Final year thesis

Comparative Analysis of Algorithms for Solving the Eternity II Puzzle

Skouzos Paschalis

Supervisors: Astaras Alexandros & Karagiannis Kostas

Computer Science Dept.
American College of Thessaloniki
Pilea, Greece

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owners, except for non-profit educational purposes in which the owners give the leading rights for pieces of work, and only if the owners are properly cited as the original designers of the system and the authors of the present project proposal.

Contact: *skouzos.paschalis@gmail.com*

© 2017 Astaras Alexandros, Karagiannis Kostas, Skouzos Paschalis
College of Thessaloniki, Pilea, Greece.

Abstract

This thesis document presents a comparative analysis between three artificial intelligence (AI) algorithms for solving the puzzle Eternity II (EII). EII falls under the general category of edge-matching puzzles and has been proven to have a solution which can be calculated in non-deterministic polynomial time (NP) [1].We initially set our design requirements and develop a mathematical representation of the EII puzzle as a system of polynomial equations. Subsequently, our AI algorithms' performance is measured as the number of edges which have been successfully matched with respect to time. We selected Genetic Algorithms (GA), Simulated Annealing and Backtracking software algorithms for our experimentation. Our results suggest that Simulated Annealing is the better suited algorithm for our puzzle.

Acknowledgements

First and foremost, I would like to thank my supervisors Astaras Alexandros and Karagiannis Kostas for the continuous help throughout my thesis. Astaras Alexandros guided my learning in the field of artificial intelligence and Karagiannis Kostas played a major role in the mathematical department of my thesis.

I take this opportunity to express gratitude to all of the Department faculty members for their help and support. I also thank my parents for their encouragement, support and attention.

I wish to express my sincere thanks to Emmanuel Maou, chair in the division of technology and science of the American College of Thessaloniki, for providing me with all the necessary facilities for the research.

I place on record, my sincere thank you to Baglavas Grigoris, Dean of the Faculty, for the continuous encouragement.

I am also grateful to my friends who made me fall in love with puzzles and riddles in general and my interaction with them was the catalyst for choosing a thesis like this. Finally, I would like to thank the person who gifted me Eternity II, probably my parents, since finding it in our attic determined my thesis project.

1 Table of Contents

Acronyms

AI	Artificial Intelligence
CSV	Comma Separated File
EII	Eternity 2
IDE	Integrated Development Environment
NP	Nondeterministic Polynomial Time
UI	User Interface

1 Introduction

It is possible to apply and compare software implementations of Simulated Annealing, Genetic and Backtracking algorithms for the solution of the Eternity II (EII) puzzle.

As the name of the puzzle indicates, EII had a predecessor. Eternity I was created by Christopher Monckton and published by Ertl Company in June 1999 with a monetary reward of $1.000.000 for the first person to solve it. The board of the puzzle was a dodecagon and it had to be filled with 209 weirdly shaped smaller polygons. Nearly a year after its release, the puzzle was solved by two mathematicians, named Alex Selby and Oliver Riordan, who approached the puzzle with a probabilistic method [2]. Selby and Riordan were then contacted by Monckton in order to help him design the second version of the puzzle. Using the previous experience they had in Eternity I, they eliminated the probabilistic exploits, increasing the difficulty of the puzzle.

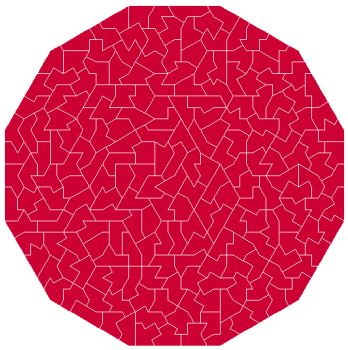


Figure 1: Eternity I solution

In July 28 of 2007, EII was released with a prize of $2.000.000. On December 31 of 2010, the puzzle was still unsolved and the monetary reward was revoked. It was stated that if the solution was not found, a smaller prize of $10.000 would be handed out to the person with the most fitting edges [3]. A partial solution of 467/480 edges was achieved by Verhaard Louis in 2009 and he won the secondary reward from the puzzle's creator [4]. As of June, 2017, Eternity II has yet to be solved.

1.1 Motivation for the project

The focal point of the project was to define the best approach for solving EII. Apart from that, there were various factors that led me to choose the specific project. First of all, AI has always fascinated me and since last summer I wanted to delve into it. This

project was a great opportunity for me in order to learn the basics and even some intermediate level of AI. Furthermore, I dream of a master in AI and, according of my personal assessment, a related thesis raises the value of my application to the universities of my choice. I also enjoy programming and I was looking forward to writing code for a solely software based project which would require optimized, efficient, well structured, readable, reliable and extensible code. Secondly, the puzzle could be formulated as a mathematical problem and with mathematics being a great love of mine since elementary school I was excited to dive deeper into that subject too. Thirdly, although I considered my chances of actually solving the puzzle close to zero, just the sheer thought of solving it captivated me. The monetary award may have been revoked but the publicity for the person who solved it will open many doors for her/his future. In addition, EII falls under the category of NP problems and mathematically proving that there is no feasible way to generate an answer with the help of a computer will have solved the P vs. NP problem, one of the seven millennium problems that the Clay Mathematics Institute offers $1.000.000 for solving [5]. Last but not least, EII remains an unsolved puzzle and the challenge of tackling something that no one was able to solve before pushed me to try harder and pour my heart and soul into my thesis.

1.1.1 Stating the problem

EII has a rectangular board which is split into a 16 by 16 grid. The goal of the puzzle is to fit all 256 rectangular pieces into that board under some specific rules. Each piece has four edges and each edge is decorated with one out of 22 possible patterns. Each edge from every piece must match with the pattern of the neighbour piece that touches that edge. In addition, each of the edges touching the border must be grey. Finally, pieces are numbered from 1 to 256 with the number written on its back side in order to enable the submission of solutions to the creators. Figure 2 displays the board of the puzzle and some connected pieces in order to grasp the main concept of our problem.

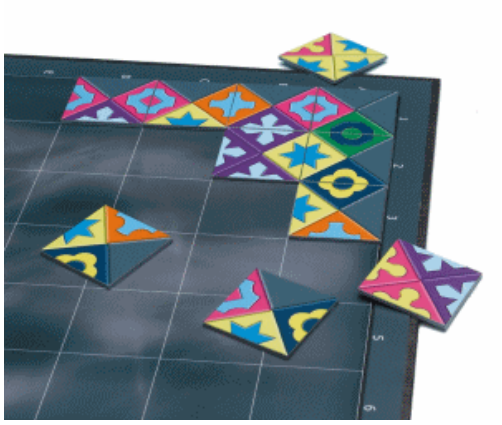


Figure 2: EII Board and Pieces

The difficulty of the puzzle lies in the number of the possible configurations of the board. The complexity stems from the large number of pieces and their four possible rotations. The above renders a brute force solution for the puzzle practically impossible. There are 256 possible placements for the pieces which yields $256! = 2.57 * 10^{506}$ combinations. In addition, each piece has four possible rotations which adds up to a total of $256! * 4^{256} = 1.15 * 10^{661}$ configurations. In order to put that number in perspective, the estimated number of atoms in the observable universe is 10^{80}. Finally, in order to raise the difficulty of the puzzle even higher, the creators of the puzzle have predetermined the piece that should be placed in the middle of the board, hence limiting the possible solutions. The latter restriction of the centre piece did not apply to the secondary reward mentioned in the introduction [1]. Because the community that was developed around the puzzle and the literature revolving it, did not take into consideration the central piece restriction, I also chose to tackle this simpler version of the problem.

1.2 Project objectives

The project was divided into five main milestones in order to achieve the desirable goal. Before any of the following was carried out, an extensive literature review was required in order to acquire a holistic approach of the task at hand. The objectives of the project in a chronological order were as follows:

1. Obtain the pieces to a digital form of my choice.
2. Selection of known AI algorithms and a subsequent implementation in a programming language.
3. Selection of a mathematical formulation of the puzzle and development.

4. Measure their performance with respect to matched edges and time.
5. Comparative performance analysis between all of the implemented algorithms.

All of the above were accomplished, and we are in a position to claim that one of the AI algorithms is better suited for solving EII.

2 Literature and State of the Art Review

2.1 Literature Review

Our literature review is divided in two main categories, related to general bibliographical entries but also related specifically to EII.

2.1.1 Literature Related to Implementing the Code

For reasons mentioned later in our document, our language of choice is going to be C++. In order to refresh our knowledge we consulted "Accelerated C++: Practical Programming by Example" [6] and as a C++ reference book we used "The C++ Programming Language" since it is much more extensive [7]. Finally, in order to write maintainable and reusable code we consulted three books by Meyers which state specific ways that lead to better C++ code [8]–[10].

Our main book for implementing simulated annealing and backtracking was "The Algorithm Design Manual" [11] and , again, whenever we wanted to dive deeper into a subject we consulted a more thorough book, in this case it was "Artificial Intelligence: a Modern Approach" [12]. In learning about GA we referred to two books, by Goldberg [13] and Mitchel [14]. Finally, we used a library in order to implement the GA and the documentation from the creator [15] but also a document related to the library's usage from an AI professor [16], were of paramount importance since the usage of the library was complex.

2.1.2 Literature Related to EII

Our most closely followed paper was "A Global Approach for Solving Edge-Matching Puzzles" [17] since this was the basis for our mathematical approach and we couldn't deviate from their method. The content of the paper is explained in more detail in 4.2.4.

There were two other attempts to tackle the puzzle with genetic algorithms [18], [19]. Munoz et al. [18] go into greater detail into explaining their methods and results, this is why we chose to follow their crossover method and some of their mutation techniques. Simulated annealing was used in combination with two other algorithms in a paper [20] described in 2.2. The parameters of their implementation will be our starting point in future experimentation with simulated annealing. Toulis [21]

performed a similar comparison to ours, between backtracking, simulated annealing and some other methods. He also found that simulated annealing had better performance than backtracking. Two more backtracking attempts followed an interesting approach through field programmable gate array logic in order to improve their speed by solving the puzzle on the hardware [22], [23]. Apart from the above which were related to our approaches in solving the puzzle, three publications view the puzzle as a constraints satisfaction problem and try to solve it as such [24]–[26].

2.2 Existing State of the Art Solutions

According to Benoist et al. [27] and my literature review, the best solution so far has 13 violations, that means that 467/480 of the inner edges are matched. Interestingly enough, it is the solution that earned Verhaard $10.000 back in 2009 and it has yet to be surpassed. His solution technique is explained in [28]. Another state of the art solution is the guide and observe hyper-heuristic algorithm by Wauters et al. [29] which manages to match 461/480 edges in less than an hour of running time and it was also the winner of the contest on EII proposed in the META'10 conference [30]. An algorithm that worked in two phases was also in the top three places of the above contest. First it solved the border pieces by performing a tabu search and in phase two, it used simulated annealing and perturbation in order to not get stuck in local minima [20].

3 Design Requirements and Deliverables

In this section of the paper, the design requirements of the algorithms are introduced and explained in greater detail.

3.1 Extracting the design requirements

In order to extract the design requirements, it would be helpful to assume the role of the algorithm or, in order to have a better grasp of the task, we can envision ourselves trying to solve the puzzle by hand. Imagining a physical perspective to the puzzle is the correct approach to extracting the design requirements since this thesis revolves around several algorithms, hence a generalized method of thinking is mandatory.

3.1.1 Example scenarios

There are six possible example scenarios for a human agent trying to solve a physical copy of EII.

1. We want to prepare the game.
2. We want to decide our next action.
3. We want to place a piece.
4. We want to undo some of our moves.
5. We want to pause the game.
6. We want to stop the game.

3.1.2 Use cases

3.1.2.1 *Prepare the game*

Description: We want to prepare a game of the physical copy of EII.

References Requirements no.: 1, 9

Course of events:

1. We acquire the puzzle.
2. We lay down the board of the puzzle.
3. We take out the pieces of the puzzle.

Pre-condition: None.

Post-condition: The game is ready to be played.

Assumptions: We have enough space to lay down the board and the pieces.

3.1.2.2 *Decide the next action*

Description: We want to decide what our next action will be, we can either try to place a piece, undo some of our moves, or stop the game.

References Requirements no.: 2, 3, 4, 8

Course of events:

1. Evaluate our current position.
2. Decide on our next action.

Pre-condition: Either the game is ready to be played or a decision must be taken.

Post-condition: An action must be taken, either place a piece, undo or stop.

Assumptions: We have decided what 'evaluation of our position' means. In other words, as we look at the board we have to know whether we are in a better or worse position than before making our last move based on some standards.

3.1.2.3 *Place a piece*

Description: We have a physical copy of the puzzle and we want to place a piece in a valid position.

References Requirements no.: 3, 8

Course of events:

1. Decide where we want to place a piece.
2. Find a matching piece to place.

Pre-condition: The action of placing a piece must be taken.

Post-condition: We must decide our next action.

Assumptions: We have enough time to find a matching piece and such a piece exists.

3.1.2.4 *Undo some of our moves*

Description: We have a physical copy of the puzzle and we want to undo some of our moves.

References Requirements no.: 3, 9

Course of events:

1. Decide on the number of pieces we want to remove.
2. Remove them.

Pre-condition: The action of undo must be taken.

Post-condition: We must decide our next action.

Assumptions: There are pieces on the board to be removed.

3.1.2.5 *Stop the game*

Description: We want to stop a game of the physical copy of EII.

References Requirements no.: 3, 5, 6

Course of events:

1. If it is our best score, save it.
2. Take a picture.
3. Remove the pieces and the board.

Pre-condition: The action of stopping the game must be taken.

Post-condition: None.

Assumptions: None.

3.1.2.6 *Pause the game*

Description: We want to pause a game of the physical copy of EII.

References Requirements no.: 3, 6, 7

Course of events:

1. Store the board intact, plus the remaining pieces.

Pre-condition: None.

Post-condition: The game is paused and the state of the game saved.

Assumptions: We have enough time between wanting to pause and performing the course of events. In relation to the algorithm, it means that under a sudden power outrage we are not able to pause the game and save its state.

3.1.3 Functional requirements

From the above use cases, we can extract the following functional requirements for an algorithm that is trying to solve EII.

Requirement	Explanation
Requirement Number	1
Requirement Title	Digital representation of the board and pieces
Requirement Type	Functional
Requirement Details and Constraints	The board of the game must be represented in an appropriate data type inside the algorithm, or in an external file. The same holds true for the pieces of the board. The only difference is that the pieces should definitely be kept in a separate file and parsed by the algorithm in order to make them available to every program that requires it.
Criticality	Must

Table 1: Functional Requirement #1

Requirement	Explanation
Requirement Number	2
Requirement Title	Decision making
Requirement Type	Functional
Requirement Details and Constraints	The algorithm must be in a position to decide what its next action will be, either try to place a piece, undo a certain number of actions or stop running.
Criticality	Must

Table 2: Functional Requirement #2

Requirement	Explanation
Requirement Number	3
Requirement Title	Represent the state of the board and available pieces
Requirement Type	Functional
Requirement Details and Constraints	The algorithm must save in an appropriate data structure the position and the rotation of the pieces already placed on the board as well as the remaining pieces to be placed.

Criticality	Must

Table 3: Functional Requirement #3

Requirement	Explanation
Requirement Number	4
Requirement Title	Evaluate the state of the board
Requirement Type	Functional
Requirement Details and Constraints	The algorithm must be in a position to evaluate its progress in order to decide if it has to continue or stop searching. For example, it has to know the number of matched edges so far.
Criticality	Must

Table 4: Functional Requirement #4

Requirement	Explanation
Requirement Number	5
Requirement Title	Graphical representation of the board
Requirement Type	Functional
Requirement Details and Constraints	The program is ideally able to represent the state of the board in a graphical way in order to be more communicative to the user.
Criticality	Want

Table 5: Functional Requirement #5

Requirement	Explanation
Requirement Number	6
Requirement Title	Export results
Requirement Type	Functional
Requirement Details and Constraints	The program must be capable of exporting the state of the board in a file for progress saving purposes.
Criticality	Must

Table 6: Functional Requirement #6

Requirement	Explanation
Requirement Number	7
Requirement Title	Continue searching from a file
Requirement Type	Functional
Requirement Details and Constraints	The program is advised to have a “continue search” function in case an error occurs during a run and we don’t want to lose our progress. In addition, this will allow us to pause searches.
Criticality	Want

Table 7: Functional Requirement #7

3.1.4 Non-functional requirements

Requirement	Explanation
Requirement Number	8
Requirement Title	Speed
Requirement Type	Performance
Requirement Details and Constraints	Because of the magnitude of the search space, algorithmic speed is of crucial importance
Criticality	Want

Table 8: Functional Requirement #8

Requirement	Explanation
Requirement Number	9
Requirement Title	Memory Usage
Requirement Type	Performance
Requirement Details and Constraints	Because we should be able to save previous states in order to undo our actions it is recommended to opt for low memory usage in order to maximize our history's size.
Criticality	Want

Table 9: Functional Requirement #9

Requirement	Explanation
Requirement Number	10
Requirement Title	Fair Comparison of Algorithms
Requirement Type	Non-Functional
Requirement Details and Constraints	We must carry out a fair comparison between the algorithms. That means that their performance must be measured with the same standards and procedures.
Criticality	Must

Table 10: Functional Requirement #10

3.2 List of design requirements

The complete list of design requirements is displayed below.

- Digital representation of the board and pieces
- Decision making capabilities
- Representation of the state of the board and the available pieces
- Evaluation of the state of the board
- Graphical representation of the board

- Exporting results functionality
- Continue searching from a file input
- High speed and low memory usage
- Selection of fair comparison methods

The only design requirement we were unable to implement was the resume capability but, in the end, we did not need it since our runs went smoothly.

3.3 Development milestones

There were four development milestones related to the project. The first one was to select the AI algorithms we were going to test. The second one concerns the implementation of the backbone of the program. This includes having the digital representation of the pieces and the board completed as well as the main skeleton of the program. The main skeleton incorporated a main function and the performance measurement code. Therefore, whenever we wanted to implement a new algorithm we were be able to write it in a different file, include that file in our main file and then run it. The third milestone was obtaining the data from our runs and the fourth and final one was performing the comparative analysis between all of the implemented algorithms.

3.4 Project Timeline

The section contains the timeline for the thesis in the form of a Gantt chart.

