Learning behaviour trees for simulated fighter pilots in airborne reconnaissance missions
– A grammatical evolution approach

Lärande av beteendeträd för simulerade stridspiloter i spaningsuppdrag

Pernilla Eilert

Supervisor: Johan Källström
Examiner: Fredrik Heintz

External supervisor: Tina Erlandsson
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Pernilla Eilert
Abstract

Fighter pilots often find themselves in situations where they need to make quick decisions. Therefore an intelligent decision support system that suggests how the fighter pilot should act in a specific situation is vital. The aim of this project is to investigate and evaluate grammatical evolution paired with behaviour trees to develop a decision support system. This support system should control a simulated fighter pilot during an airborne reconnaissance mission. This thesis evaluates the complexity of the evolved trees and the performance, and robustness of the algorithm. Key factors were identified for a successful system: scenario, fitness function, initialisation technique and control parameters. The used techniques were decided based on increasing performance of the algorithm and decreasing complexity of the tree structures. The initialisation technique, the genetic operators and the selection functions performed well but the fitness function needed more work. Most of the experiments resulted in local maxima. A desired solution could only be found if the initial population contained an individual with a BT succeeding the mission. However, the implementation behaved as expected. More and longer simulations are needed to draw a conclusion of the performance based on robustness, when testing the evolved BT:s on different scenarios. Several methods were studied to decrease the complexity of the trees and the experiments showed a promising variation of complexity through the generations when the best fitness was fixed. A feature was added to the algorithm, to promote lower complexity when equal fitness value. The results were poor and implied that pruning would be a better fit after the simulations. Nevertheless, this thesis suggests that it is suitable to implement a decision support system based on grammatical evolution paired with behaviour trees as framework.
Acknowledgments

To Tina Erlandsson, my supervisor at Saab AB, thank you for suggesting the project idea.

To my supervisors, Tina Erlandsson and Johan Källström, thank you for your constant and unstinted guidance, encouragement, insightful comments and enthusiasm shown throughout the length of the project. To my examiner, Fredrik Heintz, thank you for enthusiasm and insightful comments throughout the length of the project.

To my colleagues Christopher Bergdahl and Anders Jörgård at Saab AB, thank you for running my simulations on your computers. Thanks also to Lars Pääjärvi for your enthusiasm of the project and making me a part of the tactical team at Saab AB after my graduation.

To my family, thank you for your everyday support and encouragement.
Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures viii
List of Tables x
Abbreviations x
Vocabulary xi

1 Introduction 1
   1.1 Motivation ................................................................. 2
   1.2 Aim ........................................................................... 2
   1.3 Research questions ....................................................... 2
   1.4 Delimitations ............................................................... 2
   1.5 Thesis Outline ............................................................. 3
   1.6 Other additions ........................................................... 3

2 Theory 4
   2.1 Finite-State Machines ..................................................... 4
   2.2 Behaviour Tree ............................................................. 5
       2.2.1 Semantics ............................................................... 5
       2.2.2 Advantages ............................................................ 6
       2.2.3 Disadvantages ......................................................... 7
   2.3 Reinforcement Learning .................................................. 7
       2.3.1 Exploration and Exploitation ..................................... 8
   2.4 Evolutionary Algorithms ............................................... 8
       2.4.1 Evolutionary Cycle .................................................. 8
       2.4.2 Advantages and Disadvantages ................................ 10
       2.4.3 Genetic Programming ............................................ 11
       2.4.4 Grammatical Evolution ........................................... 15
   2.5 Evaluation Metrics ....................................................... 18
   2.6 Related Work ............................................................. 18

3 Method 20
   3.1 Pre-Study .................................................................. 20
       3.1.1 Scenarios and Behaviours ....................................... 20
       3.1.2 BT Structure .......................................................... 21
       3.1.3 Baseline ............................................................... 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.4 Fitness Function</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Implementation</td>
<td>21</td>
</tr>
<tr>
<td>3.2.1 Simulation Environment</td>
<td>21</td>
</tr>
<tr>
<td>3.2.2 Choice of Algorithm</td>
<td>22</td>
</tr>
<tr>
<td>3.2.3 Architecture and Framework</td>
<td>23</td>
</tr>
<tr>
<td>3.2.4 Motivation of techniques</td>
<td>23</td>
</tr>
<tr>
<td>3.3 Evaluation Metrics</td>
<td>26</td>
</tr>
<tr>
<td>3.4 Experiments</td>
<td>26</td>
</tr>
<tr>
<td>4 Results</td>
<td>28</td>
</tr>
<tr>
<td>4.1 Pre-Study</td>
<td>28</td>
</tr>
<tr>
<td>4.1.1 Behaviours</td>
<td>28</td>
</tr>
<tr>
<td>4.1.2 Scenarios</td>
<td>29</td>
</tr>
<tr>
<td>4.1.3 Baseline</td>
<td>32</td>
</tr>
<tr>
<td>4.1.4 Fitness Function</td>
<td>35</td>
</tr>
<tr>
<td>4.1.5 Defence potential</td>
<td>36</td>
</tr>
<tr>
<td>4.2 Implementation</td>
<td>37</td>
</tr>
<tr>
<td>4.2.1 Simulation Environment</td>
<td>38</td>
</tr>
<tr>
<td>4.2.2 Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Experiments</td>
<td>46</td>
</tr>
<tr>
<td>4.3.1 Baselines</td>
<td>46</td>
</tr>
<tr>
<td>4.3.2 Experiment 1</td>
<td>48</td>
</tr>
<tr>
<td>4.3.3 Experiment 2</td>
<td>59</td>
</tr>
<tr>
<td>4.3.4 Experiment 3</td>
<td>66</td>
</tr>
<tr>
<td>5 Discussion</td>
<td>71</td>
</tr>
<tr>
<td>5.1 Results</td>
<td>71</td>
</tr>
<tr>
<td>5.1.1 Experiments</td>
<td>71</td>
</tr>
<tr>
<td>5.1.2 Evaluation</td>
<td>72</td>
</tr>
<tr>
<td>5.2 Method</td>
<td>73</td>
</tr>
<tr>
<td>5.2.1 BT Structure</td>
<td>73</td>
</tr>
<tr>
<td>5.2.2 Implementation</td>
<td>73</td>
</tr>
<tr>
<td>5.2.3 Reliability and Validity</td>
<td>73</td>
</tr>
<tr>
<td>5.2.4 Sources</td>
<td>74</td>
</tr>
<tr>
<td>5.3 The work in a wider context</td>
<td>74</td>
</tr>
<tr>
<td>5.3.1 Replacement of fighter pilots</td>
<td>74</td>
</tr>
<tr>
<td>5.3.2 System control</td>
<td>74</td>
</tr>
<tr>
<td>5.3.3 Trustworthiness of an AI system</td>
<td>75</td>
</tr>
<tr>
<td>5.3.4 Considerations of the work</td>
<td>75</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>76</td>
</tr>
<tr>
<td>6.1 RQ 1 - Grammatical Evolution as a suitable method</td>
<td>76</td>
</tr>
<tr>
<td>6.2 RQ 2 - Important parameters of the implementation</td>
<td>76</td>
</tr>
<tr>
<td>6.3 RQ 3 - Performance due to Robustness and Complexity</td>
<td>77</td>
</tr>
<tr>
<td>6.4 Summary</td>
<td>77</td>
</tr>
<tr>
<td>6.5 Future Work</td>
<td>78</td>
</tr>
<tr>
<td>Bibliography</td>
<td>79</td>
</tr>
<tr>
<td>A Objectives and Behaviours</td>
<td>81</td>
</tr>
<tr>
<td>A.1 Objectives</td>
<td>82</td>
</tr>
<tr>
<td>A.2 Behaviours</td>
<td>82</td>
</tr>
<tr>
<td>B UML - Algorithm</td>
<td>83</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 Illustration of a behaviour tree with its components ........................................ 5  
2.2 Example of a behaviour tree .............................................................................. 6  
2.3 General evolutionary cycle ................................................................................ 9  
2.4 Illustrates the view of EA performance after D.E. Goldberg (1989) ...................... 10  
2.5 Graphical representation of one-point crossover ................................................. 14  
2.6 Illustrates an example of a BNF grammar .......................................................... 16  
2.7 Illustrates an example of a mapping process between genotype and phenotype .... 16  
2.8 Illustrates one-point crossover, n-point crossover and uniform crossover in GE. 17  
2.9 Illustrates the problem of ripple effect ............................................................... 17  

3.1 Illustration of the overview of the evolutionary cycle ........................................ 22  
3.2 Illustration of the architecture of the algorithm .................................................. 23  
3.3 Illustration of fixed one-point crossover and validation with both genotype (integer sequence) and phenotype (behaviour tree) ......................................................... 25  

4.1 Illustration of the first scenario .......................................................................... 30  
4.2 Illustrates the agent’s view in scenario 2 ............................................................. 30  
4.3 Illustration of the second scenario ..................................................................... 31  
4.4 Illustrates the waypoint routes of the airborne enemies in scenario 2 ................. 31  
4.5 Illustration of the third scenario ........................................................................ 32  
4.6 Illustrates the waypoint routes of the airborne enemies in scenario 3 ............... 32  
4.7 Illustration of the baseline BT .......................................................................... 34  
4.8 Illustration of the three different levels of threat ............................................... 36  
4.9 Illustration of how a tick travels through the baseline BT ................................... 39  
4.10 Illustration of the main loop of the implementation ........................................... 41  
4.11 The grammar .................................................................................................... 43  
4.12 Illustration of a tree structure generated by the PTC2 initialisation technique .... 44  
4.13 Illustration of a distribution of the fitness on a larger initialisation of 200 individuals ................................................................. 45  
4.14 Illustration of baseline’s graphical result and fitness value for scenario 1 .......... 47  
4.15 Illustration of baseline’s graphical result and fitness value for scenario 2 .......... 47  
4.16 Illustration of baseline’s graphical result and fitness value for scenario 3 .......... 48  
4.17 Illustrates the fitness of the trees for simulation 1, scenario 1 ............................ 49  
4.18 Illustrates the change in complexities of the trees for simulation 1, scenario 1 .... 50  
4.19 Illustrates the change in tree depth for simulation 1, scenario 1 ......................... 50  
4.20 Illustrates the fitness of the trees for simulation 2, scenario 1 ............................ 51  
4.21 Illustrates the change in complexities of the trees for simulation 2, scenario 1 .... 52  
4.22 Illustrates the change in tree depth for simulation 2, scenario 1 ......................... 52  
4.23 Illustrates the fitness of the trees for simulation 3, scenario 1 ............................ 53  
4.24 Illustrates the change in complexities of the trees for simulation 3, scenario 1 .... 54  
4.25 Illustrates the change in tree depth for simulation 3, scenario 1 ......................... 54  
4.26 Illustrates the fitness of the trees for simulation 4, scenario 1 ............................ 55
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Fitness interval</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation time</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>The extracted data during simulation for testing and plotting</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>The extracted data during simulation containing data used in the fitness function</td>
<td>40</td>
</tr>
<tr>
<td>4.5</td>
<td>The extracted data during simulation containing the result</td>
<td>41</td>
</tr>
<tr>
<td>4.6</td>
<td>Control parameters for the evolution process</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>Additional control parameters and features for Scenario 1</td>
<td>48</td>
</tr>
<tr>
<td>4.8</td>
<td>Result of the best evolved BT from Scenario 1</td>
<td>59</td>
</tr>
<tr>
<td>4.9</td>
<td>Additional control parameters for Scenario 2</td>
<td>59</td>
</tr>
<tr>
<td>4.10</td>
<td>Additional control parameters for Scenario 3</td>
<td>66</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>Behaviour Tree</td>
<td></td>
</tr>
<tr>
<td>UCAV</td>
<td>Unmanned Combat Aerial Vehicles</td>
<td></td>
</tr>
<tr>
<td>HDS</td>
<td>Hybrid Dynamical System</td>
<td></td>
</tr>
<tr>
<td>FSM</td>
<td>Finite-State Machine</td>
<td></td>
</tr>
<tr>
<td>HFSM</td>
<td>Hierarchical Finite-State Machine</td>
<td></td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
<td></td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
<td></td>
</tr>
<tr>
<td>MC</td>
<td>Monte Carlo</td>
<td></td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic Programming</td>
<td></td>
</tr>
<tr>
<td>TD</td>
<td>Temporal-difference</td>
<td></td>
</tr>
<tr>
<td>SARSA</td>
<td>State-Action-Reward-State-Action</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td>Evolutionary Computation</td>
<td></td>
</tr>
<tr>
<td>EA</td>
<td>Evolution Algorithm</td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
<td></td>
</tr>
<tr>
<td>GP</td>
<td>Genetic Programming</td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>Evolutionary Strategies</td>
<td></td>
</tr>
<tr>
<td>EP</td>
<td>Evolutionary Programming</td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>Grammatical Evolution</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>Control Architecture</td>
<td></td>
</tr>
<tr>
<td>RND</td>
<td>Random (initialisation)</td>
<td></td>
</tr>
<tr>
<td>RHH</td>
<td>Ramped-Half-and-Half (initialisation)</td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>Sensible Initialisation</td>
<td></td>
</tr>
<tr>
<td>PTC1, PTC2</td>
<td>Probabilistic Tree-Creation 1 and 2 (initialisation)</td>
<td></td>
</tr>
<tr>
<td>FPS</td>
<td>Fitness-Proportionate Selection</td>
<td></td>
</tr>
<tr>
<td>BNF</td>
<td>Backus Naur Form</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Start Symbol (BNF grammar)</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Non-Terminal Symbol (BNF grammar)</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Terminal Symbol (BNF grammar)</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Production Rules (BNF grammar)</td>
<td></td>
</tr>
<tr>
<td>SR</td>
<td>Success Rate</td>
<td></td>
</tr>
<tr>
<td>MBF</td>
<td>Mean Best Fitness</td>
<td></td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming</td>
<td></td>
</tr>
</tbody>
</table>
## Vocabulary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario</td>
<td>A scenario has an objective and an agent with its behaviours. In this project, the scenario is represented in a simulated environment.</td>
</tr>
<tr>
<td>Agent</td>
<td>In a scenario, the agent is the object that is controlled by a behaviour tree.</td>
</tr>
<tr>
<td>Target</td>
<td>In a scenario, the target is the object that the agent is searching for.</td>
</tr>
<tr>
<td>Threat</td>
<td>In a scenario, there could be both airborne and ground threats.</td>
</tr>
<tr>
<td>Unit or Units</td>
<td>In a scenario it is the term of an object, which could be of type agent, target or threat.</td>
</tr>
</tbody>
</table>
Introduction

Fighter pilots often find themselves in situations where the need to make quick decisions is of paramount importance. If the wrong decisions are taken, (in worst case) the consequences could be fatal. Therefore an intelligent decision support system that suggests how the fighter pilot should act in a specific situation is vital. Not only would this minimise human errors but also lighten the workload of the fighter pilot in stressful situations. To automatically act depending on the current situation in an intelligent way, unmanned combat aerial vehicles (UCAV) have an even greater need of this support. Such a system is not trivial to implement.

In game development, the search for a realistic game artificial intelligence (AI) has generated several techniques. One of the most common is Finite State Machines (FSM:s). A suggested improvement of the FSM when designing game AI is to use behaviour trees (BT:s). BT:s have numerous advantages over FSM:s, such as modularity, scalability and reactiveness. A BT describes the behaviour of an agent and given certain conditions, the tree states which action(s) the agent should execute.

A person with relevant domain expertise could model a BT to generate a suitable behaviour for the agent. However, in complex scenarios, it is difficult to obtain a tree that could solve all possible situations that could arise. It is also difficult to construct trees that are general and are able to perform well in similar scenarios. For that reason, one alternative is to use different machine learning algorithms to train the agent. How the agent should behave can be described with clear goals for a mission. Typical goals could be to carry out a mission, disarm the enemy and/or land before running out of fuel.

