TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics

IDK40LT
Anton Charnamord 134458IAPB

GENETIC APPROXIMATIONS FOR THE FAILURE-FREE SECURITY GAMES

Bachelor's Thesis

Supervisor: Aleksandr Lenin
Ph.D
associated lecturer

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT
Anton Charnamord 134458IAPB

GENEETILISED LÄHENDUSED VEAVABADE TURVAMÄNGUDE JAOKS

Bakalaureusetöö

Supervisor: Aleksandr Lenin
Ph.D
mittekoosseisuline õppejõud

Tallinn 2016

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Anton Charnamord

23.05.2016

Abstract

This thesis deals with computational aspects of attack trees, more precisely, evaluating the expected adversarial utility in the failure-free game. It has been shown by Buldas and Lenin that exact evaluation of this utility is an NP-complete problem, so a computationally feasible approximation is needed. The thesis considers a genetic approach for this challenge. Since genetic algorithms depend on a number of non-trivial parameters, we now have a multi-objective optimization problem. The work considers several heuristic criteria to solve them.

This thesis is written in English and is 35 pages long, including 6 chapters, 13 figures, 3 tables and 4 algorithms.

Abstract

Geneetilised Lähendused Veavabade Turvamänguge jaoks

Väitekirjas uuritakse ründepuude arvutuslikke aspekte, täpsemalt ründaja ootetulu hindamist veavaba turvamängus. Buldas ja Lenin näitasid, et ootetulu täpne arvutamine on NP-täielik probleem ja seega on vajalikud arvutuslikult tõhusad lähendid. Väitekiri vaatleb üldist lähenemist sellele väljakutsele. Et geneetilised algoritmid sõltuvad paljudest mittetriviaalsetest parameetritest, siis tekivad mitmeeesmärgilised optimiseerimisprobleemid. Töös vaadeldakse mitmeid heuristilisi kriteeriume nende lahendamiseks.

Lõputöö on kirjutatud inglise keeles keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 13 joonist, 3 tabelit ja 4 algoritmi.

List of Abbreviations and Symbols

Abbreviation	Definition
AGA	adaptive genetic algorithm
GA	genetic algorithm
PDAG	propositional directed acyclic graph
SAT	Boolean satisfiability

Symbol	Definition
$\mathcal{X}$	set of atomic attacks
x_i	atomic attack
$\mathcal{F}$	monotone Boolean function
σ	attack suite
$\mathcal{P}$	prize of a SAT game
$\mathcal{E}_i$	expenses of $\mathcal{X}_i$
p_i	probability of success of $\mathcal{X}_i$

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This thesis is an extended version of the paper[1] which covers the results of the research conducted in 2015.

1.1 Theoretical Background and Problem

It is well-known that security assessment methods use modelling approach which allows to create a model of the analysed organisation. During the modelling process, a model state, at which the model is considered to be protected, must be defined. At the next stage of the process, computational methods check whether the model is protected with respect to the model security definition.

A number of possible attack strategies is considered in order to model various adversarial actions against the system. The strategies correspond to certain assumptions of attacker's behaviour. These models are called *security games*.

It is also known that solving a security game is a complex task which tends to belong to NP-space. From a practical standpoint, heuristic methods are widely used. These methods calculate the result with an acceptable accuracy within reasonable time.

1.2 Problem Statement

Buldas, A. and Lenin, A. introduced a certain type of security games called *satisfiability games* [2, 3]. This new type of security assessment considers rational attackers that perform targeted profit-oriented attacks. Attack trees are used to represent all possible ways to attack. Based on them, expected adversarial expenses are calculated. If the profit of an attacker exceeds his expenses, it is concluded that the

system requires to introduce additional security measures.

To solve satisfiability game means to answer the question: if there exists an attack against the organisation where the total expenses of the attacker are lower then the profit. This problem was named *weighted monotone satisfiability problem* and is an NP-complete problem, so a computationally feasible approximation is needed.

Therefore, the main purpose of the work is to create an approximation method that allows to solve satisfiability games in particular cases.

1.3 Methodology

1.3.1 Optimisation Methods

It is known that there is a wide range of various optimisation algorithms and methods. Some of the most popular approaches are:

- simplex method
- Karmarkar's algorithm
- gradient ascent algorithm
- simulated annealing approach

Although the first three algorithms are very powerful and widespread, they cannot be applied to this particular case. These approximation methods use fitness landscape function which is unknown for the considered model. The last approximation algorithm is totally suitable and could be chosen to provide the research with a plausible approximation. However, genetic approach has been chosen to solve the optimisation problem. Based on the previous experience with GA implementation, it is important to compare the results of the current research with the previous ones [4].

1.3.2 Genetic Algorithm

A genetic algorithm typically depends on various parameters. The set of parameters commonly contains the following:

- A genetic representation of **chromosomes** also known as **individuals** which are feasible solutions for the optimization problem.
- A **population** of encoded solutions which is the number of used individuals.
- **Fitness function** which evaluates the optimality of the solutions.
- Genetic operators such as **selection**, **crossover**, **mutation** that generate a new population from the existing one.
- Control parameters such as **population size**, **crossover rate**, **mutation rate**, and the condition under which the reproduction process terminates.

Selecting all these loosely connected parameters is considered to be a non-trivial task. Taking into account all the possible ways of solving the problem, collecting heuristic evidence for optimal parameter selection was chosen as a basic method of optimising the considered parameters of the algorithm. In order to do it, attack tree generator was created to provide the research with suitable data set. The data set was used as a benchmark data for the conducted experiments at which heuristic evidences were collected empirically.

