ADaCS: A Tool for Analysing Data Collection Strategies

John C. Mace(✉), Nipun Thekkummal, Charles Morisset, and Aad Van Moorsel

School of Computing Science, Newcastle University, Newcastle upon Tyne NE1 7RU, UK
{john.mace,N.B.Thekkummal1,charles.morisset, aad.vanmoorsel}@newcastle.ac.uk

Abstract. Given a model with multiple input parameters, and multiple possible sources for collecting data for those parameters, a data collection strategy is a way of deciding from which sources to sample data, in order to reduce the variance on the output of the model. Cain and Van Moorsel have previously formulated the problem of optimal data collection strategy, when each parameter can be associated with a prior normal distribution, and when sampling is associated with a cost. In this paper, we present ADaCS, a new tool built as an extension of PRISM, which automatically analyses all possible data collection strategies for a model, and selects the optimal one. We illustrate ADaCS on attack trees, which are a structured approach to analyse the impact and the likelihood of success of attacks and defenses on computer and socio-technical systems. Furthermore, we introduce a new strategy exploration heuristic that significantly improves on a brute force approach.

Keywords: Security modeling · Risk management · Attack trees · Experiment design · Data collection

1 Introduction

To obtain model results that reflect realistic systems accurately, one collects data from different sources and parameterises various variables in the model. Each data collection source might have a different cost and a different accuracy. For instance, a model for political forecast might take as input polls from population samples, or aggregated data from social media. While opinions polls might bring an option to ask specific questions, collecting data from social media might bring a more global picture. Ideally, an organisation interested in political forecast should collect data from both sources, but in practice, the combined cost might be too high.

Cain and Van Moorsel proposed in [4] a general model for optimising data collection strategies, based on the variance generated by these strategies and a cost budget. We refine this approach here by presenting the tool ADaCS, which takes a model as input, together with a cost model and a description of each source for each input parameter of the model, and returns automatically the best *data collection strategy*, i.e., the optimal way to spend a given budget on data collection. We illustrate ADaCS in the domain of attack trees, which take, among others, the likelihood of success of attacks and defensive mechanisms as input parameters. We believe that ADaCS could be useful to a security architect wanting to understand the best data collection strategy for a given attack tree.

The main contribution of this paper is therefore the tool ADaCS (Sect. 3), which is implemented as an extension of the probabilistic model-checker PRISM, together with the application of this tool on attack-trees (Sect. 4). We also present a heuristic for finding an approximation of the optimal data collection strategy reducing significantly the search space (Sect. 5).

2 Problem Formulation

In this section we articulate the data collection problem as a mathematical optimization problem, based on the earlier work of Cain and Van Moorsel [4]. Since we concentrate in this paper on tool support and practical applications, our presentation of the theory will be less general, but this reduces the complexity of the notation compared to [4]. We first present a general formulation of the problem, which we then refine on attack trees.

2.1 Optimal Data Collection Strategy

Let us consider a model, taking as inputs a set of parameters P_1, ..., P_n, and outputting a random variable Y. This model can be an attack tree, as we explain below, or any other discrete-event dynamic system model [11]. In order to compute the expected value of Y, which we write $E[Y]$, we consider a simulation-based approach: we repeat a number of M runs, such that at each run j, we instantiate each P_i with a value p_{ij}, compute the corresponding value y_j; the expected value of Y is calculated as $E[Y] = \sum_{j=1}^{M} y_i / M$, and the variance by $\sum_{j=i}^{M} (y_i - E[Y])^2/(M-1)$ [6].

The crucial point here is to instantiate each P_i with a value p_{ij}. Without any information, we could simply take a random value from the domain of P_i. However, many models can use data collection to determine the value of the parameters, for instance with the environmental metrics described above. For instance, in a socio-technical setting, such as the 'USB model' introduced in [3], the model describes the different security threats related to the use of USB sticks within an organisation, a parameter might be the typical behavior patterns of users of IT systems.

Hence, we now consider that each parameter P_i can be associated with a normal distribution $\mathcal{N}(\mu_i, \sigma_i/\sqrt{M_i})$ (this follows from the Central Limit Theorem, e.g., [6]), where μ_i represents the mean value for P_i, σ_i^2 the variance, and M_i the number of samples used to compute these values. In the USB model example, we could for instance ask 30 employees about how often they carry a USB stick to their customer, and we could go through 100 travel bookings to identify the duration of travel.

One of the key points of [4] is that a small number of data source samples M_i for P_i implies a wide normal distribution, and therefore implies a large variance in the values p_i drawn, which then implies a large variance in the output $E[Y]$ (which is not desirable). Their key argument is therefore to use the Central Limit Theorem to reflect that when one increases the number of samples M_i it will narrow down the normal distribution for P_i. Given an additional number of samples $N_i > 0$, one would run simulations drawing values p_{ij} from $\mathcal{N}(\mu_i, \sigma_i^2/\sqrt{M_i + N_i})$.