Two approaches for training decision making agents are Reinforcement Learning (RL) and evolutionary algorithms (EA). The difference between the two methods is that EA:s evolve new behaviours due to randomness in the algorithm, whereas RL improves an existing behaviour while interacting with the environment. One technique within EA:s is Grammatical Evolution (GE), which is a grammar-based form of genetic programming (GP). In this project, GE was implemented and evaluated for its suitability together with BT:s, to develop a decision support system for a simulated fighter pilot.

Two factors investigated in this project was complexity and robustness. As mentioned, a more complex scenario would generate a more complex tree structure. This makes it more difficult to generate and understand the structure of the tree. The less complexity a tree has, the higher understandability it has.
1.1 Motivation

The more general a behaviour tree is or in another word, robust, the better. It could then be used in similar scenarios and not only with the scenario it was trained on. Therefore a part of this project was to find key characteristics of the algorithm and configure these to produce a more robust tree.

The master’s thesis is produced in cooperation with the company Saab AB. The company is world-leading in solutions, services and products in military defence and civil security on the global market.

At present, the domain experts are designing BT:s by hand-coding them themselves. Since it is difficult to obtain a BT for complex scenarios, the motivation behind this project is to investigate if it is possible to generate a similar or better tree by using evolutionary algorithms. The results of the trees is compared to a hand-coded BT, which served as a baseline in this thesis.

Machine learning methods are a cross-border technology. By examining the possibilities to implement a learning method together with BT:s in aerial vehicles as a decision support system, conclusions can be drawn if the technique is suitable to this problem and worth researching in other fields. One example of another field would be autonomous cars. Both are autonomous vehicles, and similar, in the way that they both have hard deadlines, critical sections and need to be executed in real-time. This puts even more pressure on the execution on the behaviour of the trees.

1.2 Aim

The aim of this project is to investigate and evaluate machine learning, more specifically, EA:s with BT:s to develop a decision support system. This support system should control a simulated fighter pilot during an airborne reconnaissance mission including enemies.

1.3 Research questions

The research questions that will be answered in this report are the following:

1. Are evolutionary algorithms paired with BT:s a suitable method to develop a decision support system for simulated fighter pilots?

2. Which parameters of the implementation are of most importance for the system and how are these implemented?

3. How well do evolutionary algorithms perform with BT:s, based on robustness and complexity compared to a baseline?

1.4 Delimitations

A simulation environment, provided by the company was used for testing purposes, including implementation of the decision support system and the evaluation of the results. By using this environment, the implementation had to be done in MATLAB\textsuperscript{1}, version R2017b. Matlab is a computing software used by engineers and scientists.

Another limitation was the definition and usage of scenarios. In the pre-study, scenarios and behaviours for an airborne reconnaissance mission were decided and summarised. The behaviour tree should be trained on a specific scenario and be able to handle similar scenarios.

\textsuperscript{1}https://se.mathworks.com/products/matlab.html
Lastly, the time restraint of the project was 20 weeks. This limited the amount of simulations. For future work in the same field and suggestions for further experiments are presented in the end of the thesis, in chapter 6.

1.5 Thesis Outline

The outline of this thesis is as follows. Chapter 2 presents relevant theory to the project. Chapter 3 presents the pre-study, implementation and the method of the experiments. In chapter 4, the results of the pre-study, various control parameters, distribution of the initialisation technique and the executed experiments are presented. This is followed by chapter 5, where a discussion of the two previous chapters are held. Finally, in chapter 6, conclusions are drawn from the discussion.

1.6 Other additions

The objects (aircraft and ground units) in the scenarios are illustrations from Saab, and the maps in the scenarios are taken from Openstreetmap. Otherwise, the figures in the thesis are made by the author.

\[\text{https://www.openstreetmap.org}\]
This chapter explains relevant concepts to the thesis. A brief introduction to finite-state machines and their relation to behaviour trees starts this chapter. Continuing with the semantics, advantages and disadvantages of behaviour trees.

In section 2.3, a brief introduction to RL is presented, followed by a more thorough introduction of EAs and their advantages and disadvantages compared to RL. Suitable evaluation metrics of EAs are explained in section 2.5.

This chapter ends with a related work to give the reader an idea of how far the research has gone until this present day.

2.1 Finite-State Machines

A finite-state machine (FSM) is a mathematical model of computation. An FSM has a finite number of states, but is limited to enter only one state at a time. The model consists of an internal state, a list of its finite states and the terms for the transitions. Transition is the common name for switching between states. The FSM is the discrete part (decision making) of a hybrid dynamical system (HDS) [2]. An HDS has, as mentioned, a discrete part but also a continuous part (motion in virtual environment) [2].

FSM:s is one of the most common techniques in game development, to generate a realistic game AI [17]. As mentioned in several papers [17, 3], an FSM has numerous disadvantages. Modularity and scalability is difficult to manage when the systems become bigger and more complex. As M. Olsson mentioned in his paper (2016) [21], to generate a fully functional system, there has to be transitions between all states. Since all nodes are connected, it can be difficult to modify or remove one state without changing the connections and the other states in the FSM.

In 1987, D. Harel developed state charts, more commonly known as hierarchical finite-state machines (HFSM) [12]. This was an attempt to facilitate the transition duplication and increase the understanding of the complex systems. In an HFSM, a node can be a superstate, which contains substates. This means that several substates share one superstate. This made a more modular FSM but otherwise, it has the same disadvantages as an FSM.
2.2 Behaviour Tree

A behaviour tree is a directed graph with a tree structure. A tree is constructed with nodes and arcs, and the top node is called root. There are six types of nodes: Selector, Sequence, Parallel, Decorator, Action and Condition. If the node is a leaf, then it could be of type Action or Condition, whereas if it is not a leaf, it is one of the others. All the node types are explained below and in figure 2.1, some of the types are illustrated.

Figure 2.1: Illustration of a behaviour tree with its components (from the top and left). Root node followed by a sequence node. The first sub-tree (to the left) represents selector node, condition node and action node. The second sub-tree represents parallel node, action node, selector node, condition node, action node, decorator with its only child, here an action node.

2.2.1 Semantics

The Selector node can be considered as an OR gate [15, Chapter 3]. It selects the first child and executes it, if the child fails, the next child will be executed. If a child was successfully executed, the execution ends and the node returns success. The node can also return the state running, depending on the current status of the child. Since the node acts as an OR gate, it can verify several conditions to see if either of them is true.

In a Sequence node, all its children is executed sequentially in a determined order. The node returns success if the execution has finished and all the children have successfully been executed, otherwise it returns failure. While a child is running, the node returns running. Since the children are executed sequentially, the behaviour of the node is similar to an AND gate [13, Chapter 3]. This makes it suitable for applications that need specific tasks to be successfully executed together. Unlike the sequence node, which executed all the children one by one in a specific order, the Parallel node executes all its children in parallel. For the node to return success, a certain fraction of all the children need to return success, otherwise it returns failure.
2.2. Behaviour Tree

The **Decorator** node differs from the earlier nodes, it has only one child. The node can modify its child in three different ways. It can change the behaviour of the child, the return status of the child, or not to tick the child [13, Chapter 3].

To perform an action, an **Action** node is used. It is also possible to state conditions that have to be met before executing the action. These conditions are placed in a **Condition** node. The action node returns success if the action is executed, failure if it is not, and running if the action is still in execution mode. It is important to note that the condition node cannot modify variables of a BT.

Figure 2.2 illustrates an example of a BT. From the root node starts a tick and the tree gets executed from the left to right. The first node to get executed is the selector node, which has two children nodes. The left child node is a decorator (RoT), and resets on each tick. This means that the result from its child has to be checked again next tick. Therefore the decorator could change the return status of the child from running to ready. The child is a sequence node with two children, one condition and one action.

If the condition node returns failure, then the next sub-tree is executed. If it returns success, then the action is performed in the current sub-tree.

Figure 2.2: Example of a behaviour tree. The first node is the root node, below is a selector node. The first child of the selector node is a decorator node with type RoT (reset on tick). Its child is a sequence node with one condition node and one action node. The second child of the selector node is an action node.

### 2.2.2 Advantages

Chong-U Lim, R. Baumgarten and S. Colton suggested in their paper (2010) [17], that behaviour trees are superior to FSM:s when designing game AI. BT:s have numerous advantages over FSM:s such as modularity [17, 3], scalability [17, 3] and reactivity [3]. According to R. de P. Pereira and P. M. Engel (2015) [25], BT:s are commonly used to model NPC:s (Non-Playable Characters controlled by the computer). They also claim that BT:s are viewed, not only as an alternative to FSM:s, but also for HFSM:s (Hierarchical Finite State Machines) and hand-coded rules done by scripting.

P. Ögren (2012) [20] argues in his paper that BT improves modularity, reusability and complexity of control systems of UAV:s. He claims that these advantages are a result of the tree structure of the BT:s. M. Colledanchise and P. Ögren (2017) [2] believe that FSM can
be complemented by BT:s in robotic software development. A. Klöckner (2013) [16] compares BT:s to FSM and conclude that due to the tree structure of BT:s, it offers increased scalability and a simple, standardised interface. The author also mention that FSM:s need to be re-initialised when updated, whereas BT:s do not. In the paper of A. Klöckner (2013) [16], the author states that BT:s can be used for UAV mission management.

The author M. Colledanchise (2017) [4] lists advantages of BT:s, and presents eight design principles of a control architecture in the report. The author claims that a BT meets the properties of the design principles, to a certain extent, due to its advantages. Some of the advantages of a BT are tree structure, how the traversing in the tree is made and to its return status. In the paper of P. Ögren (2012) [20], the author explains that when the BT is executed, the root node is ticked for each time step along the control loop. The tick travels down the tree to a specific leaf node (which is executed or modified according to the decorator node) and back again. This means that the BT is tick driven and not event driven as the FSM. It also has no states, only conditions.

2.2.3 Disadvantages

The disadvantages lifted in the paper of the author M. Colledanchise (2017) [4], are a collection of problems that BT developers have encountered while working with the trees. By using single threaded sequential programming, the implementation of the BT engine could be difficult. BT:s are also not a better option over simpler control architectures (CA:s) when executed in an environment that is predictable. It is expensive or not possible to check conditions in a closed-loop task execution. A solution however, would be to design an open-loop task execution and including memory nodes in the BT. Lastly, something that has to be taken under consideration when choosing BT over other CA:s is that BT:s are not yet as widely used. Therefore less information exists concerning development and test. A. Klöckner (2013) [16] mentions drawbacks on BT:s, that they can not foresee future actions of the tree and the reaction time (for real-time deployment) of the tree must be guaranteed.

In the paper A Framework for Constrained and Adaptive Behaviour-Based Agents (2015) [25], limitations of the BT:s are mentioned. They lack variation in behaviours and are unable to adapt to changes. However, the authors present a solution to these limitations, to use learning algorithms. According to the authors this would avoid repetitiveness and raise the adaptiveness. Y.S Janssen (2016) [15] points out in his paper that work have been done to improve BT design using different learning techniques, for example, evolutionary learning and Q-learning.

2.3 Reinforcement Learning

Reinforcement learning (RL) is a technique in machine learning where the agent learns by doing, also called trial-and-error. This means that RL has no training data and learns from its mistakes or successes. This feedback is called reward or reinforcement and can be given during or in the end of an execution. These rewards are used to find an optimal policy in Markov Decision Processes (MDP:s). [26], Chapter 22] Reinforcement learning algorithms use the framework of Markov Decision Process (MDP), as it describes the decision process of the agent. An optimal policy is a policy that maximise the expected total reward.

There are three different approaches to RL: value-based, policy-based and a combination of the two, actor-critic. These three approaches can then be model-based or model-free. Model-based RL learns a model of the environment, which can be used to generate synthetic experiences for updating the policy, whereas model-free RL attempts to directly learn the best policy.

Value-based RL optimises the value function, and are related to Dynamic Programming principles. Examples of algorithms are SARSA and Q-learning.
Policy-based RL optimises a policy function without using the value function. There are two types of policies: deterministic and stochastic. If the policy is deterministic it always returns the same action for a given state, whereas a stochastic policy will give a distribution probability over possible actions. Examples of an algorithm searching in policy space is Policy Gradient.

2.3.1 Exploration and Exploitation

In RL, the trade-off between exploration and exploitation is important and also a dilemma. Exploitation focuses on the current knowledge to maximise a reward, whereas exploration means that the agent explores to possibly improve its policy. If an algorithm always chooses the best known action with the highest reward, the algorithm is greedy. This can lead to that the algorithm does not find the optimal solution to the problem, but only a local maximum or minimum depending on the problem, hence the dilemma. The solution to this problem is to guarantee that the algorithm explores the environment and does not take the action with the highest reward every time. This is called \( \epsilon \)-greedy policy. The \( \epsilon \)-greedy policy gives a flexible approach in the absence of domain information. The value of \( \epsilon \) states the percentage of times that the agent should randomly select an action. Instead of choosing the action that is most likely to maximise the reward given what the agent knows so far.

2.4 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a subclass of Evolutionary Computation (EC) and are inspired by the natural evolution process. For example, if an agent is placed in a new environment, these algorithms help the agent to adapt, to be able to survive in its new surroundings. Often, the EAs solve optimisation problems (EAs are a population based meta-heuristic optimisation method). The four most commonly used algorithms in EA are genetic algorithms (GA), genetic programming (GP), evolutionary strategies (ES) and evolutionary programming (EP). The algorithms differ only in technical details. In Introduction to Evolutionary Computing, the authors represented the algorithms accordingly:

- GA represents strings over a finite alphabet
- GP represents trees
- ES represents real-values vectors
- (classical) EP represents state machines

2.4.1 Evolutionary Cycle

A.E. Eiben and J.E. Smith (2003) describe a general evolutionary cycle in their paper. The steps are expressed in a flow chart in figure and explained below.
As seen in figure 2.3, there are several steps in an EA. The initial step of an EA is to transfer the actual environment to the simulated environment connected to the EA. This step is called **Representation**. The authors explained the representation as a bridge between the problem and the simulation, where the problem is solved through evolution. The next step is **Initialisation** and it sets the initial population. The population can be chosen randomly (most common) or by a heuristic algorithm to obtain an initial population with higher fitness.

An important part of the algorithms is the **Evaluation function** (also called fitness function). If the ideal solution is known, it evaluates how close the given solution is to the ideal solution of the problem. The function should be efficient and generate results what one feels to be true. In the **Population** step, the possible solutions are found. It is here the unit of evolution is shaped. It is important to understand that the individuals by themselves can not be modified, it is the population that can be changed and adapted to the environment. As A.E. Eiben and J.E. Smith (2003) mentioned, the individuals can be seen as static objects. Often in EA, the population size does not change during the cycle. Since the size is fixed, the algorithms need to decide which individuals should continue to the next step in the cycle. The decision is made by comparing the results of the fitness function and/or the age (parent or offspring). This fitness-ranking takes place in the **Survivor selection** step.

**Parent Selection** separates the individuals by comparing their quality. The individuals with better quality are chosen to become the parents of the next generation. However, this does not mean that the ones with low quality will not become parents. A few will get included to generate a broader distribution. If all the parents with lower quality would have been excluded in this step, the population could end up in a local optimum.

With the help of **Variation operators**, new individuals will be generated from the parents. There are two types of variation operators: mutation and crossover. In **Crossover**, the features from two parents are randomly mixed together to obtain one or two offsprings. The algorithms generate random combinations of the features, some will be better and some will be worse than the parents. Crossover differs between the EC types. A.E. Eiben and J.E. Smith (2003) states that it is often the only variation operator in GP, whereas in GA it is the main operator for the search and in EP, it is not used. In contrast to a crossover operator, which takes two parents as input to generate an offspring, a **Mutation** operator takes only one parent as input and modifies it randomly. Even for mutation the use of the operator differs depending on the chosen algorithm.

