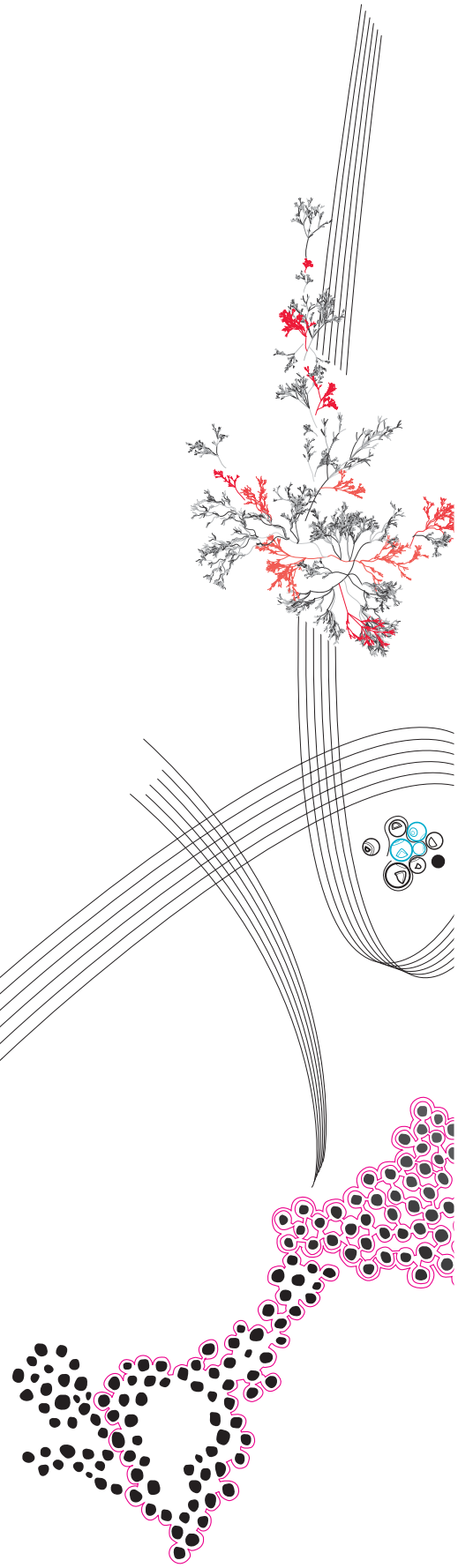

BSc Thesis Applied Mathematics
(and Technical Computer Science)

Unveiling the hidden threats

An approach to solve the cost metric of an Attack Tree

Marijke van Iperen

Supervisors:
Dr. M.A. Lopuhaä-Zwakenberg
Dr. A. Antoniadis
Y. Meng

February, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

This section is dedicated to expressing special thanks of gratitude, for the ones that contributed to the thesis.

Firstly, Dr. M.A. Lopuhaä-Zwakenberg is my supervisor on the computer science side. He proposed the subject of attack trees. He always put his enthusiasm into this project and every meeting. He was the one supervisor that was very critical on the material. Thereby he allowed me to develop a critical attitude.

Secondly, Dr. A. Antoniadis is my supervisor on the applied mathematics side. He has a critical view of presenting a mathematical subject on paper and poster.

Thirdly, Y. Meng is my second supervisor on the applied mathematics side for her feedback on the thesis and poster.

Lastly, thanks to family and friends for their support and for allowing me to do this study. Special thanks to Tudor for being my rubber duck and grammar expert.

I am grateful for all of them!

It was a big learning experience working in a research domain!

Thanks again to all who helped.
Kind regards,

Marijke van Iperen

Unveiling the hidden threats

van Iperen M.M.A.P.*

February, 2024

Abstract

In light of the growing number of cyber threats targeting critical infrastructure, it is crucial to assess the security of systems. Attack trees offer a powerful visual framework for categorizing possible avenues of attack. By delving into attack tree metrics, such as minimum cost, potential vulnerabilities can be pinpointed and gauge the level of resources required for an attack. The aim is to investigate strategies for calculating the minimal cost metric of attack trees.

The study is conducted using a self-created Python model for attack trees. In order to try different algorithms as a trial. Those algorithms are later formally proven. Thereby the existing bottom-up approach is analyzed and translated into a heuristic and two exact approaches are explored. For the latter, the concept of multi-parent nodes is defined to address the problem.

In this paper, two results are presented. Firstly the limit of the bottom-up algorithm solution for AT is defined. An example is given to illustrate the worst-case. Additionally, two exact approaches are presented for keeping track of different sets of nodes, specifically the basic attack step (BAS) nodes or the multi-parent nodes. Additionally, it is proven that the latter is better because it allows us to keep track of less nodes. Furthermore, both result sets are reduced by only retaining the potential or obvious superior sets.

The implications of this research are threefold. Firstly, this paper may be used for future research to generalize the limits of the bottom-up algorithm. Secondly, additional enhancements to the exact approach are possible. Thirdly this can be used as a base to implement an algorithm to solve the security metric of an attack tree.

Keywords: attack trees, bottom-up, multi-parent, minimal cost

*Email: marijkevaniperen@gmail.com

Contents

1 Introduction

History This paper is about attack trees(AT), this is a relatively new area. One of the first who wrote about it is B. Schneier, an internationally renowned security technologist, who addressed the importance of cybersecurity in society. He wrote about the subject of attack trees in 1999 in the Blog [5]. Before the '90s there are relatively no articles about attack trees. There are several reaseons for that:

Firstly, the flaws of security were thought of as unimportant in the early days. The importance increased in recent years, especially during COVID-19[1].

Secondly, computer systems were not as complex as they are nowadays. In the sense that if people did understand how a computer worked, they did this to a deeper level. Through the years more abstract software/programs/hardware were developed. And nowadays even a computer scientist might not be able to tell all the details, simply because of too many abstraction levels, nuanced and specific implementations, it became a substantially more complex system. Therefore models are needed to analyse cybersystems.

Thirdly, in the field of cybersecurity, attack trees serve as a valuable tool for quantitative analysis, offering a systematic way to assess and prioritize potential threats.

Therefore abstract methods are needed to analyse cybersecurity systems, one example being attack trees. An attack tree can be defined in multiple ways. In this article the same notation as in the article from Milan Lopuhaä-Zwakenberg and Mariëlle Stoelinga is used. Defining a model is the first step, but the efficient and correct analysis is the next one. Milan Lopuhaä-Zwakenberg introduced some algorithms in the articles, [2],[4], [3]. One of the algorithms is the bottom-up algorithm [2, Algorithm 1], which runs in polynomial time. This paper is based on this algorithm. The reader might recognise the structure of the algorithm from other graph problems, for example, the isomorphism problem.

Implication First, a bit more about attack trees, AT is a graphical security model which allows us to systematically categorize the different ways in which a system can be attacked. Despite their name, attack trees are directed, acyclic, rooted, finite graphs rather than trees. The graph consists of leaves, representing basic attack steps(BAS). The root is the goal to be reached. The intermediate nodes are OR/AND gates representing how they depend on their children. An example is given in Figure 1

In figure 1, an attack tree is depicted, it represents an attack on a safe, for illustration purposes. An attacker has the opportunity to activate a set of BAS nodes. Those possible options are code, hammer and screwdriver. This can lead to a successful attack on the safe if it reaches the root node. An AT can be treated as a sequence of logical operations. Examples of successful attacks are, code, screwdriver, hammer, code, screwdriver, hammer. Examples of unsuccessful attacks are, code, screwdriver.

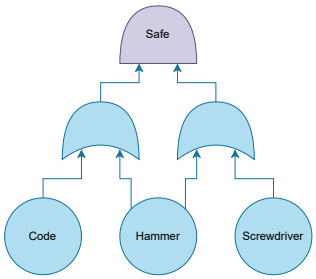

FIGURE 1: Visualisation of the system as an attack tree.

Imagine that it is desired to compare the security of this safe to the one of another safe. The other safe has different vulnerabilities, this makes it difficult to compare their securities. Therefore there exist security metrics, there are many kinds, and one of them is the minimal cost metric. This security metric can be seen as the value given to the attack tree to depict how secure the cybersecurity of the system is. This Figure 1 is a relatively simple version of an attack tree. It is easy to consider all the options and find the metric value. As given in the history paragraph, systems are getting more complex and therefore an algorithm is desired.

Literature There is not an abundance of related articles. The most closely related article that had been found is the article [3]. The main difference of the current paper and the one mentioned above is in that the later one a heuristic is drawn with Pareto front, where the outcome is still unknown. In this paper, the analysis focuses on where it goes wrong in the system and a few suggestions are given on what to do about it, including a formal algorithm to find exact security metrics.

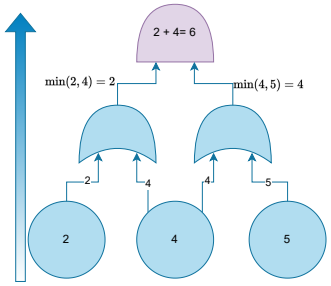

FIGURE 2: Visualisation of the bottom up.

Bottom-up algorithm The bottom-up algorithm [2, Theorem 4.4] is the basis of this research paper. It is an algorithm that goes from bottom to top following paths that start at leaves and end at the root. It runs in polynomial time.

The downside of the bottom-up algorithm is that it only gives the correct security metric for tree-shaped AT. Besides the fact that it is well known to give insufficient value for the security metric for non-tree-shaped ATs. Note that it is unknown how insufficient those results are. As given in the example 2, the bottom-up algorithm finds the value 6, while it is 4. As seen in the figure, the node with cost 4 is considered twice, even though that is not the optimal decision. Therefore a deviation is obtained instead. Later on such a node will be called a multi-parent node. To illustrate the multi-parent problem this paper uses some new terminology specific to the subject.

This raises the question: to what extent does the bottom-up algorithm deviate from the exact metric for an attack tree in the case that the minimal cost metric is used? The follow-up question is: if the (theoretical) deviation that is found is ever (practically) reached.

Lastly, the bottom-up algorithm can be altered to an exact algorithm, by keeping several different paths. One way is to do that with basic attack step(BAS) nodes. Another way is to keep track of multi-parent nodes, a new concept that narrows down up until the problem introduced in this paper. It will be shown that the multi-parent nodes strategy might be slightly better, due to the number of nodes to keep track of. Additionally, two improvements for both algorithms are shown, to keep the potential and keep the promising ones.
This article will dive deeper into the minimal cost security metric and, where possible, generalise.

2 Preliminaries

2.1 What is an Attack tree formally?

As stated in the introduction, an attack tree is a graph with AND, OR and BAS nodes. Where BAS is a leaf of the tree, without in-going edges. There exists only one vertex without outgoing edges, namely the root of the graph. Furthermore, in this paper only static attack trees are considered, thus with an attack tree, we refer to the static version of the problem. Below, a formal definition of an attack tree is provided.