1.4 Thesis Outline

The work has the following structure. First, Chapter 2 draws the state of the art. Then, Chapter 3 provides the reader with the theoretical background and initial setup of the conducted experiments. Chapter 4 presents and evaluates the genetic algorithm. This algorithm is improved with adaptiveness in Chapter 5. Finally, Summary draws some conclusions.

2 State of the Art

2.1 Related Research

Security threats can be described and assessed using attack trees. This hierarchical method for estimation of security risks represents a set of attacks against a system as a tree structure, where the root of the tree is considered to be an attacker's goal. Intermediate and terminal nodes of the tree are used to illustrate a variety of possible ways, in which the system can be attacked. Every attack against the real organisation consists of atomic actions (atomic attacks) that the attacker has to perform in order to achieve his goal.

In addition to purely descriptive approach which uses attack trees, computational aspects have been introduced since its first descriptions. Based upon the variables of an attack tree, the result of its analysis may vary. The value of a certain attack can be assessed according to different set of criteria. For instance, it can be the cost of the attack or expected profit.

Most of the earlier attack tree studies focus on the analysis of a single parameter. A substantial step forward was taken by Buldas *et al.* [5] who introduced the idea of game-theoretic modelling of the adversarial decision making process based on several interconnected parameters like the cost, risks and penalties associated with different atomic attacks. Their approach was later refined by Jürgenson *et al.* who introduce sequentiality into the adversarial decision making model [6]. However, increase in the model precision was accompanied by a significant drop in computational efficiency. To compensate for that, a genetic algorithm approach was proposed by Jürgenson *et al.* [4]. It was later shown by Lenin *et al.* that this approach is flexible enough to allow extensions like attacker models [7].

Buldas *et al.* [8] introduced the upper bound ideology by pointing out that in order to verify the security of the system, it is not necessary to compute the exact adversarial

utility but only the upper bounds. If adversarial utility has a negative upper bound in their fully adaptive model, it is safe to conclude that there are no beneficial ways of attacking the system.

Buldas *et al.* further improved the fully adaptive model by eliminating the force failure states and suggested the new model called the failure-free model [2]. The model more closely followed the upper bounds ideology originally introduced by Buldas *et al.* [8] and turned out to be computationally somewhat easier to analyse.

2.2 Algorithm Basis

Based on the previous research of Buldas *et al.* [2], exact evaluation of adversarial utility in the case of the failure-free game is an NP-complete problem, so a computationally feasible approximation is needed. The whole family of genetic algorithms is considered to be one of the most widely used approaches to provide an approximation to NP-complete problems. This typical approximation approach is aimed to significantly increase efficiency of computations. However, the increment in speed of computations is accompanied by a drop in precision of results.

Hence, the main objective of the approximation is to find the cheapest solution that does not exceed the specified value which is the prize $\mathcal{P}$ of the game. On one hand, the propagation method [2] establishes the fixed upper bound for the adversarial utility and limits real utility from above. On the other hand, the chosen genetic approximation of the precise method aims at approaching the precise result from below. Fig. 2.1 illustrates an interval which contains the real adversarial utility. The interval has a fixed upper bound and sliding lower bound. The real adversarial utility lies somewhere within the interval and cannot be well-determined during the analysis. The upper bound is determined by the propagation method while the lower bound is set by the genetic approximation. Therefore, the more precise approximation is implemented, the closer the genetics to the precise result is.

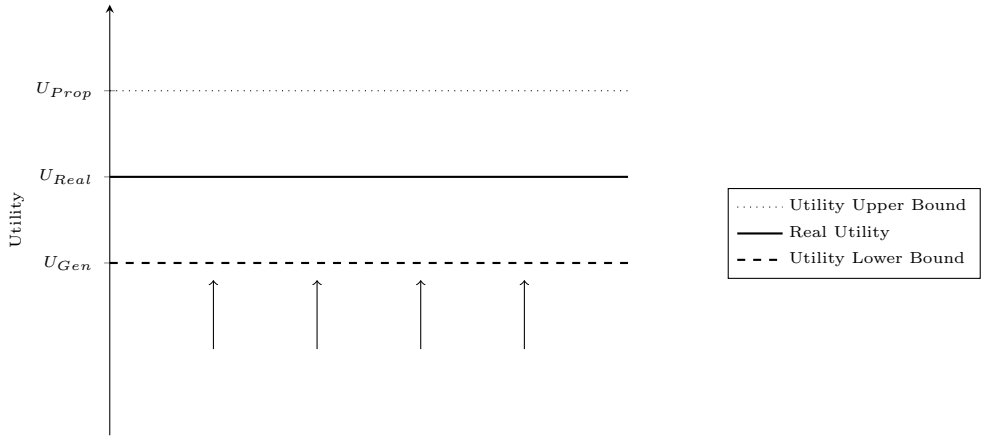

FIGURE 2.1: Estimated Adversarial Utility

3 Experiment Setup

3.1 Terms and Definitions

Every attack tree can be represented as a monotone Boolean function where the arguments of the function correspond to the tree leaves and the $\wedge$ and $\vee$ logic operators correspond to AND and OR intermediate nodes, accordingly.

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be the set of all possible atomic attacks, and $\mathcal{F}$ be a monotone Boolean function corresponding to the considered attack tree.

