Iterative Distribution-Aware Sampling for Probabilistic Symbolic Execution

Mateus Borges[†‡] Antonio Filieri[†] Marcelo d'Amorim[‡] Corina S. Păsăreanu[*]

[†] Univ. of Stuttgart Germany

[‡] Fed. Univ. of Pernambuco Brazil

[*] CMU SV/NASA Ames Research Center USA

ABSTRACT

Probabilistic symbolic execution aims at quantifying the probability of reaching program events of interest assuming that program inputs follow given probabilistic distributions. The technique collects constraints on the inputs that lead to the target events and analyzes them to quantify how likely it is for an input to satisfy the constraints. Current techniques either handle only linear constraints or only support continuous distributions using a "discretization" of the input domain, leading to imprecise and costly results.

We propose an iterative distribution-aware sampling approach to support probabilistic symbolic execution for arbitrarily complex mathematical constraints and continuous input distributions. We follow a compositional approach, where the symbolic constraints are decomposed into sub-problems whose solution can be solved independently. At each iteration the convergence rate of the computation is increased by automatically refocusing the analysis on estimating the sub-problems that mostly affect the accuracy of the results, as guided by three different ranking strategies.

Experiments on publicly available benchmarks show that the proposed technique improves on previous approaches in terms of scalability and accuracy of the results.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

Keywords

Symbolic Execution, Monte Carlo Sampling, Probabilistic Analysis

1. INTRODUCTION

Probabilistic symbolic execution is a promising technique for quantifying the probability of reaching program events of interest assuming that program inputs follow given probabilistic distributions [2, 9, 28]. The input distributions allow data from real world observations to be incorporated in the analysis of programs that interact with their environment, as well as to encode uncertainty in design assumptions about the usage profile of a program. The technique has many potential applications, e.g. it can be used in debugging, for ranking program errors, in analyzing the control software of autonomous vehicles that interact with uncertain environments, for computing software reliability, and for quantitative information flow analysis, to name a few.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2786832

The technique [2, 9, 11, 28] uses a symbolic execution of the program to collect symbolic constraints on the inputs that lead to the occurrence of target program events. The constraints are then analyzed to estimate their solution spaces and to quantify how likely it is for an input distributed according to a given probabilistic distribution to satisfy the constraints. However the current techniques are quite limited as they can either handle only linear constraints [11, 28] or, if they handle complex mathematical constraints [2], they only use uniform input distributions, while more complex distributions are handled using discretization techniques, which may be imprecise and costly in practice.

To achieve higher accuracy and scalability for the analysis of non-linear constraints under continuous input distributions we propose an iterative distribution-aware statistical analysis method. Our method is compositional and builds upon `qCORAL` [2] to decompose the analysis of program constraints into sub-problems that can be solved separately. We improve upon `qCORAL` by providing an iterative technique that re-focuses the sampling effort on the constraints that have higher impact on the accuracy of the probabilistic analysis results. Further, the method samples the inputs according to the distributions defined in the input profiles, avoiding the cost of discretizing continuous distributions, as was needed with `qCORAL`.

Statistical methods have proved to be effective in solving integration problems such as those arising from probabilistic analysis. Especially when the number of variables grows, statistical methods outperform symbolic and numerical ones [16]. Nonetheless, general statistical integration methods available off-the-shelf are not capable of exploiting all the information about a program behavior obtained through symbolic execution: e.g., certain constraints have higher impact in driving the program execution toward the occurrence of a target event. Similarly, certain constraints have higher impact on the convergence rate of statistical estimation, since they provide more information about the possibility for the target event to occur.

Based on this insight, we analyze the path conditions leading to the occurrence of the target event to *rank* the constraints according to their impact on the convergence of the statistical analysis. Exploiting this ranking, our method iteratively re-focuses the sampling to gather more information about the satisfaction of the most important constraints first, achieving higher estimation accuracy in a shorter time. We propose three ranking strategies: two of them are based on gradient descent optimization [21] while the third one uses a simple but efficient heuristic which gives more importance to the constraints whose probability estimates are farther from convergence.

We provide a formal assessment of our technique, study its convergence, and evaluate it experimentally on publicly available bench-

```
goal = new Uniform(-10, 10);
flapPosition = new Uniform(-5, 5);
windEffect = new Normal(0,0.5,-15,15); //weak wind
//windEffect = new Normal(0,7.25,-15,15); //strong wind
actuatorEffect = 5;
MAX_POSITION = 15;
MIN_POSITION = -15;

if(goal < 0){ // actuator
 flapPosition = flapPosition - actuatorEffect +
      windEffect;
} else{
 flapPosition = flapPosition + actuatorEffect +
      windEffect;
}
if (flapPosition > MAX_POSITION || // safety check
    flapPosition < MIN_POSITION){
 throw new OverrunException();
}
```

Figure 1: Flap controller.

marks from [2, 9, 28], including real world software taken from the medicine and the aero-space domains. Our experimental results show significant improvement over previous discretization-based approaches and built-in routines of general purpose mathematical tools, both in terms of accuracy of results and analysis time.

1.1 Example

To illustrate our analysis we introduce a small code snippet modeling a safety check for a simplified flap controller of an aircraft modified from [9] (see Figure 1). The controller is composed of a flap actuator and a safety check to avoid overrun of the flap. The variables influencing the behavior of the flap are the goal position, the current position of the flap, and the wind effect. The actuator performs a move towards the goal but the actuation can be hindered by the effect of the wind that can lead to an overrun of the flap.

The goal position can vary in the range $[-10,10]$, while the current position of the flap when the next control step is activated is assumed to be within $[-5,5]$. For both these variables, any value in the domain is considered equally likely, i.e. the concrete inputs are assumed as realizations of a Uniform distribution over each variable domain.

The wind effect is instead assumed to behave as a Normal distribution, with mean 0 and a standard deviation which depends on the strength of the wind. We will consider two different application scenarios: in case of weak wind, the standard deviation is assumed to be 2, meaning that the effect is most of the time quite close to 0; in case of strong wind, the standard deviation is assumed to be 7.25, so larger values of the the wind effect are more likely. In either case, the effect of the wind is bounded by the interval $[-15,15]$. This simplified model exemplifies how the *uncertainty* about this physical phenomena can be taken into account for the analysis.

In Figure 1, the random distributions for the input variables are characterized by the value of their parameters and the lower and upper bound of the domain (which are always the last two arguments). The Uniform distribution does not need additional parameters besides the domain. The Normal distribution is characterized by its mean and its standard deviation, respectively, besides the domain.

The probabilistic distribution of the wind effect can be obtained systematically from telemetry mission data. It is possible, for example, to measure the frequency over time of values occurring within certain ranges during a mission to obtain realistic usage profiles. Techniques for the automatic inference of probabilistic profiles are described elsewhere, see e.g. [12].

Analyzing the example program. Probabilistic symbolic execution computes the probability of a certain event to occur or not during the execution of the program, given a usage profile. The result of this kind of analysis is not a boolean (indicating presence or absence of an error), but a quantitative figure, whose value depends on both the program and its usage. In this example the probability of throwing an OverrunException is only 0.04% in the presence of weak wind, but it grows up to 8.43% when the wind is strong.

This paper describes a technique for the automatic estimation of the probability for a target event to occur given input profiles described by continuous probability distributions over floating-point input domains, enhanced by a set of strategies to speed up the convergence rate of the estimation.

2. BACKGROUND

2.1 Probabilistic Symbolic Execution

Probabilistic symbolic execution quantifies the probability of the software satisfying a given (safety) property [9, 11, 28]. This approach is comprised of two stages: symbolic execution [17] and solution space quantification. Tools to support the first stage are presented in e.g. [9, 11, 28]; this paper focuses on the second stage: solution space quantification. We provide here a quantification procedure that can be easily incorporated in the above tools.

Symbolic execution is a program analysis technique that executes programs on symbolic, rather than concrete inputs, and computes the values of program variables as symbolic expressions in terms of the inputs. Symbolic execution abstracts all possible program executions into a symbolic execution tree, where the nodes are the symbolic program states and the edges are the program instructions. Each symbolic execution path is uniquely identified by a *path condition* (PC), i.e. a constraint that the program inputs have to satisfy for driving the execution along the corresponding path. To avoid the problem of non-terminating executions, the approach bounds the symbolic execution tree. The paths hitting the execution bound are specifically marked and provide an index of the dependability of the analysis [9]. Let PC^T be the set of PCs identifying executions satisfying the target property; similarly PC^F denotes the set of PCs for the paths that lead to property violation.