**Survivor selection** reminds of parent selection but is executed later in the cycle. It separates the individuals based on their quality and is executed after the creation of the offspring. Survivor selection is deterministic, in contrast to parent selection, which is often stochastic.
2.4. Evolutionary Algorithms

The last step in the cycle is the **Termination condition**, according to A.E. Eiben and J.E. Smith (2003) [8], there are two suitable conditions for termination. The first is to stop the execution if the optimum value is found. The second is to choose a condition which ensures that the algorithms will stop. The authors gave four examples of suitable conditions that will stop the algorithms: limit on the number of fitness evaluations, limit on allowed CPU time, threshold value of fitness improvement and population diversity.

2.4.2 Advantages and Disadvantages

Advantages of EA:s, mentioned in the paper *Evolutionary Algorithms: A Critical Review and its Future Prospect*, are their simplicity and flexibility, since they are inspired by the natural evolution process. They take advantage of previous information and they are representation independent. The algorithms are robust, due to the successfulness of the adaptiveness of the solution in new unobserved environment. However, there is no guarantee of an optimal solution for a specific problem and the algorithm demands a lot of simulation time. [30]

In the book of D. E. Goldberg (1989), the author compares robust problem solvers to algorithms that are tailored to a specific solution and to random search. Here, EA:s are also claimed to be robust problem solvers. [11] The result of the comparison, is illustrated in the figure 2.4. As can be seen in the figure, the EA:s outperform random search. The algorithms that are tailored to a specific solution performed much better on the specific problem it was assigned to. However the algorithm looses its performance when changing the type of problem.

![Figure 2.4: Illustrates the view of EA performance after D. E. Goldberg (1989) [11].](image)

Newer research shows that there are no general problem solvers that are successful and efficient for all range of problems. Therefore the figure 2.4, does not give a completely accurate picture of the algorithm performance. But EA:s are still a more successful and efficient general algorithm than a tailored one. [8]

Related work in section 2.6, showed that a TD-method that has been frequently used together with BT:s in game development was Q-Learning. As for the EA:s, genetic programming was mentioned several times together with BT:s. Sometimes a mapping between genotype and phenotype is necessary, and to solve this, a grammar-based form of GP could be applied. A genotype is represented as an integer sequence and is mapped to the phenotype, which is the solution.

Grammatical evolution is based on GP but it handles the mapping process between genotype and phenotype. The integration with BT:s for the two algorithms are equivalent, except
that the grammar has a problem by being too flexible, which may result in invalid trees. However, this is easy to solve by setting up rules how the tree can be structured.

One disadvantage with RL techniques is the trade-off between exploitation and exploration. Since the EA techniques do not interact with the environment in the same way, they do not share this problem. Even if the environment would generate faulty information, this does not affect the EA techniques as much as the RL techniques. When an agent has ambiguity in its perception, G. D. Croon et. al (2005) show that evolutionary learning outperforms reinforcement learning when training an agent, since performance differences between reinforcement learning and evolutionary learning are related to the proportion of this ambiguity. This is an important aspect, since an airborne vehicle collects a huge amount of data from its sensors. However, the simulated environment used in this project does not produce disturbance or generate faulty data to the sensors. A disadvantage by using an EA technique instead of an RL technique, would be a less effective algorithm when using clear data. RL techniques are better suited to solve specific problems, whereas EAs are more general.

It is a problem in itself to find a valid fitness function when using EAs. Often the algorithm gets stuck in a local maximum or minimum depending on the problem. To prevent this, trees with less fitness are included in the evolution process.

Q-learning is an off-policy TD control algorithm. Off-policy is one of two classes of learning control methods, the other is on-policy (estimates the value of a policy while using it for control). In off-policy, the behaviour policy and estimation policy are separated. This is an advantage, since the behaviour policy samples all possible actions whereas estimation policy is deterministic and therefore greedy. However in Q-learning it is hard to trust good Q-values and the more complex the scenario gets, the longer simulation time the algorithm needs. Long simulation time is however a common problem in EAs too. To solve the simulation time, the scenarios could be decomposed into smaller missions. Q-Learning does not manage huge state spaces, while EAs handles it better.

To enhance the performance of EAs, research has been made on hybrids of EAs and other techniques. A hybrid of an EA and a heuristic method would perform better than its parents separately.

### 2.4.3 Genetic Programming

Genetic programming (GP) is a popular technique within EC, which evolves computer programs. The solution structure is translated into a tree-based structure which is operated on by evolutionary operators. In programming, the tree is equivalent to a function and is evaluated in a leftmost depth-first manner. In the tree-based structure, there are two types of nodes: functions and terminals. The leaves of the tree are terminals and the nodes with children are functions.

#### Initialisation

**Random (RND) initialisation** was used in the original implementation of GP. It generates random genotypes of specific length, arrays with binaries or integers. After the initialisation is made, it is needed to validate the genotypes since RND generates many invalid genotypes that can not be mapped or it generates repeated solutions. This is a disadvantage of the technique, however it is easy to implement.

**Grow, Full and Ramped-Half-and-Half (RHH)** are three initialisation techniques defined by J. Koza. The technique Grow generates a random population, with one individual at the time. The creation starts from the root node and every lower level node is randomly chosen as either a function or a terminal node. If it is a terminal node, then a random terminal is chosen. And if it is a function, a random function is chosen. The amount of children of the
function node is as many as its number of arguments. If the algorithm has reached the limit of tree depth, then the children will automatically become a randomly selected terminal. [31]

Full is similar to grow, except that it requires a maximum depth. Starting from the root node, all nodes with the depth less than the limited depth is a randomly selected function. Beyond that, the nodes become randomly selected terminals. [3]

One problem with the two methods mentioned above is that they do not provide trees with a wide range of variety of size and shape. RHH is a combination of both, developed to ensure enough diversity in the population. The method divides the population in sub-groups. The amount of sub-groups are the same as maximum depth minus one. Each sub-group is assigned an individual maximum depth size and half of the trees are generated with the method full and the other half with grow. [7, 31]

Two other initialisation techniques are called Probabilistic Tree-Creation 1 and 2 (PTC1, PTC2). S. Luke concludes in his paper (2000) [18], that PTC1 and PTC2 have a low computation time even though the algorithms provide uniform distribution of functions. According to the paper (2017) [19], the PTC2 provided a wider sampling of initial solution sizes, depths and densities. He also presented a version of PTC2, that generated even denser trees, but performance decreased. PTC2 works as follows:

1. If the control parameter of tree size is one, then generate a random terminal and return it.
2. If the control parameter of tree size is larger than one, then generate a random non-terminal as root in the tree. The children of this node are put into a queue $Q$.
3. While the sum of the size of the queue $Q$ and the size of the nodes in the tree are equal or less than the control parameter of tree size, do the following:
   - Remove a random position from the queue $Q$, fill the position in the tree with a random non-terminal $n$ and add all $n$’s children in the queue $Q$.
4. Iterate through the queue $Q$ and fill the position in the tree with a random terminal.

It is possible to apply probability when choosing non-terminals and terminals, in that way, some non-terminals and terminals have greater probability to be chosen.

Fitness Function

To rate how well an individual performs, the algorithm uses a measure called fitness test or fitness function [31]. The fitness reflects the characteristics of the environment. A fitness function is an important part of the evolutionary algorithm, since it evaluate the population based on these characteristics or requirements [8]. The requirements have to be carefully chosen, since the fitness function serves as a base for the selection functions.

In maximisation problems a higher fitness value from the fitness function is desired, whereas in minimisation problems, a lower fitness value from the fitness function is desired.

Selection Functions

As mention in section 2.4, there are two types of selection; parent selection and survival selection. These are executed at different stages in the evolutionary cycle, see figure 2.3. However the methods of parent selection could also be used in the survival selection [8].

In the paper (2015) [8], the authors present two different population management models found in literature. The models are generational model and steady-state model. The difference between the models are that the first mentioned, changes the entire population at each generation, whereas the other does not. Generational state model, starts with a population

---

of chosen size. From this population, an amount of parents are selected and together, they create a mating pool. All the individuals in the mating pool are copies of the population, but with more copies of the parents with higher fitness and less copies of the parents with lower fitness. Offsprings are then created from the mating pool with the genetic operators and the offsprings are evaluated. After each generation, the whole population is replaced with individuals, selected from the offspring. The steady-state model works similar, but does not change the entire population, but only a part of it. The part of the population that is replaced is called generational gap.

M. Walker (2001) \cite{Walker2001} mentions three selection functions in his paper: **fitness-proportionate selection** (FPS), **greedy over-selection** and **tournament selection**. He states that these three are the most frequently used by J. Koza. Both FPS and tournament selection are commonly used methods in parent selection \cite{Koza1992}. These two are explained below.

In FPS, the individuals are chosen by their absolute fitness value compared to the population. This means that it is probabilistic (the fittest individuals are more frequently selected than the individuals with worst fitness). The percentage proportional of the fitness value is computed as in equation \ref{eq:fps}.\footnote{13}

\begin{equation}
P_{FPS}(i) = \frac{f_i}{\sum_{j=1}^{n} f_j}
\end{equation}

FPS uses the technique called **roulette wheel** algorithm \cite{DeJong1975}. The roulette wheel algorithm can be resembled by a roulette wheel (hence the name). The proportion of the selection probability reflects the size of the holes in the roulette wheel. By spinning the wheel, some individuals have higher probability to be chosen.

Some disadvantages when using fitness proportional selection are mentioned in the paper \cite{Koza1992}. The better individuals can take over the population too quickly or there are almost no selection pressure. The second happens when the fitness values are very similar and therefore the mean fitness increases slowly. When the selection pressure increases, more fitter individuals are more likely to survive, which leads to less fitter are less likely to survive.

Tournament selection performs tournaments between chosen individuals and the individual with highest fitness wins. When using tournament, the number of individuals for a tournament has to be decided as well as the number of the tournaments. Probability parameter is used if the highest fitness is not desired but randomness is. The probability parameter in tournament selection adjusts the selection pressure and is pre-defined. A random number between zero and one are compared to the probability parameter. If the number is greater than the probability parameter, then the less fit individual is chosen. The probability parameter is often set to larger then 0.5 to favour the more fit individuals. \footnote{10} Tournament selection works well on large population sizes, because it does not require information about the population nor the quantifiable measure of quality. \footnote{8}

When applying the genetic operators after parent selection, individuals with high fitness can be lost. Often it is possible for the algorithm to re-discover them. A solution is to implement survival selection. Two common survival selection techniques are **fitness-based replacement** and **age-based replacement** \cite{Koza1992}.\footnote{8}

There are different features of fitness-based replacement \footnote{8}. To be sure that the individuals with highest fitness survive, a feature called elitism can be applied. Elitism moves a pre-defined fraction of the population to the next generation without going through the genetic operators. \footnote{10}

The aged-based replacement ensures that each individual exists the same amount of EA iterations. This means that it does not consider the fitness values and therefore the best fitness from one iterations can change to the next. This leads to a priority of children over its parents.\footnote{8}
Genetic Operators

One of the main genetic operators in GP, is crossover. Crossover is often the only operator used \[8\]. In the book (2012) \[14\], the authors present three types of crossover. The first is one-point crossover and is illustrated in the figure 2.5. Given two individuals, selected from different sub-groups, a cross point is randomly selected in both trees. Then the subtrees of the cross points are cut and recombined. \[31, 22\]

The second is n-point crossover, it behaves as one-point crossover but it has n crossover points. Uniform crossover is the last type and it uses a masking technique. The two chosen parents masks with a binary sequence. If the number is a zero, then the first child gets the value from the first parent, whereas if it is a one, it gets the value from the second parent. And the inverse for the second child. \[14\]

![Figure 2.5: Graphical representation of one-point crossover. First the selection function randomly chooses a cross point in both trees (parents). The trees are cut at these points and the sub-trees are switched. The result is two new trees (offsprings).](image)

Another genetic operator in GP is mutation. There are two different types of mutations. The first is micro-mutation, which affects the leaf nodes. The second is macro-mutation (also called Headless Chicken Crossover), which replaces an existing node by a randomly generated tree (with limited depth). \[27, 28\] Macro-mutation reminds of crossover, but uses only one individual and the replacing node is a randomly generated tree.

In GP, the mutation and crossover are applied in parallel and not followed by each other. This means that the selected individuals become either mutated or crossover. Whereas in other variants of GA, crossover is followed by mutation. \[8\] Crossover and mutation are not the only operators, a few other evolutionary operators worth mentioning are editing, permutation, encapsulation and decimation \[31\].
2.4. Evolutionary Algorithms

Control Parameters

In GP, the user has to decide upon control parameters. Some essential parameters are population size, maximum number of generations and probability of mutation and crossover. The more complex a problem is, the larger population size is needed. Maximum number of generations is used as a termination parameter. If the algorithm has not successfully created an individual, then the cycle ends after a pre-defined number of generations. The balance between crossover and mutation is stated as two probability parameters. In GP, often only crossover is used, therefore when using mutation, its probability is low. It is around 5-10% and sometimes even omitted. [31]

2.4.4 Grammatical Evolution

Grammatical Evolution (GE) is a grammar-based form of GP. GE is constructed by a genotype and a phenotype and a grammar, mapping between them. The representation of GP is a tree structure, which means that the phenotype is of this structure. The genotype is represented as an integer sequence. The genotype-phenotype mapping is not always required.

Backus Naur Form

Backus Naur Form (BNF) is a notation for expressing the grammar of a language in the form of production rules and is a left recursive grammar. BNF is represented by the tuple, see equation 2.2, where N stands for non-terminal symbols, T for terminal symbols, P for production rules and last, S for start symbol. [22]

When applying BNF with BT, the terminal symbols are the nodes which serve as leaf-nodes, and non-terminal symbols with including functionality are the other nodes in the tree structure. In other words, non-terminal symbols also includes symbols that are not part of the final tree structure. The start symbol S, is represented with the first symbol in the grammar. The mapping rule is represented in the equation 2.3. The rule selected, is depending on c (codon), which is an integer value, and r, the number of option rules that the current non-terminal symbol has.

\[
BNF = (N, T, P, S) \quad (2.2)
\]

\[
Rule = c \mod r \quad (2.3)
\]

The non-terminal symbols, terminal symbols and start symbol,

\[
N = \{< BT >, < Node >, < Condition >, < Action >\}
\]

\[
T = \{obstacleAhead(); enemyAhead(); moveLeft(); moveRight(); jump(); shoot();\}
\]

\[
S = \{< BT >\}
\]

together with production rules give an example grammar, illustrated in figure 2.6. This grammar could be used in a simple game, where the agent has the actions move left, move right, jump and shoot at enemies.
By changing the parameters \( \{N, T, P, S\} \) in the BNF grammar, it is possible to change the rules of the mapping. This makes the GE flexible and simple to change.

A short example of the mapping between genotype to phenotype is illustrated in figure 2.7. The example is using the grammar illustrated in figure 2.6. The figure includes an integer sequence, the result of the mapping to the left and the computation to the right.

The mapping starts with the start symbol, here \(<BT>\) and the first codon \(c = 2\) in the integer sequence. The start symbol has only two option rules, therefore \(r = 2\). By using the equation 2.3 with \(r = 2\) and \(c = 2\), the result is \(<BT> <Node>\).

Since the grammar works from left to right, the next symbol to be translated is \(<BT>\). The symbol to be translated in each step is marked as bold text. There are still two option rules, but this time \(c = 3\) and this results is \(<Node>\).

The translation continues until the end of the codons (integer sequence). As seen in the mapping example, there are several codons on one node. Therefore each codon in the genotype does not directly correspond to a node in the phenotype.