Definition 2.1. An *attack tree* (AT) is a rooted, directed, finite, connected, acyclic graph $T = (N, E, \gamma)$ where N is the set of nodes, E is the set of edges, and the *node-type function* is defined as $\gamma : N \to \{AND, OR, BAS\}$. A node is denoted by BAS, modelling a *basic attack step* if and only if it is a leaf of the AT, so without in-going edges.

This definition is drawn from [2, Definition 2.2] and [4, 3, Definition 1]. Note that in this article, v is denoting a node in the AT, $v \in N$.
Whereby $e = (p, c)$, $p, c \in N$ and $e \in E$, where p is called the parent on the tail side and c is called the child on the head side. Additionally, it can be said that $\exists p \in N$, such that p is called a leaf (BAS) node i.e. $\gamma(p) = BAS$ and $\nexists c \in N$ such that c is a leaf. An AT has a few functions, definitions 2.2, 2.3 and 2.4 are describing them:
To recognize if a node is a leaf the child function is specified.

Definition 2.2. The *child function* is defined as $ch : N \to \mathcal{P}(N)$, where $\forall p \in N$, $N(p) = \{c \in N | (c, p) \in E\}$.

And so then a BAS v of the AT is when $ch(v) = \emptyset$.
To search for the parent of a child, the parent function is specified.

Definition 2.3. The *parent function* is defined as $ch^{-1} : N \to \mathcal{P}(N)$, where $\forall c \in N, ch^{-1}(c) = \{p | (p, c) \in E\}$.

To recognize the root of an AT the following property holds.

Definition 2.4. The root node R is defined as the node of the AT, one without any outgoing edges. This node is unique in an AT because $ch^{-1}(R) = \emptyset, R \in N$.

The following lemmas are exploring properties of AT structure.

Lemma 2.5. *$\forall T, \exists v \in N$, such that $\gamma(v) = BAS$.*

Proof of Lemma 2.5. Direct proof, every directed acyclic graph has a leaf and therefore a BAS. □

Lemma 2.6. *$\forall v \in N, ch^{-1}(v) \neq \emptyset$ if and only if $v \neq R$*

Proof of Lemma 2.6. This follows directly from the definition of AT and root node. □

Lemma 2.7. *An $\gamma(R) = BAS$ if and only if $\{R\} = N$.*

Proof of Lemma 2.7. Lemma 2.7 directly follows from root node and BAS properties. □

Definition 2.8. An *attribute function*, represents the cost of a BAS, thus $\alpha : N_{BAS} \to V$, V is a space, and $N_{BAS} = \{v | ch(v) = \emptyset, v \in N\}$

Corollary 2.9. *Given $v \in N$, then $\gamma(v) = BAS$ if and only if $ch(v) = \emptyset$*

Proof of Corollary 2.9. This follows straight from the definition of BAS node. □

Definition 2.10. An sub AT T_v from AT T and the node v is defined as the AT left over by keeping all the paths from BAS to v, such that v is the new root node.

2.2 Attack

In this section the necessary definitions of attack trees are given. This is needed to have a structural concept of attack.

Definition 2.11. An *attack* A on T is a set of BAS nodes. Those nodes in such a set are referred to as attacker-activate nodes.

Definition 2.12. The *set of all attacks* is denoted by $\mathbb{A}$.

For Boolean's is used $\mathbb{B} = \{0, 1\}$, where 0 means false and 1 true.

Definition 2.13. The *strucutre function* $f_T : N \times \mathbb{A} \to \mathbb{B}$ of an AT T is given by

$$f_T(v, A) = \begin{cases} 1 & \text{if } \gamma(v) = \text{OR and } \exists u \in \text{ch}(v), f_T(u, A) = 1 \\ 1 & \text{if } \gamma(v) = \text{ AND and } \forall u \in \text{ch}(v), f_T(u, A) = 1 \\ 1 & \text{if } \gamma(v) = \text{ BAS and } v \in A, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.14. An *successful attack* A on T is an attack that reach the root through the AND and OR gates, meaning $f_T(R, A) = 1$

Definition 2.15. A is minimal if A is successful and $\neg\exists A', A' \subset A$, where A' is successful on T.

Definition 2.16. $[\![T]\!]$ is the set of all minimal attack of an AT T.

2.3 Quantitative analysis

In the section that follows the semi-ring structure is defined. It allows us to have multiple security metrics. The benefit of having multiple security metrics is to be able to analyse multiple aspects of a system, on an abstract level. This abstract-level makes sure that quantitative analysis can be done on multiple trees at the same time.

Definition 2.17. A *semi-ring* D, is defined as $D = (V, \triangledown, \triangle)$ Where the parameters are defined as V is the domain with disjunctive operator $\triangledown : V^2 \to V$ and conjunctive operator $\triangle : V^2 \to V$. Both operators are associative and commutative. Moreover $\triangle$ is distributes over $\triangledown$.
When translating the word to math, the following is deducted,
$\forall x, y, z \in V, x \triangle (y \triangledown z) = (x \triangle y) \triangledown (x \triangle z)$

Below the attribute function defined, this is for every T and security metric defined differently. An attack tree can have multiple attribute function, in order to analysis different aspects of the system by the security metric.

Definition 2.18. For an AT T, an attribute function α is defined as $\alpha := N_{BAS} \to V$, where V is a space and N_{BAS} are the BAS nodes in N.

Definition 2.19. Given AT T and semi-ring D and α an attribution on V.
The *security metric* for $[\![T]\!]$ associated to α and D is given by: Where $[\![T]\!]$ is a set of successful attacks, $\hat{\alpha}([\![T]\!]) = \bigtriangledown_{A \in [\![T]\!]} \bigtriangleup_{a \in A} \alpha(a)$.

As seen in the previous section each node of an AT has a type defined by the function γ. Each of those types is attached to an action. In the case of minimal cost security metric ($M = (\mathbb{N}, \min, +)$) the semi-ring is defined as the OR gate $\triangledown$ is the minimum, AND gate $\triangle$ adds the incoming signals and the BAS node contains a value in the $\mathbb{N}^+$. A different security metric requires a different semi-ring to generalise the algorithms.
For any semi-ring defined on an attack tree, although operators are defined, the most important part is that the minimal attack is a successful attack with the best metric solution of the attack tree. The term best is for every metric different, but in the case of the minimal cost metric, it is the smallest value.

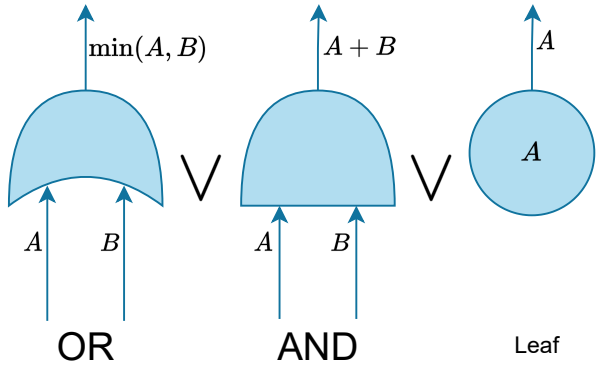

FIGURE 3: Minimum cost metric, M

The security metric is an aspect of AT. This aspect gives a level of security for a certain security metric. There exists a variety of security metrics. The relevant ones that can solved with this paper are chosen. Those are those which can have an extension by V^+. All those security metrics are from the paper [2]. The security metrics are expressed as semi-ring. An extension might be needed in some algorithms. The meaning of the extension is that they can form a group so if $(V^+, \triangle)$ is a group, the extension is only used for the algorithms.

Additionally, e is denoted to be the neutral element of the space V.
A few possible security metrics are defined below.

METRIC	V	V^+	∇	Δ
min cost	$\mathbb{N}$	$\mathbb{Z}$	min	$+$
min time (sequential)	$\mathbb{N}$	$\mathbb{Z}$	min	$+$
max damage	$\mathbb{N}$	$\mathbb{Z}$	max	$+$
discrete prob.	$[0, 1]$	$\mathbb{R}^+$	max	$\cdot$
continuous prob.	$f : \mathbb{R} \to [0, 1]$	$\exists f^{-1} : [0, 1] \to \mathbb{R}$, $e : \mathbb{R} \to 1$	max	$\cdot$

3 Multi-parent

3.1 Definitions

The following terminology is defined so that later, more complex and nuanced ideas can be proved using these terminology. Example of objects defined in the section below are the maximum number of paths and maximum multi-parent nodes path, these structures make a lot of sense visually and can help the reader build quite some intuition about them. On the other hand, to make proofs using them they need to be defined correctly, from a mathematical stance.

Definition 3.1. An AT, with $|N| - 1 = |E|$, which results in a tree is called a *tree-shaped* AT.

A child with two or more parents is defined as a multi-parent node because it has multiple parents. The proper definition:

Definition 3.2. A *multiple-parent* (mp) $p \in N$ is a node with multiple-parents, meaning $ch^{-1}(p) > 1$. The set of all mp nodes is denoted by P.

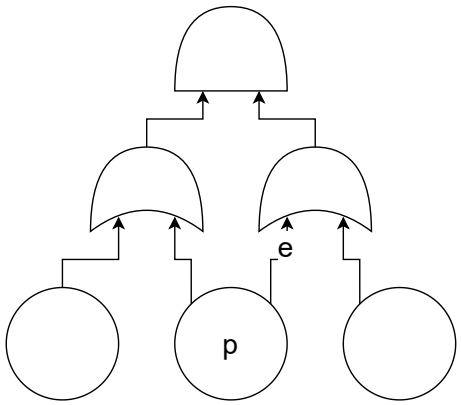

FIGURE 4: Visualisation of a multiple parent.

In Figure 4 an example of p is depicted. Without the edge e, p is not element of the set of multi-parent nodes. Additionally for the AT without the edge, e, the AT is *tree-shape* AT. A normal bottom-up algorithm BU as given in Algorithm 1 which, for tree-shaped ATs, computes the correct security metric in polynomial time. For ATs which are not tree-shaped, the algorithms introduced later are needed to obtain an exact solution.

Definition 3.3. Let the number of outgoing connections for a node v be defined as $|ch^{-1}(v)|$

The following lemma is needed to show that the mp is the problem, for analysing security metrics.

Lemma 3.4. *$\exists p \in P$ if and only if the AT is non tree-shaped AT.*

Proof of Lemma 3.4. This Lemma require a proof from two ways. In order to proof the if and only if statement.

- If there $\exists p \in P$, then $|ch^{-1}(p)| > 1$ and therefore there $\exists e_1, e_2 \in E$, such that (p_1, p), $(p_2, p) \in E$, respectively, with $\exists p_1, p_2 \in N$. Meaning that there is more than one outgoing edge. Since AT is a connect, direct graph, $|N| - 1 < |E|$, therefore non-tree-shape AT.

- If the AT is non-tree-shape AT and thus has $|N| - 1 < |E|$, there $\exists p \in \mathbb{N}$, such that $ch^{-1}(p)$.

That conclude the proof

□

For future proofs, the definition of the maximum multi-parent path is defined as.