		Task Mode	Task Name	Duration	Start	Finish	Predecessors
0			**Final year thesis**	**193 days?**	**Thu 15/9/16**	**Fri 9/6/17**	
1			**CS443 (Fall 2016)**	**62 days**	**Thu 15/9/16**	**Fri 9/12/16**	
2	✓		Thesis selection	11 days	Thu 15/9/16	Thu 29/9/16	
3	✓		Bibliographical research	6 days	Fri 30/9/16	Fri 7/10/16	2
4	✓		Literature review	30 days	Mon 10/10/16	Fri 18/11/16	3
5	✓		Writing of thesis proposal	6 days	Mon 21/11/16	Mon 28/11/16	4
6			Creation of A0 poster	4 days	Tue 29/11/16	Fri 2/12/16	5
7	✓		Powerpoint presentation	5 days	Mon 5/12/16	Fri 9/12/16	4
8	✓		**Spring I 2017**	**53 days**	**Wed 25/1/17**	**Fri 7/4/17**	**1**
9	✓		Development of previously tested algorithms	50 days	Mon 30/1/17	Fri 7/4/17	
10			**CS444 (Spring II 2017)**	**46 days**	**Mon 10/4/17**	**Fri 9/6/17**	**8**
11	✓		Comparative analysis of the algorithms	18 days	Mon 10/4/17	Wed 3/5/17	
12			Writing of thesis	24 days	Thu 4/5/17	Mon 5/6/17	11
13			Creation of final poster	2 days	Tue 6/6/17	Wed 7/6/17	12
14			Powerpoint presentation	4 days	Tue 6/6/17	Fri 9/6/17	12

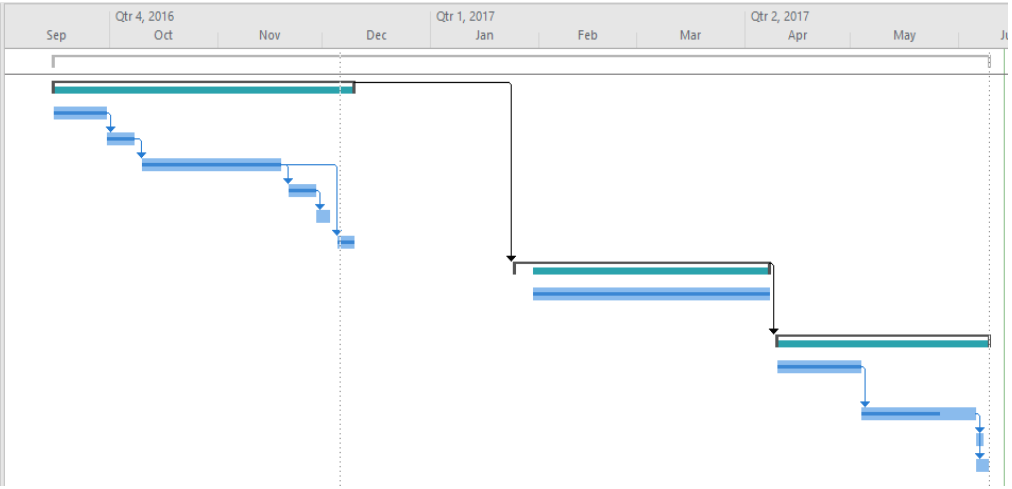


Figure 3: Gantt Chart

3.5 Contingency planning

Due to time constraints, we were unable to implement as many algorithms as we wanted and we also did not create any new ones. In addition, we had no time to test further configurations of our simulated annealing algorithm. Another aspect that we would like to test is the memory footprint of each algorithm, something that we mentioned in our proposal but we were unable to implement. Finally, except from the GA which was tested thoroughly, since it was the first one to be developed, every other algorithm has room for improvement and we propose some methods in section 6.2.

4 Software and Hardware Development

As it was mentioned in the design requirements, our major performance focal point is speed. According to various benchmarks [31] [32], C++ is one of the fastest languages and in combination with its memory management mechanisms it becomes an ideal candidate for this project. In addition, C++ is one of the languages I feel most comfortable with and have fun writing. The integrated development environment (IDE) of my choice is CLion by Jetbrains. CLion provides every functionality we can expect from a modern IDE but it also shares the interface with other IDEs from Jetbrains so changing programming languages doesn't mean we also have to change our workflow. Notepad++ was also used in a number of occurrences in order to replace characters in text files by using regular expressions.

4.1 System Level Design

The software of the thesis is split into two main programs and four auxiliary programs.

The first main program that was developed was the one that engulfed all three of the AI algorithms. This is the only program we wrote that directly tries to solve the puzzle, everything else is an intermediate software that satisfies another need. It uses as input a text file with the pieces and its output is split into three files:

1. The algorithm and the configuration we tested.
2. The number of edges that were matched with respect to CPU time.
3. The final configuration of the board pieces.

Two versions were produced. The first one has no user interface (UI) and was used in order to generate the results. Every setting regarding the configuration of the algorithms and the number of iterations was hardcoded and the program was run inside the IDE. The second version has a simple command line interface (CLI) and it can be used for demonstration purposes. The user selects the algorithm he wishes to run and the relevant configuration, then the algorithm is run once. Both of them were compiled with the GNU compiler collection under Cygwin. In order to execute them under Windows, POSIX API functionality is required and it can be provided with a DLL (cygwin1.dll) which is installed alongside Cygwin.

The second main program was the one used for our mathematical approach. The input of this program is a text file containing the pieces and the output is a text file with the system of equations we are trying to solve. The output is formatted appropriately in order to be used as input in the program we chose for solving the equations, Macaulay 2.

The four smaller, auxiliary programs were:

1. A program used to check if the pieces provided have the same two edges in more than one corner.
2. A program used to average the log files that were produced by the AI runs.
3. A program used to find the best configuration for the three versions of the GA we tested.
4. A program used to produce a 16x16 puzzle without piece rotations.

All of the above programs will be further explained in the following chapters.

4.2 Description of Algorithms

In this section, we will give an overview of the tested algorithms and explain the way they work. This segment is targeted to the general audience and requires no prior programming, or any other kind of knowledge. A more detailed view of the development process is presented in 4.3.

4.2.1 Genetic Algorithms

Genetic algorithms try to mimic the evolutionary process in nature. There are four main notions that a Darwinian evolutionary system embodies:

- One or more populations of individuals competing for limited resources
- The notion of dynamically changing populations due to birth and death of individuals
- A concept of fitness which reflects the ability of an individual to survive and reproduce, and
- A concept of variational inheritance: offspring closely resemble their parents, but are not identical.[33]

Let us take as an example the population of 20 years old humans currently inhabiting earth. That **population consists of** all the **individuals** that are 20 years old. Each one

of those **individuals** is **characterised by** her/his set of **genes** i.e. **genome**. It is the combination of those **genes that define the attributes** of that individual i.e. how smart, beautiful etc. he/she is. Based on the **level of those attributes** and the context in which that individual lives he/she **has more probabilities** of finding a mate and reproducing i.e. that individual has a **higher fitness**. The **offspring** of that human will **share** some of her/his **genes** but they may also display some level of **mutation**.

Our algorithm follows the same concepts. Each puzzle **piece** is **one gene**. The set of all the pieces form a genome, and that **genome defines** an **individual**. Each individual differs from one another since its pieces will be placed in different positions with different rotations i.e. the placement of those genes in each individual varies. In our context, the individuals with **higher fitness** are the **ones closest to our solution** i.e. highest number of matched edges. Therefore, those individuals have higher chances of being **chosen as the parents** for the next generation of individuals. The **configuration** of the offspring's pieces will be **provided by the parents**. A portion of the child will be identical with the "father" and the rest with the "mother". There is also a chance for a **small mutation** to take place. In every child, there is a possibility that a piece's location or rotation will be altered.

Figure 4: Gene, Individual and Population

There are three main functionalities that a simple GA must implement in order to produce good results in any context: reproduction, crossover and mutation [13].

4.2.1.1 *Reproduction*

Reproduction is the process through which we select the individuals that are going to become the parents of our next generation. A successful reproduction process is one that selects parents who are fit enough in order to have evolution/progress but diverge enough in order not to reach a homogenous population quickly. On the one hand, if our criteria for selection were loose, our parents would be of medium quality and so

would be the offspring. On the other hand, if the criteria were strict our parents would be the same best individuals over and over again resulting in similar offspring and therefore evolution would come at a halt. A method to overcome this and choose something in-between the two extremes is called roulette wheel selection. Each individual in our population gets a roulette wheel slot in proportion to its fitness. Whenever we want to create an offspring and we have to choose a parent, we spin the wheel and select the individual on whom the ball lands. By following this method, fitter individuals have higher chances of being selected but medium to low fitted individuals are also going to be picked from time to time leading to a balance between diversity and progress.

4.2.1.2 *Crossover*

Crossover is the method through which we combine two parents in order to produce an offspring. There is no standard way to achieve this since it depends on the nature of the problem we are facing. In our case, two individuals, the parents, are two different configurations of the board pieces. Therefore, a natural and simple way to combine them would be to replace a portion of the father with an equal portion of the mother and vice versa. The size of that portion was chosen to be at least four pieces wide/high and at maximum six pieces wide/high.

4.2.1.3 *Mutation*

Mutation is the process through which we occasionally alternate an offspring right after it is created. Again, the mutation method is problem-specific and in our case, we have created two. For every piece in the offspring configuration, there is a very small chance to either rotate it or swap its position with another piece. Mutation is a helpful procedure in our quest for an optimal solution since reproduction and crossover may become overzealous and lose some potentially useful genetic material [13].

4.2.2 Simulated Annealing

Simulated annealing is another algorithm inspired by the physical world. In metallurgy, annealing is the process of gradually cooling down a metal or glass in order to toughen it. The idea behind the algorithm is to start with a high temperature, a random starting configuration and then gradually explore the search space while also lowering the temperature. In every step of the way we either rotate a piece or swap the position of two pieces. If the new configuration is better than the old one we accept it.

Even if it is not, there is still a chance to accept the new configuration with the probability being proportionate to the temperature. The highest the temperature the highest the possibility to accept a worse solution. As the pieces on the board are cooling down, we start accepting only better movements. Following this procedure, we allow the algorithm to explore multiple directions of the search space in the beginning of the run. As a result, we do not get stuck in a bad position i.e. local minimum.

4.2.3 Backtracking

Backtracking is the simplest of the three AI algorithms. Imagine being locked in a room with 256 doors. Some of them are locked some of them are not. You start trying out the doors until you find one that is unlocked and you step through it. Then you arrive at another room with 255 doors, again, some of them are locked others are not. You keep entering through the first unlocked door until you either reach the exit or you are faced with only locked doors. If it is the latter, then you go back to the previous room and try the next unlocked door. The process continues until you are free. In our puzzle, each location on the board represents a room and each piece a door. We start trying out pieces at the bottom left corner of the puzzle until we find one that fits. Afterwards, we move to the next room, the position on the right and check the remaining 255 pieces for one that matches. Unless we are extremely lucky, we will reach a point where no piece will fit. Then, we will have to go back, remove the last piece we placed and try the others. This is a brute force algorithm that is guaranteed to find a solution given enough time. Unfortunately, the size of the puzzle renders an exhaustive search of all the paths we can follow futile. Nonetheless, we developed this algorithm in order to compare its efficiency in finding a partial solution in comparison to the other algorithms.

4.2.4 Mathematical Approach

Our inspiration for tackling the problem with a mathematical approach came from a paper by Kovalsky et al. [17]. In their research, they propose a general method for solving edge-matching puzzles and state that their “attempts to solve very simple puzzles using generic polynomial system solvers met with limited success”. We wanted to verify their findings and also try an alternative method. In this section, we will describe their polynomial representation, specialized for EII, and also introduce our view at solving the puzzle with equations.

4.2.4.1 *Polynomial Representation*

Our first goal is to mathematically represent the attributes of the board and the pieces. Therefore, we consider that the puzzle and the pieces are placed on a plane with Cartesian coordinates and each piece has width and height of 2. The bottom left corner of our puzzle is placed in the origin of our axis $(x, y) = (0,0)$. The list below states every part of our formulation:

- Centre location of *i*th piece: $t_i \in \mathbb{R}^2$
- Relative position of the centre of edge j of piece i with respect to t_i: $b_{i,j} \in \{(0,1), (1,0), (0,-1), (-1,0)\}$
- Pattern of edge j of piece i: $c_{i,j} \in \{0,1, \dots ,22\}$, where 0 is the grey pattern of the border
- Orientation of edge j of piece i: $\theta_{i,j} \in \{0, \frac{\pi}{2}, \pi, 3\pi/2\}$. We consider the right edge to have $\theta = 0$ and then we consider a counter-clockwise rotation
- Absolute position of edge j of piece i: $t_i + b_{i,j}$

An example with the top edge of a piece with $i = 5$ is displayed in Figure 5. In addition, we consider our board as one more piece with $i = 0$ and fixed centre location $t_0 = (16,16)$, since we said that each piece has width and height equal to 2. The attributes of its edge elements are $b_{0,j}, c_{0,j}$ and $\theta_{0,j}$. Finally, we start our approach by supposing that our pieces have already the correct orientation and in order to solve the puzzle we just have to place them in the correct position t_i, with no need to rotate them.

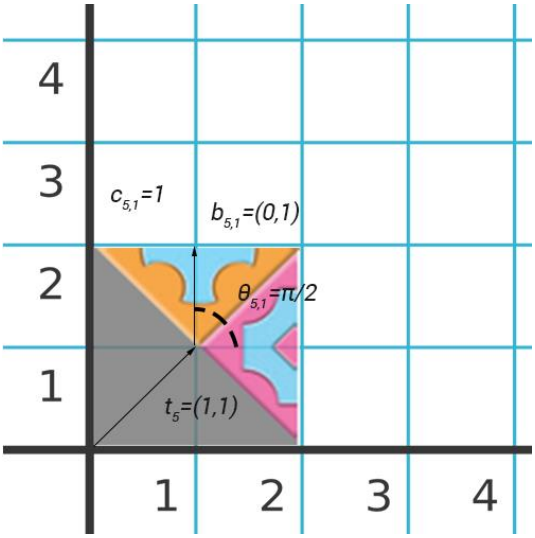


Figure 5: Representation Example for Top Edge of Piece with i=5

If $t_1, t_2, \dots, t_{256}$ is a solution to our puzzle then each attribute of every edge of our puzzle must satisfy the following equations. If we consider i and $\hat{\imath}$ two touching pieces and j and $\hat{\jmath}$ their matching edges, the edges must have:

- Equal absolute positions: $t_i + b_{i,j} = t_{\hat{\imath}} + b_{\hat{\imath},\hat{\jmath}}$
- Equal patterns: $c_{i,j} = c_{\hat{\imath},\hat{\jmath}}$
- Opposite orientations: $\theta_{i,j} \equiv \theta_{\hat{\imath},\hat{\jmath}} + \pi \ (mod\ 2\pi)$

We also define the signed indicator function $s_{i,j}(c, \theta)$ as

$$s_{i,j}(c,\theta) = \begin{cases} 1, & c_{i,j} = c \ and \ \theta_{i,j} = \theta \\ -1, & c_{i,j} = c \ and \ \theta_{i,j} = \theta + \pi \\ 0, & otherwise \end{cases}$$

Equation 1

Therefore, we have that if $t_1, t_2, \dots, t_{256}$ is a solution for EII then

$$\sum_{i,j} s_{i,j}(c,\theta) f(t_i + b_{i,j}) = 0$$

Equation 2

for every (c, θ) and for any function $f: \mathbb{R}^2 \to \mathbb{R}$.

Our f of choice has to be exponential in order for the solutions of the system to be solutions of the puzzle (see Appendix A in [17]). Following [17], we choose $f_k(u) =$

$2^{k^T u}$, instead of the proposed $f_k(u) = e^{k^T u}$, with $k \in \mathbb{R}^2$, to ensure that the solutions are rational numbers for computational reasons. Therefore, Equation 2 becomes

$$\sum_{i,j} s_{i,j}(c,\theta) 2^{k^T (t_i + b_{i,j})} = 0$$

Equation 3

We simplify our notation by setting ${a_i}^{(k,c,\theta)} = \sum_j s_{i,j}(c,\theta) 2^{k^T b_{i,j}}$ and we get

$$\sum_i {a_i}^{(k,c,\theta)} 2^{k^T t_i} = 0$$

Equation 4

We remind that $t_i \in \mathbb{R}^2$ and it is the location of the centre of piece i, hence we can rewrite Equation 4 as

$$\sum_i {a_i}^{(k,c,\theta)} 2^{k^T \binom{x_i}{y_i}} = 0 \Leftrightarrow \sum_i {a_i}^{(k,c,\theta)} 2^{kx_i + ky_i} = 0 \Leftrightarrow \sum_i {a_i}^{(k,c,\theta)} 2^{kx_i} 2^{ky_i} = 0$$

Equation 5

Finally, we set $2^{x_i} = X_i$ and $2^{y_i} = Y_i$ in order to get

$$\sum_i {a_i}^{(k,c,\theta)} X_i^k Y_i^k = 0 \Leftrightarrow \sum_i {a_i}^{(k,c,\theta)} (X_i Y_i)^k = 0$$

Equation 6

Notice that the variables in Equation 6 are $X_1, \dots, X_{256}$ and $Y_1, \dots, Y_{256}$. It is now time to define k. We derived that if $t_1, \dots, t_{256}$ are solutions to EII then Equation 6 holds. However, we also want the converse, if we find $X_1, \dots, X_{256}$ and $Y_1, \dots, Y_{256}$, we want them to be a solution to the puzzle. The authors of the paper provide a constructive proof that this holds if for each edge type (c,θ) we gather K number of equations with distinct $k = 1, \dots, K$. K is defined as the number of edges of type (c,θ).

$$K(c,\theta) = \#\{edges\ of\ type\ (c,\theta)\}.$$

Edges θ and φ where $\theta \equiv \varphi + \pi\ (mod\ 2\pi)$ count as equal i.e. opposite edges are considered to be one type.

To sum up, if we know the correct orientation of each piece, we can get their correct positions by solving a system of equations. That system is created if for every edge type (c, θ) we get K equations of the form

$$\sum_i {a_i}^{(k,c,\theta)} (X_i Y_i)^k = 0$$

Equation 7

with distinct $k = 1, \dots, K$.

Up until this point we supposed that we had the correct orientation for every piece but that doesn't hold true in EII, each piece has four possible rotations. In order to tackle this, the authors of the paper extended their framework by adding in their equations 3 more copies of every piece, each one with different rotation, and then summing them all together. Imagine it as summing 4 copies of Equation 7 but in each copy the pieces are rotated by 90 degrees more. This multiplied the complexity of the system and as was the case with them, we were also unable to produce the Gröbner basis of the system. Then we chose to follow a different approach. We wanted to see whether we could solve the system with Gröbner bases if we did have the correct rotations.

4.2.4.2 *Solving the Polynomial System of Equations*

In order to explain the attempted method of the authors [17], but also our alternative approach, we first have to explain rings, ideals and Gröbner bases. We need methods/algorithms that solve systems of polynomial equations, in an arbitrary number of variables, with arbitrary degrees. We will start by giving an overview of polynomial systems.

4.2.4.2.1 Rings and Fields

In a system of polynomial equations, the sets of coefficients and solutions are $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ etc. A **ring** R is a set equipped with two operations:

$$+: R \times R \to R \text{ and } \cdot: R \times R \to R$$

satisfying certain compatibility axioms inspired by properties of addition and multiplication e.g. $a \cdot b = b \cdot a$ or $a(b + c) = ab + ac$ [34]. If $\forall a \neq 0, a \in R, \exists b \; s.t. \; a \cdot b = 1$ then R is called a **field** and denoted by F. Hence, $\mathbb{N}$ and $\mathbb{Z}$ are rings, while $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ are fields.