For example, for the code in Figure 1, the set of path conditions PC^F leading to the occurrence of an OverrunException are:

```
goal<0 && flapPosition-actuatorEffect+windEffect > 15
goal<0 && flapPosition-actuatorEffect+windEffect < -15
goal>=0 && flapPosition+actuatorEffect+windEffect > 15
goal>=0 && flapPosition+actuatorEffect+windEffect < -15
```

The goal of the second stage – solution space quantification – is to compute the expected probability that the program inputs satisfy any of the PCs in PC^T, given a probabilistic distribution over the input domain. Formally:

$$\int_D \mathbb{1}_{PC^T}(\mathbf{x}) \cdot p(\mathbf{x}) \tag{1}$$

where D is the input domain defined as the Cartesian product of the domains of the input variables, $p(\mathbf{x})$ is the probability of an input $\mathbf{x}$ to occur according to the probability distribution over the inputs, and $\mathbb{1}_{PC^T}(\mathbf{x})$ is the indicator function on PC^T, that returns 1 when $\mathbf{x}$ satisfies any of the PCs in PC^T, and 0 otherwise [23]. The probability of failure can be computed similarly.

2.2 Monte Carlo Estimation

Exact solutions to Equation (1) are not always possible because the integral might be ill-conditioned by the presence of complex non-linear constraints or because of scalability issues to deal with high dimension problems [16]. Monte Carlo methods provide a general, approximate solution to the integration problem. The basic solution, called "hit-or-miss" [27] (HM-MC), consists in generating

n random inputs over the input domain (drawn according to the specified distributions) and to count the number of hits, i.e. the inputs satisfying any of the constraints in PC^T, and, conversely, the number of misses. The ratio $\hat{x}$ between the number of hits and n is an efficient, unbiased, and consistent estimator of the probability of satisfying the constraints [23]. The mean and the variance of $\hat{x}$ are:

$$E[\hat{x}] = \bar{x} \qquad Var[\hat{x}] = \frac{\bar{x}\cdot(1-\bar{x})}{n} \tag{2}$$

where $\bar{x} = \sum_n \mathbb{1}_{PC^T}(\mathbf{x}_i)/n$ is the sample mean. The *accuracy* of the estimate $\bar{x}$ can be assessed through the variance of $\hat{x}$: the closer the variance is to 0 the more accurate is the estimation. As evident from Equation (2), variance decreases with the number of samples, going asymptotically to 0 when $n \to \infty$.

Stratified Sampling and Interval Constraint Propagation (ICP). The slow convergence rate of Monte Carlo hit-or-miss methods [16] can be improved by using *stratified sampling*, which partitions the input domain into regions (*strata*) which can be analyzed separately, using for instance hit-or-miss Monte Carlo. The local result $\hat{x_i}$ associated to each region R_i is combined to obtain an estimator $\hat{x}$ over the global input domain with the following properties:

$$E[\hat{x}] = \sum_i w_i \cdot E[\hat{x}_i] \qquad Var[\hat{x}] = \sum_i w_i^2 \cdot Var[\hat{x}_i] \tag{3}$$

where w_i is defined as $w_i = \text{size}(R_i)/\text{size}(D)$, i.e. the ratio between the size of the strata R_i and the entire domain D. The use of stratified sampling never increases the variance of the global estimator and, in the worst case, it is equivalent to non-stratified sampling.

In previous work [2] we proposed the use of an interval constraint propagation solver, RealPaver [14] to prune out regions of the solution space that do not contain any solution of PC^T; consequently improving convergence even further. RealPaver produces as output a set of boxes whose union reliably contains all the solutions of a given (complex, non-linear) constraint. Stratified sampling is then used for composing the results of analyzing the boxes.

2.3 Compositional Quantification

In probabilistic symbolic execution the number of constraints to analyze can be very large, leading to potentially high analysis cost. To address this issue, we follow the compositional approach from [2] that splits the analysis into smaller sub-problems based on the structure of the constraints composing PC^T (or PC^F), allowing also the reuse of partial results.

PC^T contains all the PCs leading to property satisfaction. The goal is thus to quantify the probability of satisfying the disjunction $\bigvee_{pc \in PC^T} pc$. Note that all PCs are disjoint by construction.

2.3.1 Handling Disjunction

Assume that we estimated the probability of satisfying two path conditions, $pc_i, pc_j \in PC^T$, through the estimators $\hat{x}_i$ and $\hat{x}_j$, respectively, and for these estimators we know mean and variance. Let us denote by $\hat{x}$ the estimator of the disjunction $pc_i \vee pc_j$ that we want to compute. We can use the following composition rule (for disjunction) [2] to compute the mean and an upper bound for the variance of $\hat{x}$.

$$E[\hat{x}] = E[\hat{x}_i] + E[\hat{x}_j] \qquad Var[\hat{x}] \leq Var[\hat{x}_i] + Var[\hat{x}_j] \tag{4}$$

2.3.2 Handling Conjunction

Each path condition pc_i is a conjunction of simpler constraints $c_{i0} \wedge c_{i1} \wedge \ldots c_{im}$. This set of (conjoined) constraints can be partitioned in *independent* subsets; each subset including the constraints which are related through variable dependency. Intuitively, two variables i and j depend on one another if there exists a constraint predicating on both of them (e.g., $v_i > v_j + 1$) or if they both depend on a third variable. This dependency relation is symmetric and transitive by construction and is extended with reflexivity (i.e., v_i always depends on itself), becoming an equivalence relation which induces a partition $\{V_0, V_1, \ldots, V_n\}$ on the set of variables V.

For each constraint c_{ik} of pc_i, $A(c_{ik}, v)$ indicates that c_{ik} predicates on variable v. Let C_{ij} denote the set of constraints occurring in pc_i and predicating on any of the variables in V_j (i.e. $C_{ij} = \{c_{ik} \mid A(c_{ik}, v) \wedge v \in V_j\}$). Let $\hat{x}_{ij}$ and $\hat{x}_{ik}$ be the estimators of C_{ij} and C_{ik}, respectively. Then, we can use the following composition rule (for conjunction) [2] to compute the estimator $\hat{x}$ of $C_{ij} \wedge C_{ik}$:

$$\begin{aligned} E[\hat{x}] &= E[\hat{x}_{ij}] \cdot E[\hat{x}_{ik}] \\ Var[\hat{x}] &= E[\hat{x}_{ij}]^2 \cdot Var[\hat{x}_{ik}] + E[\hat{x}_{ik}]^2 \cdot Var[\hat{x}_{ij}] \\ &\quad + Var[\hat{x}_{ij}] \cdot Var[\hat{x}_{ik}] \end{aligned} \tag{5}$$

The rule extends naturally to many constraints. Note that some constraints may occur in multiple PCs; we can thus cache and re-use the analysis results, for efficiency.

For example, analyzing the PC extracted in Section 2.1 for the example from Figure 1, the following six independent constraints can be identified (notice that being `actuatorEffect` a constant, symbolic execution simplifies it):

```
c_0: goal < 0  c_1: goal >= 0
c_2: flapPosition + windEffect > 20
c_3: flapPosition + windEffect < -10
c_4: flapPosition + windEffect > 10
c_5: flapPosition + windEffect < -20
```

Each of these constraints can be analyzed independently using the Monte Carlo techniques described in Section 2.2 and their estimators can be combined according to the structure of the the PCs using the composition rules of Equations (4) and (5).

3. DISTRIBUTION-AWARE SAMPLING

In this section, we extend the compositional sampling approach from [2] with an efficient way to handle inputs that follow continuous distributions. Specifically we enhance the Monte Carlo estimation with a *distribution-aware* sampling strategy, where the random samples can be generated according to a continuous distribution instead of a uniform one. However, the use of interval constraint propagation (i.e. RealPaver) and stratified sampling requires some care, since we need to restrict the sampling to specific sub-regions of the input domains.

We assume that each input variable i is defined over a bounded continuous interval $[a,b]$ and its values are distributed according to a known distribution $Distribution_i(\theta_i)$, restricted, or *truncated* [7], to this interval. θ_i is a (possibly empty) constant vector of known parameters characterizing the distribution, e.g., a Normal distribution is characterized by the values of its mean and its variance, while an Exponential distribution only by its mean. We will use the notation $Distribution_i(\theta_i)|_{a,b}$ to represent the distribution truncated to the interval $[a,b]$. The mapping between input variables and their corresponding truncated probability distributions constitutes the usage profile (see the variable declaration in Figure 1).

As mentioned in Section 2, the output of ICP is a set of boxes containing all the inputs satisfying PC^T. Each box is defined by the conjunction of constraints of the form $v_i \in [a_i, b_i]$, where each variable is restricted to a particular interval within its domain. Since in general the range of $Distribution_i(\theta_i)$ may fall outside $[a_i, b_i]$, we need to restrict the sampling to $[a_i, b_i]$. A simple approach would be to generate for each variable v_i a set of samples $x_i^1, x_i^2, \ldots x_i^n$

(drawn according to the distribution) and then prune out all the $x_i^j \notin [a_i, b_i]$. However this may be inefficient, especially if the intersection between the range of $Distribution_i(\theta_i)$ and $[a_i, b_i]$ is small.

