Attack Time Analysis in Dynamic Attack Trees via Integer Linear Programming

Milan Lopuhaä-Zwakenberg[1(✉)] and Mariëlle Stoelinga[1,2]

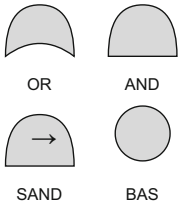
[1] University of Twente, Enschede, the Netherlands
{m.a.lopuhaa,m.i.a.stoelinga}@utwente.nl
[2] Radboud University, Nijmegen, the Netherlands

Abstract. Attack trees (ATs) are an important tool in security analysis, and an important part of AT analysis is computing metrics. However, metric computation is NP-complete in general. In this paper, we showcase the use of mixed integer linear programming (MILP) as a tool for quantitative analysis. Specifically, we use MILP to solve the open problem of calculating the *min time* metric of dynamic ATs, i.e., the minimal time to attack a system. We also present two other tools to further improve our MILP method: First, we show how the computation can be sped up by identifying the modules of an AT, i.e. subtrees connected to the rest of the AT via only one node. Second, we define a general semantics for dynamic ATs that significantly relaxes the restrictions on attack trees compared to earlier work, allowing us to apply our methods to a wide variety of ATs. Experiments on a synthetic testing set of large ATs verify that both the integer linear programming approach and modular analysis considerably decrease the computation time of attack time analysis.

Keywords: Attack trees · Quantitative analysis · Optimization · Mixed integer linear programming

1 Introduction

(Dynamic) Attack Trees. Attack trees (ATs) are a prominent methodology in security analysis. They facilitate security specialists in identifying, documenting, analyzing and prioritizing (cyber) risks. An AT is a hierarchical diagram that describes a system's vulnerabilities to an adversary's attacks. Despite their name, ATs are rooted directed acyclic graphs. Roots of ATs represent the adversary's goal, while the leaves represent basic attack steps (BAS) undertaken by the adversary. Each internal root is labeled with a gate, determining how its activation depends on that of its children. Standard ATs (SATs) feature only `OR` and `AND` gates, but many extensions have been introduced to describe more elaborate attack scenarios [16]. One



This research has been partially funded by ERC Consolidator grant 864075 CAESAR and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233.

C. Ferreira and T. A. C. Willemse (Eds.): SEFM 2023, LNCS 14323, pp. 165–183, 2023.
https://doi.org/10.1007/978-3-031-47115-5_10

of the most prominent extensions are *dynamic ATs* (DATs) [14]. DATs introduce a `SAND` (sequential `AND`) gate, which is activated only when its children are activated sequentially in the correct order. By contrast, an `AND`-node's children can be activated in parallel. An example is given in Fig. 1.

Quantitative Analysis. Quantitative analysis aims at computing *AT metrics*. Such metrics formalize how well a system performs in terms of security, and are essential when comparing alternatives or making trade-offs. Many such metrics exist, such as the minimal cost, minimal required skill, or maximal damage of a successful attack. This paper focuses on *min time*: the minimal time the adversary needs to perform a successful attack, given the duration of each BAS. This is important, since attack success crucially depends on time: attacks that take too long are not viable. Insight in timing behaviors of attacks is therefore a key to devising effective countermeasures. For instance, a security operations centre is interested in the time difference between the fastest viable attack and its average response time [1]. *Min time* is especially relevant in the context of DATs: On many metrics, such as cost/probability/skill, `SAND` and `AND` gates behave identically. Thus, to compute those metrics, algorithms for SATs immediately generalize to DATs. It is in the timing behavior that the difference between `SAND` and `AND` manifests itself, so that novel computation algorithms are needed.

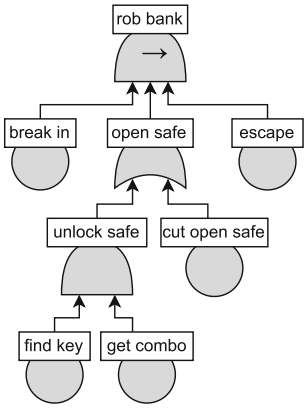
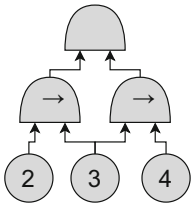


Fig. 1. A DAT for a bank robbery [4]. To rob a bank, attackers must break in, open the safe, and escape (in that order). The safe is opened by cutting it open, or by unlocking via obtaining the key and combination.

Existing Algorithms for *min time*. The naive approach to calculating *min time* is to list all attacks that reach the root, and to find the one that takes the least time; clearly this is computationally prohibitive for larger ATs. A tree-shaped DAT can be computed via a bottom-up (BU) algorithm [14,23]. This algorithm works for general attributes (e.g. cost, probability, time), by using appropriate operators at each gate. For DAG-shaped ATs, the BU algorithm does not always work, because the values in different branches are no longer independent. For SATs this is not a problem because the relevant operators are idempotent [17]. In the DAT above, however, the BU algorithm of [14] calculates *min time* as $\max(2+3, 3+4) = 7$. However, the only successful attack is the one that activates the three BAS sequentially, and so *min time* equals $2+3+4 = 9$. Thus to find *min time* for DAG-shaped DATs new approaches are needed; in [5], efficient computation for DAG-shaped DATs is left as an open problem.



Integer Linear Programming. In this paper, we present a novel method to calculate *min time* for general DATs based on MILP. We translate calculating *min time* into a real-valued optimization problem, with a set of nonlinear constraints. We rewrite these into linear constraints by introducing auxiliary integer variables at each gate; for SAND-gates this is nontrivial and requires a careful analysis of the semantics, beyond the current literature (see below). Since dedicated solvers exist for MILP, translating attack time analysis into MILP speeds up computation time considerably.

Modular Analysis. To improve performance, we combine MILP with modular analysis [9]: we identify *modules* in a DAT, i.e., subDATs whose only connections to the rest of the DAT go via their root. We prove that *min time* can be computed by analyzing the modules separately; this requires a detailed comparison of the attacks on the larger DAT to the attacks on its modules. If a module is tree-shaped or static, then we can deploy the bottom-up algorithm to further decrease computation time. We integrate these modifications into our MILP algorithm.

Generalized Semantics. Another point we settle in this paper are generalized semantics for DATs. As `SAND`-gates require their children to be executed consecutively, different branches in the DAT may impose conflicting restrictions on the execution orders. To rule out these conflicts, [5] imposed well-formedness criteria at the cost of ruling out some satisfiable DATs. Furthermore, the corresponding attack definition was overly restrictive, with some fastest attacks not being recognized. This leads to an overestimation of *min time*. In this work we extend the definition of a (successful) attack so *min time* is correctly defined. This new definition applies to all DATs, not just the well-formed ones.

Experimental Validation. For confidentiality reasons industrial DATs are typically not disclosed to the general public [7,26]. Therefore, we create a testing suite of 2400 synthetic DATs, obtained by combining smaller DATs from the literature via standard DAT composition methods, and we compare the performance of four methods (modular versus nonmodular and enumerative versus MILP). The experiments show that on larger DATs MILP outperforms enumerative, and modular outperforms nonmodular. The code for the experiments, the generated DATs and the experimental results are available in [22], and a version with proofs is available at [21].

Contributions. Summarized our main contributions are:

1. A generalization of the poset semantics of [5] that significantly relaxes the syntactic constraints on the use of `SAND`-gates.
2. A novel algorithm to calculate *min time* for general DATs based on Mixed Integer Linear Programming.
3. A modularization approach that yields significant speed ups by separately handling fragments of the DAT that are static or tree-shaped.
4. Extensive experimental validation to evaluate the performance of the algorithms.

