Genetic Approximations for the Failure-Free Security Games

Aleksandr Lenin[1,2], Jan Willemson[1] and Anton Charnamord[1,2] *

[1] Cybernetica AS, Mäealuse 2/1, Tallinn, Estonia
[2] Tallinn University of Technology, Ehitajate tee 5, Tallinn, Estonia

Abstract. This paper deals with computational aspects of attack trees, more precisely, evaluating the expected adversarial utility in the failure-free game, where the adversary is allowed to re-run failed atomic attacks an unlimited number of times. 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. In this paper we consider a genetic approach for this challenge. Since genetic algorithms depend on a number of non-trivial parameters, we face a multi-objective optimization problem and we consider several heuristic criteria to solve it.

1 Introduction

Hierarchical methods for security assessment have been used for several decades already. Called *fault trees* and applied to analyze general security-critical systems in early 1980-s [1], they were adjusted for information systems and called *threat logic trees* by Weiss in 1991 [2]. In the late 1990-s, the method was popularized by Schneier under the name *attack trees* [3].

There are several ways attack trees can be used in security assessment. The simplest way is purely descriptional. Such an approach is limited only to qualitative assessment of security. Based on such an assessment, it is difficult to talk about optimal level of security or return of security investments. Already the first descriptions of attack trees introduced computational aspects [2, 3]. The framework for a sound formal model for such computations was introduced in 2005 by Mauw and Oostdijk [4].

Most of the earlier studies focus on the analysis of a single parameter only. A substantial step forward was taken by Buldas *et al.* [5] who

* This research was supported by the European Regional Development Fund through Centre of Excellence in Computer Science (EXCS), the Estonian Research Council under Institutional Research Grant IUT27-1 and the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement ICT-318003 (TREsPASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.

introduced the idea of game-theoretic modeling 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 and Willemson [6] to achieve compliance with Mauw-Oostdijk framework [7]. However, increase in the model precision was accompanied by significant drop in computational efficiency. To compensate for that, a genetic algorithm approach was proposed by Jürgenson and Willemson [8]. It was later shown by Lenin, Willemson and Sari that this approach is flexible enough to allow extensions like attacker models [9].

Buldas and Stepanenko [10] 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 upper bounds. Buldas and Lenin further improved the fully adaptive model by eliminating the force failure states and suggested the new model called the failure-free model [11]. The model more closely followed the upper bounds ideology originally introduced by Buldas *et al.* [10] and turned out to be computationally somewhat easier to analyze. It has been shown that finding the optimal strategy is (still) an NP-complete problem, hence looking for a good heuristic approximation is an important goal. Additionally, one of the goals of the paper is to find empirical evidence for the rational choice of the parameters of the genetic algorithm.

The paper has the following structure. First, Section 2 defines the required terms. Section 3 presents and evaluates our genetic algorithms. These algorithms are improved with adaptiveness in Section 4. Finally, Section 5 draws some conclusions.

2 Definitions

Let $\mathcal{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$ 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). *By a satisfiability game we 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 modeled 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 we can define three common types of games:*

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.*
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 [11, Thm. 5].*

3 Genetic Approximations for the Failure-Free Satisfiability Games

The whole family of satisfiability games tries to maximize expected adversarial profit by solving an optimization problem: given a monotone Boolean function $\mathcal{F}(x_1, x_2, \ldots, x_n)$ optimize the utility function $\mathcal{U}(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$ over the set of all satisfying assignments fulfilling a set of model-specific conditions (in some specific cases). The models for the SAT games without move repetitions and the failure-free SAT games differ only by their corresponding utility functions, as in both cases the order in which atomic attacks are launched by an adversary is irrelevant. On the contrary, models for SAT games with repetitions (e.g. [12]) consider strategic adversarial behavior in the case of which the order in which the atomic attacks are launched does matter. In this paper we focus on the genetic approximations suitable to be applied to the SAT games without repetitions, as well as the failure-free SAT games. The suggested algorithm is practically validated by the example of the computational model for the failure-free SAT game.

3.1 Genetic algorithm (GA)

A genetic algorithm is typically characterized by the set of the following parameters: a genetic representation of chromosomes or individuals

(feasible solutions for the optimization problem), a population of encoded solutions, fitness function which evaluates the optimality of the solutions, genetic operators (selection, crossover, mutation) that generate a new population from the existing one, and control parameters (population size, crossover rate, mutation rate, condition under which the reproduction process terminates).

The reproduction process, as well as the condition, under which reproduction terminates is identical to the one described in [9]. We refer the readers to this paper for further details. An individual is any feasible solution to the considered optimization problem. Thus, for the SAT games a solution is any of the *satisfying attack suites*. We have chosen linear binary representation of individuals to facilitate the robustness of the crossover and mutation operations. The algorithm used to generate individuals is shown in Algorithm 1.