In order to add samples, given a set of parameters $X = \{P_1, \ldots, P_n\}$, we assume the existence of a set of sources $D = \{d_1, \ldots, d_m\}$, such that each source is a predicate $d_i : X \rightarrow \mathbb{B}$ (a source can sample multiple parameters at the same time), indicating which parameters it can sample. Each source d_i is associated with a cost c_i, indicating the cost of one sample of d_i (the cost is the same regardless of the number of parameters d_i can sample). A cost here represents an abstract notion, which could for instance correspond to a monetary value or the time required to sample.

A *data collection strategy* s is a set $\{N_1, \ldots, N_m\}$, indicating a number of samples $N_j(s)$ for each source d_j. Given a parameter X_i, we write $N_i(s)$ for the number of samples collected from all the sources, i.e., $N_i(s) = \sum_{d_j \in D | d_j(x_i)} N_j(s)$. The variance of Y using the strategy s, which we write $Var[E[Y] \mid s]$, is calculated by drawing for each parameter P_i the value p_i from $\mathcal{N}(\mu_i, \sigma_i^2/\sqrt{M_i + N_i(s)})$ in the simulation-based approach described above. Note that $Var[E[Y] \mid s]$ can be calculated using the equations in the first paragraph of this section, but that now the randomness is not only associated with the model but also with the parameter uncertainty under strategy s.

It is worth emphasising here that a strategy s decides which parameter distribution to narrow. Trivially, the best strategy would therefore be to add as many samples as possible for each source. However, in practice, a data collection strategy is bound by a budget, which leads us to the definition of the optimal collection strategy, simplified from [4].

Definition 1 (Optimal Strategy). *Given a budget B, a cost c_i for each sample provided by data source d_i, and a set S of all possible strategies, the optimal strategy is defined as:*

$$\arg\min_{s \in S} Var[E[Y] \mid s] \text{ subject to: } \sum_{i=1}^{m} N_i(s) \cdot c_i \leq B.$$

2.2 Attack Defence Trees

In an attack tree (AT) [17] each leaf corresponds to a basic action on the system, and each node is a composition of sub-trees using the logical operators $\wedge$ and $\vee$. For the sake of exposition, we might associate a node with a label, representing the attack corresponding to that node. For instance, a brute force attack on a password usually requires two sub-attacks to be successful: having the hash for the password, and being able to crack it. If we represent these two attacks by the atomic actions $\mathsf{gethash}$ and crack, respectively, then we can define the attack tree for a brute force attack as[1] $\mathsf{bf} = \wedge(\mathsf{gethash}, \mathsf{crack})$. Similarly, we can represent the attack of getting the password for an account by either brute-forcing it, or by stealing it: $\mathsf{getpw} = \vee(\mathsf{bf}, \mathsf{steal})$. In the remainder of this paper, we assume it is clear from the context whether a label corresponds to an atomic action (such as crack) or a composite attack (such as bf).

An attack-defence tree (ADT) [13] is an attack tree with the addition of the logical operator $\neg$, corresponding to the negation. Intuitively speaking, basic actions can either be attacks or defensive mechanisms, and the negation of a defensive mechanism corresponds to an attack. For instance, a typical defensive mechanism against someone getting the password is to two-factor authentication (TFA). Hence, getting the account can be defined as the conjunction of getting the password and not using TFA: $\mathsf{getaccount} = \wedge(\mathsf{getpw}, \neg\mathsf{tfa})$.

ADT are useful to calculate the likelihood of an attack to succeed, by breaking it down to the likelihood of all atomic actions to succeed [2]. More precisely, given an ADT t with a set $A = \{a_1, \ldots, a_n\}$ of atomic actions, such that $p_i \in [0, 1]$ represents the likelihood of action a_i to succeed, we can define the function p_s which returns the likelihood of t to succeed as:

$$p_s(t) = \begin{cases} \max(p_s(t_1), \cdots, p_s(t_k)) & \text{if } t = \vee(t_1, \ldots, t_k), \\ p_s(t_1) * \cdots * p_s(t_k) & \text{if } t = \wedge(t_1, \ldots, t_k), \\ 1 - p_s(t') & \text{if } t = \neg t', \\ p_i & \text{if } t = a_i \end{cases}$$

In general, the probabilities p_i correspond to a parameter P_i which can be collected from the environment: the probability of a hashed password to be cracked depends on the time spent by the attacker and the entropy of the password domain; the probability of a user using two-factor authentication (assuming the user has such a choice) can be statistically computed; etc. The variance of these probabilities will depend on the source for data collection: an in-depth analysis of the system (e.g., using penetration testing techniques) will provide an accurate but costly view of the system, while relying on general statistics might be cheaper but with more variance. In this paper, we formulate the problem of data collection in the context of attack-defence trees: given a tree t, the parameters we consider are the likelihood of success $X = \{P_1, \ldots, P_n\}$ for all atomic actions, and the variable Y corresponds to the probability $p_s(t)$ to be higher than a given threshold.