Definition 3.5. Let q_v and w_v be a finite list of nodes in N that is a path from a node $n \in N$ to the root. w_v is called the maximum multi-parent path of an node N, when $\forall q_v, \prod_{p \in w_v} |ch^{-1}(p)| \geq \prod_{p \in q_v} |ch^{-1}(p)|$.
This maximum multi-parent path of a node $n \in N$, is depicted by $\mu(n) = \prod_{p \in w_v} |ch^{-1}(p)|$.
Where in the case of the maximum multi-parent path, from R root to root is defined as $\mu(R) = |ch^{-1}(R)| = 1$

Definition 3.6. Let q and w be a finite list of nodes in N that is a path from a BAS node to root. w is called the maximum multi-parent path of an AT T, when $\forall q, \prod_{p \in w} |ch^{-1}(p)| \geq \prod_{p \in q} |ch^{-1}(p)|$.
Where in the case of the maximum multi-parent path, from a minimal AT(only the root node) is defined as $|ch^{-1}(R)| = 1$

w might not be unique see Figure 4

Definition 3.7. Given AT T, then let $\beta : N \to \mathbb{N}$ and $n \in N$,
$\beta(n)$ gives the number all possible paths between n and R. This can be defined recursively, by $\beta(R) = 1$ and $\beta(n) = \sum_{u \in ch^{-1}(n)} \beta(u)$

Definition 3.8. Given definition 3.7 and a node $k \in N$, $\beta(k)$ is called the maximal number of path if $\forall n \in N$, such that $\beta(k) \geq \beta(n)$.

4 Bottom up algorithm

In the next section, the bottom-up algorithm will be defined and shown to give a correct solution for tree-shaped ATs. After that, an approximation analysis will be done on the bottom-up algorithm use on the non-tree-shaped AT. The interest lies in establishing a bound on the deviations in results. Lastly, it will be shown that this bound is indeed met practically, the so called worst case bound.

4.1 Bottom up for tree shape AT

Below the sub-routines of the bottom-up algorithm are defined. The bottom-up algorithm is introduced in paper [2] and also used in [3]. Besides the idea of the bottom-up is used in the attack-tree domain. It is already used in quite some other graph problems, including the graph isomorphism problem. Of course, other operators are used to give the desired result. The flaw of the algorithm lies in the fact that the mp node is visited multiple times.

The Algorithm 1 shows the bottom-up algorithm, a recursive algorithm that finds the result. The algorithm is split up into different sub-actions. In such a way that the algorithm can easily be reused in later sections.

Algorithm 1: Bottom up for tree shape AT

input : T, v, D, α
output: $opt := opt \in V$
Function bu(T, v, D, α):
  if $\gamma_T(v) = BAS$ **then**
    result $\leftarrow$ BAS(T, v, D, α, bu)
  else if $\gamma_T(v) = OR$ **then**
    result $\leftarrow$ OR(T, v, D, α, bu)
  else if $\gamma_T(v) = AND$ **then**
    result $\leftarrow$ AND(T, v, D, α, bu)
  result $\leftarrow$ AFTER(result, v, bu)
  return : result

Algorithm 2: BAS function for tree shape AT

output: *opt*
Function BAS($T, v, D, \alpha,$ bu):
  return : $\alpha(v)$

Algorithm 3: OR function for tree shape AT

output: *opt*
Function OR($T, v, D, \alpha,$ bu):
  return : $\bigtriangledown_{u \in ch(v)}$ bu$(T, v, D, \alpha,$ bu$)$

Algorithm 4: AND function for tree shape AT

output: *opt*
Function AND($T, v, D, \alpha,$ bu):
  return : $\bigtriangleup_{u \in ch(v)}$ bu$(T, v, D, \alpha,$ bu$)$

The after function is quite trivial, but will be used in later sections.

Algorithm 5: AFTER function for tree shape AT

output: result

1 Function `AFTER`$(T, v, D, \alpha,$ `bu`$,$ *result*`)`**:**

  ⌊ **return :** result

Theorem 4.1. *Let T be an tree-shaped AT, α an attribution on V , and $D = (V, \triangledown, \triangle)$ a semi-ring attribute domain. Then $\hat{\alpha}(T) = \texttt{bu}(T, R, \alpha, D)$.*

Theorem 4.1 holds because [2, Theorem 4.4], this paper reference to multiple ways to prove this theorem. Additionally in the paper [3] more research is done related to the bottom-up algorithm, but still in approximation form. This article continues with the goal of finding the exact answer. Further, it appears that the bottom-up algorithm deviates from the exact answer. To what extent this does deviate becomes clear in the next section.

4.2 Approximation analysis

In the next two subsections analysis of deviation of the bottom-up algorithm `bu` for normal AT is analysed. When the bottom-up algorithm is translated to a dynamic program, the algorithm runs in polynomial time to determine the security metric. For tree-shaped AT, this is an exact security metric of the AT, but for non-tree-shaped AT there is a deviation. The focus is to present the issue and analyse how much it deviates from the optimal security metric. So, first, it would be shown what the deviation is. After that, it would be shown that it can construct a worst-case scenario for that deviation.

4.2.1 Worst case analysis of the BU

It turns out that the length of the maximum mp path is the bottleneck. This theorem is proven by contradiction, due to the fact that if there existed more possibilities to split up the multiplicity of the nodes would have been higher.

Definition 4.2. Let the split AT be defined as T_{split} from AT T, this AT is created as follows.
Let P be the set of all mp T and $p \in P$, such that p has no mp children and let T_p a sub-tree of T, by definition 2.10.
Let $ch^{-1}(p) = \{c_1, \ldots, c_k\}$. Then replace T_p with k copies. These copies are connected to the main AT by connecting c_i only to the root of the i-th copy. Then p is no longer present in P since it is a non-mp. There might be new mp nodes due to copying of mp nodes. This is repeated until $P = \emptyset$. Then T_{split} is obtained.

In order to have an easy mapping between the original and the copied version, the function π is defined.

Definition 4.3. The function $\pi : N_{T_{split}} \to N_T$ maps all copies of nodes in T_{split} to the corresponding original node in T.

In this paper is for simplicity chosen to simplify notation for π. First, way let $B \subseteq N_{split}$, then $\pi(B) = \bigcup_{b \in B} \pi(b)$. Second way, let T by the attack tree and T_{split} the split version, then $\pi(T_{split}) = T$. The inverse is defined similarly. Besides the mapping per node, a similar definition for the attribute function.

Definition 4.4. Define the attribute split function as α_{split}, as $\alpha_{split} = \alpha \circ \pi$.

Below there are several lemmas given, those lemmas are needed to prove Theorem 4.10. It is a direct proof that follows from all the lemmas.
The following lemma provides a connection between an attack on the original AT and the copied version. The opposite might not be true, take for example an AT with only OR gates and one OR gate mp node. Then the attack of the split version is smaller because only one node is required to fulfil the attack. This lemma is proven by induction from bottom to top following the BAS nodes up to the root. This lemma is later used in other proofs of lemmas and theorems.

Lemma 4.5. *If A is a successful attack on T_{split}, then $A_{split} = \bigcup_{a\in A} \pi^{-1}(a)$ is a successful attack on T_{split}*

Proof of Lemma 4.5. Let T be an AT and the split version T_{split}, a successful attack A, an attack $A_{split} = \bigcup_{a\in A} \pi^{-1}(a)$ and the structure-function $f_T(R, A) = 1$.

$$f_T(v, A) = \begin{cases} 1 & \text{if } \gamma(v) = \text{OR and } \exists u \in \text{ch}(v), f_T(u, A) = 1 \\ 1 & \text{if } \gamma(v) = \text{ AND and } \forall u \in \text{ch}(v), f_T(u, A) = 1 \\ 1 & \text{if } \gamma(v) = \text{ BAS and } v \in A, \\ 0 & \text{otherwise.} \end{cases}$$

Let $v_{split} \in N_{split}$, equals $\pi(v_{split}) = v$.
Then, $f_{T_{split}}(v_{split}, A_{split}) = f_{T_{split}}(\pi^{-1}(v), \bigcup_{a\in A} \pi^{-1}(a))$, by definition of π.
The statement that needs to be proven is:
$f_{T_{split}}(v_{split}, A_{split}) = f_{T_{split}}(\pi^{-1}(v), \bigcup_{a\in A} \pi^{-1}(a)) = 1$
In order to do that an induction proof is done from the bottom to the top. If the statement is true then A_{split} is a successful attack.

Basis step: In case of $\gamma(v_{split}) = BAS$ and $\pi(v_{split})$, $\forall u \in A, v_{split} \in \pi^{-1}(v)$
thus, $v \in N_{BAS}$. Therefore $f_{T_{split}}(u_{split}, A_{split}) = 1$, since both conditions hold.

Induction step: Suppose that $f_{T_{split}}(u_{split}, A_{split}) = f_{T_{split}}(\pi^{-1}(u), \bigcup_{a\in A} \pi^{-1}(a)) = 1$ is true $\forall u_{split} \in ch(v_{split})$ and $u = \pi^{-1}(u_{split})$, where $f_T(u, A) = 1$. Then it needs to be shown for v.

Case If $\gamma(u) = OR$
$f_{T_{split}}(v_{split}, A_{split}) = \mathbb{1}(\exists u_{split} \in ch(v_{split}), f_{T_{split}}(u_{split}, A_{split}) = 1)$
because $\pi(u_{split}) = u$. By induction statement $\exists u \in ch(v), f_T(u, A) = 1$, if this is not the case then A wouldn't be a valid attack. Therefore by induction hypothesis, there exists such an u_{split} for which the statement is true in the indicator function.
$\mathbb{1}(\exists u_{split} \in ch(v_{split}), f_{T_{split}}(u_{split}, A_{split}) = 1)) = 1$
Therefore $f_{T_{split}}(v_{split}, A_{split}) = f_{T_{split}}(\pi^{-1}(v), \bigcup_{a\in A} \pi^{-1}(a)) = 1$

Case If $\gamma(u) = AND$
$f_{T_{split}}(v_{split}, A_{split}) = \mathbb{1}(\forall u_{split} \in ch(v_{split}), f_{T_{split}}(u_{split}, A_{split}) = 1)$
because $\pi(u_{split}) = u$. By induction statement $\forall u \in ch(v), f_T(u, A) = 1$, if this is not the case then A wouldn't be a valid attack. Therefore by induction hypothesis, for $\forall u_{split} \in A_{split}$ the indicator function turns true.
$\mathbb{1}(\forall u_{split} \in ch(v_{split}), f_{T_{split}}(u_{split}, A_{split}) = 1)) = 1$
Therefore $f_{T_{split}}(v_{split}, A_{split}) = f_{T_{split}}(\pi^{-1}(v), \bigcup_{a\in A} \pi^{-1}(a)) = 1$

This proves the statement.

□

The following lemma provides a connection between the original AT and the copied version in relation with the bottom-up algorithm. This lemma is proven by induction from the root to the bottom.