*sequence*: \([2, 3, 6, 2, 5, 6]\)

\[\begin{align*}
&B\div 2 = 0 \\
&B <\text{Node}> \quad 3 \mod 2 = 1 \\
&\text{Node} <\text{Node}> \quad 6 \mod 2 = 0 \\
&\text{Condition} <\text{Node}> \quad 2 \mod 2 = 0 \\
&\text{obstacleAhead}(); <\text{Node}> \quad 5 \mod 2 = 1 \\
&\text{obstacleAhead}(); <\text{Action}> \quad 6 \mod 4 = 2 \\
&\text{obstacleAhead}(); \text{jump}();
\end{align*}\]
2.4. Evolutionary Algorithms

Figure 2.8: Illustrates from the top, one-point crossover, n-point crossover and uniform crossover in GE.

However the sequences might become invalid after crossover and it is not certain that the sequences in the figure 2.8 are valid after the operation. It can happen that the sequence is too short after the operation or too long. The phenomenon is called ripple effect [6]. It means that after the crossover point, the phenotype can defer from its original context. The problem is illustrated in figure 2.9.

A solution to the problem of one-point crossover if the sequence is too short could be to re-read the sequence from the beginning [13]. Otherwise, some sort of validation needs to be called after crossover. This problem can also arise after mutation.

Mutation in GE is similar to crossover in GE. However, instead of using two sequences, only one is used and after the mutation point (same as crossover point) the codons are randomly chosen. In GE, it is possible that the mutation on the genotype has no effect on the phenotype. Since the mutation generates random codons it is possible that the result stays the same. [6] For example, if the production rule has a non-terminal to be replaced with one of two variables, then it is 50% chance that the old variable gets chosen. This is also called that the mutation can be neutral [6].
2.5 Evaluation Metrics

Both the performance of the learning algorithm and the structure of the trees are important to evaluate. The four following evaluation metrics do that.

- Success Rate
- Mean Best Fitness
- Complexity, Tree Depth and Transparency
- Robustness

The success rate (SR) gives the percentage of the individuals each generation when the optimal or a sufficient solution is found. Mean best fitness (MBF), computes the average of the best and mean of the individuals fitness values over the generations. MBF can be measured for any problem that is solved with an EA, if it uses an explicit fitness function. MBF can always be applied as a valid measure, whereas for some problems, SR can not be defined. Chapter 9 Both high SR and high MBF are desired.

If the simulation runs result in low SR and high MBF, it means that the results are near but rarely reaches an optimal or a sufficient solution. By increasing the generations, it can improve the chance of finding the optimal or the desired solution. However, if the simulation runs results in the opposite, high SR and low MBF, it indicates "Murphy algorithm". "Murphy algorithm" means that if it goes wrong, it goes horribly wrong. This leads to that the simulation runs that do not reach the desired solution, ends with poor fitness values. Chapter 9

To evaluate performance, Kirk Y.W. Scheper et. al (2015) used metrics such as success rate and tree size. Tree size can be measured by its complexity (amount of nodes) and the tree depth. Analysing complexity and tree depth, gives an understanding of the structure of the tree.

Robustness measures how well the BT performs in new simulated scenarios. Also how well the algorithm changes the trees through the generations. The measures used to decide the success of robustness are SR and fitness value of a BT in a different simulated scenario.

2.6 Related Work

In the paper of Dr. C. Child and R. Dey (2013), the authors introduce behaviour trees with Q-Learning. The authors present three different BT:s. The first is a standard BT and the second uses an ε-greedy policy on learned Q-values. The last one is a BT containing Q-Condition nodes (QL-BT) instead of standard Condition nodes. They conclude that the QL-BT has an advantage over manual generations of BT:s, since it performs on the same level or better than the standard BT. However, the authors also mention drawbacks of Q-learning. The drawbacks are that you have to rely on correct Q-values and the need of heavily increased simulation time for complex games. G. de Croon et. al (2005) conclude in their paper that evolutionary algorithms have an advantage over reinforcement learning. In evaluation of learning the behaviour of an agent that has ambiguity in its perception abilities, they show that evolutionary algorithms outperform reinforcement learning.

In the paper Learning of Behaviour Trees for Autonomous Agents, the authors have proposed a control system, where a combination of a greedy-algorithm and genetic programming is used to train a BT. This control system helps Super Mario to manage both obstacles and enemies. They are also using a model-free framework to make the system more robust and not depending on previous knowledge of the environment. A model-free framework is when the framework does not need to know additionally information about the environment except what it is possible to observe. However, the authors address that there needs to be more research and analysis before applying this approach on robotics.
In the paper of D. Perez et. al (2011) \cite{24}, the authors present a Grammatical Evolution approach to evolve Behaviour Trees. They state that several of the encountered problems when using Genetic Programming, can be solved with Grammatical Evolution. However, when mapping the algorithm together with the BT syntax, the authors found that the approach was too flexible and the trees were inefficient. To avoid this, the authors had to limit the syntax, and generated three rules for the structure of the BT. The root node of the tree has to be a selector node, with a variable number of sub-trees. Each sub-tree consists of a sequence of at least one condition, followed by a sequence of actions. Lastly, if all of the conditions in the tree traversal would fail, the tree has to have a default sequence of actions but no conditions. Dr. C. Child and R. Dey (2013) \cite{1}, also introduced an unconditional fall-back behaviour, which was random movement.

D. Perez et. al (2011) \cite{24} used an and-or trees structure, which worked well with the Grammatical Evolution approach. The authors claim that their result supports the idea that GP systems are an alternative to traditional AI algorithms. Either as by themselves or as part of an hybrid. Since their approach gave positive results in enemy shooting and close range obstacle avoidance but poor results in planning and path finding, they consider a hybrid for future work. P. A. Vikhar (2016) \cite{30} gives examples of different evolutionary hybrids in his paper and states that it gives better results than only using one algorithm. L. Chong-U et. al (2010) \cite{17} also raises the question if other techniques should be added to evolutionary techniques in automating AI-bot design.

Decision support systems have mainly been researched on fighter-to-fighter or on sub-missions. A complete mission is complicated since its complexity increases drastically with more decisions and actions. An airborne reconnaissance mission is an example of such a mission.
This chapter describes the method of the three parts: pre-study, implementation and experiments. The first part, the pre-study started with a literature study to research the area and summarise relevant concepts to the thesis. The information is found in chapter 2.

Besides the literature study, the main purpose of the pre-study was to determine the scenarios for the simulation environment and the behaviours used in the algorithm. By determining the scenarios and behaviours, it was possible to find important characteristics for the fitness function and to determine the abstraction of the nodes to decide the BT structure and the baseline.

The second part was the implementation of an evolutionary algorithm. This chapter explains the choice of algorithm, design decisions, and how technical problems were solved, whereas the subchapter 4.2 explains the actual implementation.

Lastly, to evaluate results from the pre-study and the implementation, three experiments were executed. The method of the execution of the experiments are explained in this section and the results are presented in the subchapter 4.3.

3.1 Pre-Study

This section explains the method of the pre-study, whereas the subchapter 4.1 presents the results.

3.1.1 Scenarios and Behaviours

The objectives of an airborne reconnaissance mission were decided upon after a meeting with two operation domain experts, at Saab AB. From these objectives, three scenarios were decided for the simulation environment. From these scenarios important behaviours were identified. The objectives and behaviours are summarised in Appendix A.

By comparing the behaviours already implemented in the simulation environment and the identified behaviours for the scenarios, some of the functionality in the simulation environment were changed and additional functionality was implemented. However, not all behaviours were implemented, due to the time constraint of the thesis.
3.1.2 BT Structure

The first approach was to evolve sub-trees to reduce the simulation time. L. Chong-U et al. (2010) evolved behaviour trees for four individual behaviours and then combined the best performing trees to one tree. Each behaviour had its own fitness function.

However, by identifying the scenarios and behaviours, the sub-trees would have been too small, in some cases only one node. Four sub-tree behaviours were found. The first was to follow waypoint route, second to classify a target, the third was to avoid threat area, engage combat and last to communicate to home base. For example, the sub-tree of follow waypoint route, would have contained only one node. Therefore, the BT:s in this thesis are evolved as one tree and only one fitness function was used.

Decided node types to be used in the BT structure were Selector, Sequence, Parallel, Decorator, Action and Condition. The semantics of the nodes are explained in the section 2.2.1. All non-terminal nodes were decided to be used, to give the algorithm more possibilities to choose from, even if all of them were not used in the baseline BT. The rules of the structure were depending of the chosen algorithm. The connection between them are explained in the grammar, and can be seen in figure 4.11.

3.1.3 Baseline

Kirk Y.W. Schepet et al. (2015), uses a human designed BT as a baseline and compared it to an evolution BT. This was also done in this project. During the design of the baseline BT, the different functionality of the nodes were tested. The node Decorator with the functionality reset-on-tick generated different results in performance. Therefore the behaviour of reset-on-tick were more carefully tested.

Four different baselines were made with different amount and positions of the Decorator nodes. The main idea of this investigation became to determine if it had to be taken into account when designing the grammar. If non-terminals and lonely children always needed to have a Decorator before itself, to get a valid behaviour, then this had to be implemented in the grammar. Four baselines were simulated with scenario 2 while testing their fitness results. The different baselines are found in text form, in appendix C.

The baseline with the best fitness result and the least complexity was chosen to be the BT baseline.

3.1.4 Fitness Function

From the decided behaviours and scenarios, the search for suitable key characteristics started. The found characteristics (factors for the fitness function), were sorted in importance order, normalised and a weight was added to every factor. In this thesis only one fitness function was used. The fitness function was tested with the baseline BT and scenario 3 to balance the weights between the factors.

3.2 Implementation

This subchapter starts with the explanation of the simulation environment. Further, it explains the choice of the algorithm and the architecture and framework. This subchapter ends with a motivation of the techniques used and the evaluation metrics used in the thesis.

3.2.1 Simulation Environment

The simulation environment was provided by the company. In the environment, it is possible to place various units and the units can be both ground entities and aerial vehicles. In addition, the units can be threats or allies. One unit is the agent, whose role is to execute an airborne reconnaissance mission and the behaviour of the agent is determined with BT:s.
The simulation time of the trees depends on the scenario and the control parameters. Since BTs are executed with ticks, the scenarios measure time with ticks. One tick corresponds to 2.5 seconds. Since the simulation time in EAs are time consuming, the baseline was simulated with the different scenarios. This was done to get a maximum period time for the simulations and also to get an idea of how long the simulation time would be.

**Additional Functionality**

Due to the long simulation time and restart of software each week, which resulted in interrupted simulations, a checkpoint was implemented in the simulation environment. The checkpoint saved essential simulation data every generation. When one checkpoint was saved, the previous one was deleted.

The extracted data to be saved was another additional functionality to be implemented. The data to be extracted was important data for debugging, fitness data to compute the fitness function for each individual and individual data for the evaluation metrics. Lastly, as mentioned in the earlier section, functionality such as behaviours were added or changed in the simulation environment.

**3.2.2 Choice of Algorithm**

The problem to be solved was an ML problem and GP was introduced as a suitable algorithm to solve the problem. GE was used as a representation of GP. Since this thesis evaluates robustness, a more general solution was desirable and the algorithm has advantages such as the possibility to model it as a BT and was more general in its solutions.

The evolutionary cycle of GP contains a simulation part, a scenario part, the algorithm and an evaluation of the population each generation that generates the results of the evolutionary process. A simplified overview of the cycle is illustrated in figure 3.1. From the evaluation process when using GE, the tree structure was translated into a sequence of integers. Then the genetic operators were performed on the sequence. The sequence was then translated into the tree structure and used in the simulation.

![Figure 3.1: Illustration of the overview of the evolutionary cycle.](image-url)
3.2.3 Architecture and Framework

Due to the simulation environment, the code was written and executed in Matlab, version R2017b. The structure of the code was based on object-oriented programming (OOP). An overview of the code is illustrated in figure 3.2.

![Figure 3.2: Illustration of the architecture of the algorithm.](image)

The core of the algorithm is the evolution cycle block, programmed by control parameters and the grammar. The control parameters and grammar are external to facilitate the change in the behaviour of the algorithm. The block generates checkpoints, a log-file for debugging and a BT used in the simulation.

3.2.4 Motivation of techniques

In this section, a motivation and explanation of the techniques used in the evolutionary algorithm are presented.

**Initialisation**

Since there is no guarantee of an optimal solution when using EA:s, the distribution of the initial population could be crucial. Therefore, PTC2 was implemented as the initialisation technique. It is supposed to give a better distribution of the trees from the start. Since this technique generates trees (phenotype) and then interpret these to the integer sequence (genotype), it is possible go from phenotype to genotype and contrariwise.

Non-terminal and terminal probability, was not implemented in this version of PTC2. Also, since the nodes did not have input parameters, the amount of children was not stated for the non-terminal nodes. Therefore, if the current created node was not a decorator, a random amount of children was added to the node. The equation 3.1 shows how the maximum amount of children that could be added to a node was computed. The size is the maximum size of the tree and the currentSize is the current size of the tree. This was used in the equation 3.2, with a random number and modulus, to get the amount of children to be added to the nodes.

\[
\text{maxChildren} = \left\lfloor \frac{|\text{size} - \text{currentSize}| + 1}{6} \right\rfloor + 1
\]  

\[
\text{children} = (\text{randomNumber \mod maxChildren}) + 1
\]
To investigate how well the distribution of the initialisation technique was, ten populations were created with 50 individuals in each population. Five of the populations were simulated with scenario 1 and five were simulated with scenario 3. This gave the complexity, tree depth and success rate of the populations. To investigate how the fitness function behaved with the initialisation, a population of 200 individuals was simulated and the interval of the fitness was evaluated.

**Selection Functions**

Selection methods implemented were tournament and elitism. Elitism was implemented so the best individuals would not be lost during selection. Tournament was chosen to include both the individuals with high and less fitness. However, the tournament probability was set to 0.7 to promote individuals with higher fitness.

The tournament and elitism were firstly choosing from the fitness of each individual. However, if two individuals have the same fitness, the less complex tree could be chosen or a random individual of the two could be chosen. Therefore two features were implemented to be tested during the experiments. If the less fitness was chosen in tournament, because of the tournament probability, then the complexity was not counted for.

In the tournament selection and during crossover, the same individual could not be chosen. This was decided, so the algorithm would not generate the same individual twice. This was a common problem before the implementation, because it resulted in a huge amount of similar or the same trees in early generations.

A specific parent selection was not used, instead tournament and elitism was reused.

**Genetic Operators**

Fixed one-point crossover was implemented. Since crossover together with the grammar did not give the exact copy of the nodes from one tree to the next, a validation was also implemented. In the figure 3.3, an example of two trees performing one-point crossover is illustrated.

The nodes from tree $a$ are transferred to tree $b$, and vice versa. These generate the trees $c$ and $d$. However, the last node in the generated tree $c$ is undefined since the sequence of integers ends before it gets defined. After the validation, three integers is added to the sequence and gives the node its behaviour. The last node does not have any children and therefore it is randomly chosen as a condition or action node.
Two different approaches were implemented to solve the problem. The first was to re-read the whole integer sequence. This was not possible since the codons were not randomly chosen during the initialisation. In the beginning of the sequences, there are often several zeros. To go around the problem, the integer sequence was re-read from the crossover point. However it resulted in duplicate sub-trees.

Therefore, a validation process verifying tree rules was implemented. The first rule was that the node could not be a non-terminal with no children. If this occurs, the non-terminal was changed to a random terminal. The second rule was that a decorator could not have
3.3 Evaluation Metrics

more than one child. The solution was to remove all children except the first child. Similar
to the first rule, a terminal can not have children and therefore, the terminal will be changed
into a random non-terminal. The last rule was that a node could not be unspecified. Since
the grammar starts from left, it was possible that the genotype was too short and generated
unspecified nodes. If the node had no children, then it was changed into a random terminal.
If it had children, then it was changed into a random non-terminal.