Algorithm 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;
end
else if the root is an AND node then
    forall the children of the root do
        recursive call: child considered as root parameter;
    end
end
else if the root is an OR node then
    choose at least one child;
    forall the chosen children do
        recursive call: child considered as root parameter;
    end
end
```

We allow duplicate entries to be present in the population for the sake of maintaining genetic variation and keep 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.

The choice of the population size is important – too small population does not contain enough genetic variation to maintain the exploration capabilities, 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 of 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.

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. Fig. 1 on the left demonstrates the effect of the population size on the result in the case of a single attack tree. Measurements were taken for the attack tree with 100 leaves using uniform crossover operator and mutation rate 0.1.

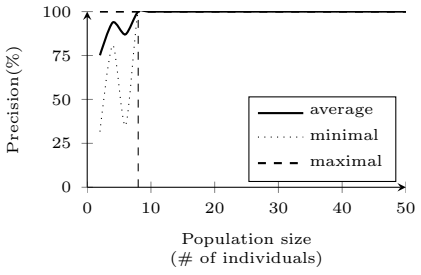

Fig. 1: Optimal population size

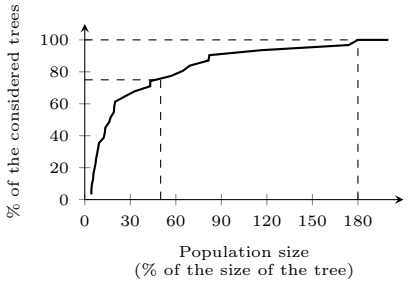

Fig. 2: Reasonable choice for population size

We have conducted experiments 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 analyzed 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. Measurements were taken with the same crossover operator and mutation rate. Fig. 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 Jürgenson and Willemson in [8] for their ApproxTree model, was a reasonable choice.

Lenin, Willemson and Sari have shown that the crossover operations take 90-99% of the time required to run the analysis [9]. Fig. 3 shows the time measurement for the suggested GA, depending on the size of the population.

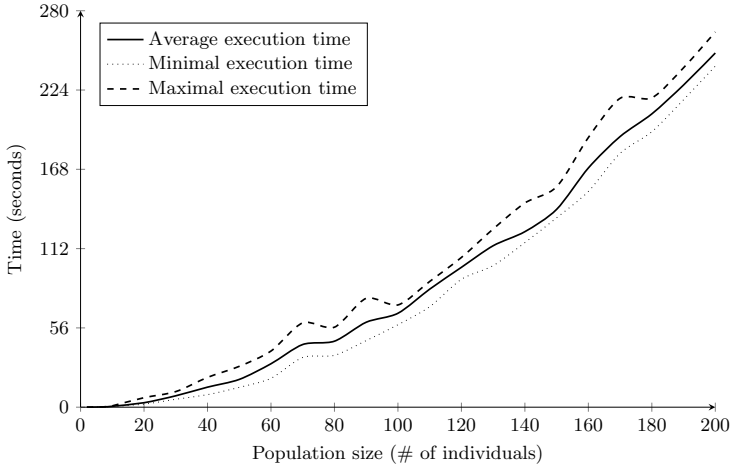

Fig. 3: Population size effect on GA execution time.

The *fitness function* is the model-specific utility function for the corresponding type of the security game. For further details we refer the reader to the detailed descriptions of the security games [7–11].

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. We have chosen 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 and 5 demonstrate the differences between the convergence speeds resulting from using various

crossover operators. It can be seen that the considered crossover operators do not have any major differences nor effect on the convergence speed of the GA.

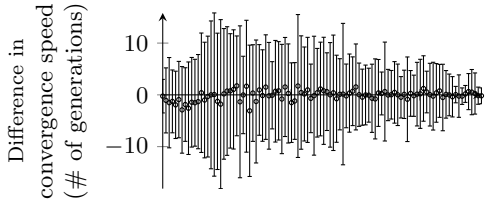

Fig. 4: Uniform crossover compared to single point crossover

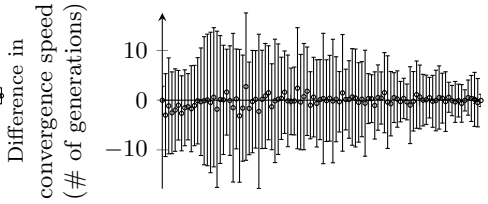

Fig. 5: Uniform crossover compared to two point crossover

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 2.

Algorithm 2: 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 the individual i in the population do
    forall the 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;
        end
    end
end
add the set of new individuals to the population;
```

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 our 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 corresponding 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) we mutate just these bits. Fig. 6 demonstrates the mutation rate effect on the utility function for the case of a specific attack tree with 100-leaves with initial population of 50 individuals. It shows that when the mutation rate exceeds value 0.1 GA turns into a random search algorithm, thus it is reasonable to keep the mutation rate rather small. We have conducted similar experiments on a larger set 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.