Lemma 4.6. *Let T be an AT and T_{split} the split version, let $n \in N$ and $n_{split} \in N_{split}$ such that $\pi(n_{split}) = n$, then* $\mathtt{bu}(T_n, n, D, \alpha) = \mathtt{bu}(T_{nsplit}, n_{split}, D, \alpha_{split})$

Proof of Lemma 4.6. Let T be an AT and the split version T_{split}. Three cases need to be considered, $n \in N_T$ and $n_{split} \in N_{T_{split}}$where $\pi(n_{split}) = n$, this exist by Definition 4.2. A proof by induction on the nodes from bottom to top, such that it holds for any n.

Basis step: Take a node $n \in N$, where $\gamma(n) = \text{BAS}$, then $\mathtt{bu}(T_n, n, D, \alpha) = \alpha(n)$ and $\mathtt{bu}(T_{n_{split}}, n_{split}, D, \alpha_{split}) = \alpha_{split}(n_{split})$. It follows that, $\alpha_{split}(n_{split}) = \alpha(\pi^{-1}(n_{split})) = \alpha(n)$

Induction step: Given $k \in \mathbb{N}$ and $\forall k_{split} \in \pi^{-1}(k)$
Suppose that Lemma 4.6 is true for some $u \in ch(k)$ and $u_{split} \in k_{split}$ where $\forall \pi(u_{split}) = u$, that it also holds for all there children.

- If $\gamma(k) = \text{AND}$, then $\mathtt{bu}(T_k, k, D, \alpha) = \bigwedge_{u \in ch(k)} \mathtt{bu}(T_u, u, D, \alpha)$ and $\mathtt{bu}(T_{k_{split}}, k_{split}, D, \alpha_{split}) = \bigwedge_{u \in ch(k_{split})} \mathtt{bu}(T_{u_{split}}, u, D, \alpha_{split})$

- If $\gamma(k) = \text{OR}$, then $\mathtt{bu}(T_k, n, D, \alpha) = \bigvee_{u \in ch(n)} \mathtt{bu}(T_u, u, D, \alpha)$ and $\mathtt{bu}(T_{split}, k_{split}, D, \alpha_{split}) = \bigvee_{u \in ch(k_{split})} \mathtt{bu}(T_{u_{split}}, u, D, \alpha_{split})$
- By induction hypothesis,
  $\forall u \in ch(k), \exists u_{split} \in ch(k_{split}) \mathtt{bu}(T_u, u, R, D, \alpha) = \mathtt{bu}(T_{u_{split}}, u, D, \alpha_{split})$.

  Claim: There is at most one copy of u in this copy of T_k within T_{split}. Proof by contradiction if there exists $z_{split} \in ch^{-1}(v_{split})$ such that $\mathtt{bu}(T_u, u, R, D, \alpha) = \mathtt{bu}(T_{split_z}, z, D, \alpha_{split})$, then u would be a multi-parent node, which means that k is not a root node. Which is a contradiction of the statement.

  Thereby there is at most one copy of u in this copy of T_k within T_{split}. This is important because that means that the bottom-up from the normal T and the split version T, have the same number of sub-results. Thereby it also shows that for each sub-result there exist another similar result on the other side.
  Thereby $\bigwedge_{u \in ch(n_{split})} \mathtt{bu}(T_{split_u}, u, D, \alpha_{split}) = \bigwedge_{u \in ch(n)} \mathtt{bu}(T_u, u, R, D, \alpha)$ and $\bigvee_{u \in ch(n_{split})} \mathtt{bu}(T_{split_u}, u, D, \alpha_{split}) = \bigvee_{u \in ch(n)} \mathtt{bu}(T_u, u, R, D, \alpha)$

This proves the lemma □

The following lemma, is proven directly with the π function. The usefulness is determined to have some correlation between the security metric of the split and the original version.

Lemma 4.7. *let T be an AT and T_{split} the split version, A and $A_{split} = \bigcup_{a \in A} \pi^{-1}(a)$ be their respective attacks and the semi-ring is M. Then $\sum_{i \in A_{split}} \alpha \circ \pi^{-1}(i) = \sum_{i \in A} \alpha(i) \cdot |\pi^{-1}(i)|$*

Proof of Lemma 4.7. Given AT T and the split version T_{split}, each copy of $a_{split} \in N_{split}$ refer uniquely to one $a \in N$, because the definition of T_{split}. This is what the function π maps.

That means that, $\forall a \in A, \forall a_{split} \in A_{split}$,where $A_{split} = \bigcup_{a \in A} \pi(a)$ is a successful attack on T_{split} as in Lemma 4.5.

Where the function π lead to $\pi(a_{split}) = a, \alpha(a) = \alpha \circ \pi(a_{split}) = \alpha_{split}(a_{split})$.
Since the π function has a relation from many to one. Let $\forall a \in A, B_{split} := \pi^{-1}(a)$
this lead to $\sum_{i \in B_{split}} \alpha_{split}(i) = \sum_{i \in B_{split}} \alpha_{split}(\pi(i)) = \sum_{i \in B_{split}} \alpha(a)$
That means that, $|B_{split}| = |\pi^{-1}(a)|$, thus $\sum_{i \in B_{split}} \alpha(a) = |\pi^{-1}(a)| \cdot \alpha(a)$.
This proves that $\sum_{i \in \pi^{-1}(a)} \alpha \circ \pi^{-1}(i) = |\pi^{-1}(a)| \cdot \alpha(a)$.
If the sum is taken for all possibilities, then
$\sum_{i \in A_{split}} \alpha \circ \pi^{-1}(i) = \sum_{i \in A} \sum_{i \in pi^{-1}(a)} \alpha \circ \pi^{-1}(i) = \sum_{i \in A} |\pi^{-1}(a)| \cdot \alpha(a)$.
This proves the statement. □

Lemma 4.8, gives a relation between the number of parents and the number of copies the split version has. This theorem is proven by induction from root to leaves.

Lemma 4.8. *Let $n \in N$, then*

$$\beta(n) = |\pi^{-1}(n)| \tag{1}$$

Proof of Lemma 4.8. Let T be an AT and T_{split} the split version and the function β be over the T Then proof Lemma 4.8 with induction over root node down.

Basis step: $\beta(R) = 1$ and $|\pi^{-1}(R)| = |R| = 1$, since R has no parents. The basis step hold.

Induction step: Suppose that Lemma 4.8 is true for $\forall u \in ch^{-1}(t)$,
meaning: $\beta(u) = |\pi^{-1}(u)|$
Then it needs to prove for t.
By definition of β 3.7,
this holds directly by definition, it is recursively defined, $\beta(t) = \sum_{u \in ch^{-1}(t)} \beta(u)$
By induction hypothesis,$\sum_{u \in ch^{-1}(t)} \beta(u) = \sum_{u \in ch^{-1}(t)} |\pi^{-1}(u)|$.
By Definition 4.2, because the split function is recursively defined as first to make copies of the children and then from the parent. Thus this is concluded $\sum_{u \in ch^{-1}(t)} |\pi^{-1}(u)| = |\pi^{-1}(t)|$.

This proves the statement □

Lemma 4.9 is proven by induction, from top to bottom, following the different paths. This theorem gives an relation of between the maximum multi-parent path and the maximum number of paths from one node in AT.

Lemma 4.9. *Given AT T, then let w the maximum multi-parent path, and $k \in N$ the node with the maximum number of paths from one node to the root of the AT in T.*

$$\beta(k) \leq \mu(k) \tag{2}$$

Proof of Lemma 4.9. Given AT T, then let w the maximum multi-parent path of a node v, $v \in N$. Claim: $\beta(v) \leq \mu(v)$ Then the proof is done with induction from root to top.

Basis step: Let R the root node of T be the starting point of this induction.
Then $\beta(R) = 1$ and the $\mu(R) = 1$. Thus $\beta(R) = \mu(R)$, thus also $\beta(R) \leq \mu(R)$

Induction step: Suppose that Lemma 4.9 is true for $\forall u \in ch^{-1}(v)$,
meaning: $\beta(u) \leq \mu(u)$
Then it needs to be proven for v.
On the side, it is given that $\beta(v) = \sum_{u \in ch^{-1}(v)} \beta(u)$, by definition 3.7.
On the other side, $\mu(v) = \prod_{p \in w_v} |ch^{-1}(p)|$
$= |ch^{-1}(v)| \cdot \max_{u \in ch^{-1}(v)}(\prod_{p \in w_u} |ch^{-1}(p)|)$
$= |ch^{-1}(v)| \cdot \max_{u \in ch^{-1}(v)}(\mu(u)) \geq \sum_{u \in ch^{-1}(v)} \mu(u)$,
by definition 3.6 and some playing around with mathematical equations.
Now the induction hypothesis comes into play, and thus
$\sum_{u \in ch^{-1}(v)} \mu(u) \geq \sum_{u \in ch^{-1}(v)} \beta(u)$.
Together with the above derivation of the two sides, it is obtained that
$\beta(v) \leq \mu(v)$

This proves the lemma. □

Then finally the main theorems, with the building blocks obtained. It will proven that this theorem hold.

Theorem 4.10. *Let w be defined as the maximum mp paths as defined in 3.6. The deviation of the security metric of T calculated by* `bu` *(T, R, M, α) is at most by of the factor $\prod_{i \in w} |ch^{-1}(i)|$, where w is defined as 3.6. Meaning that the security metric a is the correct value but the solution of* `bu` *$(T, R, M, \alpha) \leq a \cdot \prod_{i \in w} |ch^{-1}(i)|$.*

Proof of Theorem 4.10. Let $T = (N, E)$ and attribute function α and M be the minimum cost or minimum time semi-ring. Let $a = \hat{\alpha}([\![T]\!])$ the cost metric of the AT. Furthermore, let the split form of T be T_{split} as defined in Definition 4.2.
Additional, let A be a successful attack on T, such that $\hat{\alpha}([\![T]\!]) = \sum_{a \in A} \alpha(a)$. Additionally, $A_{split} = \bigcup_{a \in A} \pi(a)$ a successful attack on T_{split} by Lemma 4.5.
By Lemma 4.6 $\texttt{bu}(T, R_T, M, \alpha) = \texttt{bu}(T_{split}, R_{T_{split}}, M, \alpha_{split})$
By the fact that T_{split} is tree-shaped [2, Theorem 4.4],
$\texttt{bu}(T_{split}, R_{T_{split}}, M, \alpha_{split}) = \hat{\alpha}_{split}([\![T_{split}]\!])$.