Mutation was also implemented. The same problem occurred as for crossover, invalid trees.
Therefore the same validation was done on the mutated trees as on the trees generated with
crossover.

Control Parameters

The decided control parameters for the evolutionary process are: population size, termination
(maximum number of generations), mutation probability, crossover probability, tournament
probability, number of tournaments, number of offsprings in tournament, number of offsprings
in elitism, maximum tree depth and maximum tree size.

Since a major issue of EA:s is the difficulty to reduce the simulation time, population size
and maximum number of generations needed to be chosen carefully. The parameters were
changed during the experiments to investigate the effect the parameters had on the result.

In Grammatical evolution, mutation is not often used. If it is used, the mutation probability
is very low. The mutation probability was assigned zero at start in the simulations and then
decided before the three experiments. The number of offspring in each tournament was set to
0.2 of the population size together with number of tournaments as 0.6 of the population size.
The number of offspring in elitism was set to 0.4 of the population size.

The tree depth and size was chosen to approximately fit the baseline. However, the depth
and size were chosen to be slightly bigger. The tree depth of the baseline was eight nodes (nine
with the root node) and the tree size was 44 nodes (45 with the root node). As mentioned in
the chapter 4.1, the maximum tree depth was decided to 13 nodes and the maximum tree size
was decided to 65 nodes. The investigation of suitable values of the other control parameters
are presented in the section 4.2.2 These were chosen before the experiments.

3.3 Evaluation Metrics

The four evaluation metrics described in chapter 2 will be used to evaluate performance and
robustness of the algorithm and to evaluate the complexity of the tree structure. The metrics
were used in the end of each generation and after the termination. The method is further
described in the subchapter 3.4 and the results of the evaluation are presented as figures, in
chapter 4.

3.4 Experiments

The method of the experiments are presented in the following section. By using the evaluation
metrics, an evaluation of the results of the pre-study and implementation was made.

Experiment 1 was simulated in scenario 1, a mission to search a coast line with pre-defined
waypoints. Scenario 1 consists of several waypoints, one home base, one target and no enemies.
Since it was a simple scenario, the main idea with this experiment was to investigate how the
algorithm handles unwanted or not used nodes. Two different features (used in the selection
functions) were tested in this experiment and the most promising feature of the two, was used
in the two following experiments.

In experiment 2, scenario 2 was used. The agent should together with another unit, search
a land area. The other unit searched the west boarder whereas the agent searched the east and
north border. The scenario has waypoints through an area for the agent to search. Scenario
2 consists of several waypoints, one home base, one target and several enemies. The enemies were both static and dynamic.

Experiment 3 was similar to experiment 1, in such way that the scenario was similar but more complex. Experiment 3 was simulated in scenario 3, a mission to search a coast line with pre-defined waypoints and enemies. Scenario 3 consists of several waypoints, one home base, one target and several enemies. The enemies were both static and dynamic, as in scenario 2.

For all three experiments, the baseline BT was first simulated with the scenarios, to generate a baseline to be compare with the result of the evolved BT:s. Since scenario 1 was deterministic, the elite part of the population did not need to go through the simulation again. This saved simulation time. Since the other scenarios included enemies and therefore enabled functionality such as engage combat, the scenarios were not deterministic.

The fitness values of all individuals through the simulations and the success rate of each generation were saved. As were the factors of the fitness function for the final tree in each experiment. The change in complexity and tree depth were also saved to investigate how these changed through generations. After the simulations, the best individual from the simulations was compared to the baseline BT’s results.

If a final BT from experiment 2 or 3 could succeed the mission, then the trees were simulated with the other scenario to evaluate the robustness between the two more complex scenarios.
This chapter presents the results of the values of the control parameters, the distribution of PTC2 and the simulated scenarios. The three different scenarios are explained in the chapter 4.1.

4.1 Pre-Study

This subchapter starts with the scenarios and behaviours used in the thesis. Then it continues with the baseline BT and ends with the fitness function.

4.1.1 Behaviours

The behaviours (actions and conditions) used by the algorithm are explained below. The conditions applied in this thesis were the following:

- Has ammunition
- Has ground unit within classification range
- Has unclassified ground unit
- Sensor active air
- Sensor active ground
- Threat level air high
- Threat level air low
- Threat level air medium
- Threat level ground high
- Threat level ground low
- Threat level ground medium
- Vectoring mode
4.1. Pre-Study

- **Waypoint mode**

  The actions applied in this thesis were the following:

  - Classify ground unit
  - Deviate from threat
  - Engage combat
  - Fly along vector
  - Fly towards closest unit within range
  - Fly towards unclassified unit
  - Fly waypoint route
  - Sensor active air off
  - Sensor active air on
  - Sensor active ground off
  - Sensor active ground on

  Sensors can be used in air or on ground search. Therefore the threat area is divided into air and ground. The threat area is also divided into three threat levels: high, medium and low. More detailed information about these levels can be found in section 4.1.4. A sensor can also be active or passive, therefore these are also divided in the behaviours. If a sensor of the agent is active, it means that the enemy can detect the agent earlier. However a passive sensor can be activated during the whole flight since it does not expose the agent to the enemies. Therefore functionality for turning passive sensor on or off are not implemented.

  The sensors have a detection range and a classification range. This means that the agent can detect units further away and fly towards the unit to classify it. The agent has two air sensors and two ground sensors. The agent has one passive air sensor which can detect and classify airborne units, same for the ground. The passive ground sensor of the agent has a short detection range, whereas the other ground sensor is an active ground sensor with wider detection range but it can not classify ground units. The target to be found in the scenarios are always a ground unit. The last air sensor is an active sensor and it can detect air units and determine the position of it with a high precision.

  The agent can both engage combat or deviate from airborne threats, but only deviate from ground threats. The agent has six missiles during each simulation. More detailed information about the behaviour engage combat can be found in section 4.1.4. Behaviours, such as *Fly along vector* and *Vectoring mode* are not supposed to be used.

  The first behaviour makes the agent fly straight from the start point and the other checks if the agent is in this specific state. These are unnecessary behaviours, but are included, to see how effective the algorithm is, if it will include them or not in the final BT. Communication to home base is done by the simulation program and is not a behaviour in the BT:s.

4.1.2 Scenarios

The three chosen objectives of the scenarios for this project were the following:

1. Find and classify a target, along predetermined waypoint route. The scenario does not include any enemies.

2. Find and classify a target, along predetermined waypoint route. The scenario include enemies, both static and patrolling.
3. Slightly modified scenario of scenario 2

All the scenarios have one ground target to find and classify, and it is marked as an x in the scenarios. All scenarios also have one home base to return to communicate its finding to. The scenarios terminate when the agent reaches the last waypoint or if the period time is reached.

In the first scenario, apart from the already mentioned entities, there are 14 waypoints. The first scenario is also less complex than the other two scenarios, since it does not include any enemies. Figure 4.1 illustrates the first scenario. The left figure shows the agent’s view with the waypoint route and the home base. The right figure shows all components included in the scenario. Agent’s view illustrates the knowledge the agent has before the mission.

Figure 4.1: The figure on the left illustrates the agent’s view in scenario 1. The figure on the right illustrates scenario 1.

The second scenario has eleven waypoints, a home base, two patrolling, airborne threats and six static ground threats. In figure 4.2, both the left and right sub-figures show the agent view. The left sub-figure shows the agent’s view with the waypoint route and home base. The right sub-figure shows the allied airplane’s waypoint route (patrolling the west border). In figure 4.3, the scenario with all its components are shown and the waypoint routes of the enemies in scenario 2 are illustrated in figure 4.4.

Figure 4.2: Illustrates the agent’s view in scenario 2. The figure on the left shows the agent’s waypoint route and the figure on the right illustrates the waypoint of a friendly airborne unit.
4.1. Pre-Study

Figure 4.3: Illustration of the second scenario.

Figure 4.4: Illustrates the waypoint routes of the airborne enemies in scenario 2.

The third and last scenario is illustrated in the figure 4.5. It has 14 waypoints, a home base, two patrolling, airborne threats and seven static ground threats. The left sub-figure shows the agent’s view with the waypoint route and a home base. The right sub-figure shows all components included in the scenario. In figure 4.6, the patrolling route of the two airborne enemies are shown. The enemies have the colour coding red and the agent and its ally have the colour coding blue, in all scenarios.
4.1. Pre-Study

Another difference between the first scenario and the other two scenarios, was that the target was moved further away from the waypoint route. During the simulation of the first scenario, it showed that an early version of the baseline BT could succeed the mission without classifying all units it had found along the waypoint route. Since the target was close to the route, it did not need the functionality to move towards it as an unclassified unit. Instead it was sufficient to follow the waypoint route and the agent became within classification range of the target. By moving the target further away from the waypoint route in scenario 2 and scenario 3, it forced the BT:s to move towards all unclassified units (by executing the functionality of the node: move towards unclassified unit).

During the simulation of scenario 3 and the baseline BT another problem arose. One waypoint was placed too close to a threat. This made it impossible for the BT to fly to the waypoint and still deviate from the threat. Therefore all waypoints are placed outside threat areas in scenario 2 and 3.

4.1.3 Baseline

The results of the investigation of the decorator node type RoT (reset on tick) are summarised in appendix C and briefly described below. This node makes it possible to reset its leaf nodes at each tick, and is effective if one of them has returned running. This makes it possible for the tree to tick the whole sub-tree of the decorator node and not only the node that returned running.
Baseline 1 had no decorator nodes and gave a poor fitness result. It did not classify a target and it did not fire any missiles against the airborne threat. Therefore one decorator was included before the check of threat level air, that could result in engage combat. Another one was included before the check of unclassified units. This made the BT able to engage combat but when it encountered an unclassified target it got stuck for some reason. Baseline 2 gave even worse results than baseline 1.

More decorator nodes were included in the tree structure of baseline 3. By including decorator nodes before almost all the sub-trees in the tree, baseline 3 gave good results. It managed to classify all ground units in range, engage combat and follow the waypoint route. For baseline 4, even more decorator nodes were included. Baseline 4 gave as baseline 3, good results. However, since the complexity will be evaluated in this thesis, baseline 3 were chosen to be the baseline for this project. The baseline is illustrated in the figure 4.7 with eight nodes (nine with the root node) in tree depth and 41 nodes (42 with the root node) in tree size.

The baseline 4 had a better fitness result than baseline 3 since the weapon economy factor in the fitness function gave better result. If the baseline 3 would have been simulated with the scenario again, this could have given a better result of the weapon economy factor and therefore also get a better fitness result than baseline 4. This can happen since the functionality in the simulation generates a randomness to this factor, hence not deterministic.
Figure 4.7: Illustration of the baseline BT.
4.1.4 Fitness Function

From the decided behaviours and scenarios, five key characteristics were found. The characteristics are the following (in order of importance):

1. Find and classify target
2. Follow waypoints
3. Defence potential
4. Weapon economy
5. Time efficiency

These five characteristics are five different factors of the fitness function. To generate this order of importance in the fitness function, the factors were normalised and multiplied by a weight. This gave the fitness function a minimum and maximum value and therefore an interval for the function. The fitness interval is summarised in the table 4.1. The factors and weights of the fitness function is explained in the following subsections.

<table>
<thead>
<tr>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum fitness</td>
</tr>
<tr>
<td>Maximum fitness</td>
</tr>
</tbody>
</table>

The first scenario does not include enemies, whereas the other two do. The first scenario will therefore not collect any data of weapon economy and defence potentials, but focus on waypoint route, classification and time efficiency. Since the data is not collected, the factors of weapon economy and defence potentials will return the value of zero to the fitness function. If the scenario includes enemies the defence potential and the weapon economy are computed.

Classification of target

The most important part of an airborne reconnaissance mission is the classification of units, especially a target. There are two different types of sensors, one is active and one is passive. The difference between them is that the enemy can see you when using an active one, but not when using a passive one. Since the airborne enemy units get classified by a sensor that is not active, it is not important to include this in the fitness function. However, when detecting ground units far away, an active sensor was used.

Classification of ground units that is not a target is not included in the fitness function. If the agent changes the route, then it should not be punished for not classifying ground units. However, to miss the target that it is supposed to classify is devastating. Therefore, classification of the target is an important part of the fitness function and sets the weight if target found, otherwise zero. The weight $w_{target}$ was set to 900 and the equation 4.1 shows the computation of this part.

$$f_{target} = target \cdot w_{target}$$  \hspace{1cm} (4.1)

Follow waypoint route

Another part of an airborne reconnaissance mission is to follow a waypoint route. To understand how well the agent followed pre-determined waypoints, the shortest distance between the agent and the current waypoint was saved. A summation was made of all the distances and an average of the distances was computed. A waypoint radius $r_{waypoint}$ of all the waypoints
were predefined in the scenario as 5000 meters. The difference between the average of the summation and the radius of the waypoints was computed, see equation 4.2.

\[ f_{\text{waypoint\_average}} = \frac{\text{distance}}{\text{waypoints}} - r_{\text{waypoint}} \] (4.2)

The difference was then used in equation 4.3. The difference was compared to a threshold \( t_{\text{waypoint}} \), that was the same as the waypoint radius (5000 meters). The weight \( w_{\text{waypoint}} \) was set to 700. If the difference is greater than the threshold, then the waypoint factor becomes zero.

\[ f_{\text{waypoint}} = \frac{t_{\text{waypoint}} - \min(f_{\text{waypoint\_average}}, t_{\text{waypoint}})}{t_{\text{waypoint}}} \cdot w_{\text{waypoint}} \] (4.3)

4.1.5 Defence potential

To compute the defence potential, the amount of ticks that the agent was in threat area was saved. The threat area was divided into three threat levels: high, medium and low. The threat levels were filtered on distances, between the agent and the closest threat. The levels are illustrated in figure 4.8.

Figure 4.8: Illustration of the three levels of threat. The yellow circle symbolises high threat level, the orange medium threat level and last, the red symbolises low threat level. The levels are scaled, but not together with the airplane. The image of the airplane is taken from Saab AB.

The levels were applied in the defence potential, as three different weights. If the agent’s position was within the low threat area, then the weight \( w_{\text{low\_threat}} \) was applied and the weight was set to 500. If the agent’s position was within the medium threat area, then the weight \( w_{\text{medium\_threat}} \) was set to 1000. Lastly, if the agent’s position was within the high threat area, then the weight \( w_{\text{high\_threat}} \) was set to 1500. This leads to higher penalty, the closer the agent was to a threat. The defence potential is the only part of the fitness function which gives a negative result.
4.2 Implementation

The weights were multiplied with the amount of ticks the agent was within a specific threat level divided with the total ticks the agent was within threat area. See equation 4.4

\[ f_{\text{threat}} = -\left( (\frac{t_{\text{high\_threat}}}{t_{\text{threat}}} \cdot w_{\text{high\_threat}}) + (\frac{t_{\text{medium\_threat}}}{t_{\text{threat}}} \cdot w_{\text{medium\_threat}}) + (\frac{t_{\text{low\_threat}}}{t_{\text{threat}}} \cdot w_{\text{low\_threat}}) \right) \] (4.4)

**Weapon economy**

Weapon economy can limit the time (ticks) the agent is within threat area. If the agent fires a missile against an enemy and it gets hit, then the agent is no longer in threat area.