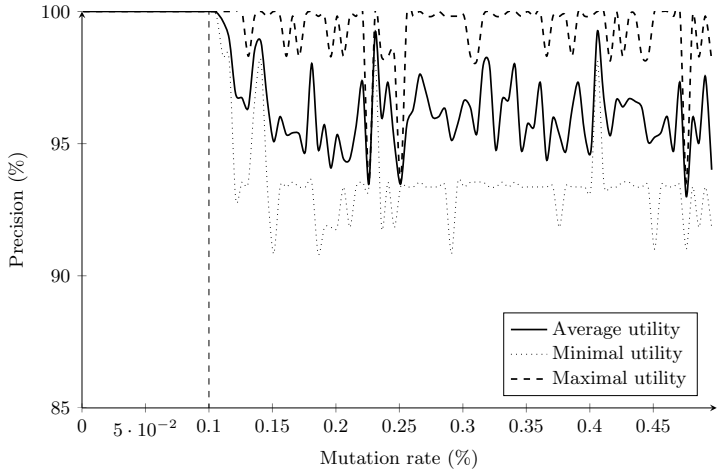

Fig. 6: GA mutation rate effect

It is important to determine the practical applicability boundaries for the suggested method. By practical applicability we mean the maximal size of the attack tree, which the computational method is capable of analyzing in reasonable time set to two hours. Extrapolating the time consumption curve in Fig. 7 we have come to a conclusion that theoretically the suggested GA is capable of analyzing attack trees containing up to 800 leaves in reasonable time. This is a major advancement compared

to the ApproxTree model [8] which would take more than 900 hours to complete such a task.

The execution time complexity estimations for GA are outlined in Table 1.

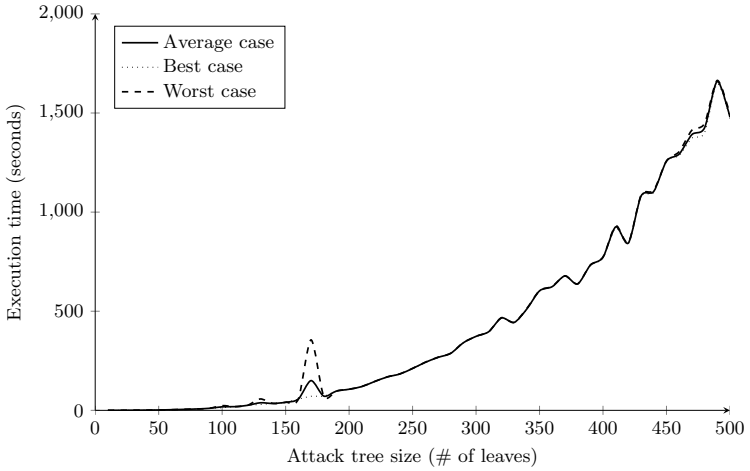

Fig. 7: GA execution time

Table 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

For comparison, the execution time complexity of the ApproxTree model [8] 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 paper 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.

4 Adaptive Genetic Algorithm (AGA)

We compare the genetic algorithm suggested in Section 3 to the adaptive genetic approach described in [13]. 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 subaverage 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. In this paper we apply the suggested method to the problem of the security games.

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. 8 shows the result corresponding to the computations using various population sizes in the experiment setup similar to the one for GA. In the case of GA the maximal value was stable with the increase in the population size, however in the case of AGA some fluctuations are present. Fig. 9 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. Based on these observations we can say 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.

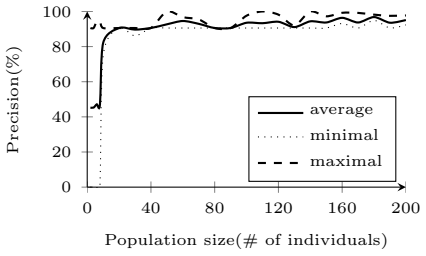

Fig. 8: Optimal population size

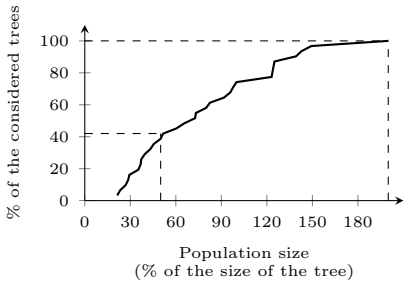

Fig. 9: Reasonable choice for population size

Similarly to the GA, we estimate the maximal size of the attack tree which AGA is capable of analyzing within reasonable timeframe set to two hours. Extrapolating the time consumption curve with the most extreme values trimmed out in Fig. 10 we have come to a conclusion that theoretically AGA is capable of analyzing attack trees containing up to