2 Related Work

Dynamic ATs were first formally defined in [14], with series-parallel graphs semantics. These assume that each node must be activated separately for each of its parents. Effectively, this makes any DAT tree-shaped, which limits the range of scenarios that can be modeled.

Poset-semantics for DATs are used in [5]; here each node can be activated only once, allowing more scenarios to be modeled. The calculation of time-related metrics such as *min time* on DAG-shaped DATs is left as an open problem.

In [2,3,18,19,31] DATs are modeled as priced-timed automata. This allows for a detailed analysis, including *min time* calculation, by activating nodes from the root either in parallel or sequentially, depending on gate type. However, this approach does not consider satisfiability; hence the *min time* found via this method can correspond to a non-existing attack. As such, this method only calculates a lower bound to the actual *min time*.

Cyber security risks are also analyzed via *time-to-compromise* [24]. This assigns an (exponential) probability distribution to the failure time of each component, from which one finds the system failure pdf. This approach can be extended to consider different attack scenarios [28]. The current paper's DAT approach allows for a more systematic way of studying different attack scenarios, but we do not consider probabilistic data. Another way to incorporate stochastics is to consider *Bayesian fault trees* [12,25], in which a node's activation depends probabilistically on that of its children. This allows for more detailed modelling, but analysis is considerably more complicated: instead of a single *min time* metric, there is a Pareto front of attack time and attack success probability. Incorporating probability in these manners would be interesting for future work.

Time analysis of DATs falls into the wider framework of quantitative analysis on ATs. Existing approaches either focus on a single metric [4,6,7] or they develop methods that apply to general classes of metrics [5,17,23]. The latter case typically use algebraic structures like semirings, defining the metric in terms of operators which are assumed to have certain properties.

3 Dynamic Attack Trees

This section reviews the definition of DATs, and develops their semantics and the *min time* metric. The notation introduced throughout the paper is summarized in Table 1. The following definition of a DAT is from [5].

Definition 1. *A* dynamic attack tree *(DAT) is a rooted directed acyclic graph* $T = (N, E)$ *where each node* $v \in N$ *has a type* $\gamma(v) \in \{\texttt{BAS}, \texttt{OR}, \texttt{AND}, \texttt{SAND}\}$ *such that* $\gamma(v) = \texttt{BAS}$ *if and only if* v *is a leaf, and every node* v *with* $\gamma(v) = \texttt{SAND}$ *has an ordering of its set of children.*

Note that a DAT is not necessarily a tree. If it is, we call it *tree-shaped*. The root is denoted R_T. For $\gamma \in \{\texttt{BAS}, \texttt{OR}, \texttt{AND}, \texttt{SAND}\}$, we write N_γ for the set of nodes v with $\gamma(v) = \gamma$. The (po)set of children of v is denoted $\mathrm{ch}(v)$. If $\gamma(v) = \texttt{SAND}$ and v has (ordered) children $v_1, \ldots, v_n$, we write $v = \texttt{SAND}(v_1, \ldots, v_n)$ for

Table 1. Notation used in this paper.

Notation	Meaning	Section	Notation	Meaning	Section
$T = (N, E)$	Dynamic attack tree	Sect. 3	$\mathrm{mt}(T, d), \mathrm{mt}(T)$	*min time* of DAT T	Sect. 3.2
$\gamma(v)$	Type of v	Sect. 3	$\mathcal{F}_T$	Time assignments of T	Sect. 4
R_T	Root of T	Sect. 3	M	*min time* upper bound	Sect. 4
$N_{\mathtt{BAS}}$	$\{v \in N \mid \gamma(v) = \mathtt{BAS}\}$	Sect. 3	Z_i^v	Consecutive BAS pairs	Sect. 4
T_v	subDAT with root v	Sect. 3	$x_i^v, y^v, z_{i,a,a'}^v$	Auxiliary MILP variables	Sect. 4
B_v	Set of BAS of T_v	Sect. 3	n_v	Number of children of v	Sect. 4
$(\mathcal{A}_T, \leq)$	Poset of attacks on T	Sect. 3.1	T_v	Sub-DAT with root v	Sect. 5
$\mathcal{S}_T$	Successful attacks	Sect. 3.1	$\bar{v}$	BAS replacement for T_v	Sect. 5
$\mathrm{t}(\mathcal{O}, d), \mathrm{t}(\mathcal{O})$	Time of attack $\mathcal{O}$	Sect. 3.2	$T^{\bar{v}}$	T with T_v replaced by $\bar{v}$	Sect. 5

convenience. We do the same for `OR` and `AND`, where the ordering of the children does not matter. We write T_v for the subDAG consisting of all *descendants* of v, i.e. all v' for which there is a path from v to v', including v itself. Furthermore, we let B_v be the set of descendants of v in $N_{\mathtt{BAS}}$. DATs can be represented graphically as in Fig. 1.

A dynamic attack tree codifies the ways an attacker can make a system fail by executing the *basic attack steps*, i.e., the nodes in $N_{\mathtt{BAS}}$. A non-`BAS` node is reached depending on its children, where `OR` and `AND` have the expected meaning, and a `SAND`-node is reached if all children are reached in their given order. The adversary's goal is to reach R_T. These semantics are defined in Sect. 3.1.

In the literature, two interpretations of nodes with multiple parent nodes exist, affecting both semantics and metrics. In the first interpretation, *multiple activation* (MA), [14,23,32] each BAS can be activated multiple times, and every parent of a node requires its own activation of that node. Thus $\mathtt{SAND}(a, a)$ succeeds only if a is activated twice consecutively. By adding a copy of each node for each of its parents, any DAT can be transformed into a tree-shaped one with equivalent semantics and metrics. As a result, metrics can be calculated quickly via a bottom-up algorithm [10], but MA cannot adequately model systems in which one action has multiple independent consequences.

In *single activation* (SA) [5,15] each BAS is executed at most once, and a node only needs to be activated once to count as an input for all its parents. In SA $\mathtt{SAND}(a, a)$ cannot be satisfied, because a cannot be activated before itself. SA is able to describe a much wider range of systems; although every SA representation can be turned into an equivalent MA representation, this process is both computationally expensive as it is done by writing the corresponding boolean function in disjunctive normal form. This rewriting also loses the meaning of the intermediate nodes in the DAT, which typically represent intermediate attacker goals. We therefore choose to analyze DATs under the SA interpretation; since every DAT is equivalent to a tree-shaped one under MA and MA and SA coincide on trees, SA can model every scenario that MA can.

3.1 Semantics

We discuss DAT semantics, extending [5]. An attack consists of a set A of attacker-activated BAS, and a strict partial order $\prec$, where $a \prec a'$ means a is executed before a'.

