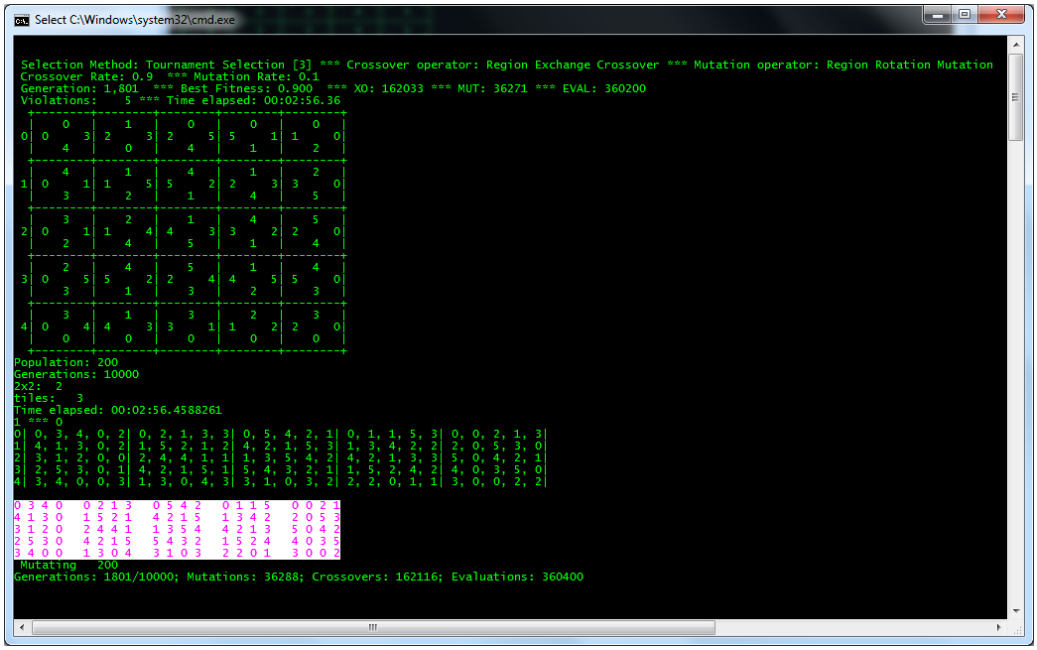

Figure 45 - Results of a 5x5 run