INC: A Scalable Incremental Weighted Sampler

Suwei Yang  
National University of Singapore  
Singapore  
suwei.yang@comp.nus.edu.sg

Victor Liang  
GrabTaxi Holdings Pte. Ltd.  
Singapore  
victor.liang@grab.com

Kuldeep S. Meel  
National University of Singapore  
Singapore  
meel@comp.nus.edu.sg

Abstract—The fundamental problem of weighted sampling involves sampling of satisfying assignments of Boolean formulas, which specify sampling sets, and according to distributions defined by pre-specified weight functions to weight functions. The tight integration of sampling routines in various applications has highlighted the need for samplers to be incremental, i.e., samplers are expected to handle updates to weight functions.

The primary contribution of this work is an efficient knowledge compilation-based weighted sampler, INC, designed for incremental sampling. INC builds on top of the recently proposed knowledge compilation language, OBDD∧, and is accompanied by rigorous theoretical guarantees. Our extensive experiments demonstrate that INC is faster than state-of-the-art approach for majority of the evaluation. In particular, we observed a median of $1.69\times$ runtime improvement over the prior state-of-the-art approach.

Index Terms—knowledge compilation, sampling, weighted sampling

I. INTRODUCTION

Given a Boolean formula $F$ and weight function $W$, weighted sampling involves sampling from the set of satisfying assignments of $F$ according to the distribution defined by $W$. Weighted sampling is a fundamental problem in many fields such as computer science, mathematics and physics, with numerous applications. In particular, constrained-random simulation forms the bedrock of modern hardware and software verification efforts [1].

Sampling techniques are fundamental building blocks, and there has been sustained interest in the development of sampling tools and techniques. Recent years witnessed the introduction of numerous sampling tools and techniques, from approximate sampling techniques to uniform samplers SPUR and KUS, and weighted sampler WAPS [2]–[6]. Sampling tools and techniques have seen continuous adoption in many applications and settings [7]–[12]. The scalability of a sampler is a consideration that directly affects its adoption rate. Therefore, improving scalability continues to be a key objective for the community focused on developing samplers.

The tight integration of sampling routines in various applications has highlighted the importance for samplers to handle incremental weight updates over multiple sampling rounds, also known as incremental weighted sampling. Existing efforts on improving scalability typically focus on single round weighted sampling, and might have overlooked the incremental setting. In particular, existing approaches involving incremental weighted sampling typically employ off-the-shelf weighted samplers which could lead to less than ideal incremental sampling performance.

The primary contribution of this work is an efficient scalable weighted sampler INC that is designed from the ground up to address scalability issues in incremental weighted sampling settings. The core architecture of INC is based on knowledge compilation (KC) paradigm, which seeks to succinctly represent all satisfying assignments of a Boolean formula with a directed acyclic graph (DAG) [13]. In the design of INC, we make two core decisions that are responsible for outperforming the current state-of-the-art weighted sampler. Firstly, we build INC on top of PROB (Probabilistic OBDD∧ [14]) which is substantially smaller than the KC diagram used in the prior state-of-the-art approaches. Secondly, INC is designed to perform annotation, which refers to the computation of joint probabilities, in log-space to avoid the slower alternative of using arbitrary precision math computations.

Given a Boolean formula $F$ and weight function $W$, INC compiles and stores the compiled PROB in the first round of sampling. The weight updates for subsequent incremental sampling rounds are processed without recompilation, amortizing the compilation cost. Furthermore, for each sampling round, INC simultaneously performs annotation and sampling in a single bottom-up pass of the PROB, achieving speedup over existing approaches. We observed that INC is significantly faster than the existing state-of-the-art in the incremental sampling routine. In our empirical evaluations, INC achieved a median of $1.69\times$ runtime improvement over the state-of-the-art weighted sampler, WAPS [6]. Additional performance breakdown analysis supports our design choices in the development of INC. In particular, PROB is on median 4.64× smaller than the KC diagram used by the competing approach, and log-space annotation computations are on median 1.12× faster than arbitrary precision computations. Furthermore, INC demonstrated significantly better handling of incremental sampling rounds, with incremental sampling rounds to be on median 5.9% of the initial round, compared to 67.6% for WAPS.

The rest of the paper is organized as follows. We first introduce the relevant background knowledge and related works in Section II. We then introduce PROB and its properties in Section III. In Section IV, we introduce our weighted sampler INC, detail important implementation decisions, and provide theoretical analysis of INC. We then describe the extensive
empirical evaluations and discuss the results in Section V. Finally, we conclude in Section VI.

II. BACKGROUND AND RELATED WORK

Knowledge Compilation: Knowledge compilation (KC) involves representing logical formulas as directed acyclic graphs (DAG), which are commonly referred to as knowledge compilation diagrams [13]. The goal of knowledge compilation is to allow for tractable computation of certain queries such as model counting and weighted sampling. There are many well-studied forms of knowledge compilation diagrams such as d-DNNF, SDD, BDD, ZDD, OBDD, AOBDD, and the likes [15]–[21]. In this work, we build our weighted sampler upon a variant of OBDD known as OBDD[∧] [14].