By definition of the minimal security metric, it follows $\hat{\alpha}_{split}([\![T_{split}]\!]) \leq \sum_{i \in A_{split}} \alpha_{split}(i)$.
By definition of π, it follows, $\sum_{i \in A_{split}} \alpha_{split}(i) = \sum_{i \in A_{split}} \alpha \circ \pi(i)$.
By Lemma 4.7, it follows $\sum_{i \in A_{split}} \alpha \circ \pi(i) = \sum_{i \in A} \alpha(i) \cdot |\pi(i)|$.
By Lemma 4.8, it follows $\sum_{i \in A} \alpha(i) \cdot |\pi(i)| = \sum_{i \in A} \alpha(i) \cdot \beta(i)$.
By Definition 3.8, it follows, for $k \in N$, which is the node with the maximum number of paths from one node to the root of the AT in T., $\sum_{i \in A} \alpha(i) \cdot \beta(i) \leq \beta(k) \cdot \sum_{i \in A} \alpha(i)$.
Then it follows by definition of attacks and security metric by using the fact that an attack consists of the attribute values of the chosen BAS nodes,
$\beta(k) \cdot \sum_{i \in A} \alpha(i) = \beta(k) \cdot \hat{\alpha}(A) = \beta(k) \cdot a$
By Lemma 4.9, $\beta(k) \cdot a \leq \prod_{i \in w} |ch^{-1}(i)| \cdot a$
This proves the statement. □

The following theorem states the obvious that in the case of the minimum cost security metric, the bottom-up algorithm does perform only to overestimate the minimum cost of an AT. To illustrate this give an AT T with minimum cost a. The bottom-up algorithm might return more than a but never less. This theorem is proven directly from the lemmas above.

Theorem 4.11. *Given T and T_{split} respectively. Then $M(T) \leq M(T_{split})$*

Proof of Theorem 4.11 . Let T AT and T_{split} the split version.
To prove Theorem 4.11, it needs to be shown $M(T) \leq M(T_{split})$. Let B minimum cost attack on T_{split}, then $\pi(B)$ is successful on T, and $\sum_{b\in B} \alpha(\pi(b)) \leq \sum_{b\in\pi(B)} \alpha_{split}(\pi(b))$.

- Let B be a minimum cost attack on T_{split},
  thus $f_{T_{split}} = (R, B) = 1$.
  This is equivalent to $f_{\pi(T_{split})} = (\pi(R), \pi(B)) = 1$, therefore $f_T = (R, \pi(B)) = 1$.

- Now it needs to be verified that the minimal cost of $\pi(B)$ is indeed smaller.
  $\sum_{b\in B} \alpha_{split}(b) = \sum_{a\in\pi(B)} \sum_{b\in\pi^{-1}(B)} \alpha_{split}(\pi(a))$
  because by writing out the statement to the attribute function of T
  and the function π.
  $\sum_{a\in\pi(B)} \sum_{b\in\pi^{-1}(B)} \alpha_{split}(\pi(a)) = \sum_{a\in\pi(B)} \sum_{b\in\pi^{-1}(B)} \alpha(a)$,
  because by the function π.
  $\sum_{b\in\pi(B)} \sum_{a\in\pi^{-1}(b)} \alpha(a) = \sum_{b\in\pi(B)} |\pi^{-1}(b) \cap B| \cdot \alpha(b)$,
  because there might be duplicates so it can be multiplied with the number of copies.
  Now the following is obtained by taking on the duplicates.
  Since they are only positive integers, in case of the minimal:
  $\sum_{b\in\pi(B)} |\pi^{-1}(b) \cap B| \cdot \alpha(b) \geq \sum_{b\in\pi(B)} \alpha(b)$

Therefore, combining the first point with definition of M.
It is obtained, $M(T) = \sum_{b\in\pi(B)} \alpha(b)$ and $M(T_{split}) = \sum_{b\in B} \alpha_{split}(b)$.
Together they concluded that $M(T) \leq M(T_{split})$.
That proves the statement. □

From Theorem 4.10 and 4.11, and a division of the maximum mp path, will lead to a lower bound. This makes sense from a practical perspective because the minimal cost might be overestimated, but never underestimated.

Corollary 4.12.

$$\frac{M(T)}{\prod_{i\in w} |ch^{-1}(i)|} \leq \frac{\texttt{bu}(T, R, M, \alpha)}{\prod_{i\in w} |ch^{-1}(i)|} \leq M(T) \tag{3}$$

Proof of corollary 4.12. This is a direct consequence of the above two theorems and dividing by $\prod_{i\in w} |ch^{-1}(i)|$

□

4.2.2 Lower bound

In this subsection a showcase of the lower bound of corollary 4.12, is demonstrated, by showing that there exists a result that equals $M(T) \cdot \prod_{i \in w} |ch^{-1}|$ in complexity. This example shows that the found bound indeed can appear in a practical situation.

The worst-case AT is built up from different parts in two directions. One part of the worst case of the AT is the blue basis depicted in Figure 5. The other part of the worst case of the AT is the green extension, depicted in Figure 6. This can be extended with a green extension depicted in Figure 7. The green extension can be added to the current root node of the blue AT. The green extension can be added similarly as depicted in Figure 8. Those green extensions will extend the number of mp nodes in the maximum mp path, as defined in Definition3.6. Besides the green extension, there is a red extension, this extension can be used to increase the number of parents of the p_i nodes for any green or blue part of the AT where p_i is the potential mp node for extension i.

In Figure 5, the basis AT is depicted.
Let the basis with red extension be defined as $G_{0,k_0} = (N_{G_{0,k_0}}, E_{G_{0,k_0}}, \gamma_{G_{0,k_0}})$, $k_0 \in \mathbb{N}$.
Let $N_{G_0,k_0} = \{n_{0,x,o} | 0 \leq x \leq k_0, x \in \mathbb{N}\} \cup \{p_1\}$
$\{n_{0,x,b} | 0 \leq x \leq k_0, x \in \mathbb{N}\}$
such that $\gamma_{G_0,k_0} = \{n_{0,x,o} \mapsto \text{OR} \cup p_1 \mapsto \text{AND} \cup n_{0,x,o} \mapsto \text{BAS}\}$.
The edges are added respectively,
such that $E_{G_0,k_0} = \{e_{0,x,0} | e_{0,x,0} = (n_{0,x,o}, p_0), 0 \leq x \leq k_0, x \in \mathbb{N}\} \cup$
$\{e_{0,x,1} | e_{0,x,1} = (n_{0,x,o}, n_{0,x,b}), 0 \leq x \leq k_0, x \in \mathbb{N}\} \cup$
$\{e_{0,x,2} | e_{0,x,2} = (p_1, n_{0,x,o}), 0 \leq x \leq k_0, x \in \mathbb{N}\}$
Further $\alpha_{G_0,k_0} = \{N_{0,x,b} \mapsto a \cup p_0 \mapsto a,\ a \in V\}$, so $\alpha(v) = a, \forall v \in N_{BAS}$
The process of adding nodes and edges is adding k_0 sets of 1 OR gate and 1 BAS node connected similarly on p_0, p_1. When $k_0 = 0$, the blue AT is obtained and p_0 is non-mp.

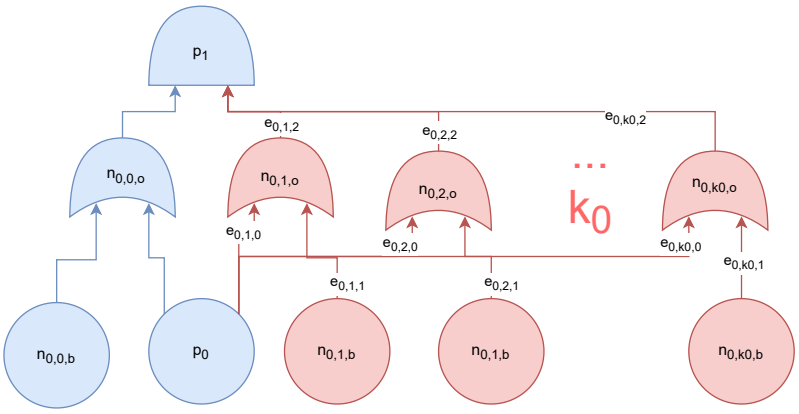

FIGURE 5: Visualisation of the basis of the worst case.

Theorem 4.13. *The blue AT (G_{0,k_0}) with k_0 red extensions, $k_0 \in \mathbb{N}$,*
gives for `bu` *(G_0, R_{G_0}, M, α) a deviation of the factor of the optimal security metric with $|ch^{-1}(p_0)|$. Meaning that the security metric $a = \hat{\alpha}([\![T]\!])$ is the correct value but the solution of* `bu` *$(T, p_1, M, \alpha) = a \cdot |ch^{-1}(p_0)|$ instead of a.*

Proof of Theorem 4.13. In Figure 5, the basis AT is depicted.
The claim to prove is Theorem 4.13
`bu` $(G_{0,K_0}, F_{G_0}, M, \alpha) = (a \triangledown a) \triangle (a \triangledown a) ... \triangle (a \triangledown a)(k_0{+}1 \text{ times}) = a \triangle a ... \triangle a(k_0{+}1 \text{ times}) = \sum_1^{k_0+1} a = a \cdot |ch^{-1}(p_0)|$ because $\triangledown := min$, $\triangle := +$ and $V := \mathbb{N}$.
$\hat{\alpha}([\![T]\!]) = a$ because the attack p_0 is a successful attack $f_{G_{0,K_0}}(R, \{p_0\}) = 1$
The security metric is less or equal a and thereby the fact that $\{p_0\}$ is a successful attack.This attack is minimal because there does not exist a smaller subset such that it is a successful attack on T. Hence, the security metric is exactly a. This proves the statement to be true.

□

Secondly, green extensions are added as depicted in Figure 6. The green extension has similar red extensions and is connected by the blue extension through the node p_1 as depicted in Figure 7.

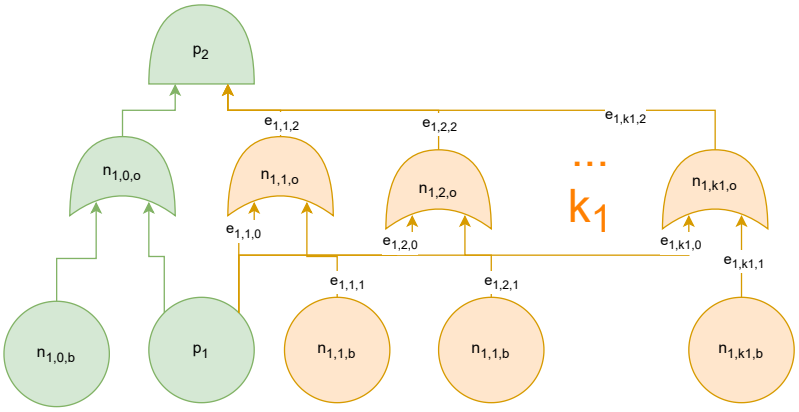

FIGURE 6: Visualisation of the extension of the worst case.