We propose an efficient solution obtained by exploiting the results of probability theory for *truncated distributions* [7]. Consider a random variable v_i with $Distribution_i(\theta_i)$. Each distribution is associated to a unique, known, *cumulative distribution function* $CDF_i(t)$ defined as $CDF_i(t) = Pr(v_i <= t)$, which we will use for the sampling. Consider also a non-empty interval $[a_i, b_i]$. Let the random variable rv_i be the restriction of v_i to the interval $[a_i, b_i]$, then the following result holds [7]:

$$CDF_{rv_i}(t) = \frac{CDF_i(max(min(t, b_i), a_i) - CDF_i(a_i)}{CDF_i(b_i) - CDF_i(a_i)} \quad (6)$$

Furthermore, the cumulative distribution has inverse:

$$CDF_{rv_i}^{-1}(u) = CDF_i^{-1}(CDF_i(a_i) + u \cdot (CDF_i(b_i) - CDF_i(a_i)) \quad (7)$$

where $0 \leq u \leq 1$. For the most common continuous distributions both $CDF(\cdot)$ and its inverse $CDF^{-1}(\cdot)$ can be computed efficiently using off-the-shelf tools or libraries (e.g., [29]).

In the following we show how to use these functions to obtain samples of any truncated distributions from samples drawn from uniform distributions which can be easily obtained from many existing off-the-shelf libraries.

Example. Recall from Figure 1 that variable `goal` follows an Uniform distribution in the interval $[-10, 10]$. According to the definition above $CDF_{\texttt{goal}}(t) = (t+10)/(20)$ for $-10 \leq t \leq 10$, 0 for $t < -10$, and 1 for $t > 10$. The reader can refer to [23] for the definition of *CDF* for the most popular continuous distributions.

The ability to compute $CDF_{rv_i}(\cdot)$ and its inverse $CDF_{rv_i}^{-1}(\cdot)$ allows us to implement a general sampling strategy for continuous distributions restricted to intervals of interest. Indeed, to take a sample from rv_i, i.e. variable x_i restricted to $[a_i, b_i]$, it is sufficient to generate a sample $\bar{u}$ from a Uniform distribution over $[0, 1]$ (by using any robust pseudo-random generator) and use this sample $\bar{u}$ to generate the sample $r\bar{v}_i = CDF_{rv_i}^{-1}(\bar{u})$ from the restricted random variable rv_i. This approach allows to bring distribution-awareness to ICP-enabled stratified sampling, thus allowing to achieve both the precision and scalability of distribution-aware sampling and the improved convergence rate due to stratified sampling.

3.1 Distribution-Aware Sampling versus Discretization

The analysis from [2, 9, 19] assume that the probability distributions over the input domain are specified by a *usage profile* (UP) defined as:

$$UP = \begin{cases} c_1 & : p_1 \\ c_2 & : p_2 \\ \dots & \dots \end{cases} \quad (8)$$

where the c_i are a partition of the input variables, i.e. $\cup_i c_i = D$ and $c_i \cap c_j \neq \emptyset \implies i = j$, and $\sum_i p_i = 1$ (we abuse the notation here by referring with c_i to both a set of inputs and the constraints uniquely characterizing such set). Each pair $c_i : p_i$ is called *usage scenario*.

This formalism for UPs allows to arbitrarily partition the input domain into a finite set of regions, each with an assigned probability. The constraints c_i can be arbitrarily complex, making the formalism expressive enough to predicate about non trivial relations among input variables. However, if the values of an input are distributed according to a continuous probability distribution, casting this case into a finite UP requires a *discretization* procedure, partitioning the domain of each variable into a finite number of intervals and assigning to each interval a probability computed from the original continuous distribution. Depending on the number and the size of the intervals, discretization may be an arbitrarily precise approximation of the continuous distribution.

Nonetheless, the unavoidable loss of precision due to discretization may introduce a bias in the analysis results, when the approximation is not fine enough. On the other hand, a finer discretization requires to partition variables domain into a larger number of intervals. Assuming v input variables are partitioned into m intervals each one, the total number of constraints to obtain a discretized version of the original UP would be v^m. Though the complexity of the analysis is linear in the number of usage scenarios, the latter grows exponentially with the required precision of discretization, limiting the scalability of the analysis.

Section 5.2 compares the accuracy and performance of our distribution-aware sampling procedure with the same analysis based on the discretization of the usage profiles showing its advantages.

4. ITERATIVE OPTIMAL SAMPLING

The divide-and-conquer procedure reported in Section 2.3 solves the problem of quantifying the solution space of PC^T in terms of simpler independent sub-problems. It further caches and reuses the results for the sub-problems to speed up the quantification. The estimates for the sub-problems are composed according to the disjunction and conjunction rules from Equations 4 and 5, respectively. The variance of such estimators is composed as well, providing an index of the accuracy of the final results. Although the asymptotic convergence of the compositional estimators is guaranteed [2], i.e., when the number of samples used to obtain the local estimators for each sub-problem grows to ∞, the convergence *rate* of the procedure is hard to quantify and may be slow in practice.

This section introduces an iterative sampling approach to speed-up the convergence rate of the quantification procedure. At each iteration, the sampling is *focused* on the parts of the input space that are likely to have the largest influence on the *variance* of the composed estimator obtained from the previous iteration, with the goal of minimizing the variance and thus increasing the overall accuracy of the estimation. We explore three iterative approaches.

The first approach is based on *gradient descent optimization* [21] (Section 4.1) which provides a natural solution to the sampling allocation problem. It uses the composition rules from Equations (2) – (5) to compute the gradient of the global variance with respect to the number of samples allocated to each local estimator. The impact of each local estimator, as quantified by the gradient, is then used to decide how many new samples to allocate for each sub-problem, aiming at minimizing the global variance.

The second approach overcomes the computational overhead related to bootstrapping new sampling procedures for multiple sub-problems. It uses a relaxed form of gradient descent optimization based on a *sensitivity analysis* (Section 4.2), where, at each iteration, new samples are allocated only for the single most influent sub-problem, as identified by the gradient.

The third approach introduces a simple heuristic sampling allocation that, at each iteration, allocates new samples for the sub-problem whose estimator has the largest variance (Section 4.3). This heuristic is computationally cheaper than the other optimization methods because it does not require the computation of the gradient of the global variance with respect to the number of samples allocated for each sub-problem. However, our experiments show that this simple heuristic works well in practice.

The three different strategies discussed in the next sections will be evaluated on several case studies in Section 5.

4.1 Gradient-Descent Variance Minimization

Gradient descent is a common optimization method for finding the *minimum* of functions for which derivatives can be defined [21]. The gradient descent method starts with an initial (random) candidate solution and iteratively discovers new candidates closer to the optimum. At each step, a new candidate solution is produced following the direction of the negative gradient of the function. As a local search method, it can get stuck in local optima. However, in our context there is a unique minimum, as the variance is guaranteed to decrease with each new sample.

The dependency of the global variance on the number of samples allocated for each sub-problem can be computed in analytical form combining Equations (2) to (5). For the sake of simplicity we will for now ignore ICP-based stratified sampling, thus assuming each sub-problem to be quantified by simple hit-or-miss Monte Carlo (Equation 2). We will bring ICP-based stratified sampling later.

As an example of this computation, consider from Section 2.1 the PC `goal < 0 && flapPosition + actuatorEffect + windEffect > 15`. The quantification problem for this PC can be reduced to the quantification of the independent constraints c_0 : `goal < 0` and c_4 : `flapPosition + windEffect > 10` (where the constant `actuatorEffect` has been already evaluated), whose solution space is quantified by the estimators $\hat{x}_0$ and $\hat{x}_1$, respectively. If n_0 samples are allocated for the estimation of $\hat{x}_0$ and n_1 for the estimation of $\hat{x}_1$, their respective variances would be (Equation 2):

$$Var[\hat{x}_0] = \frac{\bar{x}_0 \cdot (1-\bar{x}_0)}{n_0} \qquad Var[\hat{x}_1] = \frac{\bar{x}_1 \cdot (1-\bar{x}_1)}{n_1} \tag{9}$$

The variance of the estimator for the PC as a function of n_0 and n_1 can then be computed applying the conjunction composition rule (Equation 5) as follows.

$$\begin{aligned} Var(n_0,n_1) \quad = \quad & \bar{x}_0^2 \cdot \frac{\bar{x}_1 \cdot (1-\bar{x}_1)}{n_1} + \bar{x}_1^2 \cdot \frac{\bar{x}_0 \cdot (1-\bar{x}_0)}{n_0} \\ & + \frac{\bar{x}_0 \cdot (1-\bar{x}_0)}{n_0} \cdot \frac{\bar{x}_1 \cdot (1-\bar{x}_1)}{n_1} \end{aligned} \tag{10}$$

An analogous procedure can be applied to deal with disjunctive forms. Our goal is now to minimize the function $Var(n_0,n_1,\ldots,n_m)$, which denotes the variance of the global estimate as a function of the number of samples allocated for the estimation of each of the $m+1$ independent constraints the quantification problem has been split in. We want to find a sequence of sample allocations that brings global variance near 0 quickly. Note that 0 is the unique global minimum for the variance and it is reachable when the number of samples grows to infinity.