To get a higher probability of a hit, active air sensor should be active. Otherwise the precision of the enemy’s position would become less accurate, which results in less \( p_{\text{kill}} \) (probability to kill the enemy when fired missile). Distance is also important, the closer the agent is to its enemy, the higher \( p_{\text{kill}} \). Therefore weapon economy counts the number of hits divided by missiles fired. The weight \( w_{\text{weapon}} \) was set to 400. Weapon economy is described in the equation 4.5

\[ f_{\text{weapon}} = \frac{\text{hits}}{\text{missiles\_fired}} \cdot w_{\text{weapon}} \] (4.5)

**Time efficiency**

The termination of the scenarios happens when the agent has reached the last waypoint or the period of ticks is reached. Therefore, the amount of ticks is included in the fitness function. Although it can happen that the agent follows the waypoint route within less ticks, but then misses to classify units or deviate for threats. It can also happen that the agent takes a longer route than necessary. Therefore this is a tradeoff between the other parameters in the fitness function.

The equation of time efficiency is based on the period time, see equation 4.6. The time period is 13 minutes and 20 seconds (2000 ticks), see table 4.2. The difference between the elapsed time from the simulation and the period time is divided with the period time. Since the elapsed time is close to the period time, the weight \( w_{\text{time}} \) was set to 1000 to get a suitable value of the time efficiency.

\[ f_{\text{time}} = \frac{t_{\text{period}} - t_{\text{elapsed}}}{t_{\text{period}}} \cdot w_{\text{time}} \] (4.6)

**Fitness function**

The fitness function is presented in the equation 4.7 and is based on the equations 4.1, 4.3, 4.4, 4.5 and 4.6.

\[ \text{fitness} = f_{\text{target}} + f_{\text{waypoint}} + f_{\text{threat}} + f_{\text{weapon}} + f_{\text{time}} \] (4.7)

4.2 Implementation

The chapter starts with a short explanation of the simulation environment and the data extracted from it. Followed by a short information about the architecture and framework, and ends with a more detailed presentation of the algorithm.
4.2.1 Simulation Environment

The simulation time is summarised in the table 4.2 and is represented in seconds. The stop criteria for each individual in the simulation environment, for all the scenarios are one period. One period is 2000 ticks and was decided after the simulation of the baseline in all scenarios. This made it possible to stop a simulation that would not ever finish.

Table 4.2: Simulation time during simulation with baseline and decided period time of each scenario.

<table>
<thead>
<tr>
<th>Scenario number</th>
<th>Baseline</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7min 1s</td>
<td>13min 20s</td>
</tr>
<tr>
<td>2</td>
<td>9min 10s</td>
<td>13min 20s</td>
</tr>
<tr>
<td>3</td>
<td>11min</td>
<td>13min 20s</td>
</tr>
</tbody>
</table>

Figure 4.9 shows an example of how the baseline is executed during one tick. The tick travels from the left to the right and from the top (root) to bottom. The colour coding: blue is not ticked, red is ticked but the node returns failure, yellow is ticked and returns running and green is ticked and returns success. The figure shows that the tick travels first to the condition nodes checking the threat level in air. Then it continues to check the threat level on ground. The condition node has unclassified ground unit is green and returns success. This leads to that the other child node to the sequence node is ticked. However, since the unit is not in classification range, the behaviour becomes fly towards unclassified unit. It performs this action the rest of the tick since the node returns running.
Figure 4.9: Illustration of how a tick travels through the baseline BT. The resulted action node gave the agent the behaviour of *Fly towards unclassified unit*.
Additional Functionality

The additional behaviours implemented in the simulation environment are mentioned in sub-
chapter 4.1. In addition, a checkpoint was implemented and data was extracted from the
simulation environment.

The data classes saved in the checkpoint were ga_evolution, bnf_grammar and control_parameters. The classes are explained in section 4.2.2.

The extracted data, presented in the table 4.3 made it possible to debug the simulation
and the evaluation, since it showed how the agent moved during the simulation. It also showed
the scenario properties and the other actions of the agent. This data were saved in a class
called graph_simulation and was a data structure in each individual.

Table 4.3: The extracted data during simulation for testing and plotting.

<table>
<thead>
<tr>
<th>Extracted graph data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waypoint positions</td>
</tr>
<tr>
<td>Threat position</td>
</tr>
<tr>
<td>Target position</td>
</tr>
<tr>
<td>Agent position</td>
</tr>
<tr>
<td>Hit position</td>
</tr>
</tbody>
</table>

The extracted data, presented in the table 4.4, was used to compute the fitness function.
The data was saved in a class called data_simulation and was also a data structure in each
individual.

Table 4.4: The extracted data during simulation containing data used in the fitness function.

<table>
<thead>
<tr>
<th>Extracted fitness data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waypoint distance</td>
</tr>
<tr>
<td>Threat area</td>
</tr>
<tr>
<td>Threat count</td>
</tr>
<tr>
<td>Classified target</td>
</tr>
<tr>
<td>Ticks</td>
</tr>
<tr>
<td>Hits</td>
</tr>
<tr>
<td>Missiles launched</td>
</tr>
</tbody>
</table>

The computed fitness values were saved in a class called result_simulation and was a data
structure in the evolution object. In this class, the complexity and tree depth were saved. The
class was updated after each generation. This gave the best and mean fitness, best and mean
complexity and lastly the best and mean tree depth of the population. Also, for debugging
purposes, all the individuals’ fitness, complexity and three depth results were saved. This
extracted data is presented in the table 4.5.
Table 4.5: The extracted data during simulation containing the result.

<table>
<thead>
<tr>
<th>Extracted result data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Best and mean fitness</td>
<td>The mean and best fitness through all generations.</td>
</tr>
<tr>
<td>Best and mean complexity</td>
<td>The mean and best complexity through all generations.</td>
</tr>
<tr>
<td>Best and mean tree depth</td>
<td>The mean and best tree depth through all generations.</td>
</tr>
<tr>
<td>Fitness, complexity and tree depth</td>
<td>All the individuals’ fitness, complexity and tree depth.</td>
</tr>
<tr>
<td>Target</td>
<td>Amount of targets found each generation.</td>
</tr>
<tr>
<td>Individuals</td>
<td>Individuals with best fitness (first, every fifth and last generation).</td>
</tr>
</tbody>
</table>

4.2.2 Algorithm

Detailed information of the different classes and functions of the evolution process are explained in the following sections. In figure 4.10, the main loop of the implementation is illustrated.
It starts with initialisation of control parameters, parsing the grammar and initialising the evolution cycle with this knowledge. Continuing with the initialisation of the population. To get the initial fitness values of each individual, simulation was run and fitness was computed.

From here, the evolution process started. Parent selection was executed and tournament was used with crossover and mutation. When performing the genetic operators, the genotype was used. To prevent the loss of fit individuals, elitism forwards a fraction of the best individuals to the next generation. Then simulation and fitness evaluation is again performed and this process continues until termination. Classes connected to the evolution cycle and initialisation are the following:

- ga_evolution
- phenotype_node
- ge_individual
- bnf_grammar
- control_parameters

Classes connected to the representation of result are the following (explained in section 4.2.1):

- graph_simulation
- data_simulation
- result_simulation

Grammar

The grammar was saved in a txt-file and was parsed in to the evolution cycle. The start symbol is noted < BT > and the grammar is recursive. The grammar contains four different non-terminal nodes: Selector, Sequence, Parallel and Decorator. The Decorator has four behaviours: RoT, RoS, RoF and not. From the pre-study, when the behaviours were chosen, the terminals could be derived. It contains two different types of terminal nodes, namely Condition and Action. Condition, has 13 types and Action has 12 types. The BNF grammar is presented in figure 4.11.

A clarification concerning the non-terminal nodes. These are not the same as the non-terminals in the tuple of BNF grammar. The non-terminal (N) in the grammar includes all non-terminal symbols, whereas the non-terminal nodes only include the possible nodes in the behaviour tree.
To investigate the distribution of PTC2, ten initialised populations of 50 individuals each, and one population of 200 individuals were simulated. By the investigation, a sense of the complexity, tree depth and fitness distribution were given.

The figure 4.12 illustrates a tree structure generated with the PTC2 technique. Important parameters for the initialisation was maximum tree depth and maximum tree size. The maximum tree depth was set to 13 and the maximum tree size was set to 65 nodes during all simulations. The tree structure in figure 4.12 has tree depth six (seven with root node) and complexity 68 nodes (69 with root node).
The result of the distribution of PTC2 is described by evaluating the distribution of complexity and tree depth of ten initialised populations. All the trees except one had slightly bigger complexity than the decided maximum tree size. The presentation of the distribution of PTC2 can be found in appendix E. The pie-charts, figure D.1-D.10 shows that the range of complexity was 66-75 nodes. One tree had a complexity of 58 nodes. For each initialisation, the bigger part of the population had a complexity of 66-70 nodes. This means that it is possible to get a lower complexity than 66 nodes but it is very rare.

While the complexity did not differ much in the initialisation, the tree depth did. Therefore, it was the tree depth that introduced most variation of trees in the initialisation technique. The range of tree depth was 10-29 in the pie-charts, figure D.1-D.10. Even for each population the tree depth differed, however most of the trees had a tree depth of 12-19.

The first five populations were simulated with scenario 1. Three of the five populations had a success rate of 2%, 2% and 6% respectively, otherwise 0%. All the populations simulated with scenario 2 gave 0% in success rate.

In addition to these ten initialised populations, another initialisation of 200 individuals was made with scenario 2. It resulted in 13 different fitness values between -723 to 542, 0% in success rate, see figure 4.13. This was made to get an idea of the distribution of fitness values on a larger initialisation of 200 individuals. There were two peaks, one at fitness 0 and one at fitness 495.
4.2. Implementation

Figure 4.13: Illustration of a distribution of the fitness on a larger initialisation of 200 individuals.

Values of Control Parameters

When simulating to find suitable values for the control parameters, the feature where the algorithm promoted a lower complexity when equal fitness value were compared was used. By simulating scenario 1 several times with the initial control parameters, the algorithm resulted in a local maximum. The best and mean fitness became the same during early generations. The mutation was implemented to make sure new nodes could be added. However, it was only changed to 0.1 and crossover to 0.9. The percentage of elitism was changed from 40% to 20% of the population to give a higher amount of diversity between the trees. This lead to an increase of number of tournament, from 60% to 80% of the population.

The algorithm became to evolve trees with high fitness value in the first generation and increased the percentage of successful individuals throughout the generations. One problem was still that several offsprings were similar to each other after tournament. To solve this issue, the percentage of number of offspring in tournament was set to 2 individuals, independent of population size. This means that at each tournament, the one with higher fitness value had 70% chance to be chosen over the one with lower fitness value. Since the comparison was made between two individuals instead of several, the best or worst individual chosen were more often not the same.

The population size and maximum number of generations differed during the experiments. It was possible to see that the population size had more effect on the algorithm then the number of generations. Decided for the initial simulations in the experiments were a population size of 50 individuals and 20 generations. Since these changed, in some simulations, these parameters are presented under each experiment section. As for the features, such as promotion of lower complexity.

The values of the control parameters for the evolutionary process are declared in the table...
4.3 Experiments

This section presents the results of the executed experiments. All experiments start with a simulation of the baseline, continuing with the evolving individuals.

4.3.1 Baselines

This section presents the results of the baseline in scenario 1, 2 and 3. The executed behaviour of the agent during the simulations are represented with graphs. The fitness results are summarised with both the fitness factors and fitness value. Fitness and the intermediate results can differ due to rounding. Since the functionality of weapon economy is not deterministic, this factor can vary between simulations.

Scenario 1

The agent traveled along the waypoint route and classified the target within the total time of 1054 ticks. The graphical result of the behaviour and the fitness value are illustrated in figure 4.14.
4.3. Experiments

Scenario 2

The agent traveled along the waypoint route, classified the target and six threats, and shot down one airborne enemy within the total time of 1376 ticks. The graphical result of the behaviour and the fitness value are illustrated in figure 4.15.
Scenario 3

The agent traveled along the waypoint route, classified the target and six threats, and shot down two airborne enemies within the total time of 1651 ticks. The graphical result of the behaviour and the fitness value are illustrated in figure 4.16.

![Baseline, scenario 3](image)

Figure 4.16: Illustration of baseline’s graphical result and fitness value for scenario 3.

### 4.3.2 Experiment 1

This subsection present the results of the evolved BT:s simulated in scenario 1. The experiment includes four simulations. The additional control parameters (population size and termination) for the simulations are summarised in table 4.7. In this table different features are also included.

<table>
<thead>
<tr>
<th>Simulation number</th>
<th>Population Size</th>
<th>Termination (maximum number of generations)</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>20</td>
<td>Algorithm promoted a lower complexity when equal fitness value.</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
</tbody>
</table>
Simulation 1

In simulation 1, the algorithm promoted a lower complexity when equal fitness value, see table 4.3. The feature forced the algorithm to decrease both the complexity and the tree depth of the population drastically. The complexity and tree depth found their minimum value after the 14th generation. Figures 4.18 and 4.19 illustrates the complexity and tree depth through the generations for simulation 1.

The algorithm got stuck in a local maximum, see figure 4.17. The fitness value of the baseline was 2048, and the best fitness value of the evolved trees was 1148. The algorithm evolved a final BT that could navigate along a pre-defined waypoint route but could not classify units.

Figure 4.17: The plot illustrates the fitness of the trees through the generations for simulation 1, scenario 1. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.
4.3. Experiments

Figure 4.18: The plot illustrates the change in complexity through the generations for simulation 1, scenario 1. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.

Figure 4.19: The plot illustrates the change in tree depth through the generations for simulation 1, scenario 1. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 2

The algorithm in the previous simulation promoted a lower complexity when comparing two individuals that had equal fitness value. For simulation 2, this feature was replaced with a feature that chose a random individual instead. By choosing random individuals instead of
4.3. Experiments

the individual with the lowest complexity when equal fitness value, the complexity and tree depth did not decrease as in simulation 1. The mean complexity decreased until generation 12, then it started to increase again, see figure 4.21. The mean complexity was throughout the generations over baseline complexity value, whereas the best tree complexity was below the baseline complexity three times. At generation 17, a local minimum was found. This minimum corresponds to a minimum in the tree depth.

In figure 4.22, the presentation of the tree depth throughout the generations is illustrated. The mean tree depth decreased fast to baseline depth and then was steady around the baseline throughout the generations.

The algorithm evolved a final BT similar to the one in simulation 1, which could navigate along a pre-defined waypoint route but could not classify units. This means that the algorithm once again got stuck in a local maximum at a fitness value of 1148. See figure 4.20.

![Fitness over generations](image)

Figure 4.20: The plot illustrates the fitness of the trees through the generations for simulation 2, scenario 1. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.
4.3. Experiments

Figure 4.21: The plot illustrates the change in complexity through the generations for simulation 2, scenario 1. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.

Figure 4.22: The plot illustrates the change in tree depth through the generations for simulation 2, scenario 1. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 3

The population size in simulation 3 was increased to 70 individuals. This however, did not prevent the algorithm to get stuck in the same local maximum as previous simulations. The fitness over the generations are presented in figure 4.23.
The results of complexity and tree depth were similar to the results in simulation 2. A noticeable difference was a lower mean complexity and mean tree depth. The mean complexity decreased under baseline complexity after six generations and continued below throughout the generations, see figure 4.24. The mean tree depth decreased under baseline tree depth after three generations and continued below throughout the generations, see figure 4.25. The best tree depth was mostly under the baseline tree depth, part from tree peaks at generations 8-9, 15 and 20.

![Fitness over generations](image)

Figure 4.23: The plot illustrates the fitness of the trees through the generations for simulation 3, scenario 1. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.
4.3. Experiments

Figure 4.24: The plot illustrates the change in complexity through the generations for simulation 3, scenario 1. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.

Figure 4.25: The plot illustrates the change in tree depth through the generations for simulation 3, scenario 1. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 4

By increasing the individuals to 70, it did not prevent local maximum but increased the simulation time, therefore simulation 4 was made with 50 individuals. The PTC2 initialised a population where one individual could navigate along a pre-defined waypoint route and classify
units. This means that the best fitness was the same as the baseline from generation 1 until termination (the behaviour of the individual was saved as an elite). Continuing throughout the generations the individuals which could classify units increased. The fitness and success rate over the generations are presented in figures 4.26 and 4.27.