A polynomial f in the variables $x_1, x_2, \dots, x_n$ with coefficients in R is an expression of the form $f(x_1, \dots, x_n) = \sum_a c_a x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}$, $a = (a_1, \dots, a_n) \in R$. The set of all polynomials in n variables, with coefficients in a ring R, is denoted by $R[x_1, x_2, \dots, x_n]$ and it can be proven to be a ring itself [34] e.g. $x^2 + 1 \in \mathbb{Z}[x]$, $x_1 x_2 - \frac{3}{2} x_1 x_3 \in \mathbb{Q}[x_1, x_2, x_3]$. A polynomial ring $R[x_1, \dots, x_n]$ over a field F is **not** a field e.g. $\frac{1}{2}x + 1, x - \frac{1}{2} \in \mathbb{Q}[x]$ but $\frac{x/2+1}{x-1/2} \notin \mathbb{Q}[x]$.

4.2.4.2.2 Ideals

A subset $I \subseteq R[x_1, \dots, x_n]$ is called an **ideal** if $\forall r, s \in I$ and $\forall f \in R[x_1, \dots, x_n]$ we have $r - s \in I$ and $r \cdot f \in I$. We say that an ideal I is generated by some polynomials $f_1, f_2, \dots, f_s$ and we write $I =< f_1, \dots, f_s >$ if any polynomial $h \in I$ can be written as $h = g_1 f_1 + \cdots + g_s f_s$. For example, with $I =< x + 1 > \in \mathbb{Q}[x]$ we have $(x + 1)^2 \in I$, $(x^3 + 5x)(x + 1) \in I$ and $x^3 + 3x^2 + 3x + 1 = (x + 1)^3 \in I$. Another example with two variables is $I =< x, y > \in \mathbb{Q}[x, y]$ and we have $xy^2 + y^3 = y^2 x + y^2 y \in I$ or $xy^3 + x^4 + y^2 = (y^3 + x^3)x + yy \in I$ or $x = 1x + 0y \in I$, but $xy + 1 \notin I$.

If $I =< f_1, \dots, f_s > \subseteq R[x_1, \dots, x_n]$ and $h \in I$, then $h = g_1 f_1 + \cdots + g_s f_s$. If $a = (a_1, \dots, a_n) \in R^n$ is a common solution to all polynomials $f_1, \dots, f_s$ then $(a_1, \dots, a_n)$ is a solution of h.

Given $f_1, \dots, f_s \in R[x_1, \dots, x_n]$ the set of common solutions is denoted by

$$V(f_1, \dots, f_s) = \{(a_1, \dots, a_n) \in R^n : f_i(a_1, \dots, a_n) = 0, \forall i = 1, \dots, s\}$$

Given an ideal $I \subseteq R[x_1, \dots, x_n]$

$$V(I) = \{(a_1, \dots, a_n) \in R^n : h(a_1, \dots, a_n) = 0, \forall h \in I\}$$

e.g. $V(x, y) = \{(0,0)\}$ and $V(I) = \{(0,0)\}$

Generally, if $I =< f_1, \dots, f_s >$ then $V(I) = V(f_1, \dots, f_s)$.

For our approach, we would also use a theorem called **The Weak Nullstellensatz** which states that if $I \subseteq R[x_1, \dots, x_n]$ satisfying $V(I) = \emptyset$. Then $I = R[x_1, \dots, x_n]$ [35].

4.2.4.2.3 Gröbner Bases

Given a system $\{f_1 = 0, \dots, f_s = 0\}$, we construct the ideal $I =< f_1, \dots, f_s >$. We may find other polynomials $g_1, \dots, g_s$ such that $I =< g_1, \dots, g_s >$. Then, the system $\{g_1 = 0, \dots, g_s = 0\}$ has the same solutions. For example, if we have the system

$$\begin{cases} x - y = 0 \\ x + y = 0 \end{cases}$$

we have the ideal $I =< x - y, x + y >$. But we may also have the polynomials $g_1 = x, g_2 = y$ with ideal $J =< x, y >$. The solution to the latter system is $\{(0,0)\}$ and since $J = I$ those are also the solutions for the first system. $J = I$ because

$$x = \frac{1}{2}(x - y) + \frac{1}{2}(x + y) \in I$$
$$y = -\frac{1}{2}(x - y) + \frac{1}{2}(x + y) \in I$$

therefore $J \subseteq I$ and

$$x - y = 1x + (-1)y \in J$$
$$x + y = 1x + 1y \in J$$

therefore $I \subseteq J$ and as a result $I = J$.

The above is an example of what we call Gröbner basis. A Gröbner basis of an ideal is a set of polynomials $\{g_1, \dots, g_s\}$, such that $I =< g_1, \dots, g_s >$ and some extra, computationally optimal, properties. It is out of the scope of this thesis to provide detailed definition. Instead, we focus on computations and for further details on the subject readers can refer to [35].

4.2.4.2.4 Our Approach

Our approach was to split the problem in two parts. For the first part, we had to find the correct rotation of each piece and for the seconds part, we had to solve the system that would derive from those rotations. In order to find the correct rotation, theoretically, we would generate every possible combination of rotations, 4^{256}, hence the "theoretically", and then, by using The Weak Nullstellensatz, we would find the correct rotation if the ideal of our system was not equal to our ring. In order to solve the system, we would use the Gröbner basis of our ideal. A small number of systems was produced, for testing purposes, and the program we used for the equality mentioned above and the generation of the Gröbner basis was Macaulay 2.

4.3 Description of Software

In this section, we will present step by step the code development process of the thesis. In order to explain our development practices, choices and the reasoning behind them, this part has to dive deeper into the source code. Therefore, adequate programming knowledge is required. For a code-free explanation of the algorithms see 4.2.

4.3.1 Code Design: Flowcharts

The flowcharts of the most important pieces of the code are displayed below. Namely, we exhibit the three AI algorithms since they are essential to the main point of the thesis i.e. comparative analysis of algorithms in solving EII. Hence, the flowcharts of the auxiliary programs are omitted but their development is still explained in 4.3.2.

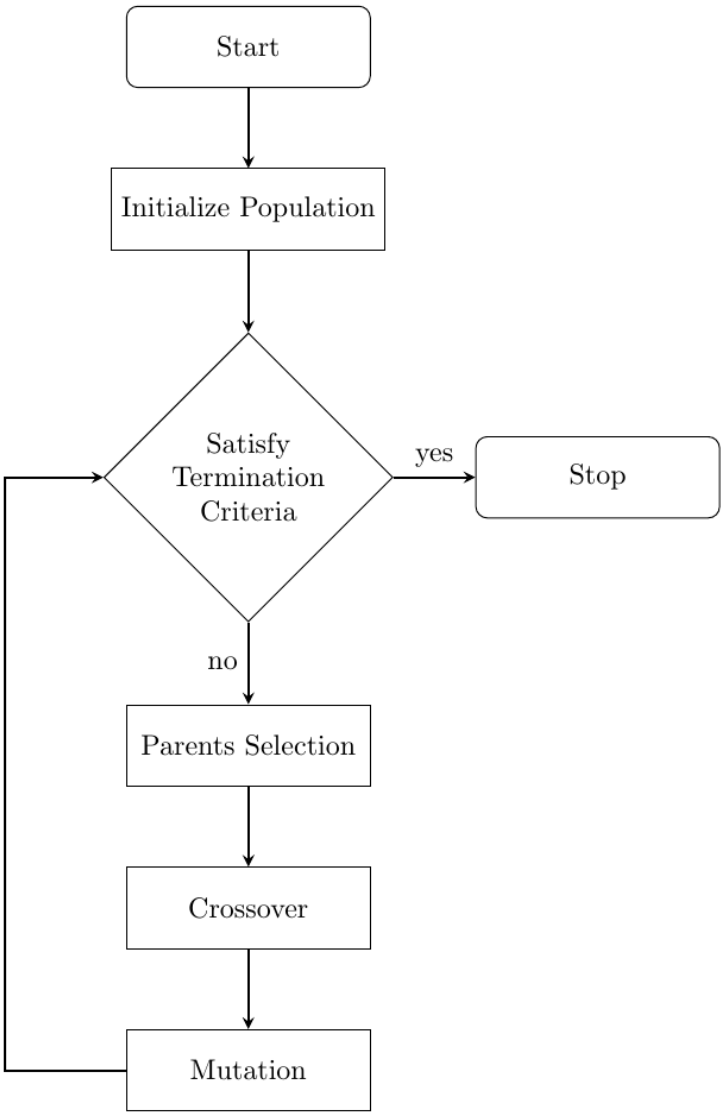


Diagram 1: Genetic Algorithm Flowchart

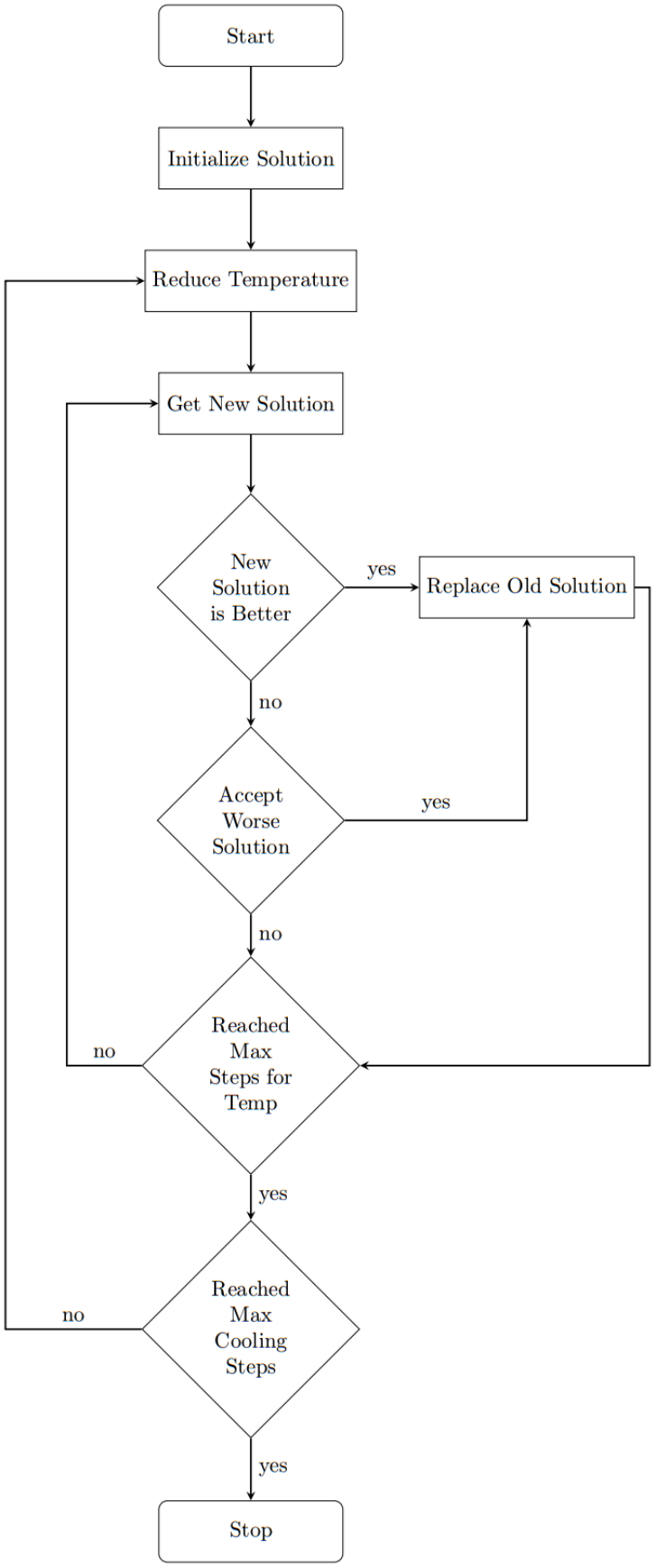


Diagram 2: Simulated Annealing Flowchart

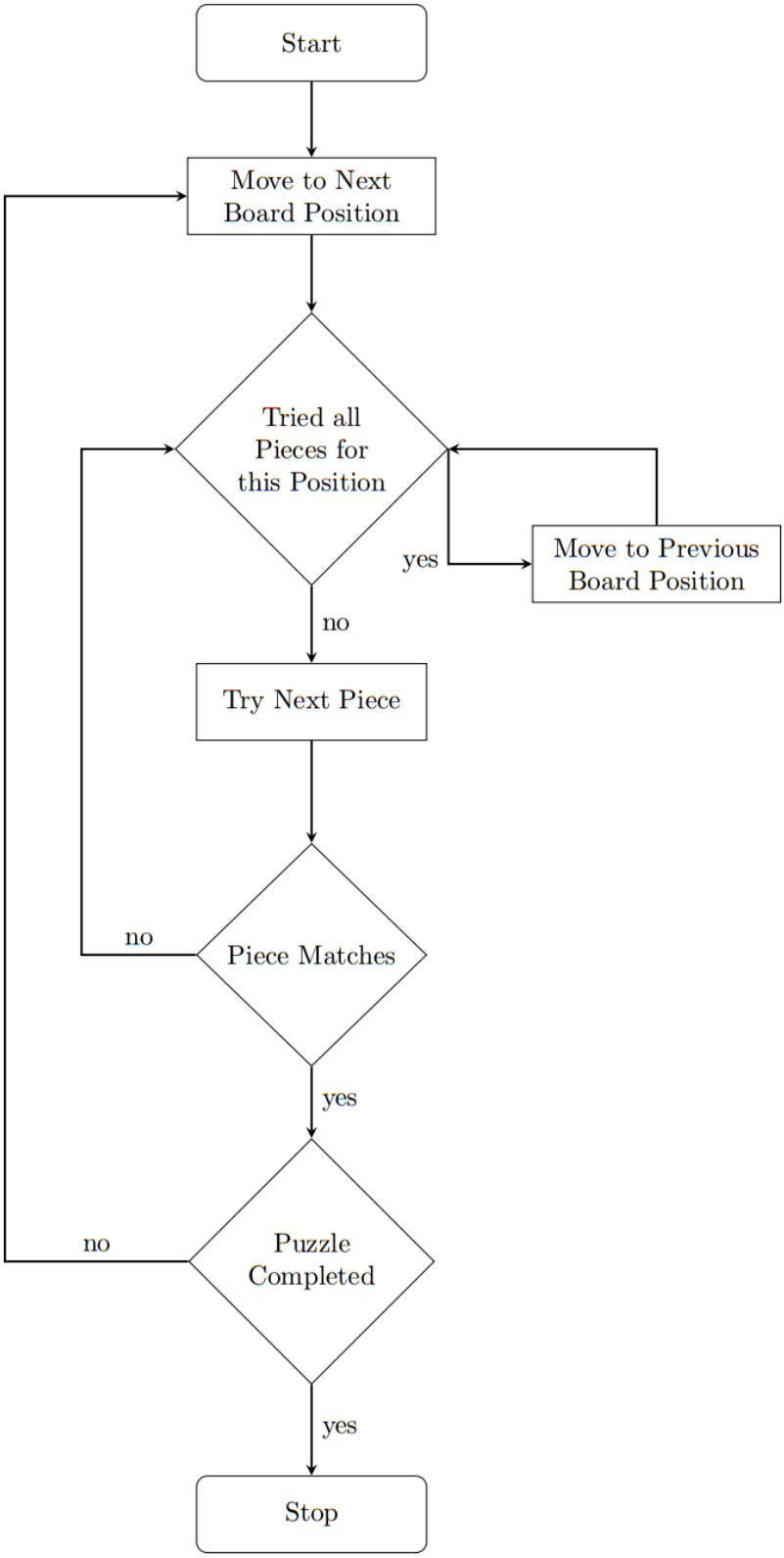


Diagram 3: Backtracking Flowchart

4.3.2 Source code development

4.3.2.1 *Pieces File*

Our first objective was to acquire a file that contained the pieces in order to import them into our programs. The convention in the EII community is to assign a number to each edge pattern from 0 to 22 with zero being assigned to the grey, border pattern. The rest of the patterns were assign numbers from 1 to 22 based on their location in the second page of the EII manual.

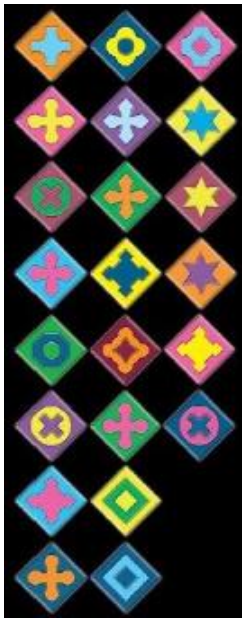


Figure 6: Patterns Image at the second Page of the Manual

The numbering starts from the top left pattern and goes downwards. Therefore, the orange edge with the grey cross is number 1, the pink edge with the yellow cross is number 2 etc. Although we own a physical copy of the puzzle, manually importing all the edges in a text file would take time and it would be prone to mistakes which are unacceptable at this stage. One wrong edge in our file would destroy the validity of our thesis and also probably render the puzzle unsolvable. Thankfully, the desired text file was found online [36]. In order to check its validity, we used another program that was also available online. The author of the program had created a checksum from the correct pieces and it was checked against the pieces found online. The program can no longer be found online so we uploaded in the Yahoo group dedicated to EII in case anyone else requests it [37]. The pieces file consists of 256 lines and each line describes a piece. The first line defines piece number one, second line piece number two and so on. In order to be consistent with the orientation, the first number defines

the top edge and then we go clockwise. For example, the line `1,17,0,0` defines the piece displayed in Figure 7.

Figure 7: Piece Encoding Example

In addition to the official pieces file, four extra files were created manually for testing purposes. Two with 4 pieces i.e. 2x2 board, and two with 9 pieces i.e. 3x3 board. In both board sizes, one set of pieces did not require any rotation in order for the puzzle to be solved. In other words, the pieces' orientation inside the file was the one required in the solution.

4.3.2.2 *Piece Class*

The second step was to create the class that would represent a piece inside our program. The class diagram is found in Figure 8.