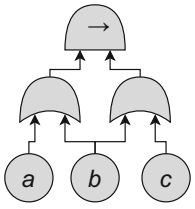
Definition 2. *The set $\mathcal{A}_T$ of* attacks *on T is the set of strictly partially ordered sets $\mathcal{O} = (A, \prec)$, where $A \subseteq N_{\texttt{BAS}}$. This set has a partial order $\leq$ given by $\mathcal{O} \leq \mathcal{O}'$, for $\mathcal{O} = (A, \prec)$ and $\mathcal{O}' = (A', \prec')$, if and only if $A \subseteq A'$ and $\prec \subseteq \prec'$.*

We are interested in successful attacks, i.e., attacks that manage to reach the root. Successful attacks, and the semantics of T, are defined as follows:

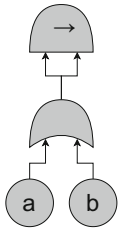
Definition 3. *Let v be a node. We say that an attack $\mathcal{O} = (A, \prec)$* reaches *v if:*

1. *$v \in N_{\texttt{BAS}}$ and $v \in A$;*
2. *$v = \texttt{OR}(v_1, \ldots, v_n)$ and $\mathcal{O}$ reaches at least one of the v_i;*
3. *$v = \texttt{AND}(v_1, \ldots, v_n)$ and $\mathcal{O}$ reaches all of the v_i;*
4. *$v = \texttt{SAND}(v_1, \ldots, v_n)$ and $\mathcal{O}$ reaches all of the v_i, and for all $a \in A \cap B_{v_i}$, $a' \in A \cap B_{v_{i+1}}$ one has $a \prec a'$.*

A is successful *if it reaches R_T. The* semantics *of T is the set $\mathcal{S}_T$ of successful attacks on T.*



A SAND-gate $v = \texttt{SAND}(v_1, \ldots, v_n)$ is only reached if all of the BAS of v_i have been (successfully) executed before any of the BAS of v_{i+1} has started. By contrast, an AND-gate allows its children to be executed in parallel. Contrary to the static case (without SAND-gates), it is possible that $\mathcal{S}_T = \varnothing$. For example, $\mathcal{S}_{\texttt{SAND}(a,a)} = \varnothing$. Also, being successful is *not* monotonous on the set of attacks, i.e., it is possible that $\mathcal{O}$ is successful while $\mathcal{O}'$ is not, even if $\mathcal{O} \leq \mathcal{O}'$. For instance, in the DAT above $(\{a, c\}, \{(a, c)\})$ is a successful attack, but $(\{a, b, c\}, \{(a, c)\})$ is not. Note that unlike the situation for SATs, a gate's activation does not simply depend on the activation of its children, but also on the relative order on the BAS associated to these children; this encodes the timing information essential to DATs.



Definition 3 is not the only way one might define the semantics of DATs. In fact, our semantics are based on those of [5], but differ on certain DATs; see Sect. 3.3. We have chosen to interpret the SAND-gate in a strict matter, so that it is activated only if the entirety of the attack on v_i has finished before the attack on v_{i+1} is started; in particular, v_i and v_{i+1} cannot share activated BAS, which may be considered unwanted behaviour. There are also other approaches, which unfortunately have other problems. For instance, one could define succesful attacks bottom-up in a compositional fashion, defining $\mathcal{O}$ to reach $\texttt{SAND}(v_1, v_2)$ if there exists attacks $\mathcal{O}_1, \mathcal{O}_2$ such that $\mathcal{O}$ is the parallel composition of $\mathcal{O}_1$ and $\mathcal{O}_2$. However, under such a definition the AT above $(\{a, b\}, \{(a, b)\})$ is a succesful attack, whereas in our opinion this AT should not be considered satisfiable. Yet another approach would be to assign a starting and finishing time to each node, similar to what we do in Definition 5, but this has the disadvantage of being more convoluted as an attack is now a function $N \to \mathbb{R}$.

3.2 The *Min Time* Metric

Min time is the minimal time it takes to perform a successful attack on a given DAT. While other metrics exist for DATs, *min time* is a fundamental time metric, and calculating it efficiently for non-tree-shaped DATs is an open problem [5].

Min time is defined as follows: There is a *duration function* $d\colon N_{\mathtt{BAS}} \to \mathbb{R}_{\geq 0}$, with $d(a)$ denoting the time it takes to execute a. If $a \prec a'$, then the BAS a' can only be started once a has been completed, while a and a' can be activated in parallel if such a relation does not exist. As such, we can define the total duration of an attack $\mathrm{t}(\mathcal{O}, d)$ and *min time* $\mathrm{mt}(T, d)$ as

$$\mathrm{t}(\mathcal{O}, d) = \max_{C \text{ max. chain in } \mathcal{O}} \sum_{a \in C} d(a), \qquad \mathrm{mt}(T, d) = \min_{\mathcal{O} \in \mathcal{S}_T} \mathrm{t}(\mathcal{O}, d)$$

where the maximum is taken over the maximal chains (i.e., maximal linearly ordered subsets) of the strict poset $\mathcal{O}$. We will often omit d from the notation and write $\mathrm{t}(\mathcal{O})$ if there is no confusion. Note that t is monotonous: if $\mathcal{O} \leq \mathcal{O}'$ one has $\mathrm{t}(\mathcal{O}) \leq \mathrm{t}(\mathcal{O}')$. Furthermore, $\mathrm{mt}(T) = \infty$ if $\mathcal{S}_T = \varnothing$.

Example 1. Figure 2 depicts the bank robbery DAT of Fig. 1 augmented with durations for the BAS (we take the expected durations from the distributions given in [4]). To calculate $\mathrm{mt}(T)$ one would first need to find $\mathcal{S}_T$. While this set is quite large, because of the monotonicity of t, the minimum is attained at one of the minimal elements of the poset $(\mathcal{S}_T, \leq)$. There are two minimal attacks, depending on whether the attackers choose to cut open the safe, or unlock it. Abbreviating BAS names, we can represent these minimal attacks as sets of chains as $\mathcal{O}_1 = \{bi \prec cos \prec e\}$ and $\mathcal{O}_2 = \{bi \prec fk \prec e, bi \prec gc \prec e\}$. These have duration $\mathrm{t}(\mathcal{O}_1) = 1.00 + 0.67 + 0.20 = 1.87$ and $\mathrm{t}(\mathcal{O}_2) = \max(1.00 + 0.50 + 0.20, 1.00 + 1.00 + 0.20) = \max(1.70, 2.20) = 2.20$. It follows that $\mathrm{mt}(T) = \min(1.87, 2.20) = 1.87$.

In the multiple activation scenario, *min time* can be calculated by reshaping a DAT into its canonical form [14], from which *min time* is easily calculated. However, this technique does not carry over to our formalism, as in the single activation scenario a canonical form does not exist.

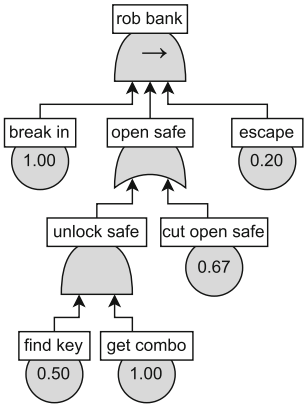
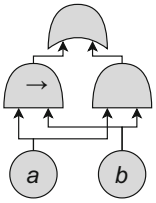


Fig. 2. The bank robbery DAT of Fig. 1 augmented with durations.

3.3 Relation to Semantics of [5]

In [5] attacks are called attacks only if they satisfy the ordering constraints imposed by *all* `SAND`-gates. This is defined only for *well-formed* DATs, i.e., all these constraints are simultaneously satisfiable. More formally, that work only considers attacks that we call *full* in the following definition.