The initial solution $\mathbf{n}^0 = [n_0^0, n_1^0, \ldots, n_m^0]$ can be computed in a bootstrap stage where an arbitrary number of samples is allocated uniformly to each estimation sub-problem. This initial round of sampling provides also an initial estimate of the expected value and the variance of each independent constraint, which will be used to estimate the value of $Var(\mathbf{n}^0)$ at the initial point.

Given the value of $\mathbf{n}^k$ at step k, the value of $\mathbf{n}^{k+1}$ is computed according to the following formula:

$$\mathbf{n}^{k+1} = \mathbf{n}^k - \gamma \cdot \nabla Var(\mathbf{n}^k) \tag{11}$$

where γ is the step size (we will get back to this later) and $\nabla Var(\mathbf{n}) = [\partial Var/\partial n_0, \partial Var/\partial n_1, \ldots, \partial Var/\partial n_m]$ is the gradient, i.e., the vector of the partial derivatives of $Var(\cdot)$ with respect to the arguments n_i. Intuitively, the gradient indicates at each step "how much" each of the independent constraints can contribute to minimize $Var(\cdot)$.

Recall that the quantification problem consists in estimating the probability that an input satisfies any of the path conditions in PC^T, given a probability distribution on the input space. In other words, we aim to estimate the probability of satisfying the disjunction of the PCs in PC^T, and each PC is the conjunction of independent constraints. Exploiting the compositional rules of Section 2.3 the gradient of the global variance with respect to the number of samples to be allocated to each independent sub-problem can be computed compositionally too.

THEOREM 1. *Derivative of disjunction compositions. The derivative of the variance of the estimator $\hat{x}_c$ for the disjunction $c = c_0 \vee c_1 \vee \ldots c_k$ with respect to the number of samples n_i allocated to the estimators $\hat{x}_i$ for the constraints c_i ($i \in [0,k]$) can be computed as:*

$$\frac{\partial Var[\hat{x}_c]}{\partial n_i} \leq \sum_{i \in [0,k]} \frac{\partial Var[\hat{x}_i]}{\partial n_i} \tag{12}$$

PROOF. *The composition rule in Equation* (4) *overestimates the variance of a disjunction as the sum of the variances of the disjuncts. The proof follows from the linearity of the derivative operator.* □

THEOREM 2. *Derivative of conjunction compositions. The derivative of the variance of the estimator $\hat{x}_c$ for the conjunction $c = c_0 \wedge c_1 \wedge \ldots c_k$ with respect to the number of samples n_i allocated to the estimators $\hat{x}_i$ for the constraints c_i ($i \in [0,k]$) can be computed, where the constraints c_i are independent, according to the definition introduced in Section 2.3, and the estimators $\hat{x}_i$ use HM-MC, as:*

$$\frac{\partial Var[\hat{x}_c]}{\partial n_i} = -\frac{Var[\hat{x}_i]}{n_i} \prod_{j \in [0,k], j \neq i} \left(E[\hat{x}_j]^2 + Var[\hat{x}_j] \right) \tag{13}$$

PROOF. *The proof is omitted for space reasons, the interested reader can refer to [3] for it.* □

The gradient descent method terminates when either the target variance is achieved or when the gradient gets close enough to $\mathbf{0}$. The latter indicates the optimum has been reached, usually within a finite accuracy. Notably, the convergence speed of gradient descent methods is proportional to the value of the gradient. This implies that the closer the gradient gets to $\mathbf{0}$ the slower the convergence is [21]. Nonetheless, for the problem at hand the improvement on the convergence rate of the global estimator is still significant, as will be shown in Section 5.

Notice that the expected values of the estimates obtained by the composition rules of Section 2 are not affected by the gradient descent procedure. The estimators keep their unbiasedness and their consistency, while only the rate of convergence of the variance to 0 is possibly optimized.

Back to our example, assume that, during the initial bootstrap phase, after allocating 1000 samples for estimating each of the constraints c_0 : `goal<0` and c_4 : `flapPosition + windEffect > 10`, we obtained the estimates $\bar{x}_0 = 0.489$ and $\bar{x}_1 = 0.088$, considering the weak wind profile. After the first iteration we obtain the following gradient $\nabla Var(n_0,n_1) = [-1.955 \cdot 10^{-9}, -1.921 \cdot 10^{-8}]$. Thus, the sampling budget available for the next iteration should be assigned to $\hat{x}_0$ and $\hat{x}_1$ proportionally to their corresponding derivatives, roughly 9.2% of the samples for c_0 and 90.8% for c_1.

Choosing the step size. The partial derivatives composing the gradient are used to decide how the *sampling budget for the next iteration* will be distributed among the independent constraints. This budget is quantified by the step size γ. In general, if γ is too small, then the algorithm will converge very slowly. On the other hand, if γ is not chosen small enough, then the algorithm may

use sampling time inefficiently. In our case, we have to take into account the overhead of starting a new sampling procedure for each independent constraint at each iteration. For a too small budget, the bootstrapping time for the sampling procedures might overcome the time for sampling, increasing the computational overhead. An optimal value for γ depends in general on the specific problem at hand. Furthermore, instead of fixing the value of γ, we fix the total number of samples to be allocated for each iteration and compute γ so to use it all, i.e. $\gamma \cdot \sum \partial Var/\partial n_i$ has to be equal to the total number of samples allowed for each iteration. Since only an integer number of samples can be allocated for each sub-problem, possible decimal results are rounded up to the smallest larger integer.

For this work we empirically evaluated different values for the total number of samples to be allocated during each iteration, and in turn γ (Section 5). Adaptive decisions for γ have been proposed too (e.g., [21, 24]), and we plan to evaluate them in the future.

ICP-based stratified sampling. ICP-based stratified sampling (see Section 2.2) can be used to reduce the variance of the estimates for the single independent sub-problems. ICP is used to partition the sampling space into a set of disjoint boxes containing all the solutions for the constraint, pruning out the domain regions containing no solutions. With the same number of samples, this approach cannot perform worse than HM-MC, and usually performs better [2]. Therefore the variance obtainable by HM-MC can be seen as an upper bound of the one of stratified sampling for the same problem.

The gradient descent procedure we defined is based on HM-MC and decides how many samples to allocate on each sub-problem during each iteration. Enhancing the local estimators with stratified sampling can only produce better results (which will be also reflected by the more accurate values for the local estimates used to evaluate $V(\cdot)$ on the next iteration) but requires an additional decision about how to distribute the samples allocated on an independent constraint among the boxes containing its solutions.

Our decision strategy is to distribute 2/3 of the samples proportionally to the product between the variance of the local estimate within the box and the size of the box, and 1/3 uniformly among all the boxes. Recalling Equation (3), both the variance and the size of the box directly affect the variance of the stratified sampling estimates. With this heuristic we take into account this dependency. The remaining third of samples is distributed uniformly to speedup the convergence of the estimators for all the boxes, and thus a better assessment of their variance for the next iterations. This is especially relevant when the probability of satisfying the target property within the box is close to 0; if no samples satisfy the constraint restricted within the box, the variance is incorrectly estimated as 0 and the box would receive 0 samples on the next iteration, preventing the local estimator to assess the actual variance.

4.2 Sensitivity Analysis and Computational Overhead

Each sampling round requires to start a sampling procedure for each box of each independent constraint for which we allocate new samples. This operation introduces a significant overhead, especially when the number of sub-problems is large and the partial derivatives in the gradient are mostly in the same order of magnitude. This implies that the budget will be distributed almost uniformly, requiring to add only a few samples for each independent sub-problem at each iteration.

A sub-optimal strategy to trade convergence rate for lower computational overhead consists in a relaxation of the gradient descent method based on sensitivity analysis. The sensitivity of the global variance $Var(\cdot)$ with respect to the number of samples n_i allocated to the constraint c_i is defined by the partial derivative $\partial Var/\partial n_i$. The single constraint mostly affecting the global variance is the one having the larger sensitivity, in absolute value (note that all of the derivatives are negative since adding more samples can only decrease the global variance). The strategy thus consists in allocating the entire sampling budget for the next iteration to the single most important constraint.

Recalling our example, if $\nabla Var(n_0, n_1) = [-1.955 \cdot 10^{-9}, -1.921 \cdot 10^{-8}]$, the whole sampling budget for the next iteration will increase n_1 because the corresponding sub-problem has the highest expected impact on the global variance.

From a mathematical viewpoint, this means that instead of following the gradient, we follow its projection on the single dimension providing the best improvement. This is in general less effective then using all the information in the gradient, but may produce valuable results at a lower computational cost (as shown in Section 5).

Table 1: Characterization of the benchmarks.