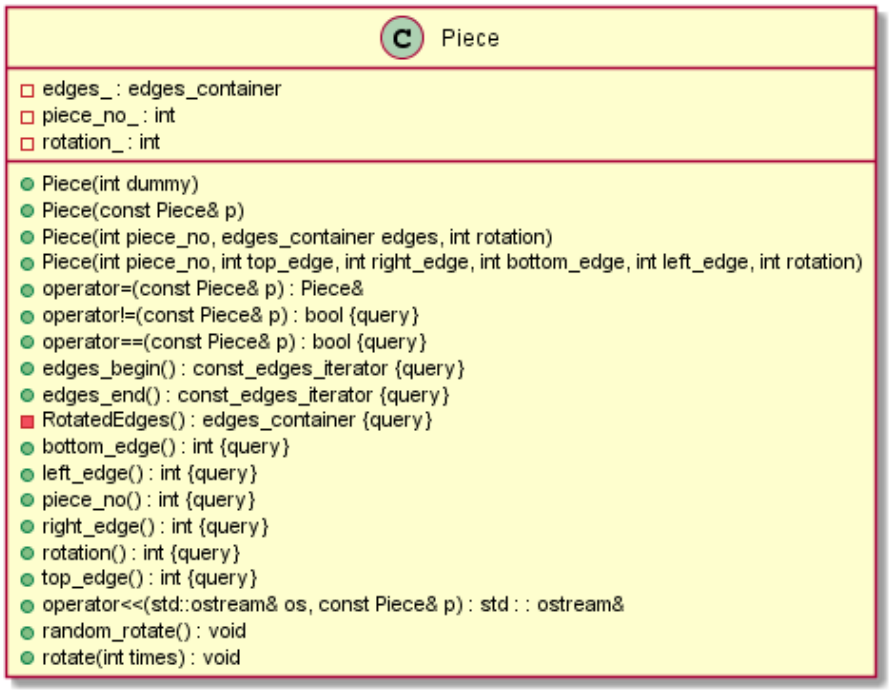


Figure 8: Piece Class

In order to have a variety of options when creating an object, we overloaded the constructor function. We can either initialize an empty object, copy a piece we

already have, or create a new one by specifying its edges and optionally, its rotation. The new feature of C++11, constructor delegation, was used in order for one constructor to call another and avoid code repetition [7]. A constructor with a dummy integer variable is implemented since it was required by the library we used for the GA. In order to encapsulate the underlying representation of the edges' data structure we employed the iterator design pattern [38] and created two functions that return a constant iterator in the beginning and ending of the structure. The only alteration a user is allowed to make on the object is to rotate it. Every other member function, except the assignment operator, is a constant and prohibits changing the class variables in order to avoid mistakes. In addition, we further encapsulated the container and iterators of the edges by using typedefs. As a result, changing from one data structure to another required editing only one line of code and correcting any compiler errors. Although our code is not completely container-independent, this is as close as we can get [8]. The data structure that we settled with was the vector since it seemed natural as the values we had to store were in the form of `1,17,0,0` and preserving their order was essential. The class also provides a visual representation of the piece in ASCII "art" form as it is shown in Figure 9. In retrospect, there is a flaw in the design of the class that hinders the performance of our algorithms but since the same class is used for all three A.I. methods the comparison is still fair. When we access any of the four edges through one of the four corresponding member functions, the private function `RotatedEdges` is called which returns the edges in the correct, rotated, order. So, if the top edge of the piece `1,17,0,0`, rotated by 1 clockwise, is requested, the `RotatedEdges` returns `0,1,17,0` and the first element, `0`, is returned to the user. This creates extra overhead whenever we request an edge pattern which could have been avoided if we rotated the edges in the container every time we rotated a piece i.e. there would be no reason to call `RotatedEdges`. Finally, there is no need for a destructor since the class does not allocate any memory [6].

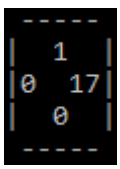


Figure 9: Piece in ASCII form

4.3.2.3 *Pieces Class*

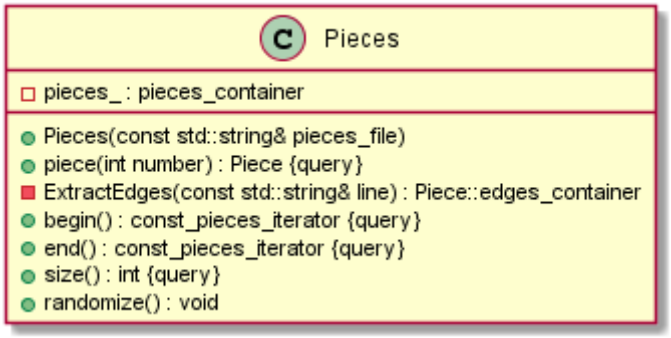


Figure 10: Pieces Class

The third step was to create a class that would read the file with the pieces and store the piece objects described above. The constructor of the class accepts the string with the pieces' filename. Afterwards, we read each line of the file and extract the edges in order to create the pieces which are placed into a vector. The class provides again an iterator in order to access the pieces sequentially but also a method that returns a piece specified by its number. In addition, it exposes a function that returns the number of pieces we have. Finally, in both the GA and the simulated annealing, we have to start with a random board, so we implemented a method that randomizes the order of the pieces inside the container but also their rotations.

4.3.2.4 *Logger Class*

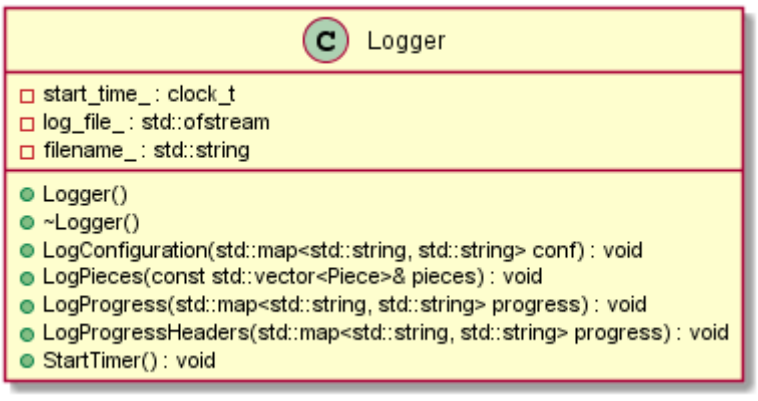


Figure 11: Logger Class

The fourth and final preparation step was to create the class that would log the results of every run. In order to ensure a unique filename for every simulation, when the object is first created, it defines a filename based on the current time and a random number between 0 and 999. Then it creates three files, all with the same name but with different extensions. The first one is used in order to save the configuration of

our algorithm. The user provides a map with the settings in the method `LogConfiguration` and they are saved into a file with a cfg extension in the form demonstrated in Figure 12.

```
algorithm = P11EIISimpleGA
elitism = 0
nConvergence = 500
pConvergence = 1.000000
pCrossover = 0.900000
pMutation = 0.010000
populationSize = 150
selector = 23GARouletteWheelSelector
```

Figure 12: Configuration File Example

The second file is used for logging our process through the runs. It is a Comma Separated File (CSV) with headers in the first row and each row contains the statistics we decided to save (Figure 13). The user has to define the headers once and then he can call `LogProgress` with a map in order to save the progress. A map was used so that the order in which we provide the statistics does not matter since we are using the key in order to place them into the correct column. The clock starts counting when the user calls the method `StartTimer` and the CPU time is calculated as the current time, minus the starting time, divided by clocks per second [39]. In order to perform less I/O operations, it would have been better to save the progress in a buffer and flush it every some interval e.g. ten measurements.

```
cpu_time,generation,maximum,mean,minimum
0.11,1,41.000000,25.836000,13.000000
0.235,2,41.000000,25.704000,14.000000
0.344,3,41.000000,25.756001,14.000000
0.469,4,43.000000,25.228001,12.000000
0.579,5,43.000000,25.184000,13.000000
0.704,6,43.000000,25.476000,16.000000
0.813,7,43.000000,26.304001,15.000000
0.938,8,43.000000,26.507999,15.000000
1.063,9,43.000000,26.596001,16.000000
1.172,10,43.000000,26.604000,15.000000
1.297,11,45.000000,26.667999,14.000000
```

Figure 13: Sample of a G.A. Progress File

The third file is created if we provide a vector of pieces in the function `LogPieces`. It is meant to be the last method we call and it saves whatever board configuration we choose, usually our best result, in a form compatible with a program called EternityEditor which can visualise the board [40]. An example of the file structure

and the corresponding visual representation is demonstrated below, using a sample 3x3 puzzle with 4 edge patterns.

```
0  1  3  0      0  3  1  1      0  0  4  3
3  2  1  0      1  1  4  2      4  0  3  1
1  4  0  0      4  2  0  4      3  0  0  2
```

Figure 14: Board Configuration File

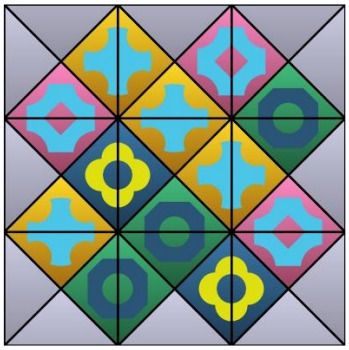


Figure 15: EternityEditor.exe Visualization

In this class, we had to implement a destructor so as to close the progress file when we are finished and the object is destroyed. Because we write multiple entries per second in our logs, the stream has to remain open during the running time of the algorithms.

4.3.2.5 *Genetic Algorithm*

In this section, we explain in more detail the implementation of our GA. We will also cover some GA related content that was intentionally omitted in 4.2.1 since it is not as essential in understanding the algorithm. The backbone of our GA is a library called GAlib. It was chosen since it provides different flavours of GA, which will be explained further below. The only thing required in order to work with the library is to define a representation for one individual and the methods responsible for his initialization, mutation, evaluation and crossover. Given that our board is two-dimensional, we created a subclass from a two-dimensional array provided by the library (see Figure 16). Each slot of that array is to be filled with a gene, in our case the genes are objects of the class Piece described above. In order to initialize the individual, we provide a set of all our pieces and the `Init` method creates a random configuration.

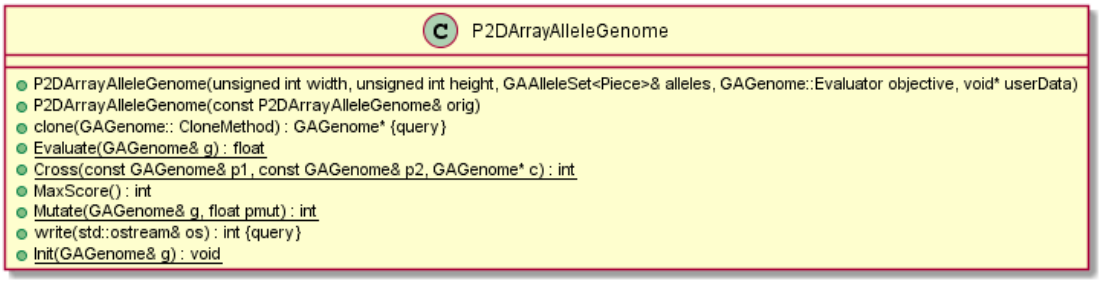


Figure 16: Genome Class

Before starting the algorithm, we have to define its behaviour.

4.3.2.5.1 Population Replacement Method

By population replacement method we denote the amount of genetic material that is going to remain the same from generation to generation. Since the population size in our algorithm has to remain constant, when a new individual is born, it has to take the place of an old individual. In other words, we have to define how many new individuals we are going to have per generation. The library provides four flavours of this method and this variety was the reason we chose it. Out of the four, we tested the performance of three.

1. Simple GA: In every new generation, we have a completely new set of individuals. If, for example, the first generation of solutions has 50 individuals, then the second one will have 50 different individuals which will be the children of the selected parents. However, there is an option called elitism which, if set to true, carries over the best individual of a generation to the next one. Experiments were performed with elitism set both to true and false. Each time we require an offspring, we use our selection method in order to select a reproduction candidate and copy her/him into a new pool of individuals which will be our parents. After a mating pool of the same size as our population is created, we randomly mate the individuals inside. In our example, this new mating pool will have 50 candidates from the old population and afterwards we will randomly mate them 50 times in order to produce our new generation [13].
2. Steady State GA: For every new generation, we specify the number of new individuals we would like to have. Afterwards, we select two parents and create one offspring at a time which immediately competes for survival with the worst offspring of the population. If the score of the offspring is better, it is

inserted into the population and the worst individual gets removed [33]. Keep in mind that there is instant insertion of an offspring in the population, therefore a new individual may take the place of one recently created if the latter has the worst fitness in the population. Hence, even if we ask for 20% new individuals in a population with size 50 we may get less than 10 offspring. However, the probability is low since we are mating good parents and usually producing better offspring.

3. Incremental GA: In every new generation, we have one or two new individuals. The number remains constant during a run and we tested both. The main difference of the incremental GA, otherwise known as GENITOR, is that it makes explicit use of ranking. Because one or two offspring are created each time, we abandon the notion of generations and we view our population as an ordered list with the best individuals at the top [41]. We chose the default replacement method which was for the new individuals to take the place of the worst genomes of the previous generation. Another available option was for the children to take the place of their parents.
4. Deme GA: This GA has multiple independent populations evolving in parallel. Each population evolves using a steady state GA and in each generation a fixed number of the best individuals migrate from one population to another. There is one "master" population which gets populated in each generation with the best individuals from all the other populations. The implementation of this GA was left as an extra and it was not developed due to time constrains.

4.3.2.5.2 Selection Method

As mentioned above, we need to define a method for selecting a parent whenever necessary. We selected one of the simplest methods available which is that of the roulette wheel. This method is conceptually equivalent to giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness [14]. Mathematically, if f_i is the fitness function which returns the fitness of one individual, then each individual has probability $pselect_i = f_i / \sum f$. Other options included ranked selection where the best member of the population is selected every time and tournament selection. Tournament selection uses the roulette wheel in order to choose two individuals and then selects the fittest. Both of those selection methods would choose better individuals than the roulette wheel but it would lead to earlier convergence of the population, for that reason, they were avoided.

4.3.2.5.3 Scaling Method

In our 16x16 board with 256 pieces we have to connect 544 edges. Therefore, in our population, each individual can be ranked based on the amount of matched edges e.g. 37/544 or 402/544. However, using this raw score inside our selection method can cause two shortcomings. Firstly, in the beginning of a run, some individuals will have far better scores than others because of the randomness during initialization. Therefore, those individuals will be selected in a much higher rate than the rest and lead to premature convergence of the population. Secondly, during the end of a run, the average fitness will be close to the best fitness and as result average individuals will have almost the same chances at reproducing as the best individuals. In order to overcome this, we use a scaled fitness which we obtain by linearly scaling our scores with an equation of the form $f' = af + b$, where f is our raw score, f' our scaled score and a, b are variables automatically calculated by our library based on Goldberg's book [13].

4.3.2.5.4 Crossover Method

After two individuals are selected for mating, there is still a small chance that they will not bear offspring. This chance is called $pCrossover$ and we experimented with values from 0.6 to 0.9 with a 0.1 step. If they do mate, then we enter the crossover phase. The way two of our individuals mate is defined inside the `Cross` function. We can consider one of the two as the mother and the other as the father. Afterwards we define a rectangular with random width and height, from four up to six pieces. Eventually, the pieces inside the specified region are swapped and we have our offspring. An example with a smaller board is presented in Figure 17. The procedure of swapping the pieces is:

1. Copy mom to child
2. Remove from the child the pieces that are going to be copied from the dad
3. Place in a stack any pieces left inside the child's selected region
4. Copy the selected region from the dad to the child
5. Fill child's empty slots with pieces from the stack

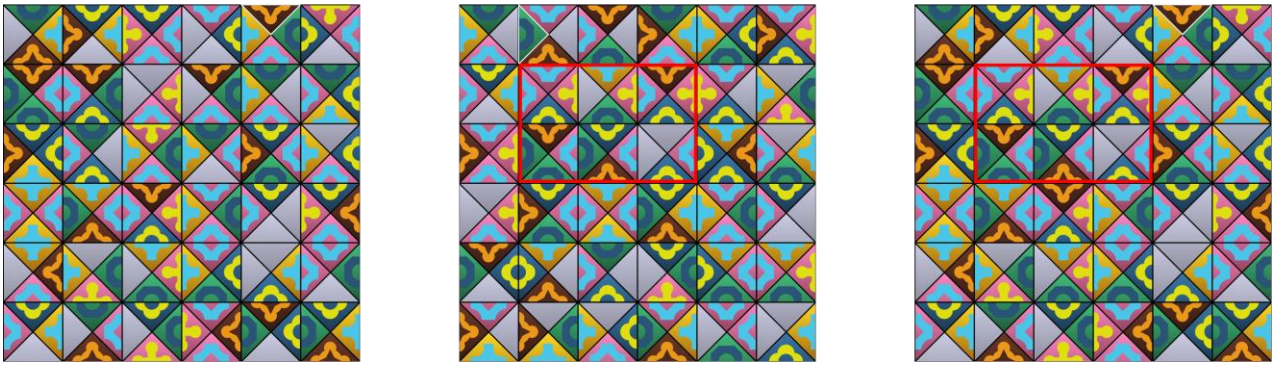


Figure 17: Mom, Dad and Child Respectively Swap Region Marked with Red

4.3.2.5.5 Mutation Method

After crossover, we enter the mutation stage. Each piece has a small chance to be mutated i.e. to change in a subtle manner. We have defined two mutation operators. The first one rotates randomly the piece and the second swaps its position with another piece. The two mutations can happen simultaneously since each one of them occurs with a certain probability. We experimented with values from 0.5% to 1.5% with a step of 0.5%.

4.3.2.5.6 Termination Criteria

Finally, we have to define when our GA will stop evolving. There were two built-in options. The first one was stopping at a specific number of generations but since the behaviour of our GA was unknown prior to experimenting, we could not guess a valid number. Therefore, we chose the second one which stopped the evolution when no evolution occurred for a number of generations. When the GA converges, all the candidates are often exactly alike, so even if we mate them there is going to be no progress [42]. The only way to improve our score would be through luck during mutation. It was decided that if after 500 generations there was no improved we would exit the evolutionary process.

4.3.2.6 *Simulated Annealing*

As mentioned in 4.2.2, simulated annealing and the acceptance criteria of a worse solution were inspired by the physical annealing process [43]. In condensed matter physics, annealing is a thermal process for obtaining low-energy states of a solid in a heat bath. The procedure is broken down in two tasks:

1. We maximize the temperature of the heat bath in order to liquidate the solid.

2. We slowly decrease the temperature until the particles of the solid arrange themselves in the ground state of the solid.

In the beginning of the procedure the particles are arranged randomly but as the temperature is lowered they form a highly structure lattice and the energy of the system is minimal [44]. Therefore, the starting state of our board is also randomly generated, this is our first step. Afterwards, we set our temperature to the maximum and start cooling down our system by a constant fraction. In each temperature, we evaluate a neighbour solution for our board. By neighbour solution, we mean that we get a solution really close to the one we have already. In order to achieve this, we either rotate a piece or swap the position of two pieces. If our new solution is better than our previous one we accept it. If not, we accept it only if

$$e^{-\frac{(C(s_i)-C(s_{i+1}))}{t_i}} \geq r,$$

where $C(s_i)$ is the cost of the previous solution i.e. our score, $C(s_{i+1})$ is the cost of our new solution, t_i is the current temperature and r is a random number $0 \leq r \leq 1$ [11]. Due to the form of the exponent, the probability decreases exponentially the worse our new solution is. In addition, the probability decreases as the value of t_i goes down. Worse moves are more likely to be accepted in the beginning when t_i is high and they become rarer as t_i is lowered [12]. As a result, we avoid getting stuck in local optima during the first stages of our run. The process of lowering the temperature is repeated a predefined amount of times and for each temperature we are going over a fixed number of neighbour solutions. We lower the temperature only when our score is not increased during a loop.