Definition 4. *Let T be a DAT. Define a relation $\sqsubseteq'$ on $N_{\texttt{BAS}}$ by $a \sqsubseteq' a'$ iff there exists a node $v = \texttt{SAND}(v_1, \ldots, v_n)$ and an $i < n$ such that $a \in B_{v_i}$ and $a' \in B_{v_{i+1}}$. Let $\sqsubseteq$ be the transitive closure of $\sqsubseteq'$. Then T is* well-formed *if $\sqsubseteq$ is a strict partial order. An attack $(A, \prec)$ on a well-formed DAT is* full *if $\prec = \sqsubseteq|_A$, the restriction of $\sqsubseteq$ to A.*



However, not all attacks will be full, because an attack may not need to reach all SAND-nodes in order to reach the root, and non-reached nodes should not put restrictions on attacks. Consider the well-formed DAT on the right. Only $(\{a, b\}, \{(a, b)\})$ is a full successful attack. However, $(\{a, b\}, \varnothing)$ is a successful attack as well. Hence non-full attacks are needed to fully describe the semantics of well-formed DATs, which motivates Definition 2. Furthermore, our definition defines the semantics of general DATs, not just the well-formed ones.

4 An MILP Approach to *Min Time*

This section describes a novel method to compute $\mathrm{mt}(T)$ based on mixed-integer linear programming (MILP). Although MILP is NP-complete, a number of good heuristics and solvers exist specifically for MILP, which can result in a low computation time. We first show that *min time* can be found by solving an optimization problem in Theorem 1, and then we describe how that optimization problem can be rewritten into the MILP framework.

The building block of the new approach is the notion of *time assignment*, which assigns to each node a completion time f_v that respects all timing constraints in the DAT. If $f_v = \infty$ then v is not reached at all. The formal definition is stated below; recall that B_v is the set of BAS-descendants of v, and N the set of nodes in the attack tree.

Definition 5. *Let T be a DAT. For a node v with children $v_1, \ldots, v_n$ and $i < n$, define $Z_i^v := B_{v_i} \times B_{v_{i+1}}$. A* time assignment *is a vector $f \in [0, \infty]^N$ satisfying:*

1. *For each $a \in N_{\texttt{BAS}}$ one has $f_a \geq d(a)$;*
2. *For each $v = \texttt{OR}(v_1, \ldots, v_n)$ one has $f_v \geq \min_i f_{v_i}$;*
3. *For each $v = \texttt{AND}(v_1, \ldots, v_n)$ one has $f_v \geq \max_i f_{v_i}$;*
4. *For each $v = \texttt{SAND}(v_1, \ldots, v_n)$, the following must hold:*
   (a) *it holds that $f_v \geq f_{v_n}$;*
   (b) *If there is a $i \leq n$ such that $f_{v_i} = \infty$, then $f_v = \infty$;*
   (c) *If there exist $i < n$ and $(a, a') \in Z_i^v$ such that $f_{a'} - d(a') < f_a < \infty$, then $f_v = \infty$.*

The set of all time assignments for T is denoted $\mathcal{F}_T$.

The SAND-conditions can be understood as follows. 4a) tells us that v cannot be reached before v_n, and 4b) tells us that v cannot be reached if any of its children is not reached. 4c) conveys that whenever there is an $a \in B_{v_i}$ that is

activated (i.e., $f_a < \infty$), then in order for v to be activated, one must have $f_{a'} - d(a') \geq f_a$ for all $a' \in B_{v_{i+1}}$. Since $f_{a'} - d(a')$ is the starting time of a', this means that a' must be started after a is finished activating. It is more subtle than simply requiring $f_{a'} - d(a') \geq f_a$ for all $(a, a') \in Z_i^v$; that would ensure that all SAND-gates impose ordering restrictions, not just those that are activated.

Note that $f_{a'} - d(a')$ is the starting time of a BAS a', so 4c) tells us that v is only reached if the BAS-descendants of v_{i+1} are started once those of v_i have been completed. We allow for a delay in completing node v, even when enough of its children have been completed. Time assignments relate to *min time*:

Theorem 1. $\mathrm{mt}(T) = \min_{f \in \mathcal{F}_T} f_{\mathrm{R}_T}$.

This result allows us to calculate $\mathrm{mt}(T)$ by solving the following optimization problem.

$$\text{minimize}_{f \in [0,\infty]^N} \ \& f_{\mathrm{R}_T} \qquad \text{s.t.} \ \ f \in \mathcal{F}_T. \tag{1}$$

This is not a linear problem, due to the nonlinear constraints of Definition 5. We use auxiliary integer variables to linearize these constraints. First, we need to get rid of the ∞ in Definition 5, which we do by replacing it with a suitably large real number. Define the constant $M = 1 + \sum_{a \in N_{\mathtt{BAS}}} d(a)$. The following lemma shows that if T is satisfiable, then to minimize (1) one can focus on the f with $f_v \in [0, M-1] \cup \infty$.

Lemma 1. *There is an f minimizing (1) for which $\forall v \colon f_v \in [0, M-1] \cup \infty$.*

This shows that we can use M to play the role of ∞ where necessary. We enforce this by demanding $f_v \in [0, M]$, and we interpret $f_v = M$ to mean that v is not reached. For a node v, let n_v be its number of children, which are denoted $v_1, \ldots, v_{n_v}$. We then use standard MILP techniques [8] to rewrite Definition 5.

To rewrite the OR-condition, we introduce an auxiliary binary variable x_i^v for each $v \in N_{\mathtt{OR}}$ and each $i \leq n_v$. The purpose of x_i^v is to represent the truthfulness of the statement "$i = \arg\min_{i'} f_{v_{i'}}$". We can then represent $f_v \geq \min_i f_{v_i}$ by

$$\sum_{i \leq n_v} x_i^v \geq 1, \qquad \forall i \leq n_v \colon f_v \geq f_{v_i} + M(x_i^v - 1).$$

The latter is automatically satisfied if $x_i^v = 0$, and reduces to $f_v \geq f_{v_i}$ if $x_i^v = 1$. The former ensures that the latter must happen for at least one i, so together these encode $f_v \geq \min_i f_{v_i}$. The condition for AND-gates can be rewritten as $\forall i \leq n_v \colon f_v \geq f_{v_i}$.

Finally, we consider SAND-gates. For $v \in N_{\mathtt{SAND}}$, we introduce an auxiliary binary variable y^v that encodes "$\exists i < n \colon f_{v_i} = \infty$ or $\exists i \exists (a, a') \in Z_i^v \colon f_{a'} - d(a') < f_a < \infty$." Then we can write Definition 5.4 as $f_v \geq f_{v_{n_v}}, f_v \geq M y^v$. To ensure $y^v = 1$ whenever one of the f_{v_i} equals ∞, we add the constraint $\forall i < n_v \colon y^v \geq \frac{1 + f_{v_i} - M}{M}$, which forces $y^v = 1$ only when $f_{v_i} > M - 1$. Furthermore,

to ensure $y^v = 1$ whenever some a, a' satisfy $f_{a'} - d(a') < f_a$, we would like to add the constraint

$$\forall i < n_v \forall (a, a') \in Z_i^v : y^v \geq \min\left\{ \tfrac{f_a - f_{a'} + d(a')}{M}, \tfrac{M - f_a}{M} \right\}. \tag{2}$$

This forces $y^v = 1$ only when both $f_{a'} - d(a') < f_a$ and $f_a < M$. To get rid of the minimum, we introduce an auxiliary variable $z^v_{i,a,a'}$ for each $i < n_v$ and $(a, a') \in Z_i^v$ as we did for the OR-condition. We then replace (2) with

$$\forall i < n_v \forall (a, a') \in Z_i^v : y^v \geq \tfrac{f_a - f_{a'} + d(a')}{M} - z^v_{i,a,a'}, \quad y^v \geq \tfrac{M - f_a}{M} - (1 - z^v_{i,a,a'}).$$

Taking all of this together, it can be shown that the constraint $f_v \in [0, M]$ holds automatically for all 'reasonable' f (i.e., if this does not hold for f, then f will not minimize f_{R_T}) and can be replaced by $f_v \in \mathbb{R}$. We then find that the optimization problem (1) can be rewritten into the following MILP problem of Fig. 3. Note that this optimization returns an f with $f_{\mathrm{R}_T} \leq M - 1$ if and only if $\mathcal{S}_T \neq \varnothing$. Hence this optimization can also be used to determine whether T can successfully be attacked.