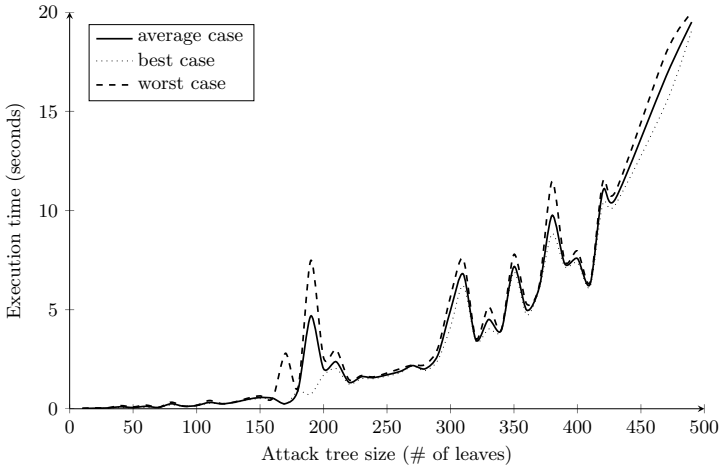

Fig. 10: AGA execution time

26000 leaves in reasonable timeframe, which is approximately 32 times more efficient compared to GA.

The execution time complexity estimations for AGA are outlined in Table 2.

Table 2: 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

5 Conclusions

This paper addressed the problem of efficient approximation of attack tree evaluation of the failure-free game. We considered the genetic approach to approximation, since it is known to have worked on similar problems previously. However, genetic algorithms depend on various loosely connected parameters (e.g. crossover and mutation operators and their corresponding rates). Selecting them all simultaneously is a non-trivial task requiring a dedicated assessment effort for each particular problem type. The current paper presents the first systematic study of GA parameter optimization for the attack tree evaluation. We have conducted a series of experiments and collected heuristic evidence for optimal parameter selection.

The second contribution of the paper is the application of adaptive genetic algorithms (AGA) to the problem domain of attack tree computations. It turns out that AGA converges generally faster than GA and provides similar level of accuracy, but with the price of potentially larger population sizes. Since usually there are no major technical obstacles to increasing the population, we conclude that AGA should be preferred to plain GA in the considered application domain.

References

1. Vesely, W., Goldberg, F., Roberts, N., Haasl, D.: Fault Tree Handbook. US Government Printing Office (January 1981) Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission.
2. Weiss, J.D.: A system security engineering process. In: Proceedings of the 14th National Computer Security Conference. (1991) 572–581
3. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobb's Journal **24**(12) (December 1999) 21–29
4. Mauw, S., Oostdijk, M.: Foundations of attack trees. In Won, D., Kim, S., eds.: International Conference on Information Security and Cryptology – ICISC 2005. Volume 3935 of LNCS., Springer (2005) 186–198
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. 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
8. 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
9. Lenin, A., Willemson, J., Sari, D.P.: Attacker profiling in quantitative security assessment based on attack trees. In Bernsmed, K., Fischer-Hübner, S., eds.: Secure IT Systems - 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings. Volume 8788 of Lecture Notes in Computer Science., Springer (2014) 199–212
10. Buldas, A., Stepanenko, R.: Upper bounds for adversaries' utility in attack trees. In Grossklags, J., Walrand, J.C., eds.: Decision and Game Theory for Security - Third International Conference, GameSec 2012, Budapest, Hungary, November 5-6, 2012. Proceedings. Volume 7638 of Lecture Notes in Computer Science., Springer (2012) 98–117
11. Buldas, A., Lenin, A.: New efficient utility upper bounds for the fully adaptive model of attack trees. In Das, S.K., Nita-Rotaru, C., Kantarcioglu, M., eds.: Decision and Game Theory for Security - 4th International Conference, GameSec 2013, Fort Worth, TX, USA, November 11-12, 2013. Proceedings. Volume 8252 of Lecture Notes in Computer Science., Springer (2013) 192–205
12. Lenin, A., Buldas, A.: Limiting adversarial budget in quantitative security assessment. In Poovendran, R., Saad, W., eds.: Decision and Game Theory for Security - 5th International Conference, GameSec 2014, Los Angeles, CA, USA, November 6-7, 2014. Proceedings. Volume 8840 of Lecture Notes in Computer Science., Springer (2014) 153–172
13. 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