Algorithm 1 Simulated Annealing

```
create initial solution
calculate its score
initialize temperature
while cooling step < max cooling steps do
    lower temperature
    while step < steps per temp do
        get new solution and calculate its score
        if new solution is better or acceptance criterion is met then
            save new solution
        end if
    end while
    if the score was increased then
        increase temperature
    end if
end while
```

Figure 18: Simulated Annealing Pseudocode

4.3.2.7 *Backtracking*

The backtracking algorithm is a brute force technique that is guaranteed to provide a solution given enough time. Although we could also develop a breadth-first search, we decided to go with a depth-first search since it uses much less space. A breadth-first search would store all the nodes at the current level and since EII has a branching factor of 382 [21] this would quickly get out of hand.

Algorithm 2 Backtracking

```
for every row do
    for every column do
        available pieces for this position ← remaining pieces
        if we backtracked to this position then
            remove the previously placed piece from the available pieces
        end if
        do
            check if next piece matches
        while piece doesn't match
        if none matches then
            backtrack
        end if
    end for
end for
```

Figure 19: Backtracking Pseudocode

We start by placing a matching piece at the bottom left corner of the board and continue matching pieces from left to right. After reaching the right border, we follow exactly the same procedure but on the above row. The movement goes on like this until the algorithm finds a solution or it is forcefully stopped. Whenever we reach a

dead end, we remove the last placed piece and try another one in its place. Each board location is assigned a list which holds all the available pieces for that slot. The first time we visit a position the eligible pieces are the ones we have yet to place. If we backtrack to a position, then the available pieces are the remaining but without the one we just removed. We have to be cautious with three pieces in order not to ignore some portions of our search space. Every time we want to place a piece, we check if any of its four corners match both the right edge of the left piece and the top edge of the bottom piece (see Figure 20).

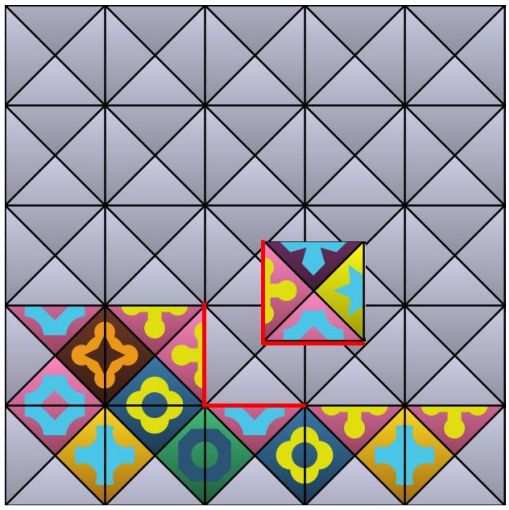


Figure 20: Backtracking Match Criteria

We also mentioned that when we backtrack to a position the available pieces to try out are the remaining pieces minus the one we had placed. The catch is that 3 pieces contain the same combination of edges in two corners. Namely pieces numbered 173, 199, 233. Hence, the first time we backtrack to those pieces, instead of removing them, we simply rotate them in order for the other identical corner to match.

Figure 21: Piece 173

Because the algorithm is not going to find a solution in order to stop, we have to kill the process after we decide that a satisfactory amount of time has passed. Since we are

comparing the speed of our algorithms, as long as we let backtracking run more time than the slowest of the rest of the algorithms, our comparison can be performed.

4.3.2.8 *Creating the System of Equations*

The goal of this program is to provide us with a text file which will include the system of equations described in 4.2.4. The string representation of those equations must be compatible with Macaulay 2 in order to use them as input. To achieve that we had to create two more classes:

1. A point class which represents a point in the Cartesian plane. The class provides the two-dimensional vector arithmetic required for the generation of equations.

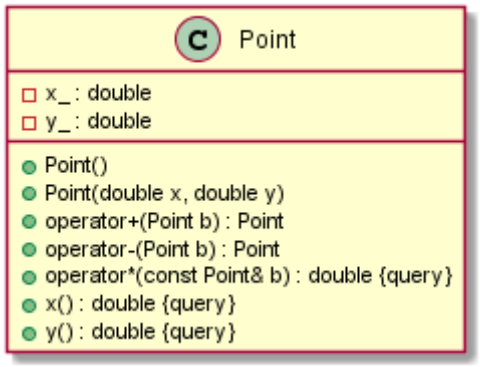


Figure 22: Point Class

2. An edge class which represents one of the four edges of a piece. Our methodology demands that we abandon the simple representation we used in our AI algorithms. While the only property that defined one edge in the AIs was the pattern's number, now we have to add our b, c and θ but also provide the functionality of function s (see 4.2.4.1)

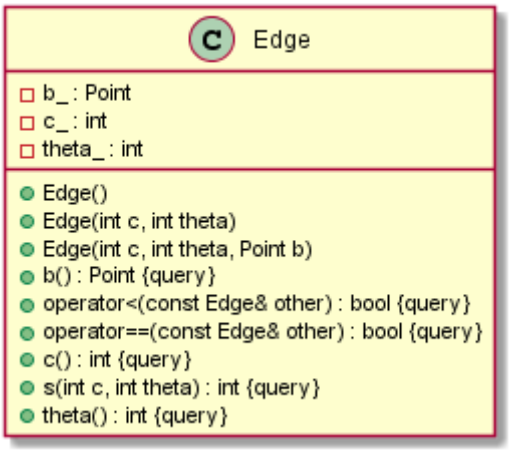


Figure 23: Edge Class

We also had to alternate two classes

1. Piece Class had to accept objects of class Edge in its constructor since now a piece consists of four Edges instead of integers.
2. Pieces Class had to create the border of our board as the piece with number 0, since in our polynomial representation the border is considered as one more piece with edges looking inwards.

Afterwards we were ready to generate our equations as described in 4.2.4.1, [17] and Figure 24.

Algorithm 3 Generate Equations

1: **for each** possible type of edge (c, θ) **do**
2: **for** $k \in \{1, \ldots, K(c, \theta)\}$ **do**
3: output file $\leftarrow \sum_i a_i^{(k,c,\theta)} (X_i Y_i)^k = 0$
4: **end for**
5: **end for**

Figure 24: Generate Equations Pseudocode

4.3.2.9 *Pieces File with Correct Rotation*

In 4.2.4.2 we explained how we can divide our problem into two easier problems. The first part is finding the correct rotation of each piece but without knowing its location. The second part is finding the location of the pieces when we already know the rotation. In evaluating the first part the only asset we needed was the pieces' file and program 4.3.2.8 Creating the System of Equations. In order to test the second part, we needed a pieces' file where a solution could be achieved merely by translating the pieces i.e. no rotations required. We could develop a program that would build such a

file but our pieces would have close to no resemblance to the real ones. Since the difficulty of our system depends on the structure of each piece, we needed a solution as close to the real one as possible. Therefore, we used the rotation of the pieces from the best-known solution which was mentioned in 2.2. The solution had only 13 errors so after "correcting" the 13 mismatched edges we had a solution for EII but with some artificial pieces. The file was downloaded from [4] and it was in the form displayed below.

```
 2/1  24/2  11/2  57/2  23/2  31/2  56/2  13/2  52/2  39/2  10/2  25/2  40/2  53/2  30/2   1/2
41/1 240/1 126/3 129/1 219/2 250/2 136/2 251/3 104/3 116/1 243/2 247/0 239/2 245/2 235/0  45/3
35/1 211/3 106/3 112/1 254/1 246/0 255/0 135/0 109/2 167/1 221/1 203/0  77/3  99/1 232/2  34/3
55/1 175/0 169/2 233/0 113/3 115/1 171/0 102/2 103/0 213/2 160/0 206/2 241/2 178/2 125/0  60/3
50/1 108/1 234/2 226/0 191/0 215/1 165/2 195/1 118/2 228/3 152/2 229/3 193/3 196/1 202/3  17/3
 8/1 142/0 180/1 210/0 190/2 154/0 205/3 121/3  63/3  83/1 225/1 253/3 248/3 173/3 122/3   6/3
51/1 105/1 238/3 200/3 201/1 144/2 182/1 170/0 130/2 218/0 237/3 183/3 166/0 179/3 131/3  15/3
22/1 111/3 101/0 120/0 256/2 222/0 220/0 161/0 124/0 209/2 134/3 110/1 168/2 242/1 249/3  20/3
12/1 204/2 128/2 133/2 216/3 207/1 208/2 139/2 127/2 230/3 117/0 119/3 114/1 188/3 194/1  59/3
43/1 174/0 198/0 186/2 223/0 184/3 123/3 107/1  76/0 192/3 132/2 187/3 177/3 172/1 181/0  58/3
18/1 176/2 185/2 231/1 227/2  78/0  87/3  95/1  71/2 212/1 214/1 189/3 197/1 224/1  68/3  49/3
54/1 236/0 217/0 244/0 252/3  93/2  86/3  62/0 155/3 162/1 199/1 137/3 164/1 156/0 146/2  38/3
44/1  81/0 148/1  70/3  84/1 150/1 140/3  61/2  82/1 158/1  79/0 138/2 143/0 149/2 147/0  36/3
28/1  67/2  66/0 145/0  97/3  75/1 159/3 141/1 153/2  80/0  88/2  89/0 151/2  92/3  65/1  29/3
27/1  73/1  74/2 163/2 100/0  96/3  90/1  94/3  72/1  85/2  91/2  64/2 157/1  69/3  98/1  32/3
 4/0  21/0  46/0  42/0   5/0  14/0  48/0  47/0  37/0   7/0   9/0  26/0  19/0  16/0  33/0   3/3
```

Figure 25: Best Solution Configuration

Each piece is described by two numbers, the number before the slash is its piece number and the digit after the slash is the amount of clockwise rotations. In addition, the top left corner of the text file, with piece `2/1`, corresponds to the top left corner of the puzzle. Therefore, our program was developed in order to get information from the above file and finally rotate the pieces inside our pieces file so their initial rotation would be the correct one. For example, in our initial file, piece number 2 was inserted as `1,5,0,0` (top edge = 1, right edge = 5, etc.), but since our solution states `2/1`, the piece will be inserted in our new file as `0,1,5,0`.

4.3.2.10 *Identical Corners in the Same Piece*

This program was created in order to find if any of the pieces had the same two edges in more than one corners (see Figure 21). In was required for our backtracking algorithm (4.3.2.7) because we had to know if it was safe to remove a piece we had already placed without checking its other rotations. The flow of the program is really simple, we just iterate over all the pieces and compare every pairing of their corners. If two corners are equal, then we report it to the user. The value of this program is twofold. Firstly, there is immediate practical usage of the output in the backtracking algorithm. Secondly, we are further analysing the problem we were given, EII, and try to define some of its attributes. As is the case with any problem, an initial

decomposition of its elements is essential and may lead us towards paths that would otherwise remain unexplored.

4.3.2.11 *Averaging Log Files*

As our AI algorithms keep on running, they produce vast amounts of files to inform us of their configuration, progress, and final results (4.3.2.4 Logger Class). In addition, each algorithm is run more than once in order to reduce the element of luck which is inherent into the genetic and simulated annealing algorithms. Hence, it is imperative that we create a program to average the scores of equal configurations and also narrow down the organizational chaos that is created. After completing our simulations, we dump every output in a folder and the program starts reading the configuration files. Whenever it encounters a configuration that has not been averaged, it starts looking for runs with identical settings. Afterwards, it prints in a new file the average score for every second of the specific configuration. This is another subtle utility of the program. In each run, we have statistics for 0.65s or 5.23s and the numbers are rarely equal. Our program normalizes the timing of the statistics by requesting for every $t = 1,2,3$ ... the closest measurement. For example, if a run has score 140 for $t = 0.8$ and 145 for $t = 1.3$ then the score that will be inserted in the calculation of the average for $t = 1$ is 140. After the scores are averaged through this method, it is time for them to be inserted into our next and final program.

4.3.2.12 *Best Configuration*

The output of Averaging Log Files is now dumped into another folder in order to be processed. Our final program is responsible for reading the type of algorithm (Figure 12) inside each configuration and collect in a CSV file the best average score for each configuration for that algorithm. We consider four different algorithms, Simple GA, Steady-State GA, Incremental GA and Simulated Annealing. We do not want to consider all the GAs as one since they have different parameters that are being altered and we want to find their best settings. In the end, each of the 4 CSV files can be opened with EXCEL and the rows can be ordered by `best_avg` in order to find the best configuration for each one of our algorithms.

algorithm	nConverger	pConverger	pCrossover	pMutation	pReplaceme	populationS	selector	best_avg
P11EIISState	500	1	0.9	0.005	0.25	250	23GARoulet	300
P11EIISState	500	1	0.9	0.005	0.35	250	23GARoulet	296
P11EIISState	500	1	0.9	0.005	0.15	250	23GARoulet	284
P11EIISState	500	1	0.9	0.005	0.15	150	23GARoulet	277
P11EIISState	500	1	0.9	0.01	0.35	250	23GARoulet	277
P11EIISState	500	1	0.8	0.005	0.25	250	23GARoulet	276
P11EIISState	500	1	0.8	0.005	0.35	250	23GARoulet	276
P11EIISState	500	1	0.9	0.005	0.35	150	23GARoulet	275
P11EIISState	500	1	0.9	0.005	0.25	150	23GARoulet	272
P11EIISState	500	1	0.7	0.005	0.15	250	23GARoulet	267
P11EIISState	500	1	0.8	0.005	0.25	150	23GARoulet	265
P11EIISState	500	1	0.8	0.005	0.15	250	23GARoulet	265
P11EIISState	500	1	0.9	0.01	0.35	150	23GARoulet	262
P11EIISState	500	1	0.8	0.005	0.35	150	23GARoulet	258
P11EIISState	500	1	0.7	0.005	0.25	150	23GARoulet	255
P11EIISState	500	1	0.7	0.005	0.25	250	23GARoulet	255
P11EIISState	500	1	0.7	0.005	0.35	250	23GARoulet	254
P11EIISState	500	1	0.9	0.01	0.25	250	23GARoulet	254

Figure 26: Best Configurations for Steady-State GA

4.4 Description of Hardware

Because the nature of the project is solely software related, the only hardware used was my personal computer. All of the above algorithms were tested in a laptop with an Intel Core i7 2630QM @ 2.00GHz processor, 4GB of DDR3 RAM and Windows 10 OS. By the naïve method of looking at the RAM consumption inside the task manager we observed that the memory usage during all of our runs was constant. There was however a slight increase during backtracking but it remained under control even for our 4GB of RAM. Thus, additional RAM would have no effect on our results. Nonetheless, a better processor would definitely yield lower run times. Since our goal was to compare the performance of our algorithms, running the algorithms in a medium-range processor was enough since the comparison was fair.

5 Experiments and Discussion

In this section, we report the configurations we tested for each algorithm and their results. Afterwards, we try to pinpoint the reasons that led to better or worse performance and also compare the methods in order to determine which one is preferable for our problem.

5.1 Experiments and Results

Due to the fact that our genetic and simulated annealing approaches are non-deterministic algorithms, meaning that with the same input they will produce different output, we tested each configuration 5 times and averaged their results. Additionally, the deterministic methods of backtracking and system of equations were only tested once.

5.1.1 Genetic Algorithms

Our genetics algorithms are divided into three categories based on their population replacement method which was explained in 4.3.2.5.1. Each one of them has a different parameter that we can be altered and after finding their corresponding best configuration we can compare all three. The parameters that remain constant in all three GAs are the selection method of the roulette wheel (4.3.2.5.2) and the termination criterion of 500 continuous generations with equal score (4.3.2.5.6).

5.1.1.1 *Simple Genetic Algorithm*

Our simple genetic algorithm was tested with all possible combinations of the values displayed below

- Elitism: on or off
- Crossover probability: 60% to 90% with a step of 10%
- Mutation probability: 0.5% to 1.5% with a step of 0.5%
- Population size: 50 to 250 with a step of 100

The crossover and mutation probabilities were selected to be in the same range that Schaffer et al. [45] found to be optimal for every problem in their test suite. However, they also suggested the population size to range for 20 to 30 but for our problem a bigger population size led to better scores. For those reasons, the crossover, mutation

and population range was equal in all three GAs. The graphs that follow display the average score out of the 544 edge matches for each of the above parameters.

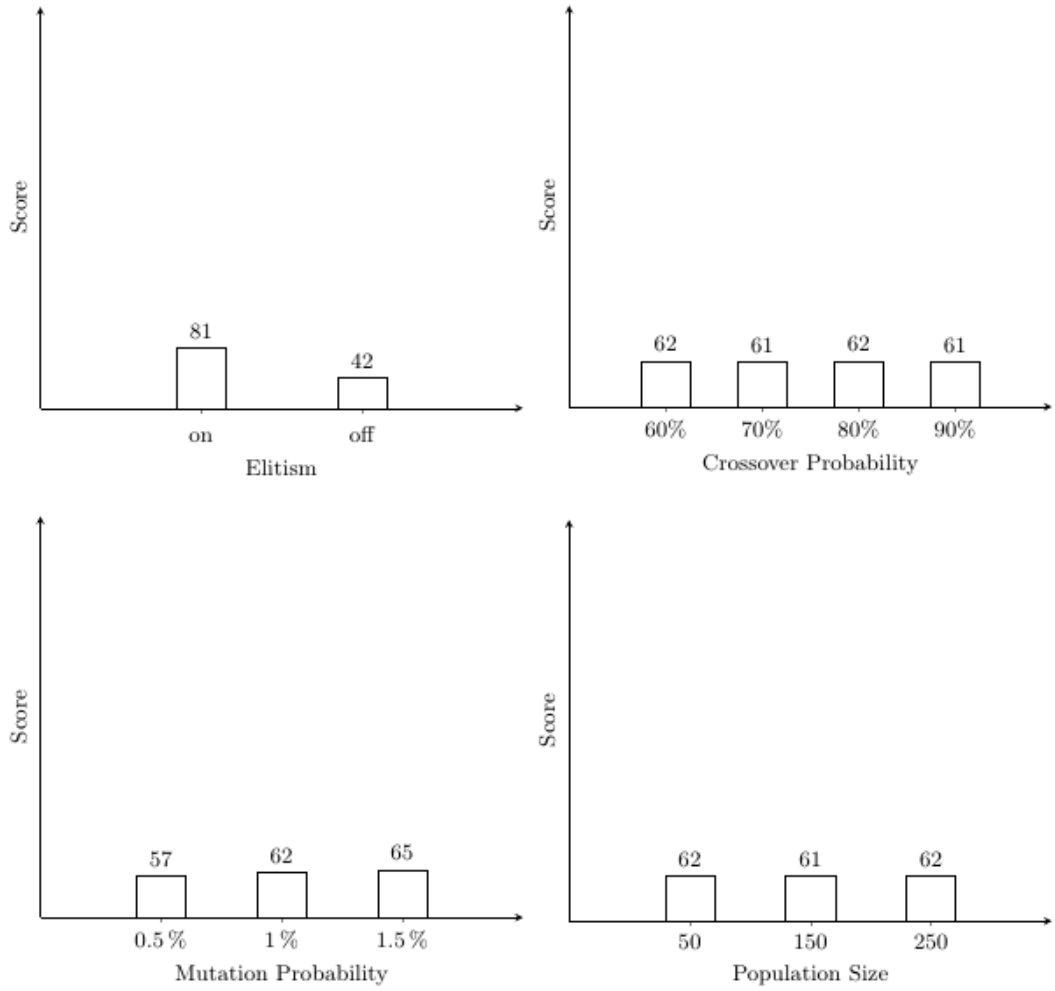


Figure 27: Simple GA - Average Scores for all Parameters

Our best average score was 102/544 edges and the configuration was

- Elitism = on
- Crossover probability = 0.7
- Mutation probability = 0.015
- Population size = 150

The time required to achieve the score was 154 seconds (Figure 28) or 2094 generations. This was the only configuration with a score over 100 and we had two more following with 97. The frequency of each score interval for all configurations is displayed in Figure 29.