Definition 1 (Attack Suite)**.** *Attack suite $\sigma \subseteq \mathcal{X}$ is a set of atomic attacks which have been chosen by the adversary to be launched and used to try to achieve the attacker's goal. Also known as individual.*

Definition 2 (Satisfying attack suite)**.** *A satisfying attack suite σ evaluates $\mathcal{F}$ to* true *when all the atomic attacks from the attack suite σ have been evaluated to* true*. Also known as live individual.*

Definition 3 (Satisfiability game)**.** *A satisfiability game is mean a single-player game in which the player's goal is to satisfy a monotone Boolean function $\mathcal{F}(x_1, x_2, \ldots, x_k)$ by picking variables x_i one at a time and assigning $x_i = 1$. Each time the player picks the variable x_i he pays some amount of expenses $\mathcal{E}_i$, which is modelled as a random variable. With a certain probability p_i the move x_i succeeds. The game ends when the condition $\mathcal{F} \equiv 1$ is satisfied and the player wins the prize $\mathcal{P} \in \mathbb{R}$, or when the condition $\mathcal{F} \equiv 0$ is satisfied, meaning the loss of the game, or when the player stops playing. Thus three common types of games can be defined:*

1. *SAT Game Without Repetitions - the type of a game where a player can perform a move only once.*
2. *SAT Game With Repetitions - the type of a game where a player can re-run failed moves an arbitrary number of times.*

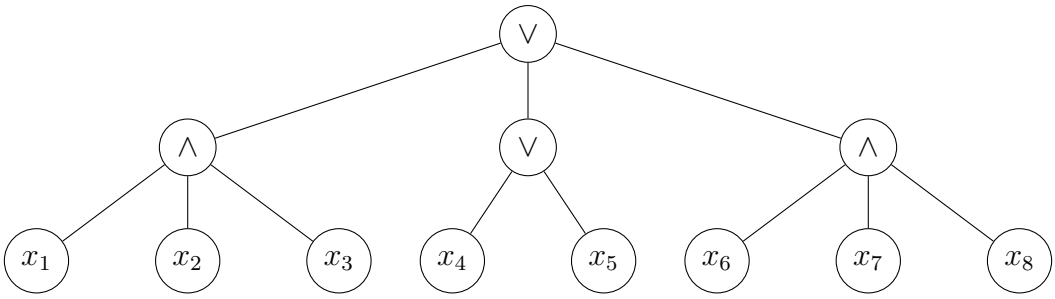

FIGURE 3.1: Attack Tree Example

3. *Failure-Free SAT Game - the type of a game in which all success probabilities are equal to 1. It has been shown that any game with repetitions is equivalent to a failure-free game [2, Thm. 5].*

For example, Fig. 3.1 shows an attack tree that can be interpreted as Boolean function $\mathcal{F} = (x_1 \wedge x_2 \wedge x_3) \vee (x_3 \vee x_4) \vee (x_6 \wedge x_7 \wedge x_8)$ with solutions: $\sigma_1 = \{x_5\}$, $\sigma_2 = \{x_1, x_2, x_3\}$, $\sigma_3 = \{x_4, x_6, x_7, x_8\}$, etc.

3.2 Tools and Environment

Let the length of an attack tree is the number of leaves in the tree. All the experiments and measurements described below have been conducted using sets of randomly generated attack trees of different lengths. Algorithm 3.2.1 illustrates the main logic of creation attack trees with specified number of leaves.

The reproduction process, as well as the condition, under which reproduction terminates, is identical to the one described in [7]. According to Lenin *et al.*, the reproduction phase terminates when the pre-defined number of last generations does not increase outcome. Result of the current generation is considered to be not improving the result of the previous generation if the difference between the results does not exceed the pre-set value.

In addition, an extra stop condition has been introduced and implemented in GA in this work. The evolution process is aborted when the algorithm run time exceeds

Algorithm 3.2.1: Attack tree generation algorithm

Data: The length of the generated attack tree $\mathcal{N}_l$

Result: The root of generated attack tree

```
List of Nodes leaves := generate N_l random leaves;
List of Nodes notLinkedNodes := empty List;
add leaves to notLinkedNodes;
while (size of notLinkedNodes != 1) do
    Node node := generate random intermediate Node;
    childrenNum := randomly choose the number of the node's children;
    for i from 0 to childrenNum do
        child := retrieve a node with a random index from notLinkedNodes;
        set child to node;
Node root := retrieve the only element from notLinkedNodes;
return root;
```

the pre-set value. This constraint allows to get results not relying on the assumption that the method will converge in the desired timeframe.

All the following computations were made with PC/Intel Core i5-4590 CPU @ 3.30 GHz, 8 GB RAM, Windows 8.1 (64 bit) operating system.

4 Genetic Algorithm

4.1 Individual

An individual is any feasible solution to the considered optimisation problem. Thus, for the SAT games a solution is any of the *satisfying attack suites*. The linear binary representation of individuals has been chosen to facilitate the robustness of the crossover and mutation operations. The algorithm used to generate individuals is shown in Algorithm 4.1.1.

Algorithm 4.1.1: Recursive individual generation algorithm

Data: The root of a propositional directed acyclic graph (PDAG) representing a monotone Boolean function. An empty individual with all bits set to 0.

Result: Live individual.

```
if the root is a leaf then
    get the index of the leaf;
    set corresponding individual's bit to 1;
else if the root is an AND node then
    forall children of the root do
        recursive call: child considered as root parameter;
else if the root is an OR node then
    choose at least one child;
    forall chosen children do
        recursive call: child considered as root parameter;
```

It is allowed for duplicate entries to be present in the population for the sake of maintaining genetic variation and keeping the population size constant throughout the reproduction process. It is well known in the field of genetic algorithms that genetic variation directly influences the chances of premature convergence – thus

increasing genetic variation in the population is one of the design goals.