3 ADaCS

In this section we introduce the tool ADaCS (*Analysing Data Collection Strategies*). ADaCS has been created by extending the probabilistic model checking

[1] We use here a prefix notation to avoid any ambiguity.

tool PRISM [14] with the functionality to analyse data collection strategies and compute the best strategy within a given budget. PRISM is used to verify the existence of different model properties such as the probability of reaching a specific system state. We begin this section by describing the process ADaCS takes to find the best data collection strategy. We then describe the main components of ADaCS in more detail.

3.1 Process of Finding Best Strategy

Figure 1 shows the main components of ADaCS and the process of computing an optimal data collection strategy for a given parameterised model: (1) files encoding the parameterised *PRISM model*, model *verification property*, and data collection *strategy configuration* are inputted to ADaCS; (2) the *strategy generator* analyses the inputted strategy configuration and generates all valid data collection strategies for the given model; (3) each valid strategy is tested M times in order to compute the model's mean output value (i.e., probability of holding verification property), and variance under that strategy. Each test run of a single data collection strategy is as follows: (3a) the *sample generator* generates the correct number of data samples per input parameter in accordance to the strategy and whose values respect the data source normal distributions encoded in the given strategy configuration; (3b) the *parameter generator* generates input parameter values for the model, where each input value represents a mean of the data source samples generated for that parameter; (3c) the *PRISM model checker* computes the maximum probability of the verification property existing in the given model under the generated input parameters; (3d) the probability value computed by PRISM is placed in the *result storage*. (4) the *strategy analyser* checks the M results generated under each strategy and computes the mean output and variance of the model under that strategy. (5) the *optimal strategy*, that is the strategy providing the minimum output variance, is outputted to a text file together with the strategy's variance and cost. We now describe the main components of ADaCS in more detail.

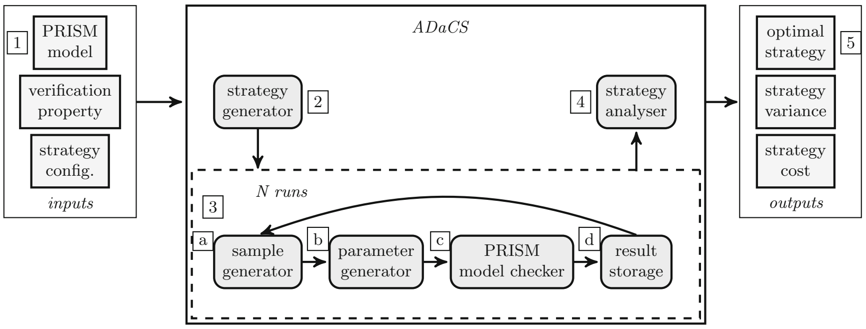

Fig. 1. Optimal strategy generation process

3.2 Strategy Configuration

Information stating how data may be collected is inputted to ADaCS by way of a *strategy configuration file* encoded in the Yet Another Markup Language (`.yaml`). YAML is a human-readable data serialisation language commonly used for configuration files such as the ones ADaCS requires. We will illustrate this by example, see Fig. 2.

```
                data_sources: [d1,d2,d3]
            input_parameters: [P1,P2,P3]
               input_mapping: [[1,0,0],[0,1,0],[0,0,1]]
                 sample_mean: [[0.5,0,0],[0,0.5,0],[0,0,0.5]]
             sample_variance: [[0.01,0,0],[0,0.01,0],[0,0,0.01]]
 pre_collected_sample_count: [0,0,0]
                 sample_cost: [1,1,1]
            sample_increment: [50,50,50]
         sample_startup_cost: [0,0,0]
                  min_values: [0,0,0]
                  max_values: [1,1,1]
                      budget: 200
```

Fig. 2. Example ADaCS strategy configuration file encoded in YAML

Figure 2 shows an example strategy configuration file for a model m_1 with three input parameters P_1, P_2, and P_3 mapped to three data sources d_1, d_2, and d_3. The internal structure of the model m_1 is not particularly relevant here, and is omitted for the sake of exposition. We can assume however that the input parameters of m_1 are probabilities therefore the min and max values are 0 and 1 respectively. The sample increment is 50 for each data source meaning samples can be drawn from each source in sets of 0, 50, 100, 150, and so on. The cost to draw each sample is 1 for each data source and the total maximum data collection budget is 200. A strategy collecting 50, 100, and 50 samples from d_1, d_2, and d_3, respectively, is valid (i.e., respecting the budget) whereas a strategy collecting 50, 100, and 100 samples from d_1, d_2, and d_3, respectively, would be invalid.