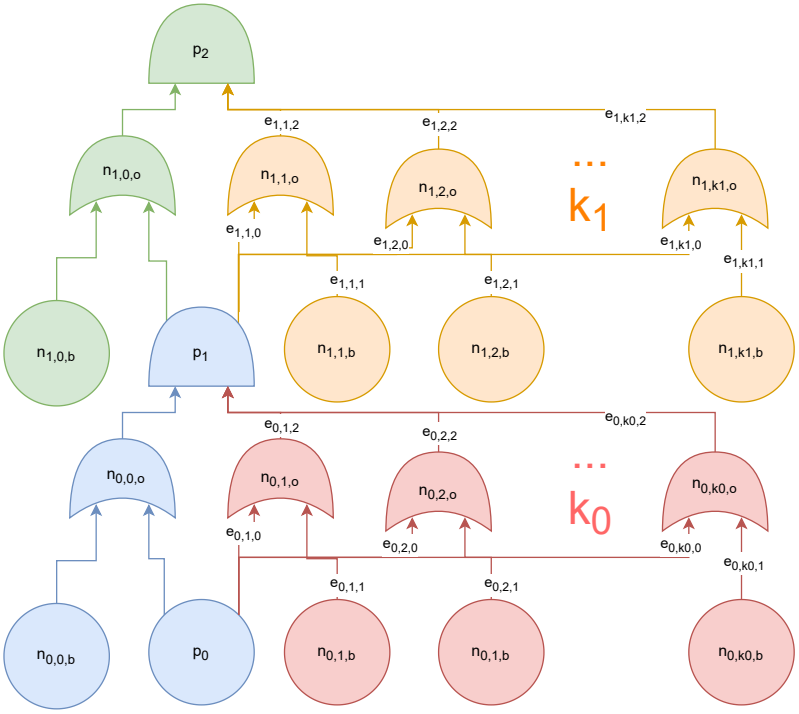

FIGURE 7: Visualisation of G_{1,k_0,k_1} the worst case.

When generalising that idea, that worst case AT can be defined
as $G_{n,k_0,...,k_n} = (N_{n,k_0,...,k_n}, E_{n,k_0,...,k_n}, \gamma_{n,k_0,...,k_n})$, where $n, k_0, ..., k_n \in \mathbb{N}$,
then the potential mp are defined as $p_i, \forall i \in \mathbb{N} i \leq n+1$.
The nodes and edges are defined as
$N_{n,k_0,...,k_n} = \{n_{z,x,w} | x, z \in \mathbb{N}, z \leq n, x \leq k_z, w \in \{o, b\}\}$ and
$E_{n,k_0,...,k_n} = \{e_{z,x,0} | e_{z,x,0} = (n_{z,x,o}, p_z) | \forall x, y, z \in \mathbb{N}, z \leq n, x \leq k_z\} \cup$
$\{e_{z,x,y} | e_{z,x,1} = (n_{z,x,o}, n_{z,x,b}) | x, z \in \mathbb{N}, z \leq n, x \leq k_z, \} \cup$
$\{e_{z,x,y} | e_{z,x,2} = (p_1, n_{z,x,o}) | x, z \in \mathbb{N}, z \leq n, x \leq k_z\}$.
As the last element of the AT, the type function is defined as
$\gamma_{n,k_0,...,k_n} = \{N_{z,x,o} \mapsto \text{OR} | N_{z,x,y} \in N\} \cup$
$\{p_j \mapsto \text{AND} | p_j \in N, j \neq 0\} \cup$
$\{N_{z,x,b} \mapsto \text{BAS} | N_{z,x,y} \in N, j \neq 0\} \cup$
$\{p_0 \mapsto \text{BAS} | p_0 \in N\}$.
The attribute function is defined as $\alpha_{G_{n,k_0,...,k_n}}(N_{z,x,b}) = \prod_{i=0}^{z} |ch^{-1}(p_z)| \cdot a = \prod_{i=0}^{z} k_z \cdot a$
The following theorem can be defined about this tree

Theorem 4.14. *The worst case AT ($G_{n,k_0,...,k_n}$) gives for* bu *($G_{n,k_0,...,k_n}, R_{G_{n,k_0,...,k_n}}, M, \alpha_{G_{n,k_0,...,k_n}}$) a deviation of the power of the optimal security metric with $\prod_{i \in w} |ch^{-1}(i)|$. Meaning that the security metric a is the correct value but the solution of* bu *($G_{G_{n,k_0,...,k_n}}, R_{G_{n,k_0,...,k_n}}, M, \alpha_{G_{n,k_0,...,k_n}}$) $= \prod_{i \in w} |ch^{-1}(i)| \cdot a$ instead of a.*

Proof of Theorem 4.14. In Figure 8, the worse case AT is depicted.
The claim to prove is theorem 4.14.
bu $(G_{n,k_0,...,k_n}, R_{G_{n,k_0,...,k_n}}, M, \alpha_{G_{n,k_0,...,k_n}}) = (((a \cdot |ch^{-1}(p_0)|) \cdot |ch^{-1}(p_1)|) \cdot ...) \cdot |ch^{-1}(p_k)|$,
this is obtained by repeating Theorem 4.13, this can be done by [2, Theorem 9.2].
This Theorem states that an AT cost metric can be obtained by first solving a proper sub-tree and then the next one.
Here are the sub-trees T_{p_i}, $i \in \mathbb{N}$.
$(((a \cdot |ch^{-1}(p_0)|) \cdot |ch^{-1}(p_1)|) \cdot ...) \cdot |ch^{-1}(p_k)| = \prod_{i \in w} |ch^{-1}(i)| \cdot a$,
by the normal multiplication rules.
More reasoning $\hat{\alpha}([\![T]\!]) = a$ because the attack p_0 is a successful attack
$f_{G_{n,k_0,...,k_n}}(R, \{p_0\}) = 1$, because M is the minimal cost.
Therefore a is the optimal security metric.
This proves the statement to be true.

□

This example from Theorem 4.14 is important because it shows for an arbitrary number, that there exists a deviation of that number. It also shows for an arbitrary number of mp nodes such deviation of the bottom-up algorithm exists.

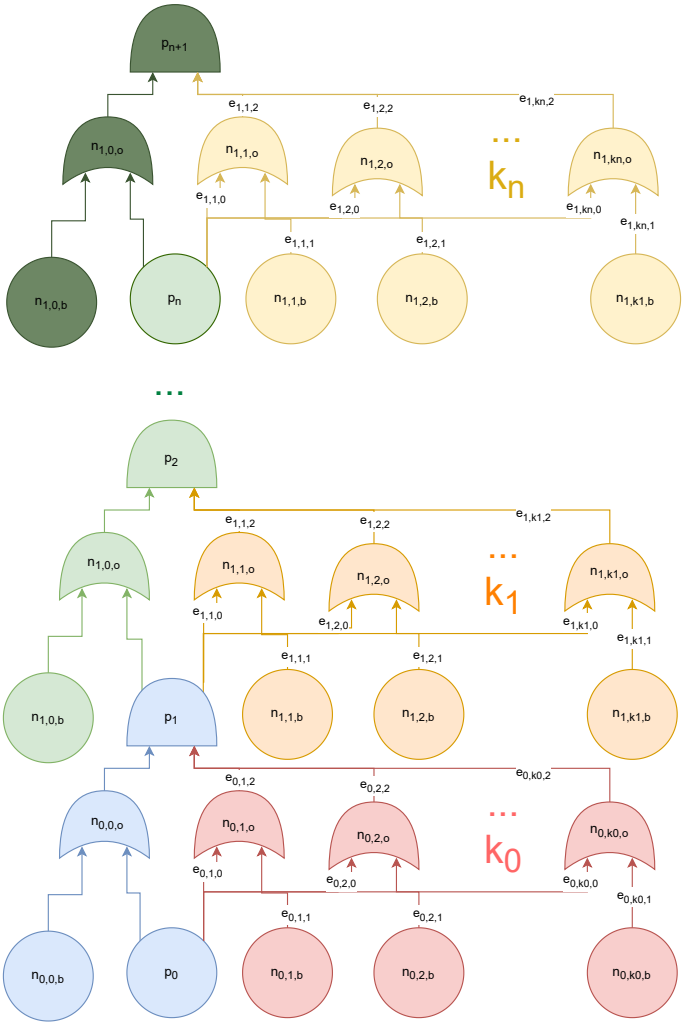

FIGURE 8: Visualisation of the worst case.

5 A first exact approach

In the following section, an exact approach is given by keeping track of BAS nodes. The goal is to prove Theorem 5.5 to show that Algorithm 6 produce a proper *opt* as defined in Definition 5.2.
First, the option is defined, this is needed in order to define options. Intuitively this option can also be seen also an option for a successful attack from a sub-attack tree $v \in N$. This is only true in a context o.a. attack tree and metric is defined etc.

Definition 5.1. Let an option be defined as (a, S, r), where $a \in V, S \subseteq N$ and $r := S \to V$.

All those different options together are collected in a set. In order to see all possible successful attacks on a sub-tree $v \in N$.

Definition 5.2. Let the set of options be defined for a node $v \in N$ as opt_v, which is a set containing the needed options to solve the security metric of T_v.
$opt_v := \{op | op := (a, S, r) a \in V, S \subseteq N, r := S \to V\}$,

In order to define that all those options are truly valid, a definition is needed. What is valid?

Definition 5.3. Let, opt_v, as defined in Definition 5.2. opt_v is called valid if and only if the following holds:

(i) $\forall o \in opt_v, \exists W \subseteq N_{BAS}, a_o = \bigwedge_{b \in W} \alpha(b)$

(ii) $\forall o \in opt_v, r_o := S_o \to V$

(iii) $\hat{\alpha}([\![T_v]\!]) \in \{a_o : o \in opt_v\}$

Then the following lemma will show that this indeed lead to a successful minimal attack. This is done by Definition 5.2.

Lemma 5.4. *Let the set of options be defined as opt and valid opt_R for a given T. Then opt is security metric solution is derived as $\hat{\alpha}([\![T]\!]) = \bigtriangledown_{op \in opt_R} a_{op}$.*

Proof of Lemma 5.4. This follows directly from the definition of Definition 5.3, the third equation. □

Now the bottom-up algorithm is redefined, in order to create those options.
This algorithm is focused on correctness, rather then the optimization.
The structure is reduced from the `bu` algorithm.

Algorithm 6: Bottom up, track BAS node

Assumption: Assumption is that $(V^+, \triangle)$ is a group.;
Output: $opt := \{op | op := (a, S_{BAS}, r) a \in V, S_{BAS} \subseteq N_{BAS}, r := N_{BAS} \to V\}$
Function `bu_BAS`(T, v, D, α)**:**
    `/* The main program is defined as bu, and the functions BAS, OR, AND are replaced as defined below. */`

The following sub-function are redefined to match the changed output.

Algorithm 7: BAS function track BAS node

output: *opt*
Function `BAS`$(T, v, D, \alpha,$ `bu`$)$**:**
    return : $(\alpha(v), \{v\}, v \mapsto \alpha(v))$

The OR function takes the union of all .

Algorithm 8: OR function track BAS node

output: *opt*
Function `OR`$(T, v, D, \alpha,$ `bu`$)$**:**
    return : $\bigcup_{u \in ch(v)}$ `bu`$(T, v, D, \alpha,$ `bu`$)$

The `AND` function creates a Cartesian product from the result of the children, such that all children are uniquely represented in each triple.