We note that this is not the only way to encode *min time* analysis into a MILP problem; for instance, instead of using the constant M, one could introduce an additional binary variable per node that denotes whether the node is activated or not. We chose for this approach since this ensures we need fewer optimization variables, even though this means that some equations such as (2) are less intuitive. Note that we get quadratically many constraints above, which is a consequence of the fact that we get a constraint for every pair (a, a') in Definition 3.4.

minimize f_{R_T} subject to:	
$\forall v \in N$	$: f_v \in \mathbb{R},$
$\forall a \in N_{\mathtt{BAS}}$	$: f_a \geq d(a),$
$\forall v \in N_{\mathtt{OR}},\ \forall i \leq n_v$	$: x_i^v \in \{0, 1\},$
$\forall v \in N_{\mathtt{OR}},\ \forall i \leq n_v$	$: f_v \geq f_{v_i} + M(x_i^v - 1),$
$\forall v \in N_{\mathtt{OR}}$	$: \sum_{i \leq n_v} x_i^v \geq 1,$
$\forall v \in N_{\mathtt{AND}},\ \forall i \leq n_v$	$: f_v \geq f_{v_i},$
$\forall v \in N_{\mathtt{SAND}}$	$: y^v \in \{0, 1\},$
$\forall v \in N_{\mathtt{SAND}},\ \forall i < n_v,\ \forall (a, a') \in Z_i^v$	$: z^v_{i,a,a'} \in \{0, 1\},$
$\forall v \in N_{\mathtt{SAND}}$	$: f_v \geq f_{v_{n_v}},$
$\forall v \in N_{\mathtt{SAND}}$	$: f_v \geq M y^v,$
$\forall v \in N_{\mathtt{SAND}},\ \forall i < n_v$	$: y^v \geq \frac{1 + f_{v_i} - M}{M},$
$\forall v \in N_{\mathtt{SAND}},\ \forall i < n_v,\ \forall (a, a') \in Z_i^v$	$: y^v \geq \frac{f_a - f_{a'} + d(a')}{M} - z^v_{i,a,a'},$
$\forall v \in N_{\mathtt{SAND}},\ \forall i < n_v,\ \forall (a, a') \in Z_i^v$	$: y^v \geq \frac{M - f_a + 1}{M} + z^v_{i,a,a'} - 1.$

Fig. 3. The MILP problem for calculating *min time*.

5 Computation Time Reduction

In this section, we introduce an algorithm reducing the complexity of computing $\mathrm{mt}(T)$. The algorithm consists of two components: First, we show that a bottom-up algorithm from [14] can be used to calculate *min time* for static (no SAND-gates) and tree-shaped DATs. As the state of the art method, based on binary decision diagrams [5], has exponential complexity, and the bottom-up algorithm has linear complexity, this is a big improvement. Second, we split up the calculation of *min time* into parts by identifying the *modules* of a DAT, i.e. subDAGs that are connected to the rest of the DAT via only one node.

5.1 Bottom-Up Computation

An important tool is the algorithm MT-BU introduced in [14] presented in Algorithm 4. It attempts to calculate $\mathrm{mt}(T)$ by traversing T bottom-up, which only has linear time complexity and is significantly faster than the MILP approach of Fig. 3. For tree-shaped T it calculates *min time* correctly, but for DAGs it fails to account for the fact that two children of a node may share BAS, which may be counted double. However, this double counting is only an issue for SAND-gates, as the operators min/max of OR/AND-gates are idempotent, i.e., $\min(x,x) = \max(x,x) = x$. This was first realized in [17], for attack-defense trees under different semantics. However, *min time* based on these *set semantics* can be proven to be equivalent to our definition in Sect. 3.2, yielding the following result (Fig. 4):

Input: Dynamic attack tree T, duration vector $d \in \mathbb{R}^{N_{\mathtt{BAS}}}$
Output: Potential min time $\mathrm{mt}(T,d)$.

if $\gamma(v) = \mathtt{BAS}$ **then**
  return $d(v)$
else if $\gamma(v) = \mathtt{OR}$ **then**
  return $\min_{v' \in \mathrm{ch}(v)}$ MT-BU$(T_{v'}, d|_{B_{v'}})$
else if $\gamma(v) = \mathtt{AND}$ **then**
  return $\max_{v' \in \mathrm{ch}(v)}$ MT-BU$(T_{v'}, d|_{B_{v'}})$
else // $\gamma(v) = \mathtt{SAND}$
  return $\sum_{v' \in \mathrm{ch}(v)}$ MT-BU$(T_{v'}, d|_{B_{v'}})$

Fig. 4. MT-BU for a DAT T.

Theorem 2 [14,17]. *If T is tree-shaped or static, then* MT-BU *calculates* $\mathrm{mt}(T)$.

5.2 Modular Analysis

Algorithm 4 only reduces complexity in the two relatively rare cases where the DAT is static or tree-shaped. However, it is possible to also reduce complexity when T is only partially static and/or tree-shaped. A well-established method in studying DATs is to consider the *modules* of T:

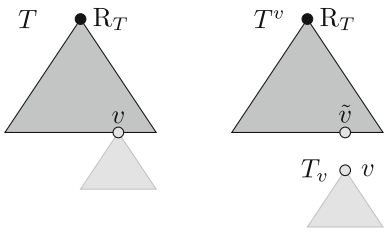


Fig. 5. Modular analysis.

Definition 6 [9]. *A* module *is a node $v \in N \setminus N_{\texttt{BAS}}$ such that all paths from $T \setminus T_v$ to T_v pass through v.*

The root of T is always a module. If v is a module, then v is the only node within T_v with parents outside of T_v. Hence we can create a tree T^v by replacing T_v within T by a new single BAS $\tilde{v}$; the parents of $\tilde{v}$ in T^v are the parents of v in T. Theorem 3 shows that *min time* can be calculated for T by first calculating it for T_v, and then for T^v. This is depicted in Fig. 5.

Theorem 3. *Let T be a DAT, and let v be a module of T. Let T^v be the node obtained by removing v and replacing v itself with a new BAS $\tilde{v}$. Then* $\mathrm{mt}(T,d) = \mathrm{mt}(T^v, d^v)$ *where d^v is a duration function for T^v given by*

$$d^v(a) = \begin{cases} d(a), & \text{if } a \in N_{\texttt{BAS}} \setminus B_v, \\ \mathrm{mt}(T_v, d|_{B_v}), & \text{if } a = \tilde{v}. \end{cases}$$

While the statement seems intuitively true, the proof requires quite a bit of work as one needs to develop machinery to relate attacks on T (and their minimal chains) to attacks on T_v and T^v. Theorem 3 reduces complexity in two ways: We split the tree into two parts whose total size is the same as the original tree. Since MILP is NP-hard, this can impact computation time. Furthermore, the smaller DAT T_v can be static or tree-shaped, in which case we can use `MT-BU` (Fig. 6).