3.3 Extending PRISM Model-Checker

ADaCS is an extension of the probabilistic model checking tool PRISM, version 4.3 [14]. PRISM is an intuitive choice as it enables the specification, construction and analysis of parameterised probabilistic models, encoded as Markov chains and Markov decision processes for example. PRISM also comes with command line functionality and an open-source Java code base which is easily adaptable for our purposes. At runtime ADaCS inputs the model being analysed into PRISM which verifies the property expressed in the inputted verification property file.

Algorithm 1. finding optimal data collection strategy

```
1: Inputs:
       files: $config, model, property$
2: Initialize:
       $opt\_strategy$ = null, $runs = 500$
3: $strategies$ = generateAllStrategies($config$)
4: $results\_data$ = emptylist
5: for $j = 0 \rightarrow strategies.length - 1$ do
6:     for $k = 1 \rightarrow runs$ do
7:         $params$ = getParameters($config$)
8:         $output$ = solveModel($model, property, params$)
9:         $results\_data$.add($output$)
10: $opt\_strategy$ = getOptimalStrategy($results\_data$)
11: $variance$ = getVariance($opt\_strategy$)
12: $cost$ = getCost($opt\_strategy$)
13: return $opt\_strategy, variance, cost$
```

For instance, assume we wish PRISM to verify the maximum probability that a state can eventually be reached in a model m_1. This property is expressed as Pmax=? [F state], where F is the eventually operator and state is the model state we are interested in reaching.

ADaCS extends PRISM with the a new `-adacs` command line switch such that the following command can be executed:

```
   $ prism m1.prism m1.props -adacs m1_config.yaml -exportresults
m1.adacs
```

The command tells PRISM to find the optimal data collection strategy under strategy configuration `m1.yaml`, given `m1.prism` and `model.props`, and output the strategy to `m1.adacs`. Algorithm 1 has been implemented in ADaCS which finds the optimal strategy, its variance and cost, by exploring all possible strategies. Example output written to `m1.adacs` is of the form:

```
 optimal strategy: [100,50,50]
strategy variance: 0.02
    strategy cost: 200
```

The optimal strategy `[100,50,50]` states that 100 budgetary units should be invested in collecting data for parameter P_1 from data source d_1, 50 units for P_2 from d_2, and 50 units for P_3 from d_3.

4 Computer Virus Attack

In this section we demonstrate ADaCS by modelling a probabilistic computer virus attack scenario in PRISM and use ADaCS to calculate the optimal data collection strategy. Such analyses could be useful to security architects by informing them where best to invest limited time or money in analysing the likelihood its systems will fall prone to attacks. We describe an attack-defence tree of a virus attack, how the virus ADT is analysed with ADaCS, and present data collection strategy analysis results and tool performance.

4.1 Attack-Defence Tree

We consider the scenario of an attacker trying to infect a computer with a virus presented by Aslanyan, Neilson, and Parker in [2]. It is assumed an attacker will try to infect a computer in two phases. Attack Phase 1 (AP1) involves trying to put the virus file on the computer system followed by Attack Phase 2 (AP2), which involves executing the virus file. Two defence mechanisms exist on the computer system to prevent such an attack. First, an anti-virus mechanism aims to prevent the success of AP1, and second, a system rollback mechanism aims to prevent the success of AP2.

Figure 3 depicts the virus attack as an attack-defence tree, where each leaf corresponds to a basic action on the computer system. More precisely, there are three attack actions: (1) email, attacker sends virus as an email attachment; (2) usb, attacker distributes virus on a usb stick; (3) exefile, attacker executes virus; and two defence actions: (1) antivirus, anti-virus detects and removes virus; (2) rollback, system rollback restores system to secure state.

In order for the virus attack to succeed both attack phases AP1 and AP2 must succeed, in other words *attack success* $= \overrightarrow{\wedge}$(AP1,AP2). The right arrow indicates AP1 must take place before AP2. For AP1 to be successful the virus must be uploaded and has not been subsequently detected or stopped by the anti-virus mechanism, such that AP1 $= \overrightarrow{\wedge}$(*virus upload*, $\neg$antivirus). It follows that *virus upload* $= \vee$(email, usb) meaning the attacker must send an email and/or distribute a usb stick containing the virus in order to upload it to the system. For AP2 to be successful the virus must be executed and the system has not been restored by the rollback mechanism, such that AP2 $= \wedge$(exefile, $\neg$rollback).