Subject	Asrt.	*#Paths*	*#Ands*	*#Ar. Ops.*	*0-var./Parts.*
`VolComp` subjects [28]					
ARTRIAL	1	442	1,484	0 (0)	23/23
	2	2,439	1,740	443 (3)	4/27
	3	2,260	68,630	19,125 (3)	10/44
CART	4	44	1,209	638 (3)	1/45
	5	47	1,296	681 (3)	1/48
CORONARY	6	320	195	62 (3)	8/15
	7	274	31	8 (3)	5/8
EGFR EPI	8	45	547	31 (3)	46/46
	9	44	422	33 (2)	38/38
EGFR EFI	10	13	163	12 (3)	13/18
SIMPLE	11	14	101	9 (3)	14/14
INVPEND	12	1	54	229 (3)	0/1
PACK	13	1,103	16,414	0 (0)	40/40
	14	906	12,080	0 (0)	37/37
	15	924	12,293	0 (0)	38/38
	16	821	840	0 (0)	38/38
	17	954	14,850	6,948 (2)	815/954
	18	1,030	16,186	7,578 (2)	935/1030
	19	1,132	17,972	8,420 (2)	1132/1132
q`CORAL` subjects [2]					
APOLLO	21	866	10037	249655 (4)	216/253
CONFLICT	22	14	70	693 (4)	8/14
TURN LOGIC	23	73	505	1718 (2)	65/73

4.3 Local Heuristic for Sampling Allocation

We also considered a low-overhead sampling heuristic that does not require to compute the gradient of $Var(\cdot)$. This heuristic prescribes to allocate at each iteration all the samples to the single estimator having the highest variance.

Back again to our example, where after allocating 1000 samples for each of the sub-problems c_0 : `goal<0` and c_4 : `flapPosition + windEffect > 10`, we obtained the estimates $\bar{x}_0 = 0.489$ and $\bar{x}_1 = 0.088$ with variance $2.498 \cdot 10^{-4}$ and $8.025 \cdot 10^{-5}$, respectively, this heuristics would require to allocate the entire sampling budget for the next iteration to increase n_0 since the estimator $\hat{x}_0$ is the one showing the highest variance.

Due to the nature of the problem, this heuristic intuitively guarantees the convergence of the estimator, since the allocation of more samples to an estimator with positive variance always strictly decreases its variance (Equation 2). This guarantees that all the estimators with non-zero variance will eventually receive additional samples, following the convergence of the global estimator as well.

5. EVALUATION

We described distribution-aware sampling and iterative sampling allocation for improving the convergence rate of a compositional simulation-based probabilistic symbolic execution. We implemented

these techniques in the Java based tool qCORAL and evaluated their effectiveness in Section 5.2 and 5.3, respectively.

5.1 Experimental Settings

For our evaluation, we used the publicly available benchmarks of VolComp [30] and qCORAL [26]. The subjects from the VolComp benchmark only contain linear constraints (that we translated in the input format for our tool). These subjects are: a heart fibrillation risk calculator (ARTRIAL), a steering controller to deal with wind disturbances (CART), a coronary disease risk calculator (CORONARY), an estimator of chronic kidney's disease (EGFR−EPI and EGFR−EPI−SIMPLE), an inverted pendulum (INVPEND), and a model of a robot deciding how to pack goods of different weights into envelopes with limited capacity (PACK). The subjects from the qCORAL benchmark contain non-linear constraints and more complex mathematical functions (e.g., sinus); they are a model of the Apollo lunar vehicle autopilot (APOLLO) and two core modules of an aircraft collision-detection monitor (CONFLICT and TURN LOGIC). These subjects were implemented in Java and analyzed using Symbolic PathFinder [25] to compute the PCs (of paths leading to assert violations).

Table 1 reports a characterization of these subjects. Column "Asrt" denotes a unique id we used to identify the checked assertions during the discussion. Column "*#Paths*" indicates the number of symbolic paths leading to the violation of the assertion. Columns "*#Ands*" and "*#Ar. Ops*" denote respectively the number of conjuncts and the number of arithmetic operations. For column "*#Ar. Ops*", in parenthesis the number of different operators (e.g., + or *sin*()). A value of 0 in this column means that all the constraints involve only comparisons (e.g., $x < y$ or $x < 5$). Finally, "*Parts.*" indicates the number of independent sub-problems identified for compositional analysis and "*0-var*" indicates the number of sub-problems with exact solutions after applying ICP. Column "*0-var./Parts.*" indicates how many of the sub-problems we obtained variance 0 on an execution of qCORAL with 100k samples without iterative sampling allocation. A sub-problem can obtain 0 variance after sampling when either ICP returned an exact solution or if all the samples returned the same truth value (cf. Equation 9).

Baseline. To compare the accuracy of the different approaches, we solved the quantification problem with the commercial tool Mathematica (version 10). We used the off-the-shelf function NProbability with default arguments. This procedure is designed to provide solutions to a broad range of problems. In contrast our method is tailored to probabilistic symbolic execution. This results (in some cases) in both a slower performance and exceptions, reported by Mathematica, where NProbability fails to achieve precise results. Different configurations of NProbability may avoid these exceptions, however they would require human expertise beyond the off-the-shelf use of the tool.

Execution environment. We run qCORAL on an Intel Core i7 920 (2.67GHz, 8M cache) machine, with 8GB of RAM. Considering the higher computational demand for Numerical Integration with Mathematica, we performed this operation on an r3.large Amazon EC2 machine, running on an Intel Xeon E5-2670 v2 (2.50 GHz, 25M cache) with 16GB RAM. Both machines run Ubuntu 64 bits.

5.2 Distribution-Aware Sampling

Distribution-aware sampling aims at providing direct support for input variables characterized by continuous probability distributions. This section reports on two different experiments we conducted to evaluate our proposed technique. The first experiment compares the results of qCORAL and NProbability with respect to precision and efficiency. The second experiment compares the precision, lack of bias, and scalability of distribution-aware sampling against the analysis with discretized usage profiles.

5.2.1 Comparison with NProbability

Table 3 reports the results of analyzing each subject with qCORAL and NProbability. qCORAL performs a single sampling round with 100k samples, thus *no iterative sampling allocation* is used for this experiment. The cells shadowed in grey highlight the cases for which Mathematica reported an exception and the results might thus not meet the default prescribed accuracy of at least five decimal digits. A "-" is used to mark the cases where the analysis failed to return results within 40 hours of processing.

To evaluate distribution-aware sampling, we experimented with different continuous input distributions. In the first set of experiments we assigned to each variable a truncated Normal distribution centered in the middle of the variable domain and with standard deviation equal to 1/6 of the domain length, and truncated by the bounds of the domain. We set the standard deviation to 1/6 so that already the non-truncated distribution has probability 99% to generate a sample within the domain. Truncation introduces a small correction to guarantee that all samples fall within the bounds. In the second set of experiments we assigned to each variable a truncated Exponential distribution. The rate parameter of the distribution has been tuned again to make 99% of the samples fall within the original variable domain. Since Exponential distributions are defined over positive domains, we excluded the subjects whose values have negative domains.

For all the cases where NProbability terminated without exceptions the results of qCORAL are consistent, at the reported accuracy (σ). In several cases the results of qCORAL are exact $\sigma = 0$ (up to Java double accuracy) and match those of NProbability. The execution time of qCORAL is significantly shorter than the one required by NProbability for the same subjects, especially on the more complex ones where the difference is by several orders of magnitude. qCORAL also produces more robust results compared to NProbability, completing the analysis for all the subjects without exceptions. This is due to the intrinsic robustness of simulation-based approaches.

A final note concerns the result of qCORAL for assertion 5 with Normal usage profile. This result is indeed larger than 1, which is clearly not a valid probability. This is due to the accumulation of inaccuracies of the sub-problems estimates, when the result is close to 1. However, the corresponding σ makes the result compatible with NProbability. Cutting it to 1 would alter the confidence intervals computable with the estimate and σ, thus we report the estimate as is, including accumulated errors.

5.2.2 Comparison with Discretization

This section evaluates the tradeoff between accuracy and scalability of distribution-aware sampling with the same analyses performed on discretized usage profiles. For the experiments reported in Table 2 we assigned for each subject a truncated Normal distribution to two of the input variables, following the same parameterization procedure described for the previous experiments. All the other variables have the original Uniform distribution defined in [26, 30]. Only two variables have been assigned a non-uniform profile for scalability reasons, since the size of discretized usage profiles grows exponentially in the number of non-uniform variables (cf. Section 3). This results in a complexity already sufficient for comparing the approaches.

There are several possibilities for discretizing a continuous distribution. A simple solution requiring no prior knowledge on the problem consists in dividing the domain into a certain number of

Table 2: Comparison of different discretization methods. `Discretization` invokes `qCORAL` [2] once for every region within each constraint partition. For every subject, two input variables are normally distributed; the others are uniformly distributed.