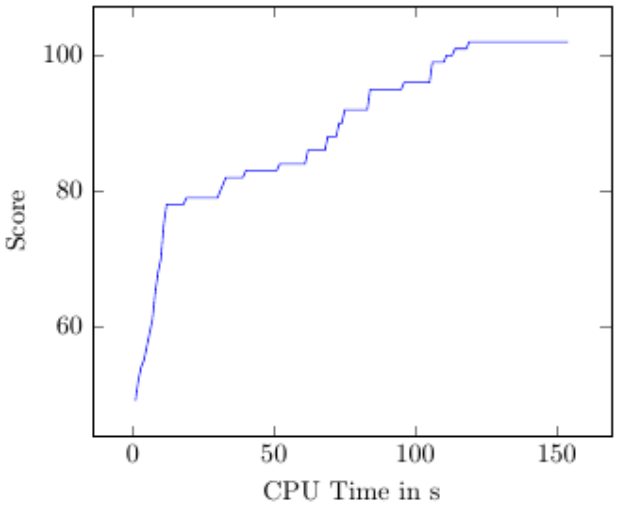


Figure 28: Simple GA - Best Result

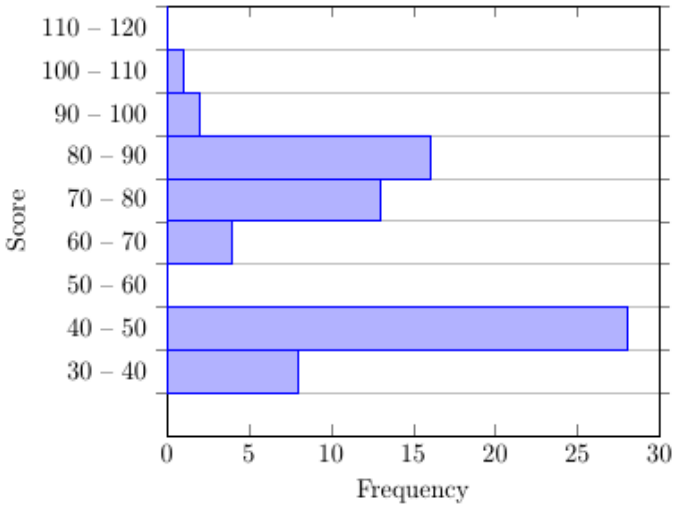


Figure 29: Simple GA - Score Frequency

5.1.1.2 *Steady-State Genetic Algorithm*

In our steady-state GA we have to define the number or percentage of the population that is going to be replaced in the next generation. We used percentage replacement since our population size varies. We experimented with the values mentioned below

- Replacement percentage: 15% to 35% with a step of 10%
- Crossover probability: 60% to 90% with a step of 10%
- Mutation probability: 0.5% to 1.5% with a step of 0.5%
- Population size: 50 to 250 with a step of 100

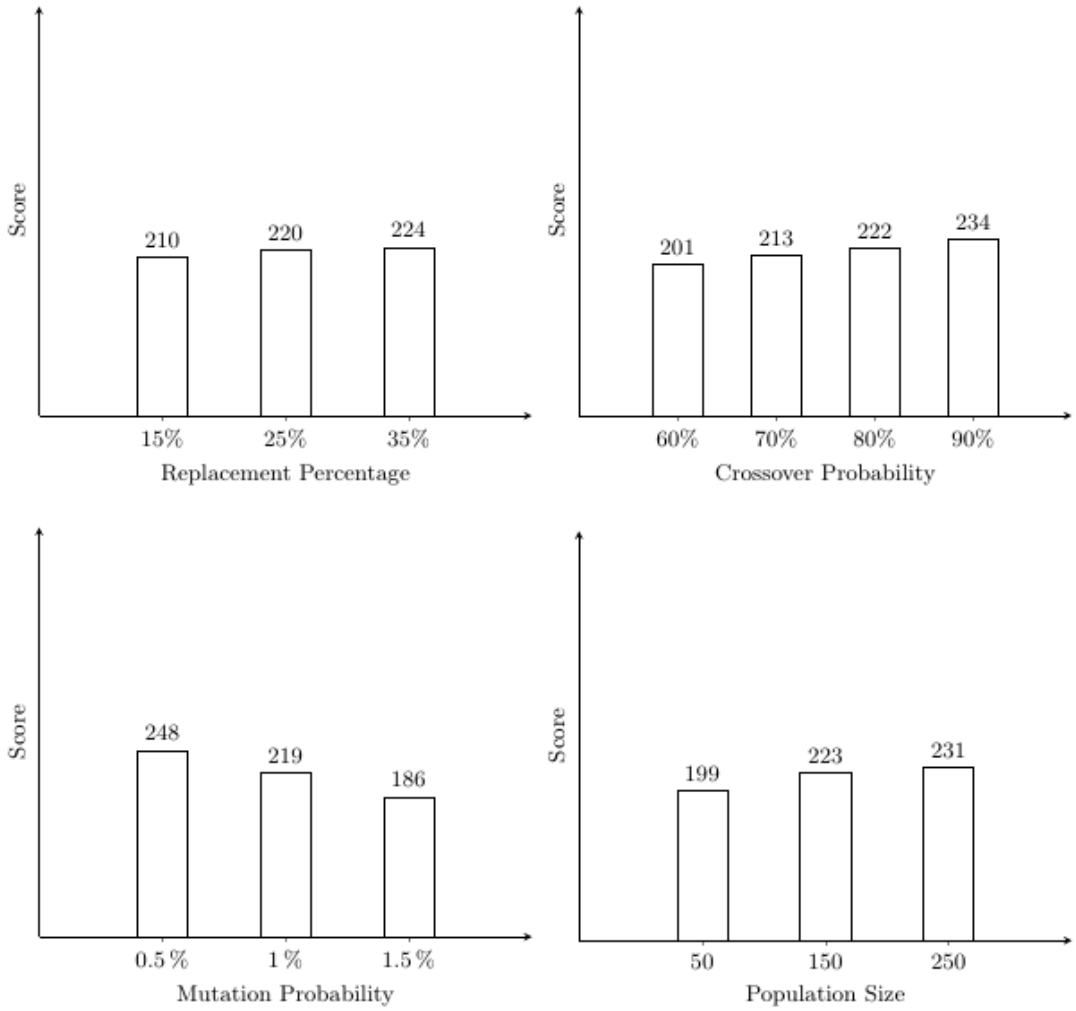


Figure 30: Steady-State GA - Average Scores for all Parameters

Our best score was 300/544 and it required 272 seconds or 8390 generations. The configuration was the following

- Replacement Percentage: 25%
- Crossover probability: 90%
- Mutation probability: 0.5%
- Population size: 250

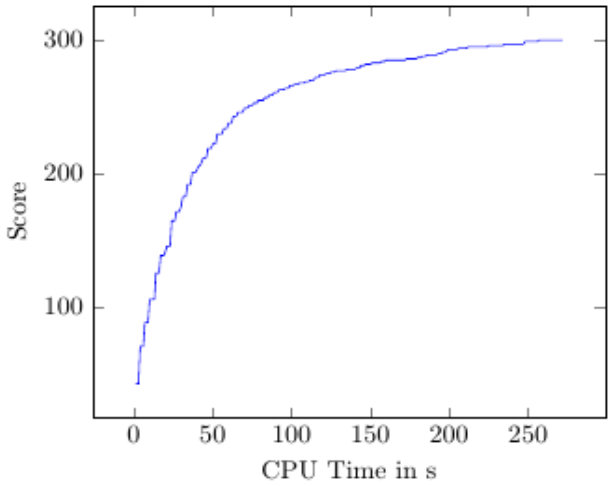


Figure 31: Steady State GA - Best Result

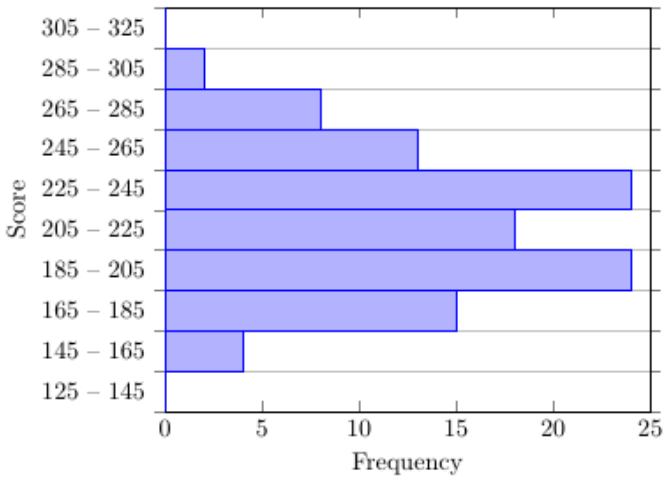


Figure 32: Steady-State GA - Score Frequency

5.1.1.3 *Incremental Genetic Algorithm*

In our last genetic algorithm, we had to define whether the next generation would have 1 or 2 new individuals. We experimented with both and the crossover, mutation and population ranges were the same as above.

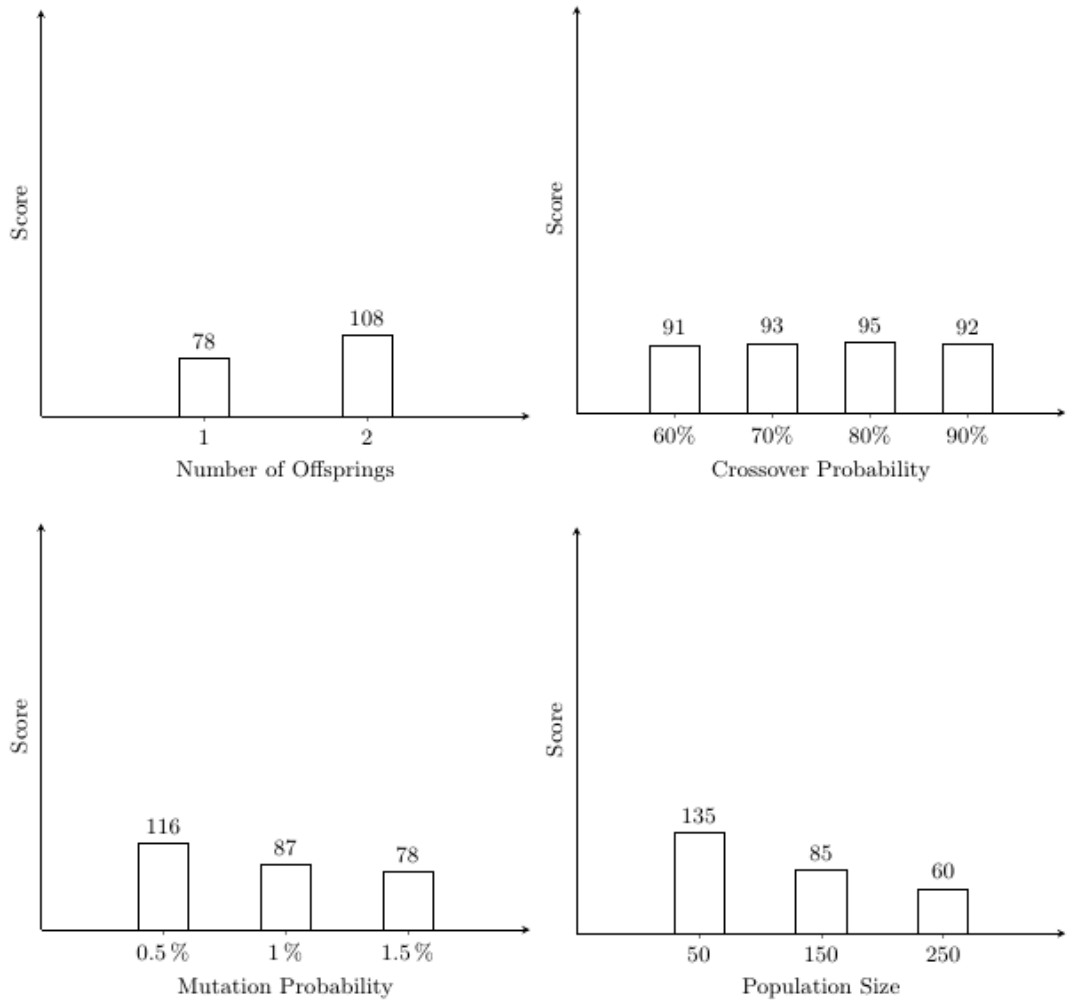


Figure 33: Incremental GA - Average Scores for all Parameters

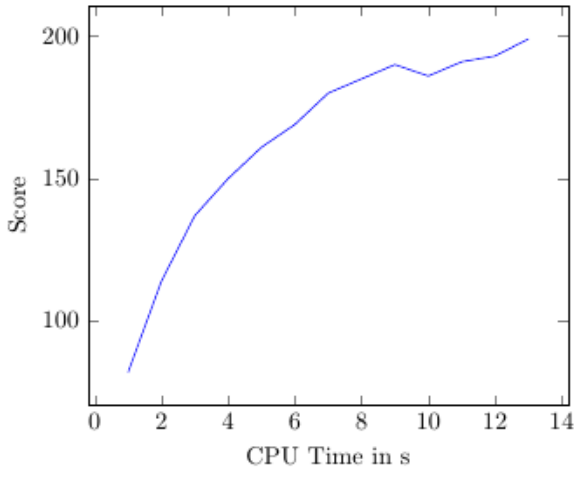


Figure 34: Incremental GA - Best Result

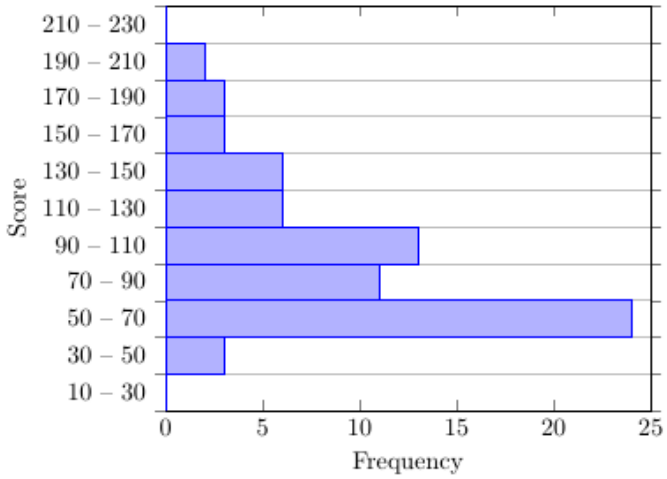


Figure 35: Incremental GA - Score Frequency

Our best score was 199/544 and it was found after just 13 seconds or 12,121 generations. The configuration was

- Number of Offspring: 2

- Crossover probability: 90%
- Mutation probability: 0.5%
- Population size: 50

5.1.2 Simulated Annealing

Simulated annealing was the only algorithm to almost reach 400 matched edges but unfortunately due to time constraints we had to stick to the bibliographical configuration [11]. Our initial temperature was 1 and every time we wanted to reduce it we did so by 10%. Our cooling process iterated for 500 times and in each loop, we checked 1000 neighbour solutions. The above configuration was run 5 times and the average score and temperature in relation to time is displayed below. Our cooling fraction of 0.9 was able to reduce our temperature from 1 to $4*10^{-6}$ in just 55 seconds. The best run was able to match 394 edges in 125 seconds.

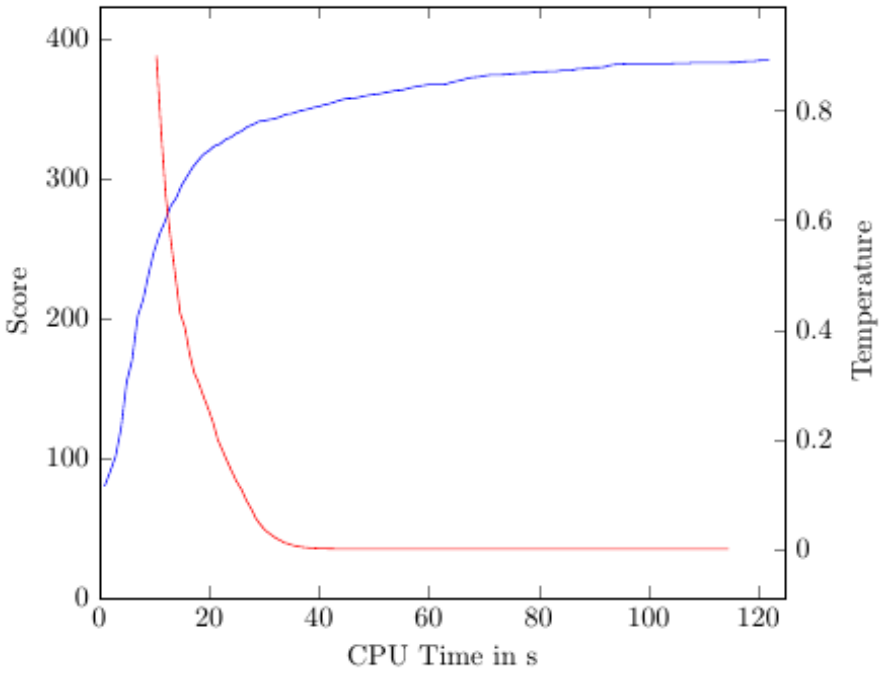


Figure 36: Average Score & Temperature for 5 SA runs

5.1.3 Backtracking

Our backtracking algorithm has no termination criteria and for that reason we had to manually shut down the program whenever we decided. Our longest running algorithm until now required 272 seconds so anything above that number would be sufficient. However, we let it run for 90 minutes in order to identify its behaviour. During that time, the algorithm performed 1,269,539 steps and the maximum score

was 389/544 edges, achieved at the 48-minute mark. The algorithm demonstrates a fast start as it is able to match 350 edges in 15 seconds but it quickly runs out of steam. After the first 15 seconds, the algorithm bounces between 270 and 370 matched edges.