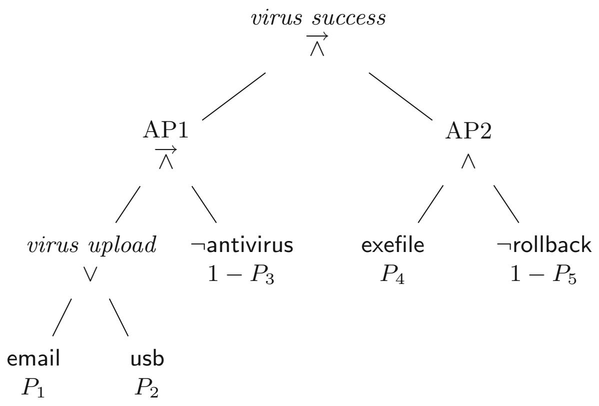

Fig. 3. Attack-defence tree representing virus attack on a computer system

4.2 Analysing Attack-Defence Tree with ADaCS

We model the virus ADT as a Markov decision process shown in Fig. 4 where the darker nodes represent virus attack failure states. The virus attack is successful if the state [¬rollback, exefile] can be reached. We assume the attacker's choice of virus deployment method is a non-deterministic attack action deployvirus whilst all other actions, email, usb, antivirus, rollback, and exefile, are probabilistic. The model is encoded in the PRISM language and has five input parameters $P_1, \ldots, P_5$ representing the probability of each of the five probabilistic actions succeeding. Each parameter is mapped to an action as follows: (email:P_1), (usb:P_2), (antivirus:P_3), (exefile:P_4), (rollback:P_5) as shown in Fig. 3. For instance, if the state [*virus upload*] is reached, the virus has been placed on the computer system. In this state, anti-virus removes the virus with probability P_3 to reach failure state [antivirus], or does not remove the virus with probability $1 - P_3$ to reach state [¬antivirus]. The model property we verify using the PRISM model checker is Pmax=? [F success], that is the maximum probability of eventually (F is the eventually operator) reaching the state [¬rollback, exefile], labelled as *virus success* in this case.

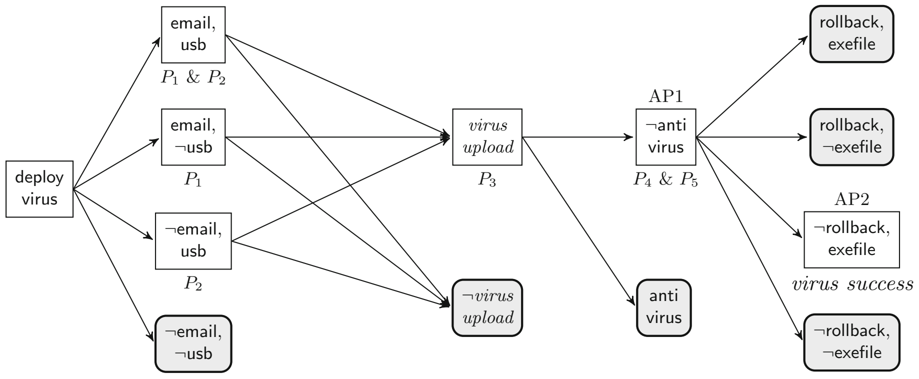

Fig. 4. Markov model style representation of the attack defence tree; darker nodes represent attack failure states.

Next we consider three data sources d_1, d_2 and d_3 from where data can be collected to provide values for the model's five input parameters. Table 1 shows the mapping of parameters to data sources, the normal distribution $\mathcal{N}(\mu, \sigma^2)$ for parameter P_i mapped to data source d_i, and the increment and cost to sample each data source. For instance, data can be collected for P_1 and P_2 from either d_1 or d_2, whereas data for P_5 can only be collected from d_3. Data collected from d_3 for P_5 is drawn from a distribution whose mean $\mu = 0.7$ and variance $\sigma^2 = 0.01$.

We assume attack method data can be sampled from a wide range of literature encompassing scientific and anecdotal evidence of computer virus attacks. Such data is likely to be easy to access but may not be highly accurate, and

Table 1. Virus attack-defence tree data collection configuration

	d_1	d2	d3
email:P_1	(0.5,0.2500)	(0.5,0.0025)	()
usb:P_2	(0.5,0.2500)	(0.5,0.0025)	()
antivirus:P_3	()	(0.7,0.0025)	()
exefile:P_4	(0.5,0.2500)	()	(0.5,0.0100)
rollback:P_5	()	()	(0.7,0.0100)
Sample increment	5	5	5
Sample cost	1	5	3

therefore comes with low cost and high variance. We represent literature-based data as data source $d1$. The values for parameters $P1$, P_2, and P_4, representing the probabilities of attack actions email, usb, and exefile being successful, can be drawn from d_1 in this instance. Next we assume attack data related to attack actions email and usb, and defence action antivirus can be sampled from the results of penetration testing. Such data is likely to be expensive and take a large amount of effort to obtain, but gives accurate scientific evidence and therefore comes with high cost and low variance. We represent penetration test results as data source d_2 from which values for parameters $P1$, P_2, and P_3 can be derived. Lastly, data relating to the actions exefile and rollback can be sampled from the results of system testing, coming with medium cost and variance. Data source d_3 represents the results of such system tests, and can provide values for parameters P_4 and P_5. We further assume data may be collected from each data source in increments of 5 samples (e.g. 5, 10, 15, ...), no samples have been pre-collected, there is no data source startup cost, and the maximum data collection budget is 100.