4.2 Population Size

The choice of the population size is important – too small population does not contain enough genetic variation to maintain the exploration capabilities, whereas too big population already contains enough genetic variation to efficiently explore the search space, and only results in the performance overhead in the crossover operator. This means that there exists an optimal population size corresponding to the minimal population size capable of producing the best result. Thus the optimal size of the population sets the lower bound for reasonable choice for the population size, and the upper bound is solely based on performance considerations – what is the reasonable time the analysts would agree to wait for the analysis to produce the result. If the population size is suboptimal, there is a high risk to converge to suboptimal solutions, and if the population is bigger than the optimal size it does not add anything, except for the increase in the time required to run the analysis. If the optimal population in some certain case is $k\%$ of the size of the attack tree (the number of leaves in an attack tree), then any population size greater than $k\%$ and capable of producing the result in reasonable time, would suit to be used for analysis.

Fig. 4.1 demonstrates the effect of the population size on the result in the case of a single attack tree. The measurements were taken for the attack tree with 100 leaves using uniform crossover operator and mutation rate 0.1.

The experiments have been conducted on the set of attack trees of different sizes (ranging from 10 to 100 leaves with steps of size 3) and observed that there is no obvious relation between the size of the analysed tree and the optimal population size. Apart from the size of the tree, the optimal population size might depend on, at the very least, the structure of the tree itself. The measurements were taken with the same crossover operator and the same mutation rate. Fig. 4.2 shows how many trees (%) from the conducted experiment the considered population size would fit. It can be seen that, in general, the population size equal to 180% of the size of the tree would fit every considered attack tree. The population size 200%, chosen by

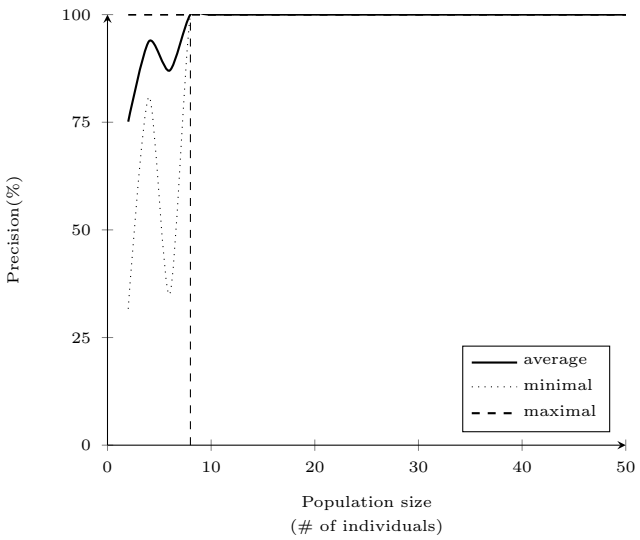

FIGURE 4.1: Optimal population size

Jürgenson *et al.* in [4] for their ApproxTree model, was reasonable enough to be optimal for the majority of the trees one might wish to analyse using their model. Fig. 4.2 shows that in the case the population size was chosen (based on practical or performance considerations) to be 50% of the size of the attack tree, this choice would be optimal for approximately 75% of attack trees, and for the rest 25% of the cases this choice might be suboptimal.

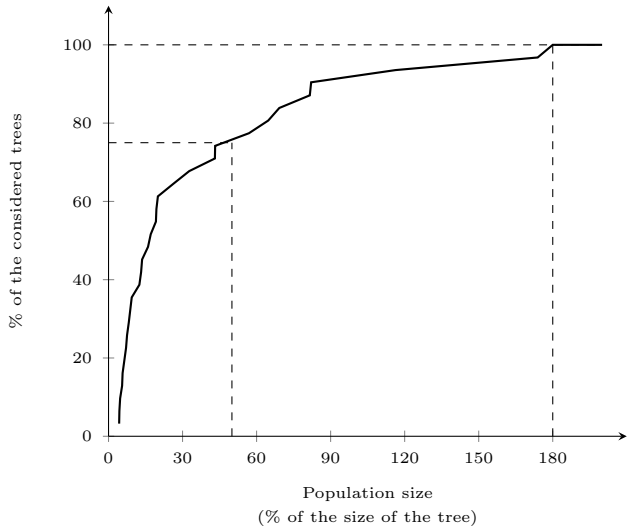

FIGURE 4.2: Reasonable choice for population size

Fig. 4.3 shows the time measurement for the suggested GA, depending on the size of the population.

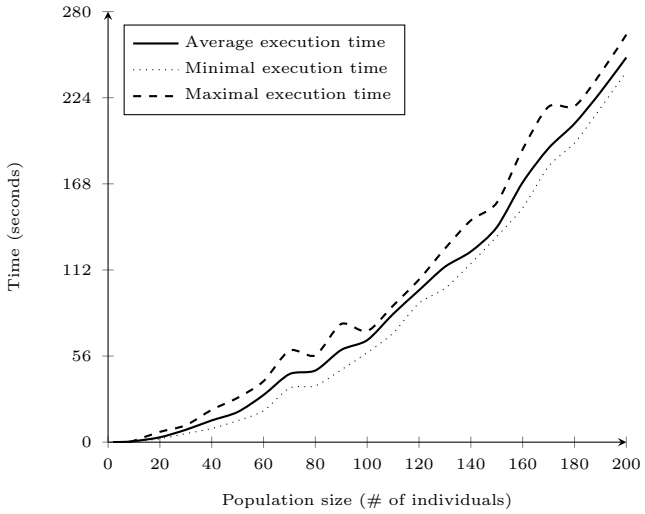

FIGURE 4.3: Population size effect on GA execution time.