Input: Dynamic AT T, duration vector $d \in \mathbb{R}^{N_{\texttt{BAS}}}$, Algorithm $\mathcal{A}$ to calculate *min time*
Output: Min time $\mathrm{mt}(T)$.

$\mathcal{V} \leftarrow$ `Module`(T);
while $\mathcal{V} \neq \varnothing$ **do**
  Pick $v \in \mathcal{V}$ of minimal height;
  if T_v *is static* **then**
    $d^v(\tilde{v}) \leftarrow$ `MT-BU`$(T_v, d|_{B_v})$;
  else
    $d^v(\tilde{v}) \leftarrow \mathcal{A}(T_v, d|_{B_v})$;
  for $a \in N_{\mathrm{BAS}} \setminus B_v$ **do**
    $d^v(a) \leftarrow d(a)$;
  $(T, d) \leftarrow (T^v, d^v)$;
  $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v\}$;
return $d(\mathrm{R}_T)$ // R_T `is a BAS now`

Fig. 6. $\mathcal{A}_{\texttt{Mod}}$ for a DAT T. The notation T^v, d^v is from Theorem 3.

The resulting algorithm is displayed in Algorithm 6. Here `Module` refers to an algorithm that finds the modules of T; this can be done with linear time complexity [9]. Algorithm $\mathcal{A}_{\texttt{Mod}}$ makes use of an algorithm $\mathcal{A}$ that calculates *min time*. For this, one can use naive enumeration or the MILP approach of Fig. 3, or potentially any new algorithm. Since the calculation of a module's *min time* value depends on its own modules, we act on the lower modules first, so Algorithm 6 handles the modules by ascending height. Note that when T is tree-shaped, every inner node is a module, so $\mathcal{A}_{\texttt{Mod}}$ is equivalent to `MT-BU` for any $\mathcal{A}$.

We note that other definitions of *min time*, such as the automata-approach of [18] and the multiple-activation definition of [14], also allow for modular decomposition. However, as these definitions are not compatible with ours, we cannot directly use these results, and we require a novel proof for Theorem 3.

6 Experiments

This section evaluates the performance of our methods. We compare the MILP approach of Fig. 3 (`MT-MILP`) to the enumerative approach (`MT-Enum`). For the latter, rather than exhaustively generating all succesful attacks, we generate bottom-up a set of candidate attacks that include all minimal succesful attacks, hence certainly the optimal attack by the monotonicity of t. For this we generalize the set semantics of [17] to dynamic ATs. We also compare `MT-MILP` and `MT-Enum` to their modular counterparts.

Existing methods in the literature are based on series-parallel graphs [14] and priced timed automata [18]. Their definitions of *min time* are not equivalent to ours. In our view, methods with different definitions of *min time* can only be compared with respect to computation time if one of them is designed to be an approximation or bound of the other; then one can compare the gain in computation time versus the loss in accuracy. However, this is not the case here: the multiple activation definition is fundamentally different, and a DAT constructed under this model represents a system different from the same DAT in the single activation model. Therefore, we cannot directly compare performance to that of existing approaches.

Source	$\|N\|$	tree	Durations
[18] Fig. 1	12	no	unknown
[18] Fig. 8	20	no	unknown
[18] Fig. 9	12	no	unknown
[3] Fig. 1	16	yes	unknown
[4] Fig. 3	8	yes	known
[4] Fig. 5	21	yes	known
[4] Fig. 7	25	yes	known
[11] Fig. 2	20	yes	unknown
[17] Fig. 1	15	yes	unknown

Fig. 7. DATs from the literature used as building blocks. Trees from [11,17] are adapted from attack-defense Tree.

In practice, attack trees can be very large [26,30]; however, for confidentiality reasons these are typically not disclosed to the general public [7,26]. Hence to our knowledge no established benchmark suites of DATs exist, and the existing literature typically considers test cases with only ≤ 25 nodes [4,18]. For such small DATs, the computation of *min time* takes less than a second no matter which algorithm is being used, which makes them unsuitable for testing difference in algorithm performance. To address the deficiency of a benchmark suite of large DATs, we create a synthetic set of testing DATs. These are created by combining DATs from the literature into larger ones. Then, we compare (1) the MILP method `MT-MILP` to the enumerative algorithm `MT-Enum` and (2) the effect of modular analysis on performance time.

All experiments are performed on a PC with an Intel Core i7-10750HQ 2.8 GHz processor and 16 GB memory. All algorithms are implemented in Matlab, and for MILP we use the YALMIP environment [20] to translate the optimization problem into the Gurobi solver [13], a state-of-the-art optimizer that can handle MILP problems. The code and results are available in [22].

6.1 Generation of Testing DATs

To create a testing suite large enough for a meaningful performance comparison, we do the following. As building blocks, we use a selection of DATs from

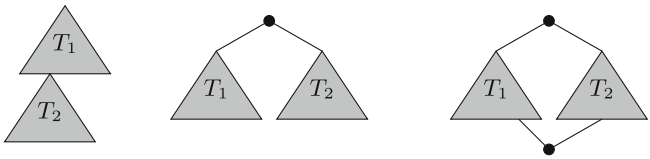


Fig. 8. The three ways of combining DATs.

the literature, shown in Fig. 7. For some, the duration of the BAS are random variables, and we take the expected value for the duration; otherwise we take a random duration from $\{1, 2, \ldots, 10\}$. We use three methods for combining two DATs T_1, T_2 into a larger one (see Fig. 8):

1. We take a random BAS v from T_1 and consider the modular composition by replacing v in T_1 by T_2. This represents a larger system, in which one subsystem, represented by v in T_1, is given its own DAT for more fine-grained analysis.
2. We introduce a new root node v with a random label, and add edges (v, R_{T_1}) and (v, R_{T_2}). This represents a system consisting of two separate subsystems.
3. We introduce a new root node v with a random label, and add edges (v, R_{T_1}) and (v, R_{T_2}); we then pick random BAS b_1 from T_1 and b_2 from T_2 and identify them (with a new random duration in $\{1, 2, \ldots, 10\}$). This represents a system consisting of two subsystems that have a shared attack step.

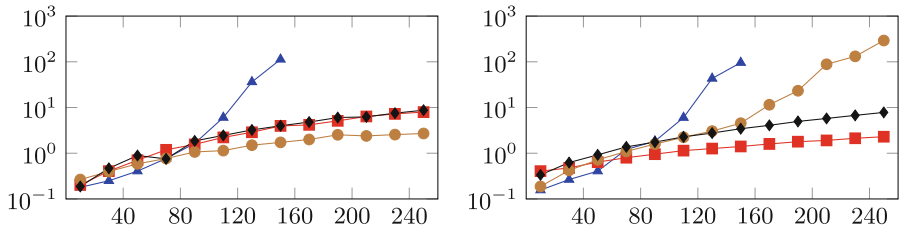
These are not the only ways by which multiple DATs can be combined; for instance, T_1 and T_2 could share multiple BAS. We selected these three methods to capture some of the common ways DATs are created by experts. Creating a benchmark suite of large DATs that resemble DATs from industry is an important avenue for further research, but beyond the scope of this paper.

We create two suites of testing DATs by combining the DATs from Fig. 7. For the first suite, $\mathcal{A}$, we combine DATs using one of the three methods above (drawn randomly) until the result has a given number of nodes. The resulting will have many modules, as T_1 is a module under the first method, and both T_1 and T_2 are modules under the second method. Therefore, we expect the modular approaches to be very fast on the DATs in $\mathcal{A}$. To also study DATs with less modules, we create the second suite, $\mathcal{B}$, by combining DATs using only the third method. Again, one could assign other weights to the three combination methods to obtain yet different testing suites, but $\mathcal{A}$ and $\mathcal{B}$ represent two of the extremes of what DATs can look like.