4.3 Data Collection Strategy Analysis

We analyse data collection strategies by executing ADaCS on a MacBook Pro with 2.7 GHz Intel Core i5 processor and 16 GB RAM. For each strategy we conduct M runs, as explained in Sect. 2, and we choose $M = 500$ based on trial runs, from which we concluded that $M = 500$ is large enough to compute the output variance sufficiently accurate. The choice for M is not the subject of study in this paper, but one potentially could further improve our approach by selecting a different number of runs M for different strategies, as long as for each strategy the calculated variance $Var[E[Y]|s]$ has a tight enough confidence interval. In our case, the virus ADT model is solved by PRISM 500 times for each strategy using input parameter values generated from samples drawn from the Normal distributions corresponding to the relevant data sources which come with a known mean μ and standard deviation σ^2. The variance of a strategy is calculated from the M output values computed by PRISM. Results generated

Table 2. ADaCS data collection strategy analysis results and performance

Strategy type	Samples [d1, d2, d3]	Strategy variance	Strategy cost	Strategies analysed	Runtime (m:s:ms)
Min. cost	[0,0,0]	1.81479E–4	0	1	00:22:067
Max. σ^2/max. cost	[100,0,0]	2.38172E–4	100	540	10:44:674
Min. σ^2	[0,5,25]	1.165047E–5	100	4617	58:16:819

by ADaCS and its performance are given in Table 2. In particular we focus on 3 analysis scenarios in order to illustrate the potential benefit of using ADaCS.

The first scenario highlights the case where no further data is collected to that given in Table 1. Analysis is carried out using ADaCS on the single data collection strategy `[0,0,0]` which indicates no samples are to be collected from any of the 3 data sources. In order to do so the total collection budget is simply set to 0. ADaCS runs the analysis in 22.067 s and computes the output variance of the model using this strategy to be 1.81479E–4.

The second strategy highlights the case where the total budget is spent without optimising the data collection strategy. We use ADaCS to identify the worst case strategy, that is the strategy that maximises the output variance of the virus ADT model with maximum cost (i.e., cost = budget). ADaCS analyses 540 strategies whose cost equals budget in 10 m 45s and returns the worst case strategy to be `[100,0,0]`. This strategy indicates the entire budget to be invested in collected data from data source d_1, that is literature based data regarding attack methods. Note, if this strategy was implemented, the output variance is roughly as bad as when no data would have been collected. In fact, the table shows a small increase in variance, but this difference is caused by the inherent uncertainty introduced by simulation.

The third scenario highlights ADaCS carrying out a full analysis of data collection strategies in order to compute the optimal, that is the strategy within budget which minimises the variance $Var[E[Y]|s]$. To find the optimal strategy, ADaCS analyses all strategies within budget, a total of 4617 strategies in this case. Taking this brute force approach comes with certain computational costs, for instance the runtime of ADaCS is 58 m 17s. The strategy computed as the optimal is `[0,5,25]` indicating 0 samples to be collected from d_1, 5 samples from d_2, and 25 samples from d_3. This is equivalent to 75 budgetary units, or 75% of time invested on the value for P_3 and 25 units, or 25% of time on the value for P_2. The strategy would indicate that system testing of the rollback defence mechanism is of most importance followed by some penetration testing of the antivirus mechanism. No more literature based data on attack methods need be collected which is in complete conflict with the worst case strategy highlighted in the second scenario above. The output variance of the virus ADT model, under the optimal strategy, reduces from 1.81479E–4 to 1.165047E–5.

5 Heuristic

In Sect. 4 we explained that by default ADaCS takes a brute force approach by analysing all data collection strategies within budget in order to find the optimal. We showed calculating the optimal strategy can be computationally expensive, even for models with a relatively small number of parameters and data sources. In this section we introduce a new strategy analysis heuristic that significantly reduces the strategy exploration space in order to find the best strategy.

We introduce the heuristic by example first. Figure 5 illustrates the heuristic for a model m_1, which can be parameterized with values coming from 3 data sources d_1, d_2, and d_3. Each data source has a sample set increment size of 50, cost per sample is 1 and total collection budget is 200. Rather than generating all valid strategies and analysing them, the heuristic generates strategies to analyse as necessary in a series of rounds; 5 rounds in the case of the example. In round 1 the base strategy is generated and analysed, this is the strategy `[0,0,0]` where no data is collected from any data source. Imagine the model output variance is computed to be 0.0676, this is set as the best strategy. In round 2 the next possible strategies are generated. As the sample increment size is 50, the next strategies are `[50,0,0]`, `[0,50,0]`, and `[0,0,50]` which are analysed. If a strategy s in round 2 has a lower output variance than the current best strategy then s is set as the best strategy, `[50,0,0]` in the case of the example. The heuristic moves on to the next round and keeps doing so until no strategy in round i has a lower output variance than the current best strategy, or the cost of the current best strategy equals the budget. The heuristic in this example cuts the brute force strategy space of 35 strategies to 13 strategies.