The success rate increased until generation 14, to 20%, the same amount of percentage of elits of the population. At generation 16 it reaches a maximum of 28% and then it decreases, however it stays over 20%.

Since the selected feature did not promote a lower complexity when equal fitness value, the complexity and thee depth varied throughout the generations. See figure 4.28 and 4.30.

At generation 10 and 12, both the complexity and tree depth had found a tree with minimal amount of nodes. The trees had only three nodes, see figure 4.29.

In the last generation there were ten individuals with the highest fitness (2048). Because of the chosen feature, an individual with fitness 2048, tree depth 8 and complexity 70 was chosen by the algorithm as the final best tree in the last generation. However, since the thesis emphasis on complexity and robustness, the individual with highest fitness, lowest complexity and tree depth was chosen as the best final individual. The BT corresponding to this individual was chosen as the best final BT of scenario 1.

Figure 4.26: The plot illustrates the fitness of the trees through the generations for simulation 4, scenario 1. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.
4.3. Experiments

Figure 4.27: Illustrates the success rate of a mission (found target) per generation for simulation 4, scenario 1.

Figure 4.28: The plot illustrates the change in complexity through the generations for simulation 4, scenario 1. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.
4.3. Experiments

Figure 4.29: The plot illustrates the change in tree depth through the generations for simulation 4, scenario 1. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Figure 4.30: The plot illustrates the change in complexity through the generations for simulation 4, scenario 1. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Best final BT - Scenario 1

The best final individual from the four simulations were found in simulation 4 with fitness 2048, tree depth 6 and complexity 30. The agent traveled along the waypoint route and classified the target. The graphical result of the behaviour is illustrated in figure 4.31. The BT of the final individual is illustrated in figure 4.32 and the results from the fitness function is summarised in table 4.8.

The evolved BT in figure 4.32 does only tick the first sub-tree. Therefore the nodes not included in this sub-tree are of minor importance. The ticked sub-tree includes key nodes such
as *Continue waypoint route* and *Classify ground unit* but also several nodes not important for the scenario. However, the complexity and tree depth of the final evolved BT are less than the baseline.

![Figure 4.31: Illustration of the graphical result of the best evolved tree of the last generation for scenario 1.](image)

![Figure 4.32: Illustrates the BT of the best evolved individual of the last generation. The BT has tree depth 6 (7 with root node) and the complexity was 30 nodes (31 with root node).](image)
4.3. Experiments

Table 4.8: Result of the best evolved BT from Scenario 1.

<table>
<thead>
<tr>
<th>Best evolved BT from scenario 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Follow waypoint route</td>
<td>674</td>
</tr>
<tr>
<td>Classification of target</td>
<td>900</td>
</tr>
<tr>
<td>Time efficiency</td>
<td>474</td>
</tr>
<tr>
<td>Fitness</td>
<td>2048</td>
</tr>
</tbody>
</table>

4.3.3 Experiment 2

This subsection presents the results of the evolved BT:s simulated in scenario 2. The experiment includes three simulations. The additional control parameters (population size and termination) for the simulations are summarised in table 4.9. The most promising feature from experiment 1 was chosen for the whole experiment.

Table 4.9: Additional control parameters for Scenario 2.

<table>
<thead>
<tr>
<th>Simulation number</th>
<th>Population Size</th>
<th>Termination (maximum number of generations)</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>20</td>
<td>Algorithm chose a random individual when equal fitness value.</td>
</tr>
</tbody>
</table>

Simulation 1

The first simulation of scenario 2 got stuck at a local maximum with fitness value 495, see figure 4.33. The algorithm evolved a final BT that could navigate along a pre-defined waypoint route but could not classify units, deviate from threats or shoot at airborne enemies.

The complexity throughout the generations behaved similar to the complexity in experiment 1, simulation 2. The tree depth decreased until 7, then it slowly increased again and stayed over the baseline tree depth. However, the best tree depth had two valleys under the baseline, at generation 6 and 20. Since the algorithm chose a random individual when equal fitness, the best complexity and tree depth varied. See figure 4.34 and figure 4.35.
4.3. Experiments

Figure 4.33: The plot illustrates the fitness of the trees through the generations for simulation 1, scenario 2. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.

Figure 4.34: The plot illustrates the change in complexity through the generations for simulation 1, scenario 2. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.
4.3. Experiments

Figure 4.35: The plot illustrates the change in tree depth through the generations for simulation 1, scenario 2. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 2

By increasing the population size, the mean fitness seems to increase slowly but steady to the local maximum, see figure 4.36. This local maximum was the same as in the previous simulation.

The complexity and tree depth did not change much even with the change in population size, see figure 4.37 and 4.38. The graphs suggested a more steady mean complexity and mean tree depth, compared to the previous simulation.
4.3. Experiments

Figure 4.36: The plot illustrates the fitness of the trees through the generations for simulation 2, scenario 2. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.

Figure 4.37: The plot illustrates the change in complexity through the generations for simulation 2, scenario 2. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.
4.3. Experiments

Figure 4.38: The plot illustrates the change in tree depth through the generations for simulation 2, scenario 2. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 3

The last simulation ended up in a local maximum, the same as simulation 1 and 2. No new observations were made on fitness, complexity or tree depth, see figures 4.39, 4.40 and 4.41.

Figure 4.39: The plot illustrates the fitness of the trees through the generations for simulation 3, scenario 2. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.
4.3. Experiments

Figure 4.40: The plot illustrates the change in complexity through the generations for simulation 3, scenario 2. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.

Figure 4.41: The plot illustrates the change in tree depth through the generations for simulation 3, scenario 2. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

**Cause of local maximum**

All simulations with scenario 2 got stuck in a local maximum. By evaluating the tree structure of several trees from the last generation in simulation 3, there were certain reasons for this. The most dominating reason was that the trees only executed behaviours from the first
subtree. This behaviour was expected since this was noticed when studying the final BT from experiment 1.

One tree structure generating this problem is illustrated in figure 4.42. The problem was the positioning of a selector node, followed by a parallel node. Since the parallel node returns running from its children, the tick will never continue to the next sub-tree. A selector node needs failure to let the tick proceed to the next node.

Figure 4.42: Illustration of an evolved tree structure with only first sub-tree ticked.

The selector node itself could generate problems. If the first child node performed an action that returned true if executed, then the rest of the tree did not tick. Another problem was when the node \textit{Sensor active ground off} was placed in the beginning of the BT and therefore made it impossible for the behaviour to classify ground units. See figure 4.43, illustrating both problems.

Figure 4.43: Illustration of an evolved tree structure with two noticeable problems. The first problem is the selector node, it makes the tick not traverse more than to the first child node of the selector. Another problem with this tree is the positioning of the two nodes \textit{Sensor active ground off} and \textit{Classify ground unit}.

When evaluating the remaining trees, several trees lacked the two key nodes, \textit{Fly towards unclassified unit} followed by \textit{Classify ground unit}. Even though the trees had either one of
the nodes, it did not suffice, since the target in scenario 2 was moved further away from the waypoint route it had to include both in right order. The amount of individuals in the populations and the amount of generations were not enough to evolve this behaviour.

Another disclosure was the poor variation of fitness values. When choosing the elites at each generation, the fitness values were sorted. While sorted, it was seen that the individuals received similar values and the distribution was poor. At generation 20, all the trees had the same fitness value except one. This was a good result since the population tried to find the optimal solution (which it thought was the local maximum).

The drawback however was the poor distribution in the earlier generations. In the first generation there were only four different fitness values distributed over 50 individuals. At generation 12, these decreased to three different and at generation 15 there were two. The total of five different values.

4.3.4 Experiment 3

This subsection presents the results of the evolved BT:s simulated in scenario 3. The experiment includes two simulations. The additional control parameters (population size and termination) for the simulations are summarised in table 4.10. The same feature used in experiment 2 was chosen for the whole experiment. An additional feature for the last simulation was to include the baseline BT in the initial generation.

Table 4.10: Additional control parameters for Scenario 3.

<table>
<thead>
<tr>
<th>Simulation number</th>
<th>Population Size</th>
<th>Termination (maximum number of generations)</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Simulation 1

Due to the initialisation technique, hopefully the simulation would have found a desirable solution. The first simulation of scenario 3 gave an even worse result than the simulations with scenario 2. The algorithm got stuck at a local maximum of 0, see figure 4.44 and the complexity and tree depth behaved as previous simulations. For the complexity and tree depth throughout the generations, see figures 4.45 and 4.46. This behaviour was expected since the scenario 3 resembles scenario 2.
4.3. Experiments

Figure 4.44: The plot illustrates the fitness of the trees through the generations for simulation 1, scenario 3. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.

Figure 4.45: The plot illustrates the change in complexity through the generations for simulation 1, scenario 3. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.
4.3. Experiments

Figure 4.46: The plot illustrates the change in tree depth through the generations for simulation 1, scenario 3. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.

Simulation 2

Due to the common problem with local maximum for the algorithm in experiment 1 and 2, the baseline BT was included in the simulation to try to mimic the behaviour of simulation 4 in experiment 1. Since the population and termination could not increase much more due to time restraints of the thesis, including the baseline BT in the initialisation hopefully would force a better population at termination.

The baseline was included in the initial population for this simulation. In experiment 1, simulation 4, it was possible to see that the algorithm worked when the initialisation generated a tree that could classify ground units. The same behaviour occurred in this simulation. In figure 4.47, the best fitness found the desired solution after five generations. The mean fitness continues slowly to increase until termination (except from few expected valleys).

The success rate started at 2% of the population (one tree succeeded the mission, hence the baseline BT). At the sixth generation, all elites had succeeded to classify the target and the maximum success rate was 34% of the population at generation 13, 15 and 16.

The lowest complexity and tree depth were the same as the baseline BT throughout the generations, except of a peak at generation 17. See figures 4.49 and 4.50. This meant that the baseline BT survived throughout the generations and became the best tree in the last generation, in simulation 2. It had the highest fitness value, lowest complexity and lowest tree depth of the population. The fitness shows a higher value than the baseline. This is because the scenario is not deterministic due to the weapon economy in the fitness function. There were also evolved trees with complexity of 43 and 45 nodes, however these were very similar to the baseline BT. The results from experiment 2 and 3 resulted in local maximum or the baseline BT as the final BT, thus it was impossible to evaluate robustness between scenario 2 and 3.
Figure 4.47: The plot illustrates the fitness of the trees through the generations for simulation 2, scenario 3. It shows both the mean fitness of the population and the best fitness of each generation. The fitness of the baseline is the horizontal line.

Figure 4.48: Illustrates the success rate of a missions (found target) per generation for simulation 2, scenario 3.
4.3. Experiments

Figure 4.49: The plot illustrates the change in complexity through the generations for simulation 2, scenario 3. It shows both the mean complexity of the population and the complexity of the best tree of each generation. The complexity of the baseline is the horizontal line.

Figure 4.50: The plot illustrates the change in tree depth through the generations for simulation 2, scenario 3. It shows both the mean tree depth of the population and the tree depth of the best tree of each generation. The tree depth of the baseline is the horizontal line.
5 Discussion

This chapter explains the significance of the results in context of the thesis, other work and wider literature on the topic. It starts with a discussion of the experiments and implementation. It then continues with a discussion of the methodology of the project. The chapter ends with a discussion of the work in a wider context.

5.1 Results

The results of this section are based on the findings in the literature study and from the results of the experiments.

5.1.1 Experiments

Experiment 1 was able to find a solution that did not get stuck in a local maximum. The algorithm evolved a BT with complexity 30 and tree depth 6, see figure 4.32 for the illustration of the BT structure. The agent traveled along the waypoint route and classified the target. After 14 generations the success rate was at 20% (the same amount of percentage of elites of the population).

The final evolved BT included also unnecessary nodes such as Vectoring mode? and Fly along vector. The removal of these nodes could be difficult for the algorithm since these are in between other nodes. Probably an optimisation technique or another algorithm could be used for optimising the behaviour. A solution could be to remove nodes not ticked or when removed, the behaviour does not change. This would also lower the complexity.

In experiment 2, all the simulations got stuck in a local maximum. There were certain reasons for this to occur. By investigating the tree structure, it was found that only parts of the tree were ticked. A significant concern is the highest-priority behaviour of a BT. The tick in the tree is traversed from top to bottom and from left to right. The grammar with BT:s was investigated briefly, the experiments proves a more thorough investigation is needed. It suggests that the grammar needs more restrictions when used together with BT:s. The selector node could force the behaviour to only tick one node or specific node sequences made the tick stop in the first sub-tree as mentioned before.

Another problem with the execution of the first sub-tree is that the other part of the tree never gets evaluated by the fitness function. Behaviour not wanted or needed for the scenario
can stay in the population throughout the generations. Lastly, the genetic operators change the right side of the crossover or mutation point, thus this change could be ineffective.

Another reason was the difficulties of the algorithm to find better fitness values. This was a result of the few different fitness values. When evaluating PTC2 with the fitness function, it was proved that only 13 different fitness values between -723 to 524 was found in a population of 200 individuals. This was problematic and the function should use a more soft function. To achieve this, an idea is to identify key-sequences of nodes. However, it is a difficult balance, since a fitness function should not manipulate the trees into a specific pattern, only guide it. The graphs illustrating the mean best fitness of the simulations suggest that the behaviour of the algorithm is correct when comparing to the literature. For example, in figure 4.26 in experiment 1, mean fitness slowly increases to the best fitness value, hence the algorithm finds key characteristics in the tree structure.

The experiment 3 included two simulations. The first simulation got stuck in a local maximum. Since the second and third scenario were similar, it was concluded that the remaining simulation would try to mimic the last simulation of experiment 1. This was succeeded by including the baseline BT in the first generation. The baseline BT survived throughout the generations and was found as the best BT in the final generation. All elites were similar to the baseline, thus the algorithm had difficulties to find new solutions.

Due to the time restraint of the project, a small population and low termination were used. Since the initialisation technique gave good distribution, the algorithm could still find a solution. To get a more just evaluation of the algorithm longer simulations needs to be done. A larger population seems to be more vital than more generations.

Because of the high abstraction of the nodes, overfitting has not been taken into account. For example, the whole functionality for the action, to fly a waypoint route, is in one node. Another factor was the shape of the scenarios. The trees were not evolved against a single opponent. Fighting against a single opponent could cause over-fitting.

5.1.2 Evaluation

The most used metrics in the experiments were MBF, complexity, tree depth. These metrics were used to calculate the individuals’ fitness value and their complexity and tree depth of their corresponding BT. The best complexity and best tree depth belonged to the individual with the best fitness.

It was expected that the complexity of the scenario would affect the complexity and tree depth. In experiment 1, the complexity and tree depth was less than the baseline. Only the first sub-tree were executed, hence the other part of the tree could easily been removed without changing performance. By combining the algorithm with a pruning technique, the result of complexity and tree depth could become better. K. Y.W. Scheper et. al (2015) optimised their BT after 150 generations by pruning the final BT (best individual tree). This resulted in a less complex tree with eight nodes instead of 32 nodes. The technique showed that it was as effective, and even better than the user-designed BT.

In the experiments that produced a solution to the problem the SR was presented. It was used to evaluate how the algorithm handled the found solutions. The algorithm did save as many found solutions as the amount of elites. This made it possible to see that found solutions did not get lost but still there were space for the algorithm to investigate other solutions.

Neither of the experiments 2 and 3 gave a desired solution (succeed with the presented mission), thus the performance of robustness could not be evaluated. Otherwise the SR would have been used to calculate the succeeded missions of each final BT. Even though (Kirk Y.W. Scheper et al.) managed to get the success rate 88% in simulation, they still only got 54% in real-world performance in their experiments. Therefore, it is important to keep in mind that simulation results are not the same as real world results.
5.2 Method

This section discusses and criticises the method of the thesis.