For a given $n_{\min}$, we combine DATs randomly drawn from Fig. 7 (either via randomly drawn methods from the 3 above, or by method 3 only) until $|N| \geq n_{\min}$. We do this 5 times for each $1 \leq n_{\min} \leq 240$, giving us two testing sets $\mathcal{A}, \mathcal{B}$ of 1200 DATs with $8 \leq |N| \leq 262$. On average 26.6% of the nodes of ATs in $\mathcal{A}$, and 16.5% of the nodes of ATs in $\mathcal{B}$ are modules. Furthermore 54.2% of the nodes of ATs in $\mathcal{A}$, and 52.5% of the nodes of ATs in $\mathcal{B}$ are BAS.

Table 2. Summary of the results. All times are in seconds. *Failure* denotes failure to compute within 10^4 seconds. $\mathcal{A}_{\text{small}}$ contains 754 DATs with ≤ 160 nodes, and $\mathcal{A}$ contains 1200 DATs with ≤ 262 nodes (including those of $\mathcal{A}_{\text{small}}$). The sets $\mathcal{B}_{\text{small}}$ and $\mathcal{B}$ hold the same amount of DATs of the same size; they are designed to contain less modules.

	$\mathcal{A}_{\text{small}}$				$\mathcal{A}$		
	MT-Enum	MT-MILP	MT-Enum$_{\text{Mod}}$	MT-MILP$_{\text{Mod}}$	MT-MILP	MT-Enum$_{\text{Mod}}$	MT-MILP$_{\text{Mod}}$
Median time	1.234	0.906	1.461	1.680	1.422	2.797	3.070
Max time	10000	7.984	12.656	6.656	19.125	10000	30.469
Failure	3.71%	0%	0%	0%	0%	0.08%	0%
	$\mathcal{B}_{\text{small}}$				$\mathcal{B}$		
Median time	1.391	0.938	1.469	1.656	1.266	3.203	2.773
Max time	10000	4.75	2326	9.484	4.75	10000	9.484
Failure	3.81%	0%	0%	0%	0%	3.08%	0%



(a) $\mathcal{A}$ (b) $\mathcal{B}$

Fig. 9. Median time (in seconds) of MT-Enum, MT-MILP, MT-Enum$_{\text{Mod}}$, MT-MILP$_{\text{Mod}}$, grouped by the number of nodes $|N|$.

6.2 Time Comparisons

We measure the computation time of the four algorithms on the testing set; we cap computation time per DAT at 10^4 s. We group the DATs depending on their value of $\lceil |N|/20 \rceil$ and calculate the median per group: these are presented in Fig. 9. We use the median because it allows us to incorporate the computations that were cancelled after 10^4 s. Since already 21.3% of the DATs of $\mathcal{A}$, and 13.8% of DATs of $\mathcal{B}$, with $141 \leq |N| \leq 160$ fail to compute for MT-Enum, we do not continue testing this method for larger DATs. The subsets of $\mathcal{A}, \mathcal{B}$ of DATs with $|N| \leq 160$ is called $\mathcal{A}_{\text{small}}, \mathcal{B}_{\text{small}}$, and consist of 754 resp. 761 DATs. The results are also summarized in Table 2, and pairwise comparisons are presented in Fig. 10.

On the testing set $\mathcal{A}$, we see from Fig. 9 that MT-Enum is by far the slowest method, while MT-MILP is the fastest; the two modular approaches are slightly slower than MT-MILP and have similar efficiency. While the inefficiency of MT-Enum is to be expected, it is surprising that modular analysis for MILP has a net negative effect on computation time. One possible reason is that the Gurobi solver,

which we treat as a black box, might incorporate strategies to reduce the MILP problem complexity that are equivalent to modular analysis on the DAT side. At any rate, the enumerative approach clearly shows the advantage of incorporating the modular approach. These results are also reflected in Fig. 10(a)–(d).

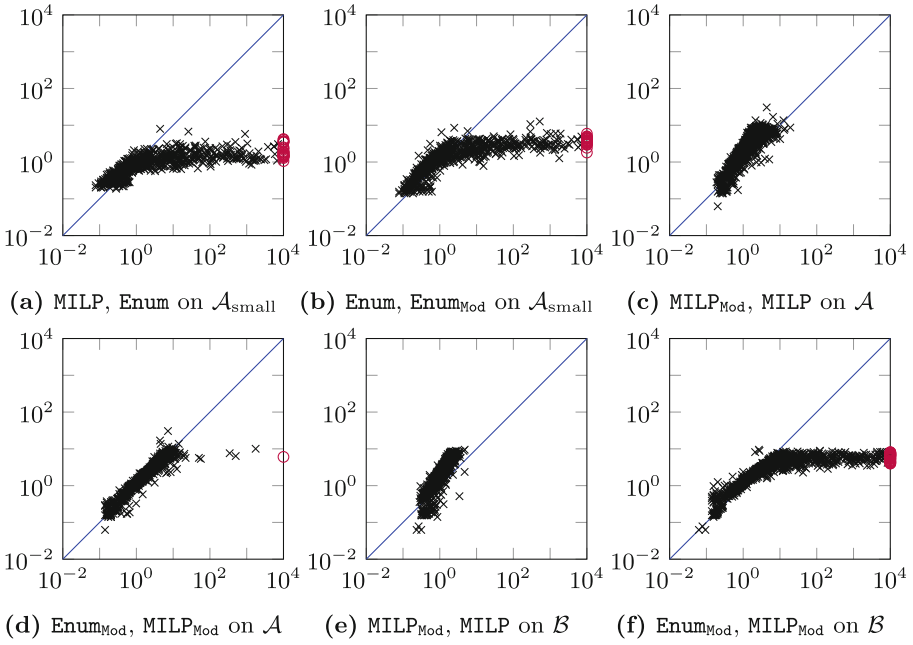


Fig. 10. Pairwise computation time comparisons of the four algorithms. The first algorithm is the vertical axis while the second is the horizontal axis. Each mark is a DAT; purple circles are computations aborted for exceeding 10^4 s. (Color figure online)

Interestingly, the difference in median computation time between MT-Enum and MT-MILP disappears when considering the modular versions of these algorithms, although the worst-case behaviour of $\text{MT-Enum}_{\text{Mod}}$ is considerably worse than that of $\text{MT-MILP}_{\text{Mod}}$ (see Table 2). We hypothesize that this is due to the fact that the DATs of $\mathcal{A}$ contain many modules. As a result, the 'indecomposable' sub-DATs on which the algorithms MT-Enum and MT-MILP are called will typically be small. Since the difference in computation time between these algorithms only appears for larger DATs, we do not see it in these experiments.

For testing set $\mathcal{B}$, we again see that MT-Enum is by far the slowest. Furthermore, for larger DATs $\text{MT-MILP}_{\text{Mod}}$ outpaces $\text{MT-Enum}_{\text{Mod}}$ considerably; see also Fig. 10(f). This shows that also in a modular setting the MILP approach significantly speeds up calculations for large enough DATs. This is to be expected from our results on set $\mathcal{A}$ as for larger DATs the 'indecomposable' subDATs on which MT-MILP is invoked will be larger as well. Interestingly, on this dataset MT-MILP is slightly

faster than $\texttt{MT-MILP}_{\texttt{Mod}}$, as can also be seen from Fig. 10(e). This might be due to the fact that on wide DATs, the MILP methods of Gurobi are more efficient at splitting up DATs into modules than our Matlab implementation of the modular decomposition algorithm. A detailed study into this difference in performance would entail a comprehensive analysis into Gurobi's Matlab implementation, which is beyond the scope of this paper.