The general heuristic is presented in Algorithm 2. The heuristic has been implemented in ADaCS and results for 6 strategy analyses of the virus ADT model are shown in Table 3. Note the significant reduction in strategies analysed, 4617 in the brute force approach, to between 40 and 60 and a computation run-time of under 1 min. Note also the best strategy returned in each case indicates that most investment should be made in collecting data from source d_3, followed

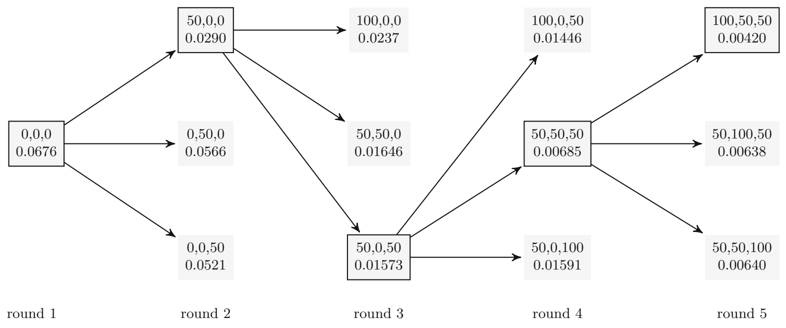

Fig. 5. Illustration of heuristic to find optimal data collection strategy for example model

Algorithm 2. Heuristic for finding best strategy

```
1: Inputs:
       files: $config, model, property$
2: Initialize:
       $best\_strategy$ = null, $runs = 500$
3: while true do
4:     $next\_strategies$ = emptylist
5:     if $best\_strategy$ == null then
6:         $base\_strategy$ = generateBaseStrategy()
7:         $next\_strategies$.add($base\_strategy$)
8:     else
9:         $next\_strategies$ = generateNextStrategies($best\_strategy, config$)
10:    $results\_data$ = emptylist
11:    for $j = 0 \rightarrow next\_strategies.length - 1$ do
12:        for $k = 1 \rightarrow runs$ do
13:            $params$ = getParameters($config$)
14:            $output$ = solveModel($model, property, params$)
15:            $results\_data$.add($output$)
16:    $strategy$ = getBestStrategy($results\_data$)
17:    if getVariance($strategy$) < getVariance($best\_strategy$) &
18: getCost($strategy$) $\leq$ getBudget($config$) then
19:        $best\_strategy = strategy$
20:    else
21:        break
22: $variance$ = getVariance($best\_strategy$)
23: $cost$ = getCost($best\_strategy$)
24: return $best\_strategy, variance, cost$
```

by d_2, and none from d_0 which matches the result of the brute force approach in Sect. 4.3. The heuristic does however only find a best strategy, and not the optimal one as shown by the increase in model output variance in Table 3. This would indicate the existence of a trade-off between strategy optimality (brute force) and computation time (heuristic).

Table 3. Best data collection strategies found using heuristic data collection method

Collection method	Samples [d1, d2, d3]	Strategy variance	Strategy cost	Strategies analysed	Runtime (m:s:ms)
Heuristic$_1$	[0,5,20]	2.56808E-5	85	49	00:46:856
Heuristic$_2$	[0,5,25]	2.19972E-5	100	57	00:52:432
Heuristic$_3$	[0,10,15]	2.65431E-5	95	49	00:45:999
Heuristic$_4$	[0,5,15]	2.68426E-5	70	41	00:39:711
Heuristic$_5$	[0,5,25]	1.99201E-5	100	57	00:51:534
Heuristic$_6$	[0,10,15]	2.61067E-5	95	49	00:46:047

6 Related Work

The data collection strategy optimization approach in this paper combines aspects of two main strands of analysis, namely sensitivity and uncertainty analysis [1,5,16], and adds to that specific detailed techniques based on statistics (the use of the central limit theorem) and optimization. In [4] we presented the related work, and explained that compared to known literature, the problem formulation in this paper is different because of its focus on strategies for deciding on data sources. This leads to a specific optimization problem not found in literature. For more details, please refer to [4]. The contributions in this paper focus on the practical application of and tool support for identifying data collection strategies, and this will also be the focus of this related work section.