4.3 Fitness Function

The *fitness function* is the model-specific utility function for the Failure-Free SAT game. The function's result is the cost of the considered attack tree without actual costs of all atomic attacks that are picked by the attacker. The pseudo-code of the Failure-Free *fitness function* is shown in Algorithm 4.3.1.

Algorithm 4.3.1: Fitness function

Data: The indivividual that is an attack suite for the PDAG.
Result: Numeric value of the individual's utility.

utility := the cost of the PDAG;
forall *bits of the individual* **do**
    if *the bit is active* **then**
        leaf := PDAG's leaf corresponding to the individual's bit with expenses $\mathcal{E}_{leaf}$ and probability p_{leaf};
        utility := utility - $\frac{\mathcal{E}_{leaf}}{p_{leaf}}$;
return utility;

For more information about the function the reader is referred to the detailed descriptions of the security games [9, 4, 7, 8, 2].

4.4 Crossover

Lenin *et al.* have shown that the crossover operations take 90-99% of the time required to run the analysis [7].

The power of GA arises from crossover, which causes randomized, but still structured exchange of genetic material between individuals in assumption that 'good' individuals will produce even better ones. The *crossover rate* controls the probability at which individuals are subjected to crossover. Individuals, not subjected to crossover, remain unmodified. The higher the crossover rate is, the quicker the new solutions get introduced into the population. At the same time, chances increase for the solutions to get disrupted faster than selection can exploit them. The selection operator selects individuals for crossing and its role is to direct the search towards promising solutions. It was decided to disable parent selection entirely thus defaulting to crossing every individual with every other individual in the population (crossover rate equal to 1), as scalable selection pressure comes along with the selection mechanisms after reproduction.

Notable crossover techniques include the single-point, the two-point, and the uniform crossover types. Figures 4.4, 4.5 and 4.6 demonstrate the differences between the convergence speeds resulting from using various crossover operators. These diagrams show the difference between mean convergence speed and standard deviation from it. It can be seen that even though there are cases when the choice of the crossover operator does matter, graphs show that the considered crossover operators are, in general, similar enough and do not have any major differences nor effect on the convergence speed of the GA.

Our choice fell upon using the uniform crossover – this enables a more exploratory approach to crossover than the traditional exploitative approach, resulting in a more complete exploration of the search space with maintaining the exchange of good information. The algorithm for the crossover operator is shown in Algorithm 4.4.1.

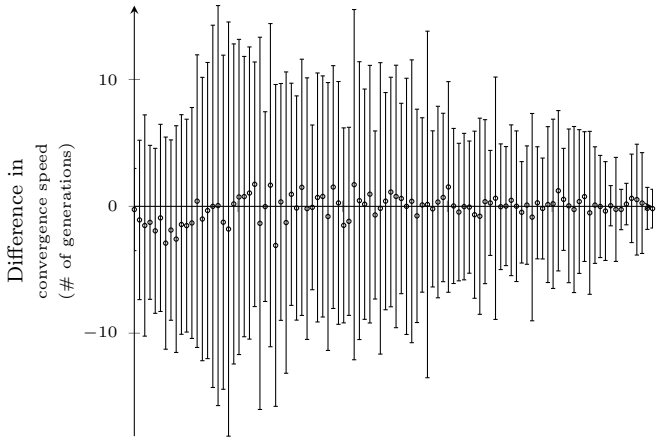

FIGURE 4.4: Uniform crossover compared to single point crossover

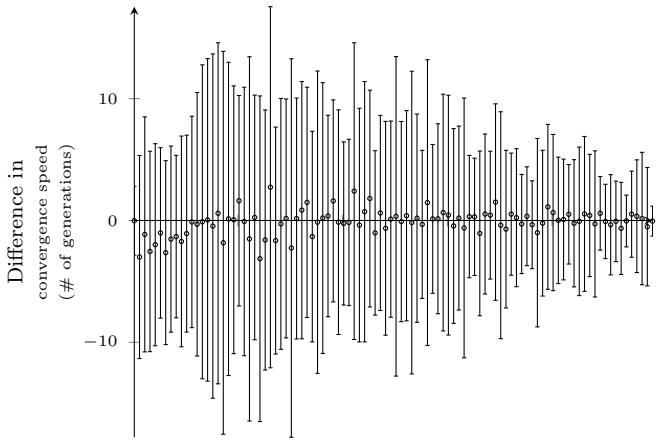

FIGURE 4.5: Uniform crossover compared to two point crossover

4.5 Mutation Operator and Mutation Rate

The role of the mutation operator is to restore lost or unexplored genetic material into the population thus increasing the genetic variance and preventing premature convergence to suboptimal solutions. The mutation rate controls the rate at which 'genes' are subjected to mutation. High levels of mutation rate turn GA into a random search algorithm, while too low levels of mutation rates are unable to restore genetic material efficiently enough thus the algorithm risks converging to suboptimal solutions. Typically the mutation rate is kept rather small, in the range $0.005 - 0.05$.