Taking $\mathcal{A}$ and $\mathcal{B}$ together, we can conclude that both the MILP approach and modular analysis create a large decrease in computation time. While these methods are slightly slower for small DATs, computation time for such DATs only takes a few seconds anyway. By contrast, for larger DATs the difference in computation time can go up to a factor 10^3. For DATs with large modules, $\texttt{MT-Enum}_{\texttt{Mod}}$ loses out against MT-MILP and $\texttt{MT-MILP}_{\texttt{Mod}}$, which behave similarly.

7 Conclusion and Discussion

This paper introduced two novel tools to calculate *min time* for DATs. First, we introduced a novel MILP-based approach that finds *min time* by phrasing it as an optimization problem. Second, we show how modular analysis can be used to reduce the computation time of any *min time* calculation algorithm. In the experiments, we compared these to the enumerative method. The experiments show that for large DATs both MILP and modular analysis can have a big impact on computation time. In particular, the MILP approach is consistently fast on any input DAT, making it a reliable tool for quantitative DAT analysis in practice.

There are several directions in which this work can be expanded. First, a benchmark suite of DATs is needed. For this it is important to find out what sizes and properties are typical for DATs used in industry, even if industry DATs themselves may not be published due to confidentiality reasons.

Second, modular analysis can also be used for other metrics, as has been done for fault trees [27,29]. Since modular analysis is a very general idea, a good approach would be to develop an axiomatization of metrics that can be handled via modular analysis, so that the method can be applied to a large set of metrics at once. Such a result is probably not hard to prove for metrics that are defined bottom-up as in [17]; the challenge lies in metrics that are defined directly from the semantics as in [5].

Third, our MILP approach can be combined with a Monte Carlo approach in a stochastic setting where the precise BAS values are unknown. A more thorough investigation can explore what guarantees such simulations can give for *min time*. As Monte Carlo methods involve sampling a large sample, performance of the *min time* calculation algorithm is important in such a study.

References

1. Agyepong, E., Cherdantseva, Y., Reinecke, P., Burnap, P.: Challenges and performance metrics for security operations center analysts: a systematic review. J. Cyber Secur. Technol. **4**(3), 125–152 (2020)
2. Ali, A.T., Gruska, D.P.: Attack trees with time constraints. In: CS&P, pp. 93–105 (2021)
3. Arnold, F., Guck, D., Kumar, R., Stoelinga, M.: Sequential and parallel attack tree modelling. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 291–299. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_25
4. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-dependent analysis of attacks. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 285–305. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_16
5. Budde, C.E., Stoelinga, M.: Efficient algorithms for quantitative attack tree analysis. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF), pp. 1–15 (2021). https://doi.org/10.1109/CSF51468.2021.00041
6. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational choice of security measures via multi-parameter attack trees. In: Lopez, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 235–248. Springer, Heidelberg (2006). https://doi.org/10.1007/11962977_19
7. Byres, E.J., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: Proceedings of the International Infrastructure Survivability Workshop, pp. 3–10. Citeseer (2004)
8. Chen, D.S., Batson, R.G., Dang, Y.: Applied Integer Programming: Modeling and Solution. Wiley, New York (2011)
9. Dutuit, Y., Rauzy, A.: A linear-time algorithm to find modules of fault trees. IEEE Trans. Reliab. **45**(3), 422–425 (1996)
10. Fila, B., Wideł, W.: Exploiting attack-defense trees to find an optimal set of countermeasures. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF), pp. 395–410. IEEE (2020)
11. Fraile, M., Ford, M., Gadyatskaya, O., Kumar, R., Stoelinga, M., Trujillo-Rasua, R.: Using attack-defense trees to analyze threats and countermeasures in an ATM: a case study. In: Horkoff, J., Jeusfeld, M.A., Persson, A. (eds.) PoEM 2016. LNBIP, vol. 267, pp. 326–334. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48393-1_24
12. François-Xavier, A., Olivier, B., Grégory, B., Vania, C., Hervé, D.: Bayesian attack model for dynamic risk assessment. arXiv:1606.09042 (2016). Preprint
13. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022). https://www.gurobi.com
14. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 339–353. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_23
15. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1036–1051. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_8

16. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: DAG-based attack and defense modeling: don't miss the forest for the attack trees. Comput. Sci. Rev. **13**, 1–38 (2014)
17. Kordy, B., Widel, W.: On quantitative analysis of attack–defense trees with repeated labels. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 325–346. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89722-6_14
18. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 156–171. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22975-1_11
19. Kumar, R., et al.: Effective analysis of attack trees: a model-driven approach. In: Russo, A., Schürr, A. (eds.) FASE 2018. LNCS, vol. 10802, pp. 56–73. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89363-1_4
20. Lofberg, J.: YALMIP: a toolbox for modeling and optimization in MATLAB. In: 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No. 04CH37508), pp. 284–289. IEEE (2004)
21. Lopuhaä-Zwakenberg, M., Stoelinga, M.: Attack time analysis in dynamic attack trees via integer linear programming. arXiv:2111.05114 (2021). Preprint
22. Lopuhaä-Zwakenberg, M.: Attack time analysis in dynamic attack trees via integer linear programming (2023). https://zenodo.org/record/8173951
23. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_17
24. McQueen, M.A., Boyer, W.F., Flynn, M.A., Beitel, G.A.: Time-to-compromise model for cyber risk reduction estimation. In: Gollmann, D., Massacci, F., Yautsiukhin, A. (eds.) Quality of Protection. Advances in Information Security, vol. 23, pp. 49–64. Springer, Cham (2006). https://doi.org/10.1007/978-0-387-36584-8_5
25. Meyur, R.: A Bayesian attack tree based approach to assess cyber-physical security of power system. In: 2020 IEEE Texas Power and Energy Conference (TPEC), pp. 1–6. IEEE (2020)
26. Paul, S.: Towards automating the construction & maintenance of attack trees: a feasibility study. arXiv:1404.1986 (2014). Preprint
27. Reay, K.A., Andrews, J.D.: A fault tree analysis strategy using binary decision diagrams. Reliab. Eng. Syst. Saf. **78**(1), 45–56 (2002)
28. Rencelj Ling, E., Ekstedt, M.: Estimating the time-to-compromise of exploiting industrial control system vulnerabilities. In: 8th International Conference on Information Systems Security and Privacy-ICISSP, vol. 1, pp. 96–107 (2022)
29. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15**, 29–62 (2015)
30. Vigo, R., Nielson, F., Nielson, H.R.: Automated generation of attack trees. In: 2014 IEEE 27th Computer Security Foundations Symposium, pp. 337–350. IEEE (2014)
31. Vitkus, D., Salter, J., Goranin, N., Čeponis, D.: Method for attack tree data transformation and import into it risk analysis expert systems. Appl. Sci. **10**(23), 8423 (2020)
32. Widel, W., Audinot, M., Fila, B., Pinchinat, S.: Beyond 2014: formal methods for attack tree-based security modeling. ACM Comput. Surv. (CSUR) **52**(4), 1–36 (2019)