The performance and dependability community has developed an important set of software tools that support the quantitative analysis of computer systems and networks, e.g., [15]. The specific problem formulation of the optimization problem in fact assumes that a performance or dependability model is built, and tool support for optimizing data collection would therefore naturally fit with tools such as Möbius [7], PRISM [14] or the PEPA workbench [9]. The data collection problem presented in this paper is related to sensitivity analysis. Sensitivity analysis aims at identifying the most important parameters, using techniques such as those within the classes of screening methods [12] and variance-based methods [5]. Augmenting the mentioned class of quantitative analysis tools for sensitivity analysis was first pursued by [10], which also articulates the computational challenge of exploring the parameter space when conducting sensitivity analysis.

The heuristics identified in this paper to improve the efficiency of exploring the 'space' of strategies is of importance in any of the related approaches in uncertainty and sensitivity analysis. For closed-form mathematical and simulation models with specific structure, advanced methods can be devised for the specific problem, see [12] and other references in [4]. Exploration of the strategy 'space' relates to exploration techniques in very different settings, such as codesign [8], and it may be mutually beneficial for the evaluation and codesign communities to jointly explore ideas and algorithms.

7 Conclusion

In this paper we presented the software tool ADaCS, which, to the best of our knowledge, is the first tool facilitating the calculation for optimal data collection strategies for model parameterization. In particular, ADaCS extends the probabilistic model-checker PRISM and includes both graphical and command line elements, and should therefore be easily usable by a model developer using PRISM. We also illustrated ADaCS on attack trees, which is a standard way to model security attacks and defenses in a system.

Since the bottleneck in determining data collection strategies is in traversing the space of all possible strategies, we elaborated on heuristics that limited the number of strategies for which the output variance needs to be computed. The algorithms can reduce the computation time by an order of magnitude, which may mean the difference between a feasible and intractable optimization.

Acknowledgements. The research in this paper was supported in part by UK EPSRC through 'Choice Architecture for Information Security' (EP/K006568/1) within the Research Institute for the Science of Cyber Security and by US NSA through the University of Illinois at Urbana-Champaign lablet in 'Science of Security Systems' (H98230-14-C-0141).

References

1. Ascough, J., Green, T., Ma, L., Ahuja, L.: Key criteria and selection of sensitivity analysis methods applied to natural resource models. In: Proceedings of the International Congress on Modeling and Simulation (2005)
2. Aslanyan, Z., Nielson, F., Parker, D.: Quantitative verification and synthesis of attack-defence scenarios. In: Proceedings of the 29th IEEE Computer Security Foundations Symposium (CSF 2016), pp. 105–119, June 2016
3. Beautement, A., et al.: Modelling the human and technological costs and benefits of USB memory stick security. In: Johnson, M.E. (ed.) Managing Information Risk and the Economics of Security, pp. 141–163. Springer, Boston (2009). doi:10.1007/978-0-387-09762-6_7
4. Cain, R., Van Moorsel, A.: Optimization of data collection strategies for model-based evaluation and decision-making. In: Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), pp. 1–10 (2012)
5. Chan, K., Saltelli, A., Tarantola, S.: Sensitivity analysis of model output: variance-based methods make the difference. In: Proceedings of the 29th Conference on Winter Simulation, pp. 261–268 (1997)
6. Cochran, W.: Sampling Techniques, 3rd edn. Wiley, New York (1977)
7. Deavours, D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J., Sanders, W., Webster, P.: The Möbius framework and its implementation. IEEE Trans. Softw. Eng. **28**(10), 956–969 (2002)
8. Gamble, C., Pierce, K.: Design space exploration for embedded systems using co-simulation. In: Fitzgerald, J., Larsen, P.G., Verhoef, M. (eds.) Collaborative Design for Embedded Systems, pp. 199–222. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54118-6_10
9. Gilmore, S., Hillston, J.: The PEPA workbench: a tool to support a process algebra-based approach to performance modelling. In: Haring, G., Kotsis, G. (eds.) TOOLS 1994. LNCS, vol. 794, pp. 353–368. Springer, Heidelberg (1994). doi:10.1007/3-540-58021-2_20
10. Haverkort, B., Meeuwissen, A.: Sensitivity and uncertainty analysis of Markov-reward models. IEEE Trans. Reliab. **44**(1), 147–154 (1995)
11. Ho, Y.-C.: Introduction to special issue on dynamics of discrete event systems. Proc. IEEE **77**(1), 3–6 (1989)
12. Kleijnen, J.P.: Sensitivity analysis and related analyses: a review of some statistical techniques. J. Stat. Comput. Simul. **57**(1), 111–142 (1997)
13. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 80–95. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19751-2_6

14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
15. Sahner, R., Trivedi, K., Puliafito, A.: Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package. Kluwer, Boston (1996)
16. Saltelli, A., Ratto, M., Andre, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global Sensitivity Analysis. The Primer. Wiley, West Sussex (2008)
17. Schneier, B.: Attack trees. Dr. Dobb's J. **24**(12), 21–29 (1999)