In the implementation of the genetic algorithm, the mutation operator is a part of the crossover operation, mutating the genes, having same value in the corresponding positions in both parent individuals. The uniform crossover randomly picks corre-

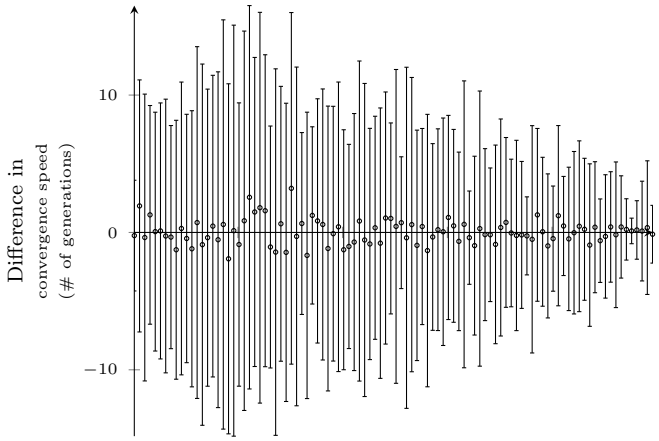

FIGURE 4.6: One point crossover compared to two point crossover

Algorithm 4.4.1: The uniform crossover operation

Data: The population of individuals represented as a sorted set.

Result: The population with new added individuals, created during the crossover operation.

```
initialize a new set of individuals;
forall individual i in the population do
    forall individual j different from i do
        new individual := the result of cross operation between
        individuals i and j;
        if new individual is alive then
            add the new individual to the set of new individuals;
add the set of new individuals to the population;
```

sponding bits in the parent individuals to be used in the new individual, and thus in the case bits are different, this already provides sufficient genetic variation. However, in the case when bits have the same value this yields just a single choice and in order to increase the genetic variation (compared to its parents) just these bits are mutated.

Fig. 4.7 demonstrates the mutation rate effect on the utility function for the case of an attack tree with 100-leaves with initial population of 50 individuals. It shows, that when the mutation rate exceeds 0.1, GA turns into a random search algorithm, thus it is reasonable to keep the mutation rate rather small. Similar experiments were conducted on larger sets of attack trees and the results have shown that the

optimal value for the mutation rate is not necessarily small – in some cases the optimal mutation rate was 0.6 or even higher. This means that the optimal value for the mutation rate cannot be set from the very beginning – it highly depends on the structure of the fitness landscape. However, it is still reasonable to follow the general rule of thumb to keep the mutation rate small, assuming that this should work for the majority of the cases.

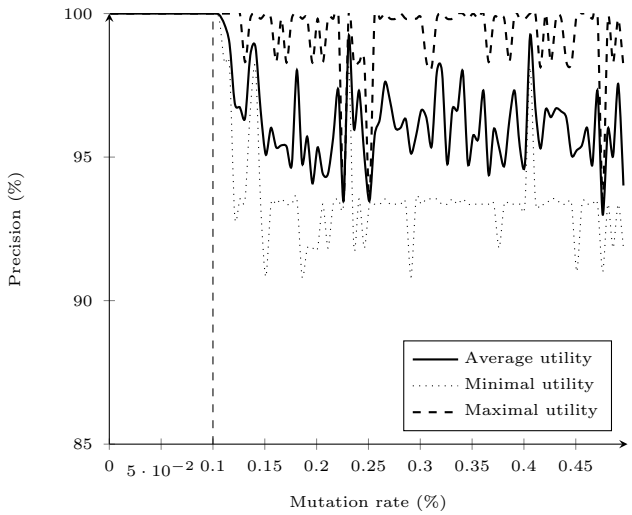

FIGURE 4.7: GA mutation rate effect

4.6 Summary for GA Parameter Choice

Thus, the following set of GA parameters was found out.

- Crossover operator: uniform crossover.
- Crossover rate: 1.
- Selection operator: missing.
- Mutation operator: uniform mutation.
- Mutation rate: 0.1.

It is important to determine the practical applicability boundaries for the suggested method. Practical applicability means the maximal size of the attack tree, which

the computational method is capable of analysing in reasonable time, set to two hours. Extrapolating the time consumption curve in Fig. 4.8 it was concluded that theoretically the suggested GA is capable of analysing attack trees containing up to 800 leaves in reasonable time. This is a major advancement compared to the ApproxTree model [4] which would take more than 900 hours to complete such a task. Such a considerable increase in efficiency is due to the fact that the Failure-Free model [2], where the adversaries are allowed to re-run failed attacks again an arbitrary number of times, is easier to analyse than the Jürgenson-Willemson parallel model [9] even with all the optimizations suggested in [4].

The execution time complexity estimations for GA are outlined in Table 4.1.

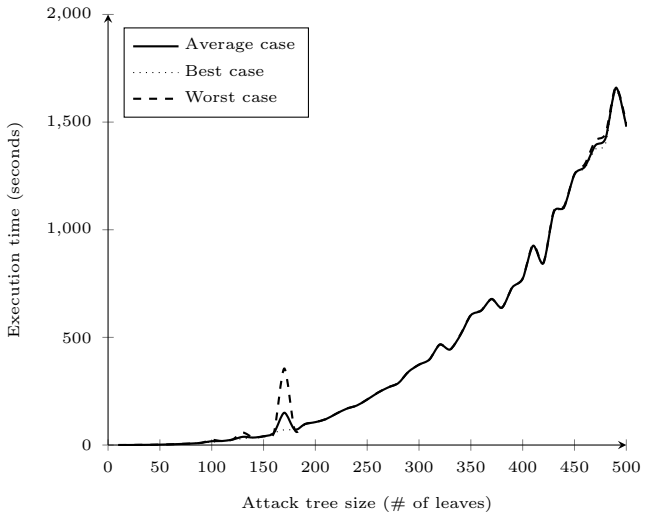

FIGURE 4.8: GA execution time

For comparison, the execution time complexity of the ApproxTree model [4] was estimated to be $\mathcal{O}(n^4)$, where n is the number of leaves in the attack tree. This difference comes from the fact that ApproxTree runs for a fixed number of generations, whereas the computations presented in this work run until local convergence, as well as the fact that the utility function used in ApproxTree is considerably more complex, compared to the corresponding utility function used in the Failure-Free model.