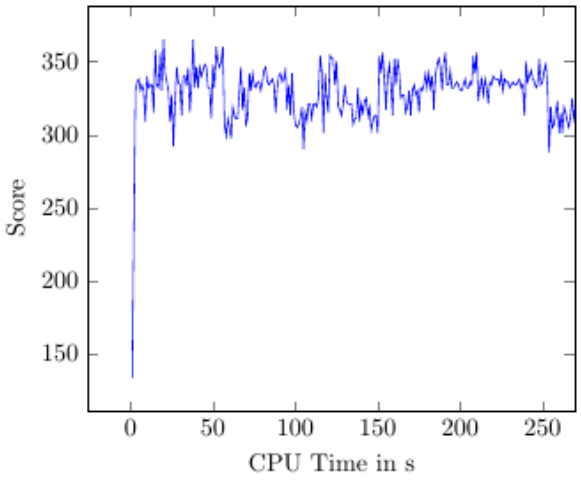


Figure 37: Backtracking up to 272s

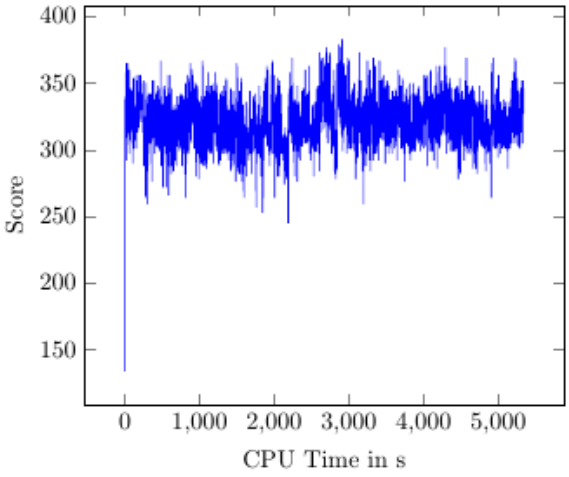


Figure 38: Backtracking for 90 minutes

5.1.4 System of Equations

Our program described in 4.3.2.8 receives a text file of our pieces as input and outputs the equations required by Macaulay 2. On average, this procedure takes 1.8 seconds, hence theoretically, it would require $4^{256}*1.8$ seconds, or $2.4*10^{153}$, in order to produce every possible combination, a technically impossible task. In addition, it takes about 35 seconds for Macaulay 2 to load the file but only $3*10^{-4}$ seconds, or 0.03ms, in order to prove that this rotation does not lead to a solution. In case we have produced the correct rotation for all the pieces, the process is killed. Unfortunately, Macaulay also terminates when we try to compute the Gröbner basis which would be our main tool in finding the solution.

5.2 Discussion of Results

In this section, we discuss the results for each algorithm separately and then we compare their performance in order to determine which one best fits our problem.

5.2.1 Simple Genetic Algorithm

From the graphs in Figure 27 we can conclude that the only dependent variable for our simple GA's performance is elitism. Except elitism, the rest of the configuration in our best score appears to be just the "luckiest" run with elitism on. By keeping our

best individual in each generation, we get, on average, a double score. By this fact, but also by the increased performance of steady-state and incremental, we can derive that overlapping populations have a better fitness for our puzzle. However, even with elitism on, the simple genetic algorithm had the worst performance of all, since our genetic operations are probably not robust enough to improve our score when almost all of our population is to be replaced. In addition, it is really slow since for every next generation we have to produce new offspring in the size of the previous population and also evaluate them. The algorithm required 154 seconds in order to create 2094 generations so, on average, we are producing 13.6 gens/sec.

5.2.2 Steady-State Genetic Algorithm

The best score we achieved with a genetic algorithm was with the steady-state replacement method. From Figure 30, we conclude that the algorithm favors higher replacement percentage, crossover probability and population size but lower mutation rate. Unfortunately, we did not keep on increasing/decreasing the above parameters in order to find their optimal values. The selection and mating are more efficient in the steady-state since the new offspring may not be inserted into the next generation if they are not fit enough. Hence, we protect our generations from bad offspring and keep their average fitness high. One of the reasons this algorithm performed so well is also the time saved because we are not generating and replacing all the individuals which leads to 30.8 gens/sec.

5.2.3 Incremental Genetic Algorithm

The incremental genetic algorithm was the GA that converged the fastest and achieved a mediocre score of 199 edges. Since in each generation we are replacing only one or two individuals we are evolving really fast, 932.4 gens/sec, but the chances of producing a better individual are lowered because we are introducing fewer individuals. As a result, the best score was repeated for 500 generations in just 13 seconds. Since we are replacing the worst individuals and the parents usually have high fitness, the offspring are inserted into the next generation alongside their parents and the genetic material of our population converges. The algorithm seems to favor smaller population sizes since if it is going to create only one or two offspring then parents with higher fitness are more likely to be chosen in a smaller population size.

5.2.4 Simulated Annealing

Even with the default configuration, simulated annealing was able to match, on average, 380/544 edges in just 130 seconds. Its advantage lies on the fact that in each iteration we are performing very few operations, so we are guaranteeing speed, but in the same time we are rarely choosing worse moves. In essence, the algorithm spends most of its time exploring good elements of the solution space and it evades getting repeatedly stuck in local minima [11].

5.2.5 Backtracking

Although backtracking had a great start with 350 edges in 15 seconds there was no major improvement after that point. The algorithm keeps on trying to fit more pieces but none of them matches so it steps back and forth to no avail. Its deterministic nature guarantees a solution if it manages to explore the entirety of the search space but the size of the puzzle renders this practically impossible. At least in our machine and with our simple implementation of the algorithm, trying is futile. Backtracking performs 237 moves/sec and our search space has a size of $256! * 4^{256} = 1.15 * 10^{661}$. At this rate, it would take us $1.53*10^{651}$ years to exhaust our search space. In comparison, our universe will be destroyed in roughly $6*10^{9}$ years.

5.2.6 System of Equations

Our mathematical approach shares the same fate as backtracking. Both of them are theoretically able to provide a solution but they fall short in practice due to the complexity of our problem. The main bottleneck of this method is that in order to import all 550 to 650 equations into Macaulay we need about 35 seconds, the number of equations depends on the orientation of each piece. Even if we did manage to overcome this and find the correct rotation for each piece, our search space would still be $256! = 8.57*10^{506}$. Even after the reduction, Macaulay is unable to compute the Gröbner basis and backtracking would require $1.148*10^{497}$ years.

5.2.7 Comparison of Methods

After defining the best flavor of GA, we are going to compare the three main AI algorithms we developed. Due to the inability of our mathematical approach to produce results, we are not capable to include it into our score-time comparison but we are still going to mention its potential.

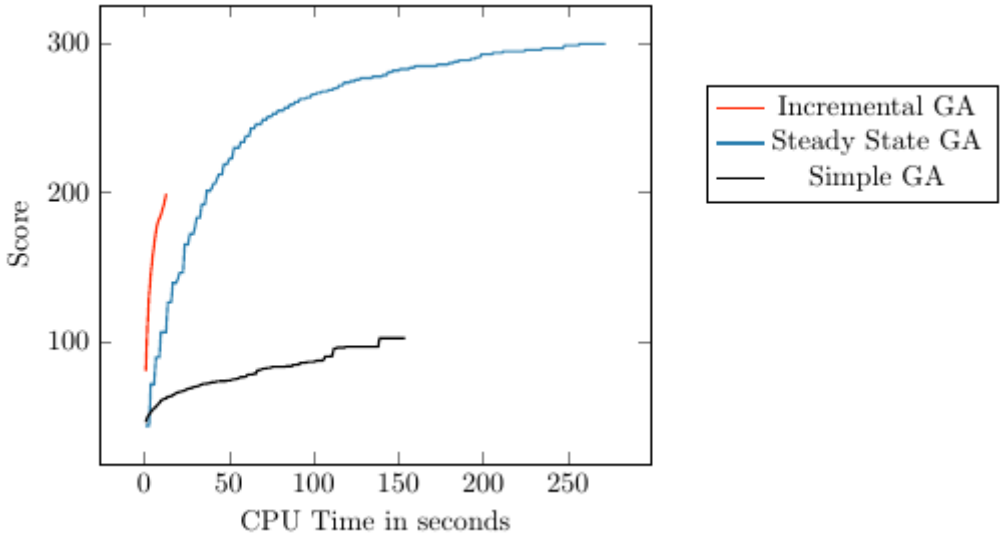


Figure 39: Average Score of Best Configurations for each GA Flavor

The steady-state replacement method was by far the most successful algorithm among our GAs. It was 1.94 times better than the simple one and 0.5 times better than the incremental. The advantage of steady state and incremental lies on the fact that they are inserting new individuals one by one so our populations are not polluted with worse individuals than what we had on the previous generation. It can be argued that by not accepting worse individuals we are losing useful genetic material that inhabited inside those unfit solutions and this may well be the case. Maybe accepting worse individuals in early generations and with lower probability in later generations, following the logic behind simulated annealing, would lead to better results. Still, the fact remains that being selective in what offspring we accept yields higher scores.

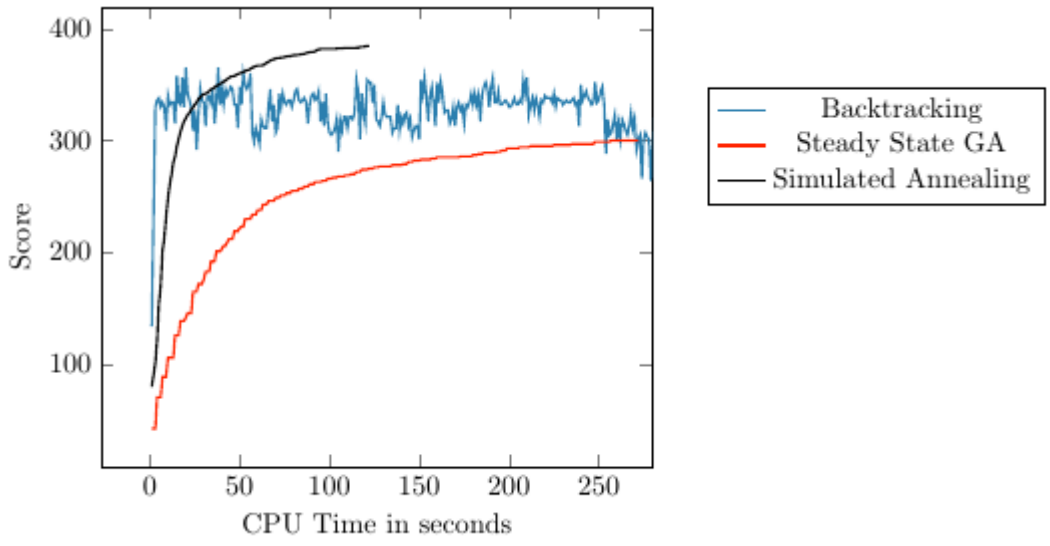


Figure 40: Average Score for each AI Algorithm

Although backtracking is the first one to break the barrier of 300 edges, it is completely overshadowed by simulated annealing after 50 seconds.

The genetic algorithm has the worst performance of all and it is probably because it is the only algorithm that spends so much time not actually improving our score. We have hundreds of individuals in thousands of generations evolving but the average contribution of each one of them in the overall fitness is minimal. We are performing countless crossovers and mutations which are eventually wasted since they do not improve our result. Maybe a more carefully selected mutation and crossover method, with higher chance of producing better offspring, could lead to higher performance. On the other hand, simulated annealing and backtracking are proceeding with baby steps, but most of the time in the correct direction, and they are able to perform better.

The superiority of simulated annealing over backtracking was also demonstrated by Toulis [21]. In his paper, the backtracking algorithm matches 384 edges in 88 seconds and his simulated annealing implementation 400 edges in 54 seconds. The performance of our backtracking was almost equal and the difference in simulated annealing can be attributed to the optimizations he performed in his puzzle's representation.

5.3 Conclusions

After trying out three different replacement methods for our GA, we concluded that the best option was replacing a medium percentage, 25%-35%, of our population. Our replacement size was small enough so that our new generation would not take over the old one and large enough in order to make progress and avoid convergence.

The winner between all of our AI algorithms was simulated annealing with backtracking coming second and the GA being close in the third place. It is worth nothing that simulated annealing was run only with the default parameters proposed by Skiena in his implementation [11]. Therefore, additional experiments with different combinations between the cooling fraction, cooling steps and steps per temperature would be of interest.

Unfortunately, our mathematical approach was brought to its knees for the same reason backtracking was unable to perform better. The search space is enormous for any deterministic algorithm to produce a satisfying result.

As we conclude with the experiments and results of our thesis, we demonstrate the best outcome our algorithms were able to produce. Figure 41 shows the final configuration of the simulated annealing run that matched 394/544 edges.

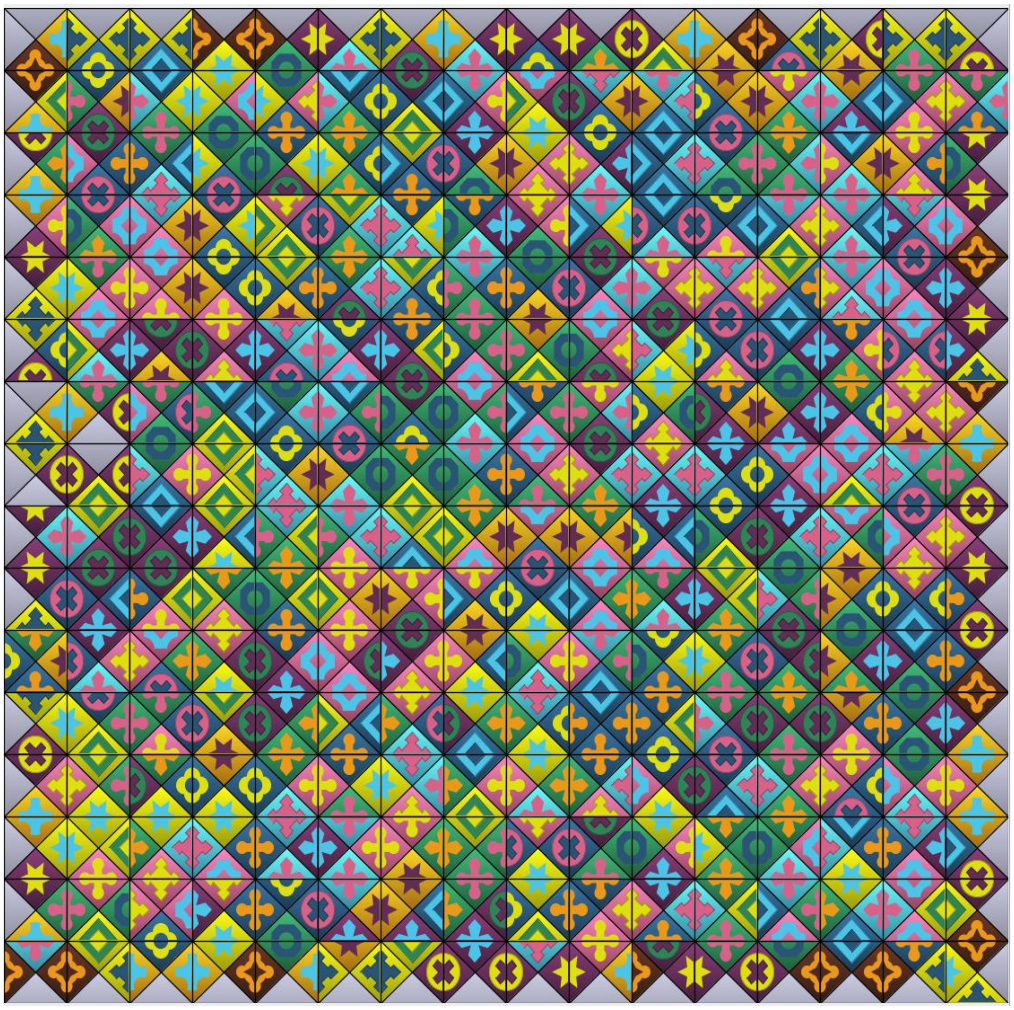


Figure 41: Best Solution Found by Simulated Annealing

6 A look into the future

In this final chapter, we examine what the future holds in relation to technologies relevant to our project, what enhancements can be made to it and what applications our algorithms may have in the future.

6.1 Advances in the technologies relevant to the project

AI is a field of continuous and rapid development. In my opinion, AI and human genetic engineering, will be the two fields with the greatest impact on our civilization. Human genetic engineering will transform the human race itself and AI will shape the world around us. AI is already implemented in the majority of apparatus that we use in our everyday life. Smartphones, cars, watches, televisions and almost every new device with a computer chip in it comes equipped with some form of AI in order to enhance their functionality. Probably the most imminent and substantial change AI will bring is the self-driving cars. With Google and Tesla being the frontrunners on the field we are slowly but surely being driven, for the time metaphorically, to a society in which human drivers will be considered a relic of the past. The advantages of self-driving cars will be numerous. First of all, self-driving cars do not get drunk or sleepy and since the majority of car accidents is due to human error, the number of crashes will be reduced and human lives will be spared. This will have as a negative side-effect our declining need of car mechanics and also lesser demand for replacement parts but their production is already mostly automated. In addition, we can consider braking and therefore accelerating as an unnecessary overhead to driving. Human drivers communicate with each other indirectly with traffic lights and some common priority rules but with AI behind the wheel it doesn't have to be this way. Driverless cars will be able to communicate with each other through a network and as a result traffic lights may become obsolete. Traffic at junctions will be handled through AI algorithms so there will be no need for brakes leading to lower fuel consumption, less traffic jams and therefore less time required for our transportation and less stress as a side-effect. The most prominent disadvantage of self-driving cars will be the enormous loss of job positions. It will be a world where the occupations of taxi and track drivers will have no place and we should be able to accommodate for that before it is too late. But the disposable human jobs won't stop there, AI systems will be able to write articles, provide technical support or even diagnose diseases. The human mind is the most complex system that we know of and we might think that

there is no possible way a computer will be able to perform our job but if there is one thing mankind gets right, is being wrong. Our civilization should have a mature discussion on the subject and come up with viable solutions for our economy or we will be overwhelmed by high rates of unemployment and individuals being uncertain of their role in our society.

6.2 Future improvements (on this project)

The most important improvement to be implemented on this project would be cloud computing. If one computer is capable of completing the puzzle with only 13 violations, imagine the results that millions of processors working together for the same cause could yield. With the infrastructure provided nowadays by technology giants like Google and Amazon, there is no need to rely on individuals' support and computers. Those companies can provide their processing power at low cost with little to no configuration requirements. The sole addition that would be required, is the usage of their APIs and the implementation of the parallel versions of our programs.