Subject	Asrt.	NProbability		Discretization				qCORAL [100k]	
				3 intervals		6 intervals			
		solution	*time(s)*	*avg. est.*	*time(s)*	*avg. est.*	*time(s)*	*est.*	*time(s)*
ARTRIAL	1	0.052928	15.83	0.067016	0.66	0.059714	0.73	0.052928	0.64
	2	0.000284	15.05	0.000320	0.74	0.000279	0.83	0.000285	0.71
	3	0.927292	648.34	0.923024	2.85	0.917593	2.87	0.924958	2.31
CART	4	0.974609	11.10	0.973542	1.63	0.964577	2.29	0.991601	1.12
	5	0.982561	11.87	0.984726	1.57	0.982017	2.28	0.992781	1.15
CORONARY	6	0.000201	1.81	0.000310	0.55	0.000241	0.61	0.000202	0.53
	7	0.000033	0.35	0.000033	0.43	0.000059	0.45	0.000033	0.42
EGFR EPI	8	0.130741	5.73	0.128302	0.65	0.128934	0.71	0.130741	0.68
	9	0.099552	4.30	0.100637	0.63	0.099036	0.65	0.099553	0.62
EGFR EFI (SIMPLE)	10	0.597568	1.74	0.596731	0.56	0.595728	0.70	0.597526	0.48
	11	0.159820	1.05	0.198692	0.50	0.178238	0.56	0.159819	0.43
INVPEND	12	0.051223	24.59	0.051119	13.54	0.051276	50.24	0.051096	2.14
PACK	13	0.984252	186.83	0.964020	1.44	0.990221	1.47	0.984252	1.43
	14	0.284253	136.10	0.314348	1.21	0.303445	1.24	0.284253	1.20
	15	0.089651	140.28	1.039513	1.23	0.096642	1.27	0.089651	1.22
	16	0.000099	10.26	0.000136	0.65	0.000113	0.69	0.000099	0.63
	17	0.156305	$\approx 2h$	0.201841	6.02	0.200773	10.84	0.186872	3.60
	18	0.623169	334.15	0.645441	6.72	0.631922	12.20	0.632369	3.91
	19	0.985956	300.59	0.945867	8.89	0.969026	14.08	0.985955	4.41
APOLLO	21	0.600787	$\approx 2h$	0.633724	3.15	0.618542	3.61	0.063177	3.51
CONFLICT	22	0.049456	101.09	0.499715	5.77	0.500178	18.10	0.500100	1.87
TURNLOGIC	23	0.370931	443.62	0.731781	1.70	0.725658	3.94	0.727923	1.13

equally large intervals and to assign each interval the probability it would have according to the original distribution, i.e. the probability for an interval $[a,b]$ would be $CDF(b) - CDF(a)$, where $CDF(\cdot)$ is the cumulative distribution function of the original continuous distribution. A deeper knowledge of the constraints to be quantified may allow more effective discretization where smaller intervals are used to increase the resolution of the approximate distributions around the points mostly affecting the satisfaction of the constraints to be quantified. However, this would require in general human expertise on the specific problem.

Table 2 reports our experimental results. We keep the result of NProbability as reference value. Notice that the results in this table differ from those on Table 3 due to the different usage profiles. Furthermore, with the simplified usage profile we use for these experiments NProbability always terminates correctly.

We discretized the domain of the two non-uniform variables in 3 and 6 intervals. The total number of usage scenarios composing the discretized profile is thus 9 and 36, respectively (cf. Section 3.1). A too coarse-grained discretization may introduce a bias in the result due to the loss of information. Finer discretization improves the precision of the result, but does not scale (due to the exponential blowup in the number of usage scenarios). This is visible in Table 2, where the results for a 3 intervals discretization deviate from the reference value more than those for the 6 intervals. Distribution-aware sampling prevents the risk of introducing such biases.

Finally, finer discretization requires a higher computation cost. Though this cost might be reduced leveraging the caching of partial results for independent sub-problems shared by the different usage scenarios, the worst case complexity remains exponential. Even with the relatively coarse-grained discretization we applied on only two non-uniform variables, the analysis time for the discretized profiles takes longer than with distribution-aware sampling. The latter grows instead only linearly with the number of variables, thus scaling to significantly larger problems.

5.3 Iterative Optimal Sampling Allocation

Iterative sampling allocation aims to improve the convergence rate of simulation-based quantification by allocating samples to sub-problems according to their predicted importance. Section 4 described three strategies to decide how to allocate samples to non-initial iterations of the quantification procedure; we evaluate these strategies here. The first strategy – *gradient* – decides how many samples to allocate to each sub-problem pursuing a gradient descent minimization of the global variance. The second strategy – *sensitivity* – performs a gradient-based sensitivity analysis to identify the single partial sub-problem with highest impact on the global variance and allocate new samples to improve the estimate. The third strategy – *local* – uses a low-overhead heuristic selecting the sub-problem to improve only based on the variance of its local estimator, thus not requiring a global impact analysis.

The goal of the three strategies is to increase the convergence rate of the compositional simulation-based quantification described previously. We take as *baseline* the distribution-aware sampling described and evaluated in the previous sections. We evaluate the iterative techniques with a sampling budget per iteration of 1k and 10k samples (which determines the step size for the gradient-based methods). We report the results of our experiments on selected subjects in Figure 2 and Table 4. Each subject is identified by the name of the program and the id of the assertion. For the experiments in this section, we assigned each variable a truncated Normal distribution with mean in the center of the variable domain (defined in the original papers of the benchmarks) and standard deviation equal to 1/6 of the domain length, as we did in the previous section.

5.3.1 Convergence Rate

Figure 2 shows the convergence rate of the three approaches and the baseline through plots having the wall-clock time on the x-axis and the average standard deviation of the global estimator (i.e., the square root of its variance) on the y-axis, in logarithmic scale. All the experiments have been ran for 30 minutes. The initial bootstrapping has been performed by taking 50k samples uniformly among all the sub-problems.

Gradient and sensitivity outperformed the baseline for all of the subjects and both 1k and 10k sampling budget per iteration. When more samples are allowed, these two approaches performs almost equally. The best improvement is achieved for APOLLO (21), while the worst is for ARTRIAL(3). These two subject are the most complex in terms of number of PCs and number of conjuncts per PC. However, while in the case of APOLLO only a few sub-problems have a high impact on the global variance, for ARTRIAL the sub-

Table 3: Distribution-aware sampling: comparison of `NProbability` and `qCORAL`.

Subject	Asrt.	NProbability		qCORAL [100k]		
		solution	*time(s)*	*estimate*	*avg. σ*	*time(s)*
Gaussian distributions						
ARTRIAL	1	0.031483	28.14	0.031483	0.00e+00	0.60
	2	0.000118	37.08	0.000119	0.000001	0.76
	3	0.968101	≈ 26*m*	0.964187	0.001159	2.31
CART	4	0.999896	≈ 12*m*	0.993029	0.014424	1.23
	5	0.997998	≈ 13*m*	1.005696	0.014901	1.30
CORONARY	6	0.000005	73.03	0.000005	5.26e-08	0.49
	7	0.000000	12.41	0.000000	2.91e-09	0.40
EGFR EPI	8	0.008055	10.35	0.008055	0.00e+00	0.72
	9	0.006240	7.97	0.006240	0.00e+00	0.63
EGFR EPI SIMPLE	10	0.592727	3.29	0.592715	0.000229	0.48
	11	0.171350	2.09	0.171350	0.00e+00	0.42
INVPEND	12	0.002235	≈ 23*m*	0.002248	0.000030	2.35
PACK	13	0.999788	164.52	0.999788	0.00e+00	1.42
	14	0.065674	148.88	0.065674	0.00e+00	1.22
	15	0.008084	154.95	0.008084	0.00e+00	1.25
	16	0.000000	8.82	0.000000	0.00e+00	0.66
	17	0.036888	≈ 31*h*	0.036883	0.000077	3.67
	18	0.525443	≈ 32*h*	0.515872	0.023695	4.01
	19	0.986260	≈ 36*h*	0.999885	0.00e+00	4.35
APOLLO	21	0.702961	≈ 4*h*	0.622393	0.032056	3.02
CONFLICT	22	0.000091	103.93	0.500701	0.000702	1.52
TURN LOGIC	23	0.176554	≈ 10*m*	0.717674	0.029120	0.87
Exponential distributions						
ARTRIAL	1	0.053505	29.07	0.053505	0.00e+00	0.64
	2	0.00e+00	35.51	0.00e+00	0.00e+00	0.70
	3	0.995979	≈ 15*m*	0.996645	0.000219	2.26
CORONARY	6	0.000047	32.99	0.000050	0.000007	0.48
	7	0.00e+00	5.62	0.00e+00	0.00e+00	0.34
EGFR EPI	8	0.731059	0.16	0.078329	0.00e+00	0.61
	9	0.116840	15.36	0.116840	0.00e+00	0.59
EGFR EPI SIMPLE	10	0.698996	2.91	0.698996	0.00e+00	0.40
	11	0.032105	2.04	0.032105	0.00e+00	0.37
PACK	13	0.948792	329.84	0.948792	0.00e+00	1.46
	14	0.393585	242.62	0.393585	0.00e+00	1.23
	15	0.149316	254.56	0.149316	0.00e+00	1.26
	16	0.000214	18.77	0.000214	0.00e+00	0.65
	17	0.243622	≈ 1.2*h*	0.243619	0.000377	3.59
	18	0.641769	578.15	0.640926	0.002324	3.86
	19	0.952801	303.06	0.952801	0.00e+00	4.21
APOLLO	21	-	+40*h*	0.643080	0.063153	2.68
CONFLICT	22	0.00e+00	36.02	0.024934	0.005130	1.04
TURNLOGIC	23	0.031814	≈ 2*h*	0.736356	0.059437	0.77

problems to be analyzed have similar impact on it. In particular, for APOLLO only a few sub-problems have a large local variance and a high impact on the global result. This is an optimal condition for all three allocation strategies, which perform similarly. The baseline approach is instead unable to exploit this information and allocates samples on sub-problems already close to convergence or with low impact on the global result. For ARTRIAL (3), there is not a big difference in the impact of the different sub-problems, though a few of them have slightly larger effect on the global result. In this case the benefit of gradient-based techniques is limited and also local does not provide a significant improvement.