5.2.1 BT Structure

EAs are known for its heavy simulation time, therefore, this was taken into account in all decisions made during the pre-study. The initial approach of sub-trees was one example of this kind of decisions. However, the functionality of the simulation environment made it impossible.

By using the simulation environment from the company, the abstraction of the nodes in the BT was fixed. The abstraction of the nodes had a great impact on the results. Since the nodes are the building blocks in the BT structure. It was decided to implement additional nodes and modify existing functionality in the simulation environment. BT structure made limitations to the fitness function, scenario and behaviours. The classification of key characteristics for the fitness function was done by investigating the scenario and behaviours.

For the training of the algorithm and the evaluation of it, three scenarios were made. These were modeled after a meeting with domain operator experts. This gave the scenarios high credibility.

5.2.2 Implementation

The foundation of the implementation was built on a literature study. Results of this study determined the techniques to be implemented with BT:s to develop a decision support system. Due to time constraints only a high level investigation of classic RL and evolutionary algorithms was made. The results would have been better if two different approaches had been implemented. However, due to robustness and general approach of the algorithm, EAs was suggested to be a better choice.

The same approach of research for EAs techniques was made. By comparing different initialisation techniques, genetic operators and selection functions, these were decided. Probably a different choice could have made differences in the results, but the decisions were made in favour of robustness and complexity. The result of the evaluation of the initialisation technique were good. PTC2 gave a good distribution for each population and for the first scenario, three out of five populations had a success rate equal or larger than 2%.

Instead of only using GP, a grammar based form of it was used. Due to the behaviour of the genetic operators, it needed a validation function. How this validation function was implemented affected the results of crossover and mutation. Crossover did not only change sub-trees between two individuals, it also added randomness to the sub-trees. Also, by using a grammar the flexibility could have triggered potential consequences for the results.

Optimisation was decided to be added as a feature in the algorithm. Instead of working on the tree structure after applied algorithm, it was run during the evaluation. By adding it as a feature it changes the behaviour of the algorithm. Depending on genetic operators, this feature could have a huge impact on the tree structure.

5.2.3 Reliability and Validity

Concerning reliability of the experiments, checkpoints were used and saved all vital data. However it is not possible to redo the exact experiment. Therefore, random number generator (RNG) should have been used to ensure the replicability. By using RNG, it would have been possible to repeat the same sequence of random numbers.

To test the complexity of the evolved tree structures, and robustness and generalisation of the algorithm, metrics were decided upon. These effected the results, since the results are derived from them. The metrics are known and well-tried, therefore the results have high validity.
To test the robustness, the best BT from scenario 3 would have been tested on scenario 2. However, to get a more accurate evaluation, small changes could also have been done in scenario 2. Such as, move the enemies, waypoints and target within a circle area. Then, vary the radius even more to see how well the individual handles variation.

5.2.4 Sources

In short, several articles of the sources are written by the authors P. Ögren and M. Colledan-chise. The authors have written several articles and even books about BTs in robotics and are well cited. Informations about the techniques used in this thesis are mainly found in books, however when comparing concepts, articles were read. An example is the validation of crossover and mutation when implemented with grammatical evolution.

Most of the articles are up-to-date, since they are published during the twenty-first century. The books used are however older but still very relevant.

5.3 The work in a wider context

This section addresses the ethical part of this thesis and puts the work in a wider context.

5.3.1 Replacement of fighter pilots

This thesis includes an investigation of the suitability of evolutionary algorithms paired with BTs as a decision support system for fighter pilots. A system can be manual or autonomous and the level of autonomy could vary. When a system is manual, it is fully under human control, whereas if it is autonomous, a computer carries out the same functions as the human under human control. However there are several levels in between these extremes. In the paper of R. Parasuraman et al (2000) [23], ten levels of automation were presented. The lower levels let the computer suggest decisions or actions for the pilot to choose from if desired. The higher lever, the more support the system gives the pilot. From level 6 and above, the system executes actions automatically but includes the pilot to change it. However, at the higher level, the pilot only gets information of what is done, and in last level the system exclude the pilot.

A fully automated decision support system would result in digital fighter pilots. The replacement of fighter pilots highlights questions concerning system control.

5.3.2 System control

To what extent should the system be automated and what is crucial for the fighter pilot to control? A flow chart of how to decide if the system should be automated or not or at what level, is presented in the paper of R. Parasuraman et al (2000) [23]. The flow chart includes acquisition, analysis, decision and action as types of automation. This can be used as an initial approach when determining a system’s automation level.

The system implemented in this thesis was tested in different scenarios. Two scenarios included enemies and the agent had the possibility to engage combat. By respecting rules of engagement, a fighter pilot should always have the final call for missile launch. Not a computer. Therefore the part in the mission critical system handling the engagement should never be fully automated. However, certain decisions concerning functionality of reconnaissance could have a higher level of autonomy.

The fighter pilot has to make tactical and strategical reasoning. A decision support system could minimise human errors but also lighten the workload of the fighter pilot in stressful situations. To find the right balance between automation and supporting fighter pilot decisions is difficult. Factors such as safety and trusting the system to execute the right action in a
specific situation must be regulated. For this, technologies for supporting human safety and control are introduced.

5.3.3 Trustworthiness of an AI system

To increase the trustworthiness of an AI system, standards, procedures and policies, and enabling technologies are being derived. This work takes long time since the laws need to be adjusted and all parts need to be satisfied.

An AI system also needs to have a safety assessment, in other words, it has to guarantee safety and control when used. Military systems have a different type of regulations than in civil autonomous airborne systems. In civil unmanned aerial vehicles, the safety of the passengers also needs to be guaranteed. This makes it even more complex. However, if a UCAV should fly in common airspace with civil unmanned aerial vehicles, guarantees of safety have to be decided and met before this is possible.

5.3.4 Considerations of the work

Since the master’s thesis was produced in cooperation with Saab AB, the implementation, testing and execution of code had to be done within SAAB’s facilities and not leave the premises. Detailed code will therefore not be included in this thesis. The implementation is instead presented as a UML diagram in appendix B and this has been approved by SAAB.
The chapter reconnects the result and discussion with the research questions, to draw conclusion of the project. The chapter ends with suggestions of improvements and possible future research.

6.1 RQ 1 - Grammatical Evolution as a suitable method

Earlier research and concepts prove that grammatical evolution is a suitable algorithm to be paired with BT:s \[3, 24\]. This thesis suggests that it is possible to implement a system based on GE with BT as framework.

GE is a grammar-based form of GP. GE is constructed by a genotype and a phenotype and a grammar, mapping between them. The representation of GP is a tree structure, which means that the phenotype is of this structure. Since BT and GP is of the same structure, the method fits well with BT:s.

The grammatical approach makes it easy to alter the tree structure. Since the grammar makes the algorithm flexible when designing the tree structure, rules have to be set up.

Research shows that EA:s often get stuck in local maximum. To counteract this behaviour and make the algorithm more robust, more research was put on initialisation techniques, selection functions and genetic operators. However, more work has to be done on the fitness function. It did not have a soft transmission between the possible fitness values. Even if the interval of the fitness values was between -1500 to 3000, not many different values could be found by the algorithm. Of 200 individuals, only 13 different fitness values were counted.

The fitness function is closely connected to the different behaviours in the BT. By extending the nodes or adding more functionality i.e. more nodes, this could generate a more smooth fitness function.

6.2 RQ 2 - Important parameters of the implementation

From the implementation of the evolutionary algorithms, key characteristics could be found while executing the experiments. Identified key characteristics of the implementation were the following:

- Scenario
6.3 RQ 3 - Performance due to Robustness and Complexity

- Fitness function
- Initialisation technique
- Control parameters

It is important to have a well designed scenario, since the behaviour is trained on this. To be mentioned, during the design of the scenarios important discoveries were made. The target had to be outside the enemy area. As the behaviour gets trained on a scenario, it gets evaluated due to its fitness value. A proper fitness function has to be chosen carefully.

Solutions to EA problems are likely to end up in a local maximum. Therefore it is vital to get the best start as possible for the population. This means that the trees should have a good distribution. The investigation made in this thesis promotes the initialisation technique of PTC2.

Last, the balance between parameters is important. The experiments has shown that a bigger population was needed. Due to time constraints in the project, populations of 50-70 individuals was simulated. However, even larger populations are needed.

According to theory, GP does not normally include mutation, but this thesis suggest that it should be included. Mutation could prevent starvation of essential nodes for the behaviour.

6.3 RQ 3 - Performance due to Robustness and Complexity

The design decisions of the algorithm made the algorithm robust to a certain extent. The initialisation technique PTC2 gave a good distribution of the initial population. The genetic operators and the validation were chosen carefully.

However, due to the results of the experiments, it was not possible to evaluate the performance based on robustness. The foundation of the experiments were poor, in the amount of simulations. Even the number of individuals in each population and the amount of generations needed to be more numerous. A factor of the results was the fitness function. Even though the thesis suggests that it is suitable to implement a system based on GE with BT as framework, the experiments of this thesis could not conclude any results about its performance based on robustness.

The experiments were analysed by the evaluation the complexity and tree depth. During the generations, the results of complexity and tree depth varied while the best fitness was fixed. This concludes that the algorithm could find a solution with low complexity and tree depth. However, due to the poor results in experiment 2 and 3, it is not possible to conclude anything on a found solution.

Concerning complexity, the algorithm used two different features connected to complexity. The algorithm promoted a lower complexity when equal fitness value or to chose a random individual when equal fitness value. The experiments prove that optimisation on the algorithm did perform well, however it caused less possibilities to find a desired solution of the problem. Instead optimisation of the tree structure could be a more suitable approach.

6.4 Summary

In summary, most of the simulations resulted in local maximums. A desired solution was found, only if the initial population contained an individual with a BT succeeding the mission. The used techniques was decided based on increasing performance of the algorithm and complexity of the tree structures. PTC2, the genetic operators and the selection functions performed well but the fitness function needs more work.

The results of the experiments were expected, the behaviour of the algorithm relates to the material covered in chapter 2. The experiments showed a promising variation of complexity when the best fitness was fixed. However, more simulations are needed to draw a conclusion of performance based on robustness when testing the evolved BT:s on different scenarios.
6.5 Future Work

A vital improvement of this work would be to add more behaviour to the simulation environment together with expanding the fitness function. This would make the fitness function more smooth and give a more just evaluation of the algorithm. This in addition to expand the experiments with larger populations and numerous generations should improve the simulation results.

An early approach in the implementation was to include optimisation of complexity in the algorithm. This backfired and the evolved trees’ complexity became small and the variety of individuals were inhibited. This feature was removed early in the experiments. Instead, a possible improvement of the complexity could be to implement pruning. By using pruning as an optimising technique nodes in the tree that are not used or are duplicates could be removed without affecting the final behaviour.

An initial idea to lower the simulation time was to define several sub-missions of the reconnaissance mission. Or different BT:s could be used in different modes or depending on where the agent is in its mission. By dividing the mission into smaller sub-missions, could this promote efficient behaviour? L. Chong-U et. al (2010) [17] evolved behaviour trees for four individual behaviours and then combined the best performing trees to one tree. Each behaviour had its own fitness function. They find that the mean fitness seems to have reached its highest at approximately 100 generations. And therefore performing the evolution for a greater number of generations may not show significant improvement in mean fitness.

In paper [17], an improvement could have been to perform experiments against other AI-bots or human players. In this simulation environment, the behaviour of the enemies were only to follow a predefined waypoint route.

A direction for future research would be to use emotional nodes, to change the behaviour of the tree depending on the state of the agent. In the paper of R. de P. Pereira and P. M. Engel [25], they present a new node, called the learning node, which embeds a local RL model without changing the overall tree. Several other authors have done this as well: emotional node, query node. This node could be used to mimic the emotional behaviour of a fighter pilot in different situations. This could for example be used for training fighter pilots.

An investigation in line with the structure of BT:s would be to compare the tree structure of the evolved BT:s with human designed trees, to evaluate if the structure is similar or diverges. And in more complex scenarios, investigate if the algorithm evolve less complex trees than the human designed trees. Are there scenarios where the human could not think about all the possible situations and therefore need an evolved tree?

Lastly, evaluating EAs merged with other techniques to give better results would be an interesting area of research. L. Chong-U et. al (2010) [17] raises the question if other techniques should be added to evolutionary techniques in automating AI-bot design. P. A. Vikhar (2016) [30] gives examples of different evolutionary hybrids in his paper and states that it gives better results than only using one algorithm.


Objectives and Behaviours
A.1 Objectives

- Communication to home base
- Fuel limitation
- Identify a target (with threat)
- Identify a target (without threat)

A.2 Behaviours

Conditions
- Has ammunition
- Has fuel
- Has active sensors
- Home base within sensor reach
- Unclassified unit within sensor reach
- Enemy air defence within sensor reach
- Enemy aircraft within sensor reach
- Target within sensor reach
- Within threat area
- Within combat zone

Actions
- Activate sensors
- Classify units into target, enemy or ally
- Collect target data
- Collect enemy data
- Continue follow a route
- Deactivate sensors
- Destroy reconnaissance elements
- Deviate from threat area
- End mission
- Engage combat (only in combat zone and with airborne units)
- Failed mission
- Fly towards home base
- Fly towards target
- Fly towards unclassified unit
- Follow a route
- Transmit collected data to home base
UML - Algorithm
C.1 Baseline 1
C.2 Baseline 2

![Diagram of Baseline, scenario 2](image)

- Start position
- Route
- Waypoints
- Threats from start
- Hits

---

Follow waypoint route: 0
Defence potential: -471
Classification of target: 0
Weapon economy: 400
Time efficiency: 0

Fitness: -7
Total distance waypoints: 4363380
Total threat area: 7
Total time: 2000
C.3 Baseline 3
C.4 Baseline 4

Baseline, scenario 2

Start position
Route
Waypoints
Threats from start
Classified threats
Hits
Classified target

Follow waypoint route: 678
Classification of targets: 900
Weapon economy: 488
Time efficiency: 124

Fitness: 1583
Total distance waypoints: 72187
Total threat area: 1152
Total time: 1751
PTC2 - Distribution
Figure D.1: Population 1 had range 66-74 in complexity (left) and 10-28 in tree depth (right). The population was simulated with scenario 1 and gave 0% in success rate.

Figure D.2: Population 2 had range 66-74 in complexity (left) and 11-29 in tree depth (right). The population was simulated with scenario 1 and gave 0% in success rate.

Figure D.3: Population 3 had range 66-74 in complexity (left) and 10-28 in tree depth (right). The population was simulated with scenario 1 and gave 2% in success rate.
Figure D.4: Population 4 had range 66-75 in complexity (left) and 10-24 in tree depth (right). The population was simulated with scenario 1 and gave 6% in success rate.

Figure D.5: Population 5 had range 66-73 in complexity (left) and 10-25 in tree depth (right). The population was simulated with scenario 1 and gave 2% in success rate.

Figure D.6: Population 6 had range 66-73 in complexity (left) and 11-22 in tree depth (right). The population was simulated with scenario 2 and gave 0% in success rate.
Figure D.7: Population 7 had range 66-73 in complexity (left) and 10-23 in tree depth (right). The population was simulated with scenario 2 and gave 0% in success rate.

Figure D.8: Population 8 had range 66-72 in complexity (left) and 11-25 in tree depth (right). The population was simulated with scenario 2 and gave 0% in success rate.

Figure D.9: Population 9 had range 66-75 in complexity (left) and 10-28 in tree depth (right). The population was simulated with scenario 2 and gave 0% in success rate.
Figure D.10: Population 10 had range 66-73 in complexity (left) and 10-24 in tree depth (right). The population was simulated with scenario 2 and gave 0% in success rate.