Another enhancement to our results would be to further explore our algorithms. For example, in our GAs, our library gives us the option to define what it calls allele sets. That means that we can restrict the permissible pieces for each location on the board. Therefore, we can restrict the pieces that can be placed in the border to those that have a grey edge and also implement our center piece restriction. In addition, we mentioned that the incremental GA had the problem of converging too fast, we also stated that the new individuals take the place of the worst individuals. As a result, new individuals populate the same generation as their parents and the divergence of the population is low. Therefore, if we used the other insertion option provided by the library and replace the parents, the population would have higher differentiation and converge at a later stage. In addition, if we consider the incremental GA as a steady-state GA with replacement of 1 or 2 offspring, we could employ the methods used in [46] in order to preserve useful diversity. In regards to our algorithm with the best results, simulated annealing, time constrains prohibited us from trying out further configurations so in a follow up research this should be the first course of action. Furthermore, although backtracking is probably futile if we want to find a solution, there are still ways to improve its performance. In our implementation, whenever we wanted to find a suitable piece for a location, we tried out every possible available piece. An alternative would be to pre-hash the pieces based on the combination of the

patterns on each edge an achieve an O(1) insertion time using a hash table [47]. This methodology was also used by Bourreau and Benoist [48] but they limited the size of their board, they experimented from sizes 7x7 to 10x10, so our findings could not be compared. A related paper on generating instances of EII in any size, for testing or comparison purposes, has been published and we could follow it if we want to experiment in different grid sizes [49]. Our last suggestion on improving backtracking is vague but really promising since it comes from Donald Knuth and his infamous series of "The Art of Computer Programming". In his Volume 4B, Fascicle 5B, titled "Introduction to Backtracking", which has yet to be released, he states various new methods of creating state of the art backtracking algorithms [50]. Although I tried to read it, I gave up after some point, since I could not follow. The fascicle is available online for alpha-testing at [51]. Finally, in regards to our mathematical approach, we could take a closer look in the structure of our equations in order to simplify them or try a different, more appropriate, technique.

The third improvement would be further optimization of the code. C++ is a language that with better code and compiler options could always perform faster but mastering its ways is a long and treacherous endeavor. Because the majority of the algorithms are based on loops, one small speed improvement in an expression of that loop will have exponential benefits for the code since its benefits will be harvested numerous times. In addition, the latest stages of our code were hastily developed so their quality is inferior to the first parts. Therefore, they should be refactored and their appearance should be in par with the rest of our source code. To paraphrase Stroustrup, the creator of C++, reading code should make you smile the way a well-crafted music box or well-designed car would [52].

Another interesting improvement would be to try out the algorithms with the center piece restriction and compare the performance implications of one extra constraint. In my opinion, the implementation would be straight-forward by just removing the center piece from the available pieces from the beginning of the run, adding it to the board, and restricting it from being removed.

6.3 Insights into future applications

Another relevant technology to my thesis, due to its performance capabilities, is the new types of computers that large technological, scientific and financial companies

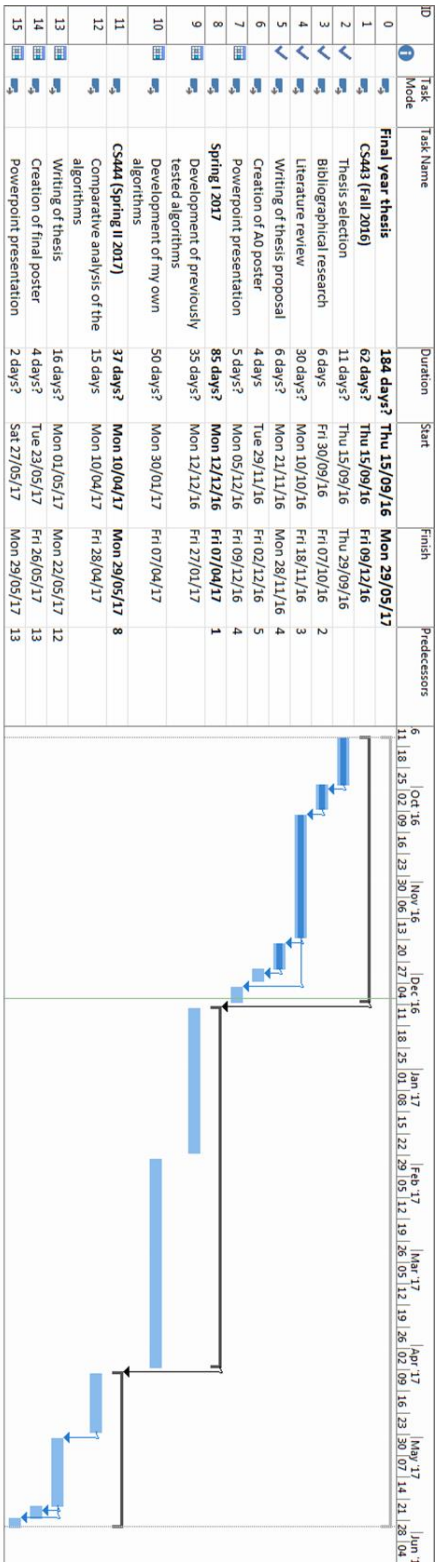
are buying for around 15 to 20 million dollars called quantum computers. Quantum computers are using concepts of physics belonging to the realm of quantum physics like superposition, the ability of particles to be in two states at the same time, both zero and one, allowing the simultaneous process of a much larger set of data. Other concepts include quantum entanglement, the ability of one particle to affect another particle's state even if they are apart. Multi-verses, which is the concept of universes similar to ours but with alternate courses of history, and quantum-tunneling, the slippage between those universes. Although, the technology is in a primal state right now, there is a great analogy to be made with horses and airplanes. A fast horse could run faster than the first plane invented by the Wright brothers but as technology progressed airplanes could perform tasks that were impossible for any horse since they were using a different domain to execute them, air versus land. In a similar fashion, in the future, quantum computers will be in a position to tackle problems conventional computers are unable to, since they are using the different domain of quantum physics. The above machines in combination with AI will be able to introduce humanity to a new era of computational power and intelligence which will not only help as solve problems that were previously deemed impossible, but also help us explore the cosmos. Due to the young age of this technology, any specific predictions would be hasted but it certainly is a giant step towards achieving general AI, or in simpler terms, human like intelligence.

Appendices

Initially Proposed Design Requirements

The only initially proposed design requirement we were unable to implement was measuring the RAM usage of our algorithms since when we searched for a way to do it there was no out-of-the box solution. As a result, we left it for later and eventually we run out of time.

Initially Proposed Timeline



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
0		**Final year thesis**	**184 days?**	**Thu 15/09/16**	**Mon 29/05/17**	
1		**CS443 (Fall 2016)**	**62 days?**	**Thu 15/09/16**	**Fri 09/12/16**	
2		Thesis selection	11 days?	Thu 15/09/16	Thu 29/09/16	
3		Bibliographical research	6 days	Fri 30/09/16	Fri 07/10/16	2
4		Literature review	30 days?	Mon 10/10/16	Fri 18/11/16	3
5		Writing of thesis proposal	6 days?	Mon 21/11/16	Mon 28/11/16	4
6		Creation of A0 poster	4 days	Tue 29/11/16	Fri 02/12/16	5
7		Powerpoint presentation	5 days?	Mon 05/12/16	Fri 09/12/16	4
8		**Spring I 2017**	**85 days?**	**Mon 12/12/16**	**Fri 07/04/17**	1
9		Development of previously tested algorithms	35 days?	Mon 12/12/16	Fri 27/01/17	
10		Development of my own algorithms	50 days?	Mon 30/01/17	Fri 07/04/17	
11		**CS444 (Spring II 2017)**	**37 days?**	**Mon 10/04/17**	**Mon 29/05/17**	8
12		Comparative analysis of the algorithms	15 days	Mon 10/04/17	Fri 28/04/17	
13		Writing of thesis	16 days?	Mon 01/05/17	Mon 22/05/17	12
14		Creation of final poster	4 days?	Tue 23/05/17	Fri 26/05/17	13
15		Powerpoint presentation	2 days?	Sat 27/05/17	Mon 29/05/17	13

Figure 42: Initially Proposed Timeline

Flowchart of planned code

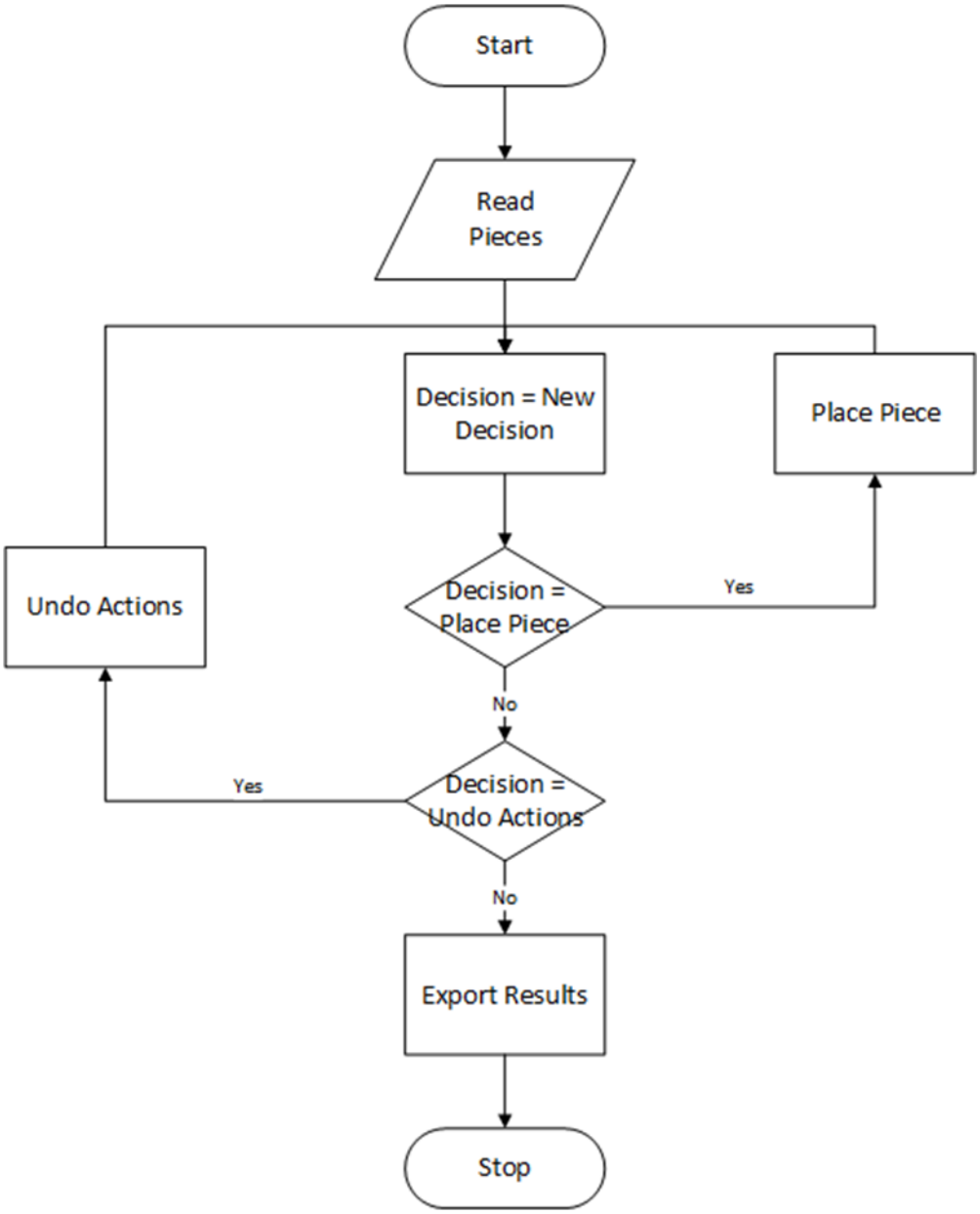


Figure 43: Flowchart of Plant Code

Parts list

Personal Computer: HP Pavilion dv6-6115ev

Data sheets for software components

- C++ Programming Language: http://devdocs.io/cpp/
- CLion IDE: https://www.jetbrains.com/clion/documentation/
- Macaulay 2: http://www.math.uiuc.edu/Macaulay2/doc/Macaulay2-1.10/share/doc/Macaulay2/Macaulay2Doc/html/

Table of figures

Table of equations

Table of tables

Table of diagrams

Bibliography

[1] E. D. Demaine and M. L. Demaine, “Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity,” *Graphs Comb.*, vol. 23, no. 1, pp. 195–208, 2007.

[2] A. Selby, “Description of method.” [Online]. Available: http://www.archduke.org/eternity/method/desc.html. [Accessed: 27-May-2017].

[3] L. Verhaard, “EII Solver.” [Online]. Available: http://www.shortestpath.se/eii/index.html. [Accessed: 08-Jun-2017].

[4] L. Verhaard, “Best Results.” [Online]. Available: http://www.shortestpath.se/eii/results.html. [Accessed: 08-Jun-2017].

[5] Clay Mathematics Institute, “P vs NP problem | Clay Mathematics Institute.” [Online]. Available: http://claymath.org/millennium-problems/p-vs-np-problem. [Accessed: 03-Mar-2017].

[6] A. Koenig and B. E. Moo, *Accelerated C++: Practical Programming by Example*, 1 edition. Boston, MA: Addison-Wesley Professional, 2000.

[7] B. Stroustrup, *The C++ Programming Language, 4th Edition*, 4 edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2013.

[8] S. Meyers, *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*, 1 edition. Boston: Addison-Wesley Professional, 2001.

[9] S. Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*, 3 edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2005.

[10] S. Meyers, *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*, 1 edition. Beijing ; Sebastopol, CA: O’Reilly Media, 2014.

[11] S. S. Skiena, *The Algorithm Design Manual*, 2nd edition. London: Springer, 2008.

[12] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. 2016.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[14] M. Mitchell, *An Introduction to Genetic Algorithms*, Reprint edition. Cambridge (Ma) ; London: MIT Press, 1998.

[15] M. Wall, “GAlib: A C++ Library of Genetic Algorithm Components.” Aug-1996.

[16] W. Bein, “The Use of GAlib.” 2002.

[17] S. Kovalsky, D. Glasner, and R. Basri, “A Global Approach for Solving Edge-Matching Puzzles,” *SIAM J. Imaging Sci.*, vol. 8, no. 2, pp. 916–938, Jan. 2015.

[18] J. Muñoz, G. Gutierrez, and A. Sanchis, “Evolutionary Genetic Algorithms in a Constraint Satisfaction Problem: Puzzle Eternity II,” in *Bio-Inspired Systems: Computational and Ambient Intelligence*, 2009, pp. 720–727.

[19] P. O. Niang, “Solving the Eternity II Puzzle using Evolutionary Computing Techniques,” masters, Concordia University, 2010.

[20] W.-S. Wang and T.-C. Chiang, “Solving Eternity-II puzzles with a tabu search algorithm,” presented at the Proceedings of the 3rd International Conference on Metaheuristics and Nature Inspired Computing, META, 2010, vol. 10.

[21] P. Toulis, “The Eternity Puzzle,” 2009.

[22] V. Kašík, “Acceleration of Backtracking Algorithm with FPGA,” presented at the Applied Electronics (AE), 2010 International Conference on, 2010, pp. 1–4.

[23] P. Malakonakis and A. Dollas, “Exploitation of parallel search space evaluation with fpgas in combinatorial problems: the eternity II case,” presented at the 2011 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 264–268.

[24] M. J. H. Heule, “Solving edge-matching problems with satisfiability solvers,” *SAT*, pp. 69–82, 2009.
[25] P. Schaus, Y. Deville, and others, “Hybridization of CP and VLNS for Eternity II,” *Journ. Francoph. Program. Par Contraintes JFPC’08*, 2008.
[26] A. J. F. M. A. Ramos and D. F. P. Dias, “Constraint Logic Programming, Solution and Optimization: Eternity II.”
[27] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua, “Localsolver 1. x: a black-box local-search solver for 0-1 programming,” *4OR*, vol. 9, no. 3, pp. 299–316, 2011.
[28] L. Verhaard, “Details of Eternity II Solver eii.”
[29] T. Wauters, W. Vancroonenburg, and G. V. Berghe, “A guide-and-observe hyper-heuristic approach to the Eternity II puzzle,” *J. Math. Model. Algorithms*, vol. 11, no. 3, pp. 217–233, 2012.
[30] L. Jourdan El-Ghazali, “Editorial,” *J. Math. Model. Algorithms*, 2012.
[31] M. Fourment, “Language benchmark - Results.” [Online]. Available: http://www.bioinformatics.org/benchmark/results.html. [Accessed: 27-May-2017].
[32] I. Gouy, “The Computer Language Benchmarks Game.”
[33] K. A. D. Jong, *Evolutionary Computation: A Unified Approach*, Reprint edition. A Bradford Book, 2016.
[34] J. B. Fraleigh, *A First Course in Abstract Algebra, 7th Edition*, 7 edition. Boston: Pearson, 2002.
[35] D. A. Cox, J. Little, and D. O’Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 3rd edition. New York: Springer, 2008.
[36] *e2pieces.txt in Stuff/math-progs/E2Solver/trunk/samples – MyStuff*. [Online]. Available: http://stuff.nativeboinc.org/wiki/browser/Stuff/math-progs/E2Solver/trunk/samples/e2pieces.txt?order=size&desc=1. [Accessed: 04-Jun-2017].
[37] *Discussion of the “Eternity II” puzzle. - Yahoo Groups*. [Online]. Available: https://groups.yahoo.com/neo/groups/eternity_two/files/e2editor.exe/. [Accessed: 04-Jun-2017].
[38] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 edition. Reading, Mass: Addison-Wesley Professional, 1994.
[39] “The GNU C Library: CPU Time.” [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html. [Accessed: 05-Jun-2017].
[40] “Eternity II Editor,” *SourceForge*. [Online]. Available: https://sourceforge.net/projects/eternityii/. [Accessed: 05-Jun-2017].
[41] D. Whitley, “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best,” in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 116–121.
[42] G. J. E. Rawlins, Ed., *Foundations of Genetic Algorithms 1991*, 1 edition. San Mateo, Calif.: Morgan Kaufmann, 1991.
[43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[44] E. Aarts and J. K. Lenstra, Eds., *Local Search in Combinatorial Optimization*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1997.
[45] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, “A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function

Optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, San Francisco, CA, USA, 1989, pp. 51–60.
[46] M. Lozano, F. Herrera, and J. R. Cano, "Replacement strategies to preserve useful diversity in steady-state genetic algorithms," *Inf. Sci.*, vol. 178, no. 23, pp. 4421–4433, Dec. 2008.
[47] R. Sedgewick and K. Wayne, *Algorithms*, 4th edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2011.
[48] E. Bourreau and T. Benoist, "Fast Global Filtering for Eternity II," *Constraint Program. Lett. CPL*, vol. 3, pp. 036–049, 2008.
[49] C. Ansótegui, R. Béjar, C. Fernández, and C. Mateu, "How Hard is a Commercial Puzzle: the Eternity II Challenge.," presented at the CCIA, 2008, pp. 99–108.
[50] D. E. Knuth, *The Art of Computer Programming, Volume 4B, Fascicle 5: The: Mathematical Preliminaries Redux; Backtracking; Dancing Links*, 1 edition. Place of publication not identified: Addison-Wesley Professional, 2017.
[51] D. E. Knuth, "The Art of Computer Programming." [Online]. Available: http://www-cs-faculty.stanford.edu/~uno/taocp.html. [Accessed: 27-May-2017].
[52] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1 edition. Prentice Hall, 2008.