The local strategy fails to improve the convergence rate, and actually slows it down, when the sub-problems with highest variance have a low impact on the global result: ARTRIAL (2), and CORONARY(7). Indeed, the impact of a sub-problem depends not only on its local variance but also on the estimates and variance of the other sub-problems that are in conjunction within the PCs under analysis. In CORONARY (7) it is possible to observe a small spike for the local strategy. The initial sampling provides indeed only approximate estimates for the various sub-problems. These estimates are improved through the subsequent iterations. The greediness of local makes the method to keep sampling from a single sub-problem until its variance is reduced enough to move to another one. When more samples are allocated to a sub-problem, not only the variance of the corresponding estimator is decreased, but also the approximation of the global result is improved through the composition rules, possibly

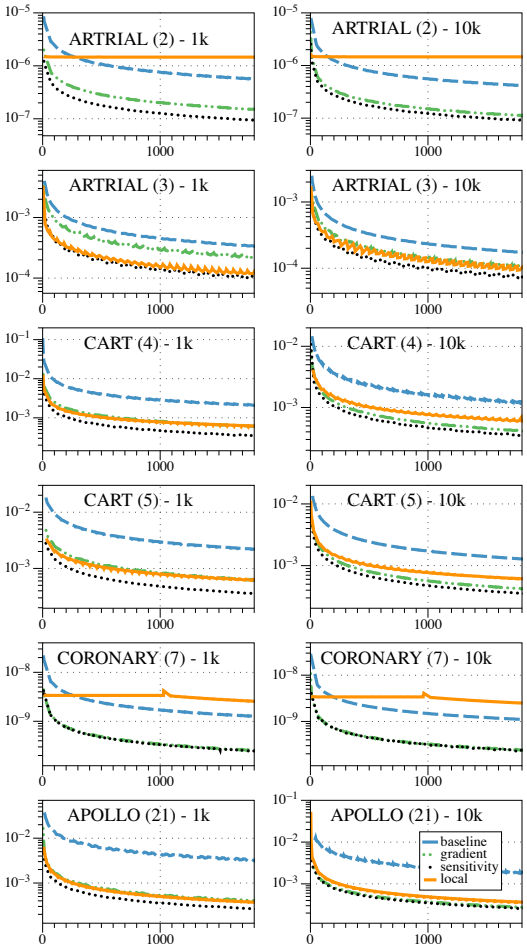

Figure 2: Convergence rate of iterative sampling and the baseline approach on selected subjects (y-axis in logarithmic scale).

correcting a wrong initial assessment. For this reason it is possible to obtain small spikes when moving from one sub-problem to another.

5.3.2 *Time to Converge*

Table 4 reports in tabular form the results of some of the experiments for a deeper investigation. All the runs have been interrupted after 30 minutes. Target σ represents different target accuracies in terms of standard deviation of the global result. We reported the computation time required by the three iterative allocation strategies and the baseline approach to achieve the target accuracy.

For lower accuracy (larger σ), the gradient-based methods performs better with 1k sampling budget per iteration, while for higher accuracy 10k performs slightly better. In both cases the convergence rate gets slower while moving toward higher accuracy. This observation is consistent with the theory on gradient-descent optimization, since this optimization approach gets less efficient when the result approaches its optimum. Nonetheless, the two methods outperform baseline and local even for higher accuracies, since they are capable of exploiting more information about the problem. Indeed, though all the derivatives tend to converge to 0 when approaching the optimum, the small differences among them are still enough to improve on the uninformed uniform allocation performed by baseline, while local may waste time sampling from low impact sub-problems having high variance. For higher accuracy, when the differences among the derivatives get smaller, the expected impact of the different sub-problems tends to become similar. In this case, since there is no

Table 4: Time to reach a target accuracy for incremental sampling techniques plus baseline for 1k and 10k sampling budget per iteration. Initial uniform sampling bootstrap 50k.

Subject	Asrt.	Prc. 10^{-x}	Time (s)							
			Baseline		Local		Sensitivity		Gradient	
			1k	10k	1k	10k	1k	10k	1k	10k
APOLLO	21	1	3.56	3.40	3.34	3.36	3.35	3.36	3.00	3.05
		2	191.49	64.13	5.97	6.11	4.73	5.20	6.13	4.46
		3	+30m	+30m	253.03	235.47	130.79	124.79	297.78	138.54
CART	4	1	1.48	1.61	1.45	1.58	1.39	1.53	1.44	1.56
		2	83.40	28.08	4.10	3.63	3.25	3.29	6.99	4.36
		3	+30m	+30m	527.37	526.14	217.71	221.28	641.56	310.85
	5	1	1.53	1.58	1.47	1.52	1.48	1.55	1.50	1.56
		2	86.51	31.00	4.17	3.77	3.35	3.31	7.32	4.49
		3	+30m	+30m	541.47	525.43	227.99	224.31	686.19	320.69
ARTRIAL	2	4	0.96	1.02	0.95	1.02	0.95	1.01	0.96	1.05
		5	7.10	4.44	0.95	1.02	1.11	1.08	1.57	1.46
		6	580.69	309.14	+30m	+30m	17.25	15.53	40.28	24.03
		7	+30m	+30m	+30m	+30m	1603.58	1475.79	+30m	+30m
	3	1	2.97	3.13	3.14	3.26	3.21	3.28	3.06	3.19
		2	4.93	3.72	3.14	3.26	3.21	3.28	3.62	3.36
		3	207.40	57.62	13.12	8.93	20.81	11.97	92.26	22.94
		4	+30m	+30m	+30m	1565.44	1730.84	972.12	+30m	1727.74
CORONARY	6	5	0.64	0.82	0.58	0.82	0.61	0.83	0.61	0.64
		6	2.99	2.33	0.58	0.82	0.65	0.84	0.74	0.69
		7	167.43	114.38	0.63	0.83	4.55	4.40	6.16	5.64
		8	+30m	+30m	+30m	+30m	354.03	346.11	527.16	450.78
	7	6	0.53	0.49	0.83	0.57	0.84	0.58	1.10	0.60
		7	1.08	0.81	0.83	0.57	0.85	0.58	1.11	0.60
		8	30.38	23.13	0.84	0.59	2.31	1.59	3.00	2.05
		9	+30m	+30m	+30m	+30m	109.84	100.68	120.60	105.37

significantly better choice, it is more efficient to take more samples per iteration instead of consuming analysis time for updating the gradient and taking new decisions frequently. We plan to investigate in the future dynamic techniques to handle this situation.

According to our experiments, sensitivity overall provides the best strategy for iterative sampling allocation, combining the effectiveness of gradient-based impact analysis with a reduced overhead for decision making, eventually leading to more accurate quantification results in a shorter time.

6. RELATED WORK

Our work is related to probabilistic program analysis [13], probabilistic abstract interpretation [20], probabilistic model checking [15] and volume computations [8]. We discuss here some of the most closely related work.

Probabilistic analysis based on symbolic execution has been described in e.g. [9, 11, 28]. Geldenhuys et al. [11] considered uniform distributions for the inputs, linear integer arithmetic constraints, and used LattE Machiato [8] to count solutions of path conditions produced during symbolic execution. Sankaranarayanan et al. [28] and Filieri et al. [9] proposed similar techniques to compute probabilities of violating program assertions. Both techniques remove the restriction of uniform distributions, although in the latter case it is by discretizing the domain into small uniform regions. As with [11] both approaches only consider linear constraints. Sankaranarayanan et al. developed algorithms for under and over-approximations of probabilities. They use Linear Programming (LP) solvers to compute over-approximations and heuristics-based "ray-shooting" algorithms to compute under-approximations, which is applicable for convex polyhedra. Filieri et al. used the LattE tool to compute probabilities. Furthermore the approach in [9] provides treatment of multi-threading and input data structures (it uses the Korat tool [5] for counting the input structures). Follow-on work provides thorough treatment of nondeterminism [19] and describes alternative statistical exploration of symbolic paths [10].