Algorithm 9: AND function track BAS node

output: *opt*
Function `AND`$(T, v, D, \alpha,$ `bu`$)$**:**
    result $\leftarrow [(e, \emptyset, \emptyset)]$
    for $u \in ch(v)$ **do**
        $w \leftarrow \emptyset$
        for $bu \in$ `bu`$(T, v, D, \alpha,$ `bu`$)$ **do**
            for *re in result* **do**
                $r_{new} \leftarrow r_{re} \cup r_{bu}$
                $S_{BASnew} \leftarrow S_{BASre} \cup S_{BASbu}$
                $\delta \leftarrow S_{BASbu} \cap S_{BASre}$
                $a_{new} \leftarrow (\bigwedge_{i \in \delta} r_{new}(i))^{-1} \triangle a_{re} \triangle a_{bu}$
                $w \leftarrow w \cup \{(a_{new}, S_{BASnew}, r_{new})\}$
        $result \leftarrow w$;
    return : result

The after function is the same as for the original bottom-up.

Below a new theorem is defined this will help to proof the valid from Definition 5.3, such that it create a list were the security metric can derived from.

Theorem 5.5. *Let,* $\texttt{bu\_BAS}(T, v, D, \alpha) = opt_v$ *the following holds:*

(i) $\forall o \in opt_v, a_o = \bigtriangleup_{b \in S_{BAS_o}} \alpha(b)$

(ii) $\forall o \in opt_v, r_o := S_{BAS_o} \to V, b \mapsto \alpha(b)$

(iii) $[\![T_v]\!] \subseteq \bigcup_{o \in opt_v} \{S_{BAS_o}\}$

Proof of Theorem 5.5. Let T be an AT and $v \in N$, D is a semi-ring where $(V^+ \bigtriangleup)$ is a group and α and γ are given functions belonging to AT T. The algorithm gives an output, defined as:

$$\texttt{bu\_BAS}(T, v, D, \alpha) = opt_v \tag{4}$$

the claim about that output is defined in theorem 5.5

base case: Given a node $\exists k \in N$, such that $\gamma(k) = BAS$, by definition of AT such a node exists since there is no layer to go deeper into the AT, Lemma 2.6. $\forall k \in N$, where $\gamma(k) = BAS$, then $\texttt{bu\_BAS}(T, k, D, \alpha) = (\alpha(k), \{k\}, k \mapsto \alpha(k)) = opt_k$
Then $\forall o \in opt_k, \bigtriangleup_{b \in S_{BAS_o}} \alpha(b) = \alpha(k) = a_o$ also $\forall o \in opt_k, \forall b \in S_{BAS}, b \mapsto \alpha(b)$, because $r := k \mapsto \alpha(v)$ additionally, $T_k = (N_k, E_k) = (k, \emptyset)$ and $[\![T_k]\!] = \{\{k\}\} = \{S_{BAS_o}\}$

Induction step Suppose that the following, Theorem 5.5 is true about the output of $\texttt{bu\_BAS}(T, v, D, \alpha) = opt_c$ for $\forall c \in ch(v), v \in N$. Now it needs to be proven for v that it is true.
Please note $\gamma(v) \neq BAS$, otherwise it will follow the base case. By definition, a BAS node has no children.

case: If $\gamma(v) = OR$, then
$\texttt{bu\_BAS}(T, v, D, \alpha) = opt_v = \texttt{OR}(T, v, D, \alpha, \texttt{bu\_BAS}) = \bigcup_{c \in ch(v)} opt_c$.
Using the induction hypothesis, all three properties hold for all set elements. Taking the union of the elements, to keep all options open for possible attacks. It leaves the elements unchanged and therefore all properties hold for opt_v. This is trivial for theorem 5.5 (i) and (ii). For theorem 5.5 (iii) it holds because all possible incoming edges are expressed in each sub-answer $o \in opt_v$. Further, the node is an OR gate and therefore only one answer needs to be chosen. That is expressed with the options. Up until the node v. The opt_v expressed at least all minimal attacks of T_v and $[\![T_v]\!]$ is just a subset of all minimal attack possible[2, Lemma 3.6].

case: If $\gamma(v) = AND$, then $\texttt{bu\_BAS}(T, v, D, \alpha) = opt_v$. Following the algorithm, it gives $opt_v = \texttt{AND}(T, v, D, ch, ch^{-1}, \alpha, \gamma)$. A description of the part of the AND action, Algorithm 9. First, it creates an empty result, then it iterates through all children. For each child the $\texttt{bu\_BAS}(T, c, D, \alpha) = opt_c$ returns a set of possible paths. Since an AND gate is considered. Every child is represented in every $o \in opt_c$. That is done through the fact that each element in line 3 iterates through all child results ($\forall o \in opt_c$) nodes and is added to a copy of the result, creating the new result. By now, it is agreed that the for loops and initialising make sure that all children are represented uniquely in every result.

Then adding different $o \in opt_c$, where the properties hold as stated in the induction hypothesis, this is done in Line 6 of the AND part of the algorithm.
Below each of the statement of Theorem 5.5 will be proven separately.

- This block will prove Theorem 5.5(i).

  - By induction hypothesis, $\forall c \in ch(v) \forall k \in opt_c, a_k = \bigtriangleup_{b \in S_{BAS_k}} \alpha(b)$
  - In set theory is the rule that $|A \cup B| = |A + B| - |A \cap B|$

  Let $A = S_{BASre}$ and $B = S_{BASbu}$, then $\delta = A \cap B$. From this a is obtained $a_{new} = a_{re \cup bu} = a_{re} \bigtriangleup a_{bu} \bigtriangleup \{\bigtriangleup_{d \in \delta} r_v(d)\}^{-1}$. This shows a valid way to add certain sets together As shown earlier the a_{new} is updated until all children are seen. This concludes the induction hypothesis and follows the rule above, $\bigtriangleup_{b \in S_{BAS_o}} \alpha(b) = a_o, \forall o \in opt_v$.

  Additionally, all values earlier in the chain are correct and now are shown to stay within the boundaries of the properties. The inverse exists with the extension of the field.

- This block will prove Theorem 5.5(ii).
  $\forall o \in opt_v, r_o := N \to V, b \mapsto \alpha(b)$
  By definition of the algorithm, that is equivalent with,
  $\{r_b | c \in ch(v), \exists! b \in opt_c\} := \{S_{BASb} | c \in ch(v), \exists! b \in opt_c\} \to V, b \mapsto \alpha(b)$
  Because not only r_o is concatenated with the options of children as well S_{BASo}.
  By induction hypothesis and the fact that each child is uniquely represented in r_o.
  Therefore $r_o := S_{BASo} \to V, b \mapsto \alpha(b)$.

- This block will prove Theorem 5.5(iii).
  $[\![T_v]\!]$, is obtained by the fact that $\gamma(v) = AND$. Then by the induction hypothesis, $[\![T_u]\!] \subseteq \{S_{BAS_o} | o \in opt_u\}$. Unfortunately, the logical next step to take the union of both sides of the equation might be insufficient.
  The small piece missing is that all possible minimal attacks are represented in $\{S_{BAS_o} | o \in opt_u\}$, that is proven in [2, Lemma 3.6] Therefore, the $\bigcup$ operator can be taken on both sides, considering all possible combinations and remove duplicates(which there are not because sets are considered) and make sure that from each child one option is considered since it an AND operator.
  Theorem 5.5(ii) provides with a function to remove duplicate.
  $\{d | k = |ch(v)|; \bigcup_{u \in ch(v)} [\![T_u]\!]\} \leq$
  $\{d | k = |ch(v)|; 0 \leq i \leq k; \{u_0, ..., u_k\} = ch(v), \forall o \in o_{u_i}, d = \bigcup_{i=0}^{k} S_{o_{u_i}}\}$
  Since this actual done in the algorithm, it concludes that
  $\{d | k = |ch(v)|; 0 \leq i \leq k; \{u_0, ..., u_k\} = ch(v), \forall o \in o_{u_i}, d = \bigcup_{i=0}^{k} S_{o_{u_i}}\} = \bigcup_{o \in opt_v} S_{BAS_o}$.
  This should become clear from Algorithm 5.5 and the description given.
  It is left over for the reader to prove formally if this is deemed to be insufficient.

This proofs the theorem □

The proof has shown that Theorem 5.5 holds. This has not yet shown that it is valid. Therefore the following corollary is proven.

Corollary 5.6. `bu_BAS`$(T, v, D, \alpha) = opt_v$ *is a valid opt.*

Proof of Corollary 5.6. Due to the fact that Theorem 5.5 holds, and is a more strict theorem then the definition of valid, Definition 5.3.

- From Theorem 5.5(i), is equivalent to Definition 5.3(i) in terms of the `bu_BAS` algorithm. Because it is required that there exist such set W, that defines a_0. In Theorem 5.5(i), this set already is provided by S_{BAS}.
- From Theorem 5.5(ii), is equivalent to Definition 5.3(ii) in terms of the `bu_BAS` algorithm. Because $\forall o \in opt_v, S_o = S_{BAS}$, this proven.
- From Theorem 5.5(iii), is equivalent to Definition 5.3(iii) in terms of the `bu_BAS` algorithm. Because, if from 5.5(iii), the security metric is taken, then Definition 5.3(iii) is obtained.

This proofs the corollary. □

From this chapter on nothing is formally proven, this is just a representation of the work done. It is all available in the Python model to try it out.

5.1 Dynamic programming

In the section below the program `bu` is translated in a dynamic programming problem. It is done in such a way that each nodes of the AT needs to be visited only once. This will speed up time complexity quite a bit for non-tree AT simply because the intermediate results are only calculated once.
The Algorithm 10 is valid because the logic of calculating comparing with Algorithm 6 is unchanged only better managed.

Algorithm 10: Bottom up for tree shape AT

```
1  Assumption: Assumption is that (V^+, △) is a group.;
2  storage_result ← ∅
3  Function buDP(T, v, D, α):
4      if v = R then
5          storage_result ← ∅
6      if node in storage_result then
7          return storage_result[v]
8      if γ_T(v) = BAS then
9          result ← BAS(T, v, D, ch, ch^-1, α, γ, buDP)
10     else if γ_T(v) = OR then
11         result ← OR(T, v, D, ch, ch^-1, α, γ, buDP)
12     else if γ_T(v) = AND then
13         result ← AND(T, v, D, ch, ch^-1, α, γ, buDP)
14     result ← AFTER(result, v, buDP)
15     storage_result ← storage_result ∪{node ↦ result}
       return : result
```

Algorithm 11: Bottom up, track BAS node

```
1  Assumption: Assumption is that (V^+, △) is a group.;
3  Output: opt := {op|op := {a, S_BAS, r}a ∈ V, S_BAS ⊆ N_BAS, r := N_BAS → V}
4  Function bu_BASDP(T, v, D, α):
6      /* The main program is defined as buDP, and the functions BAS, OR,
          AND are replaced as defined earlier.                          */
```

5.2 Time complexity, BU_BASDP

In the section below, a sketch of the time complexity is given from the last dynamic program.