TABLE 4.1: GA execution time complexity estimations

Case	Approximation polynomial	R^2 coefficient
Worst	$1.68 \cdot 10^{-5}n^3 - 0.003n^2 + 0.7015n - 23.03$	0.99
Average	$1.41 \cdot 10^{-5}n^3 - 0.001n^2 + 0.25n - 8.81$	0.99
Best	$1.26 \cdot 10^{-5}n^3 + 1.62 \cdot 10^{-5}n^2 + 0.047n - 2.55$	0.99

5 Adaptive Genetic Algorithm

5.1 Adaptive Genetic Approach

During the research, two algorithms were compared. The genetic algorithm suggested in Chapter 4 was compared to the adaptive genetic approach described in [10]. The authors suggest to adaptively vary the values of crossover and mutation rates, depending on the fitness values of the solutions in the population. High fitness solutions are 'protected' and solutions with sub-average fitness are totally disrupted. It was suggested to detect whether the algorithm is converging to an optimum by evaluating the difference between the maximal and the average fitness values in the population $f_{\max} - \bar{f}$, which is likely to be less for the population which is converging to an optimum solution than for a population scattered across the solution space. Thus the corresponding values of the mutation and crossover rates are increased when the algorithm is converging to an optimum and decreased when the population gets too scattered. The authors concluded that the performance of AGA is in general superior to the performance of GA but varies considerably from problem to problem. The suggested method was applied to the problem of the security games.

5.2 Population Size

In the case of the adaptive genetic algorithm, the crossover and mutation rate parameters are assigned their initial values and are changed adaptively during the runtime of the algorithm, and the only parameter which remains fixed is the population size. Similarly to the GA, there exists an optimal population size, corresponding to the minimal population size, capable of producing the maximal result. Fig. 5.1 shows the result corresponding to the computations using various population sizes in the experiment setup similar to the one for GA. It can be observed that the optimal population size cannot be easily determined (as was the case in GA). In the case of

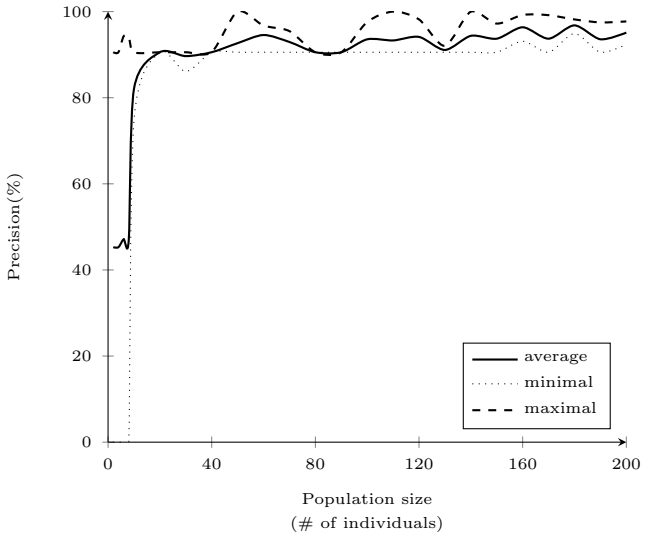

FIGURE 5.1: Optimal population size

GA, the maximal value was stable with the increase in the population size, however in the case of AGA some fluctuations are present. Fig. 5.2 shows how many trees (%) from the conducted experiment the considered population size would fit. It can be seen that, in general, the population size equal to 200% of the size of the tree would fit every considered attack tree. If the population size was chosen (based on practical or performance considerations) to be 50% of the size of the attack tree, this choice would be optimal for approximately 42% of attack trees (in the case of GA the corresponding value was 75%), and for the rest 68% of the cases this choice might be suboptimal. Based on these observations, it can be concluded that AGA seems to be more robust, but less stable, compared to GA and requires bigger population sizes in order to produce optimal results for the majority of the cases.

5.3 Summary for AGA

Similarly to the GA, the maximal size of the attack tree which AGA is capable of analysing within reasonable timeframe was estimated, set to two hours. Extrapolating the time consumption curve with the most extreme values trimmed out in Fig. 5.3 it was concluded that theoretically AGA is capable of analysing attack trees containing up to 26000 leaves in reasonable timeframe, which is approximately 32 times more efficient compared to GA.