Another simulation-based approach has been proposed in [22] for the analysis of probabilistic programs. In that work Markov Chain Monte Carlo estimation [27] is enhanced with a preliminary program analysis aiming at generating verification conditions for all the operations involving probabilistic variables and operators. A violation of any of these conditions implies the violation of the program assertions and, since these conditions may be processed before reaching the assertions, the simulations can terminate earlier reducing the overall analysis time. The work targets small probabilistic programs which describe complex probability distributions and the probabilistic analysis is not compositional itself. In turn our work provides compositional sampling techniques for the analysis of constraints generated from arbitrary programs, enhanced with iterative techniques that focus the sampling on the constraints deemed the most important.

The technique in [2] proposed a compositional quantification of the solution space based on Monte Carlo estimation (briefly recalled in Section 2). This approach can deal with arbitrarily complex numeric constraints over floating-point domains and scales better than previous approaches, however it is limited to discretized profiles and, as a simulation-based approach, may suffer from slow convergence rate. We extend that work in two directions: direct handling of non-uniform distributions and focused, iterative sampling.

Bouissou *et al.* [4] and Adje *et al.* [1] handle non-linear constraints with a combination of abstraction based on affine and p-box arithmetic. The approach relies on the use of noise variables to represent the uncertainty of non-linear computations. They use an abstraction based approach whereas we use a statistical approach. We can thus handle a wider set of non-linear constraints such as complex mathematical functions (sine, cosine, etc.). In future work we would like to do an empirical comparison between these two approaches on the examples that both can handle.

Our work is also related to weighted model counting and distribution aware sampling [6], which has many applications beyond probabilistic symbolic execution, e.g. in machine learning and constrained random verification. Chakraborty et al. [6] address the problem in the context of CNF Boolean formulas, while our work is concerned with arbitrarily complex mathematical constraints.

7. CONCLUSIONS

We presented an iterative distribution-aware sampling technique for probabilistic symbolic execution. Given a set of complex mathematical constraints representing the path conditions collected with symbolic execution over a program, we use a statistical technique to estimate the probability of satisfying them assuming the values of the constrained inputs follow given continuous probability distributions. To speed-up the convergence rate of the analysis, we proposed three strategies for iterative sampling allocation, which focus the sampling on the constraints that have the largest influence on the estimated results. Experimental evaluation of the three iterative allocation strategies show their respective effectiveness.

In the future we plan extend our tool with distribution-aware sampling from multivariate input distributions (that relate multiple input variables). In theory our proposed approach works for such cases, however efficient sampling from multivariate distributions is an open problem in statistics, often involving more sophisticated techniques such as Gibbs sampling [18, 27]. We also plan to explore the automatic tuning of the sampling budget to allocate for each iteration for reducing the decision overhead by taking into account the relative gain of each possible decision over the others.

8. ACKNOWLEDGEMENTS

This work is supported by NSF Awards CCF-1329278 and CCF-1319858 (Păsăreanu). Mateus Borges is supported by FACEPE fellowship number IBPG-0668-1.03/12.

9. REFERENCES

[1] A. Adje, O. Bouissou, J. Goubault-Larrecq, E. Goubault, and S. Putot. "Static Analysis of Programs with Imprecise Probabilistic Inputs." In: *Verified Software: Theories, Tools, Experiments*. Vol. 8164. LNCS. Springer, 2014, pp. 22–47. DOI: 10.1007/978-3-642-54108-7_2.

[2] M. Borges, A. Filieri, M. d'Amorim, C. S. Păsăreanu, and W. Visser. "Compositional Solution Space Quantification for Probabilistic Software Analysis." In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. ACM, 2014, pp. 123–132. DOI: 10.1145/2594291.2594329.

[3] M. Borges, A. Filieri, M. d'Amorim, and C. S. Păsăreanu. *Iterative Distribution-Aware Sampling for Probabilistic Symbolic Execution - extended version*. http://www.antonio.filieri.name/publications/preprints/2015-fse-qcoral.pdf. 2015.

[4] O. Bouissou, E. Goubault, J. Goubault-Larrecq, and S. Putot. "A generalization of p-boxes to affine arithmetic." In: *Computing* 94.2-4 (2012), pp. 189–201. DOI: 10.1007/s00607-011-0182-8.

[5] C. Boyapati, S. Khurshid, and D. Marinov. "Korat: Automated Testing Based on Java Predicates." In: *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA '02. ACM, 2002, pp. 123–133. DOI: 10.1145/566172.566191.

[6] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. "Distribution-Aware Sampling and Weighted Model Counting for SAT." In: *AAAI*. 2014, pp. 1722–1730.

[7] A. Cohen. *Truncated and Censored Samples: Theory and Applications*. Statistics: A Series of Textbooks and Monographs. Taylor & Francis, 1991.

[8] J. A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, and J. Wu. "Software for Exact Integration of Polynomials over Polyhedra." In: *ACM Commun. Comput. Algebra* 45.3/4 (Jan. 2012), pp. 169–172. DOI: 10.1145/2110170.2110175.

[9] A. Filieri, C. S. Păsăreanu, and W. Visser. "Reliability Analysis in Symbolic Pathfinder." In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. IEEE Press, 2013, pp. 622–631. DOI: 10.1109/ICSE.2013.6606608.

[10] A. Filieri, C. S. Păsăreanu, W. Visser, and J. Geldenhuys. "Statistical Symbolic Execution with Informed Sampling." In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE '14. ACM, 2014, pp. 437–448. DOI: 10.1145/2635868.2635899.

[11] J. Geldenhuys, M. B. Dwyer, and W. Visser. "Probabilistic Symbolic Execution." In: *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ISSTA '12. ACM, 2012, pp. 166–176. DOI: 10.1145/2338965.2336773.

[12] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli. "Mining Behavior Models from User-intensive Web Applications." In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE '14. ACM, 2014, pp. 277–287. DOI: 10.1145/2568225.2568234.

[13] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. "Probabilistic Programming." In: *Proceedings of the on Future of Software Engineering*. FOSE '14. ACM, 2014, pp. 167–181. DOI: 10.1145/2593882.2593900.

[14] L. Granvilliers and F. Benhamou. "Algorithm 852: RealPaver: An Interval Solver Using Constraint Satisfaction Techniques." In: *ACM Trans. Math. Softw.* 32.1 (Mar. 2006), pp. 138–156. DOI: 10.1145/1132973.1132980.

[15] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. "PRISM: A Tool for Automatic Verification of Probabilistic Systems." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 3920. LNCS. Springer, 2006, pp. 441–444. DOI: 10.1007/11691372_29.

[16] F James. "Monte Carlo theory and practice." In: *Reports on Progress in Physics* 43.9 (1980), p. 1145. DOI: 10.1088/0034-4885/43/9/002.

[17] J. C. King. "Symbolic Execution and Program Testing." In: *Commun. ACM* 19.7 (July 1976), pp. 385–394. DOI: 10.1145/360248.360252.

[18] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. Vol. 706. John Wiley & Sons, 2011.

[19] K. Luckow, C. S. Păsăreanu, M. B. Dwyer, A. Filieri, and W. Visser. "Exact and Approximate Probabilistic Symbolic Execution for Nondeterministic Programs." In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ASE '14. ACM, 2014, pp. 575–586. DOI: 10.1145/2642937.2643011.

[20] D. Monniaux. "An Abstract Monte-Carlo Method for the Analysis of Probabilistic Programs." In: *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '01. ACM, 2001, pp. 93–101. DOI: 10.1145/360204.360211.

[21] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2006.

[22] A. V. Nori, C.-K. Hur, S. K. Rajamani, and S. Samuel. "R2: An Efficient MCMC Sampler for Probabilistic Programs." In: *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2014.

[23] W. Pestman. *Mathematical Statistics*. De Gruyter, 2009.

[24] W. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[25] C. S. Păsăreanu, W. Visser, D. Bushnell, J. Geldenhuys, P. Mehlitz, and N. Rungta. "Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis." In: *Automated Software Engineering* 20.3 (2013), pp. 391–425. DOI: 10.1007/s10515-013-0122-2.

[26] *qCORAL*. pan.cin.ufpe.br/qcoral. 2014.

[27] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, Inc., 2005.

[28] S. Sankaranarayanan, A. Chakarov, and S. Gulwani. "Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths." In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. ACM, 2013, pp. 447–458. DOI: 10.1145/2491956.2462179.

[29] The Apache Software Foundation. *Commons Math*. http://commons.apache.org/proper/commons-math/. Accessed: 2014-12-16.

[30] *VolComp*. http://systems.cs.colorado.edu/research/cyberphysical/probabilistic-program-analysis/. 2013.