Let $n = |N|$, $n_{\texttt{AND}} = |\{v|v \in N, \gamma(v) = \texttt{AND}\}|$and $n_{\texttt{BAS}} = |\{v|v \in N, \gamma(v) = \texttt{BAS}\}|$

The time complexity of $\texttt{bu\_BASDP}(T, R, D, \alpha)$ is approximately
$\mathcal{O}(n \cdot (n - n_{\texttt{AND}}) + n_{\texttt{AND}} \cdot ({n_{\texttt{BAS}}}^{n_{\texttt{BAS}}} \cdot n)^2)$

Here the reason why is given:

In this dynamic program, it is given that the nodes are calculated once because the results are stored in the *storage_result* variable of the Algorithm 11.

To give a rough estimation. The `OR` and `BAS` functions contain simple actions, which is the opposite of the `AND` function. For the `AND` function, the first ones are overall children of v, so that is a maximum of $|n| - 1$.

The second "for" loop loops through all possibilities given by the earlier BAS nodes, which is at most ${n_{\texttt{BAS}}}^{n_{\texttt{BAS}}} \cdot n$. The first factor ${n_{\texttt{BAS}}}^{n_{\texttt{BAS}}}$ stands for all possible combinations that can be made and the factor n takes into consideration that they might be duplicated, they are not filtered out. The last FOR loop goes through all the results again, which might be as big as ${n_{\texttt{BAS}}}^{n_{\texttt{BAS}}} \cdot n$, since this is the maximum size, it is often smaller and thus quicker. So the time complexity of this algorithm, is approximately $\mathcal{O}(n \cdot (n - n_{\texttt{AND}}) + n_{\texttt{AND}} \cdot ({n_{\texttt{BAS}}}^{n_{\texttt{BAS}}} \cdot n)^2)$

6 Multiple-parent approach

Instead of keeping track of BAS nodes, it is also can keep track of mp nodes. Since those are the centre of the problem. The following algorithm is obtained.

Algorithm 12: Bottom up, track multi-parent node

Assumption: Assumption is that $(V^+, \triangle)$ is a group.;
Input: T, v, D, α
Output: $opt := \{op | op := (a, S_P, r)\} a \in V, S_P \subseteq P, r := P \to V\}$
Function `bu_MP(`T, v, D, α`):`
    `/* The main program is defined as bu_BASDP, and the functions BAS, AFTER are replaced as defined below. */`

Algorithm 13: BAS function for track MP node

input : T, v, D, α
output: *opt*
Function `BAS(`T, v, D, α, `bu):`
    result $\leftarrow (\alpha(v), \emptyset, \emptyset)$
    return : result

The BAS action is only giving an element of the space V. That is because in the $AFTER$ function the respective node is added if and only if the node is mp and the element of the respectively is not equal to the neutral element.

Algorithm 14: AFTER function for track MP node

input : T, v, D, α, `bu`, result
output: *opt*
Function `AFTER(`T, v, D, α, `bu`, *result*`):`
    if $ch^{-1}(v) > 1$ **then**
        for $o \in result$ **do**
            $a \leftarrow a_o \triangle (\bigtriangleup_{u \in S_{P_o}} r_o(u))^{-1}$
            if a *!=* e **then**
                result $\leftarrow (a_o, S_{P_o} \cup \{v\}, r_v \cup v \mapsto a)$
    return : result

6.1 Improvement bu_MP vs bu_BAS

Due to the fact that only the node of an option is added in S and r if it is not a neutral element it makes the mp node to represent multiple BAS nodes. The expectation is that the algorithm will remain valid. This can be proven by defining a proper theorem, as in Theorem 5.5. Then there are two strategies to prove this theorem, one is by induction, similar to Theorem 5.5, for the other strategy one can argue that the options stay similar enough from Theorem 5.5 to obtain the minimal attack out of it. The latter seems less work, due to the fact that keeping track of mp nodes is a relatively small change from the Algorithm 5.6.

Due to the fact that BU_MP algorithm keeps track of mp nodes and represents one or multiple BAs nodes, if one lets $h = \min(n\_AND, |P|)$, where P is the set of all mp parents, then the approximate time complexity of the BU_MP algorithm is,
$\mathcal{O}(n \cdot (n - n_{\texttt{AND}}) + n_{\texttt{AND}} \cdot (h^h \cdot n)^2)$

7 Filter improvement

In the previous sections, an exact algorithm of the bottom-up algorithm was defined. It is crucial to take into account all potential scenarios and anticipate any unforeseen variables, as alternative approaches may not necessarily yield the most optimal outcome. On the other side of the coin, for some scenarios, it can be guaranteed that there exists a better alternative. In such cases, options can be stored without affecting the result. This creates a so-called filter. Two filters are constructed. The word filter is used because given a set, only a selection is kept. Considering the time complexity, a few observations are made. Firstly, with this filter, it can do better by deselecting quite some parts. On the other hand, the filter could check and select out nothing.

This can be coded by the pseudo-code of the `AFTER` function. This can be done in combination with the `bu_MP` `AFTER` function, by first doing the `AFTER` function from `bu_MP`, then this `AFTER` function.

Algorithm 15: AFTER function

input : T, v, D, α, `bu`, result
output: *opt*
1 **Function** `AFTER`(T, v, D, α, `bu`, *result*)**:**
2 $result \leftarrow \{c | c \in result, \forall b \in result, S_c = S_b \implies \nabla(a_c, a_b) = a_c \vee a_c = a_b\}$
3 $result \leftarrow \{c | c \in result, \neg\exists b \in result, S_b \subseteq S_c \wedge \nabla(a_c, a_b) = a_c \vee a_c = a_b\}$
return : *result*

A bit more description about the two filters is provided in the last part of this section. An option contains an attack, S and a value a as defined in Definition 5.2:

1. (Correspond to Algorithm 15 Line 2)The first one filters out the elements where the same attack is defined but a different value is obtained. Of course, the best value is kept track of. In case the minimal cost metric is used then the value obtained will be the smallest number.

2. (Correspond to Algorithm 15 Line 3)The second one filters out the elements where the attack is a subset or equal to another attack, but this attack obtains the worst metric value. Then it is chosen to delete the attack which is a subset or equal to the other one. This can be done because the latter option has more potential since all necessary nodes are present that might reappear later on. In the first attack, there are also some nodes but they are all contained in the latter one as well. Therefore it is better to keep the latter one.

Filter 1 seems to contain filter 2, on the first side, but if the discrete math rules are followed using CNF, it will become clear that they are different. That has to do with the quantifiers and negations.

Those filters can be translated to a theorem. That would allow us to formally prove the statement, probably with contradiction.

8 Conclusions

Firstly, it is seen that the bottom-up is bounded, by a factor that can increase exponentially. This could help us to define a heuristic. After that, a worst-case introduced to see that the bottom-up algorithm bound sometimes is met.

In the second part, an exact approach is defined by keeping track of the BAS nodes. Later it appeared that keeping track of the mp nodes gives in more cases a better result. Lastly, the exact approach is improved by a so-called filter, to keep track of fewer nodes.

9 Future work

Several recommendations can be made:

Approximation approach

- The result also holds for minimum time since that is the same security metric but just a different name.
- To prove that the bound of the bottom-up algorithm is the maximum number of paths from a `BAS` node in an AT, from Definition 3.8. This is a slightly tighter statement as shown in this thesis. It is probably already a sub-result of Theorem 4.10.
- Although the paper is made for minimal cost, it might also work for the maximum cost by taking negative cost, the observation is that it will go through.
- For the minimum cost metric and minimum time metric it is proven that a deviation of maximal of the factor $\triangle \prod_{i \in w} |ch^{-1}(i)|$ for the bottom-up algorithm. This might be similar in the case of metrics as well. A proof would be similar but the maximum and minimum are not defined properly in all those security metrics.
- Another open problem would be to determine a heuristic depending on the attack that is chosen by the bottom-up algorithm the maximum multi-parent path and the difference between the `AND` or `OR` gate.

Exact approach

- Proving valid, from Definition 5.3, for the other algorithms given in their paper.
- It would be interesting to see if there exist more optimise opportunities such that the amount of options that are kept track narrows down.
- If it is desired to implement the algorithm, it might be useful to look at the Python model that is programmed. It also might be useful to consider the bottom-up algorithm, depending on the situation.

References

[1] Aws Naser Jaber and Lothar Fritsch. Covid-19 and global increases in cybersecurity attacks: Review of possible adverse artificial intelligence attacks. In *2021 25th International Computer Science and Engineering Conference (ICSEC)*, page 434–442, November 2021.

[2] Milan Lopuhaä-Zwakenberg, Carlos E. Budde, and Mariëlle Stoelinga. Efficient and generic algorithms for quantitative attack tree analysis. *IEEE Transactions on Dependable and Secure Computing*, 20(5):4169–4187, September 2023. arXiv:2212.05358 [cs].

[3] Milan Lopuhaä-Zwakenberg and Mariëlle Stoelinga. Attack time analysis in dynamic attack trees via integer linear programming. *arXiv*, 0(arXiv:2111.05114), September 2023. arXiv:2111.05114 [cs].

[4] Milan Lopuhaä-Zwakenberg and Mariëlle Stoelinga. Cost-damage analysis of attack trees. *arXiv*, 0(arXiv:2304.05812), April 2023. arXiv:2304.05812 [cs, math].

[5] B. Schneier. Academic: Attack trees - schneier on security, 1999.

10 Abbreviations

Definition	Meaning
AT(s)	Attack tree(s)
mp(s)	Multi-parent(node(s))
DP	Dynamic programming
bu	bottom-up

11 Symbols

Symbole	Meaning
T	represent the attack tree(AT)
N	represent the set of nodes
E	represent the set of edges
R	Root node /the goal that needs to be achieved
ch	the function to receive the children of the node
ch^{-1}	the function to receive the parents of the node
BAS	BAS is a basic attack step.
AND	binary AND gate
OR	binary OR gate
A	the set of all sets of attacks
α	an attack set
γ	node type function of an AT
S	is the structure function
P	the set of multi-parent nodes

A Use of AI

- During the preparation of this work the author used Grammarly in order to hide the flaws of dyslexia. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.
- During the preparation of this work the author used ChatGPT in order to speed up the process of finding the proper latex function/libraries. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.
- During the preparation of this work the author used Grammerly extensively in order to write grammatically correct english. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

B Programming

All algorithms are modeled and programmed on a high level. The Github AT can be found here or with the QR-code.

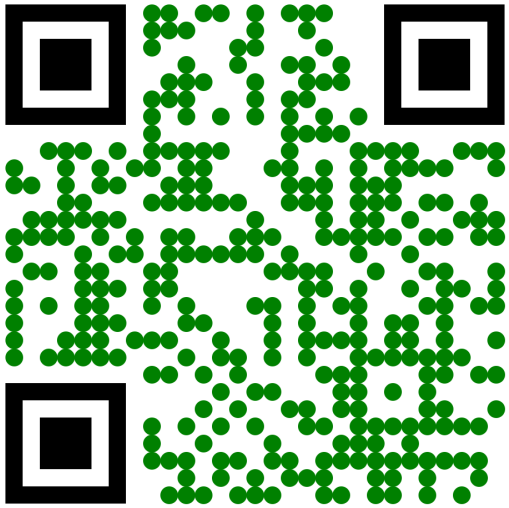

FIGURE 9: Caption