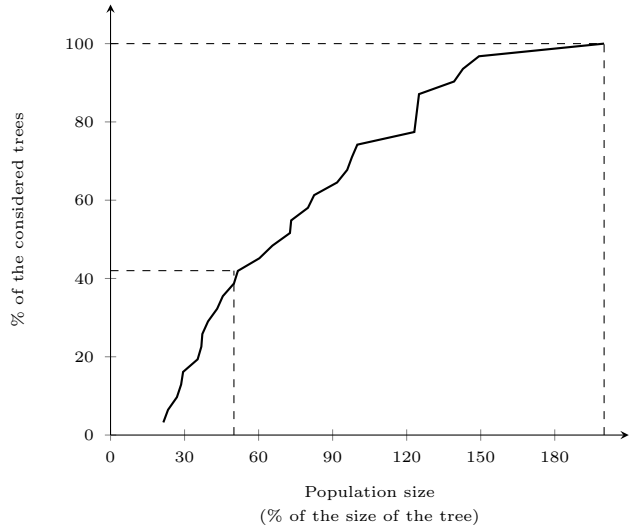

FIGURE 5.2: Reasonable choice for population size

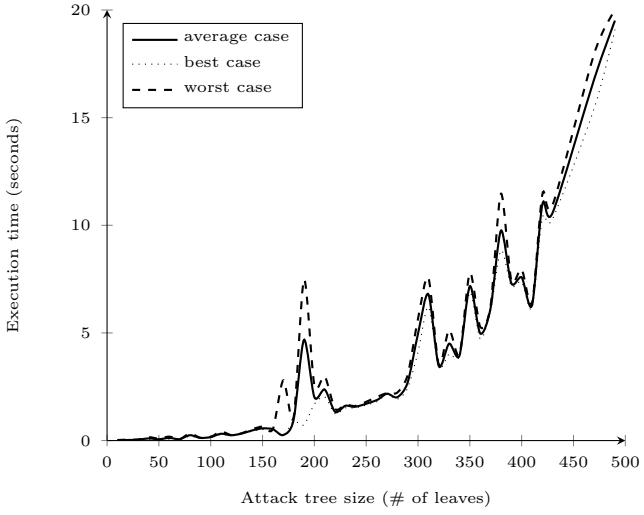

FIGURE 5.3: AGA execution time

The execution time complexity estimations for AGA are outlined in Table 5.1.

TABLE 5.1: AGA execution time complexity estimations

Case	Approximation polynomial	R^2 coefficient
Worst	$3.985x^3 - 0.0001x^2 + 0.0358x - 1.1970$	0.90
Average	$3.5731x^3 - 0.0001x^2 + 0.0267x - 0.8786$	0.94
Best	$3.1892x^3 - 0.0001x^2 + 0.0192x - 0.6115$	0.96

6 Summary

The objective of the work was to develop a new algorithm to solve security games. Among numerous alternatives, genetic approach to approximation was considered, since it is known to have worked on similar problems previously. Though, GA is a domain specific approach depending on various loosely connected parameters, a number of experiments was conducted and empirical evidence was collected. This heuristic evidence helped to select optimal algorithm parameters.

In the course of the research, two genetics algorithm implementation were compared. Table 6.1 illustrates the advantages and disadvantages of both GA and AGA implementations.

Table 6.1: GA vs AGA

	GA	AGA
Pros	more stable smaller population size	generally faster convergence
Cons	generally slower convergence	less stable larger population size

In comparison to GA, AGA converges generally faster and is more robust. Based on the assumption that execution time is the most critical parameter, it is concluded that AGA should be preferred to plain GA to achieve the practical goals.

From a practical standpoint, a new algorithm for solving satisfiability games was developed. In comparison to algorithms, that were previously implemented to solve similar problems [4], the considered algorithm is more effective providing comparable accuracy level.

References

[1] Lenin, A., Willemson, J., Charnamord, A.: Genetic approximations for the failure-free security games. In: Decision and Game Theory for Security - 6th International Conference, GameSec 2015, London, UK, November 4-5, 2015, Proceedings. (2015) 311–321

[2] Buldas, A., Lenin, A.: New efficient utility upper bounds for the fully adaptive model of attack trees. In: Decision and Game Theory for Security - 4th International Conference, GameSec 2013, Fort Worth, TX, USA, November 11-12, 2013. Proceedings. (2013) 192–205

[3] Lenin, A.: Reliable and efficient determination of the likelihood of rational attacks. PhD thesis, Tallinn University of Technology, Tallinn, TUT Press, 2015 (12 2015)

[4] Jürgenson, A., Willemson, J.: On Fast and Approximate Attack Tree Computations. In Kwak, J., Deng, R.H., Won, Y., Wang, G., eds.: ISPEC. Volume 6047 of Lecture Notes in Computer Science., Springer (2010) 56–66

[5] Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures via Multi-Parameter Attack Trees. In: Critical Information Infrastructures Security. First International Workshop, CRITIS 2006. Volume 4347 of LNCS., Springer (2006) 235–248

[6] Jürgenson, A., Willemson, J.: Serial Model for Attack Tree Computations. In Lee, D., Hong, S., eds.: ICISC. Volume 5984 of Lecture Notes in Computer Science., Springer (2009) 118–128

[7] Lenin, A., Willemson, J., Sari, D.P.: Attacker profiling in quantitative security assessment based on attack trees. In: Secure IT Systems - 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings. (2014) 199–212

[8] Buldas, A., Stepanenko, R.: Upper bounds for adversaries' utility in attack trees. In: Decision and Game Theory for Security - Third International Conference, GameSec 2012, Budapest, Hungary, November 5-6, 2012. Proceedings. (2012) 98–117

[9] Jürgenson, A., Willemson, J.: Computing Exact Outcomes of Multi-parameter Attack Trees. In Meersman, R., Tari, Z., eds.: OTM Conferences (2). Volume 5332 of Lecture Notes in Computer Science., Springer (2008) 1036–1051

[10] Srinivas, M., Patnaik, L.M.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics **24**(4) (1994) 656–667