OBDD[∧]: Lee [15] introduced Binary Decision Diagram (BDD) as a way to represent Shannon expansion [22]. [16] introduced fixed variable orderings to BDDs (known as OBDD) [16] for canonical representation and compression of BDDs via shared sub-graphs. Lai et al. [14] introduced conjunction nodes to OBDDs (known as OBDD[∧]) [14] to further reduce the size of the resultant DAG to represent a given Boolean formula. In this work, we parameterize an OBDD[∧] to form a PROB that is used for weighted sampling.

Sampling: A Boolean variable \( x \) can be assigned either true or false, and its literal refers to either \( x \) or its negation. A Boolean formula in conjunctive normal form (CNF) if it is a conjunction of clauses, with each clause being a disjunction of literals. A Boolean formula \( F \) is satisfiable if there exists an assignment \( \tau \) of its variables such that the \( F \) evaluates to true. The model count of Boolean formula \( F \) refers to the number of distinct satisfying assignments of \( F \).

Weighed sampling concerns with sampling elements from a distribution according to non-negative weights provided by a user-defined weight function \( W \). In the context of this work, weighted sampling refers to the process of sampling from the space of satisfying assignments of a Boolean formula \( F \). The weight function \( W \) assigns a non-negative weight to each literal \( F \). The weight of an assignment \( \tau \) is defined as the product of the weight of its literals.

WAPS: KUS [5] utilizes knowledge compilation techniques, specifically Deterministic Decomposable Negation Normal Form (d-DNNF) [19], to perform uniform sampling in 2 passes of the d-DNNF. Annotation is performed in the first pass, followed by sampling. WAPS [6] improves upon KUS by enabling weighted sampling via parameterization of the d-DNNF. WAPS performs sampling in a similar manner to KUS, the main difference being that the annotation step in WAPS takes into account the provided weight function. In contrast, we introduce INC which performs weighted sampling in a single pass by leveraging the DAG structure of PROB.

Knowledge compilation-based samplers typically perform incremental sampling as follows. The sampling space is first expressed as satisfying assignments of a Boolean formula, which is then compiled into the respective knowledge compilation form. In the following step, samples are drawn according to the given weight function \( W \). Subsequently, the weights are updated depending on application logic and weighted sampling is performed again. The process is repeated until an application-specific stopping criterion is met. An example of such an application would be the Baital framework [10], developed to use incremental weighted sampling to generate test cases for configurable systems.

III. PROB: PROBABLISTIC OBDD[∧]

PROB is a DAG composed of four types of nodes - conjunction, decision, true and false nodes. The internal nodes of a PROB consist of conjunction and decision nodes whereas the leaf nodes of the PROB consist of true and false nodes. A PROB is recursively made up of sub-PROBs that represent sub-formulas of Boolean formula \( F \). We use \( \text{VarSet}(n) \) to refer to the set of variables of \( F \) represented by a PROB with \( n \) as the root node. \( \text{Subdiagram}(n) \) refers to the sub-PROB starting at node \( n \) and \( \text{Parent}(n) \) refers to the immediate parent of node \( n \) in PROB.

A. PROB Structure

Conjunction node (\( \land \)-node): A \( \land \)-node \( n_c \) represents conjunctions in the assignment space. There are no limits to the number of child nodes that \( n_c \) can have. However, the set of variables \( \text{VarSet}(\cdot) \) of each child node of \( n_c \) must be disjoint. An example of a \( \land \)-node would be \( n_2 \) in Figure 1. Notice that \( \text{VarSet}(n4) = \{ z \} \) and \( \text{VarSet}(n5) = \{ y \} \) are disjoint.

Decision node: A decision node \( n_d \) represents decisions on the associated Boolean variable \( \text{Var}(n_d) \) in Boolean formula \( F \) that the PROB represents. A decision node can have exactly two children - \( \text{lo-child} \) \( \text{Lo}(n_d) \) and \( \text{hi-child} \) \( \text{Hi}(n_d) \). \( \text{Lo}(n_d) \) represents the assignment space when \( \text{Var}(n_d) \) is set to false and \( \text{Hi}(n_d) \) represents otherwise. \( \theta_{n_d} \) and \( \theta_{n_d'} \) refer to the parameters associated with the edge connecting decision node \( n_d \) with \( \text{Hi}(n_d) \) and \( \text{Lo}(n_d) \) respectively in a PROB. Node \( n1 \) in Figure 1 is a decision node with \( \text{Var}(n1) = \{ x \} \), \( \text{Hi}(n1) = n3 \) and \( \text{Lo}(n1) = n2 \).

True and false nodes: True (\( \top \)) and false (\( \bot \)) nodes are leaf nodes in a PROB. Let \( \tau \) be an assignment of all variables of Boolean formula \( F \) and let PROB \( \psi \) represent \( F \). \( \tau \) corresponds to a traversal of \( \psi \) from the root node to leaf nodes. The traversal follows \( \tau \) at every decision node and visits all child nodes of every conjunction node encountered along the way. \( \tau \) is a satisfying assignment if all parts of the traversal eventually lead to the true node. \( \tau \) is not a satisfying assignment if any part of the traversal leads to the false node. With reference to Figure 1, let \( \tau_1 = \{ x, y, \neg z \} \) and \( \tau_2 = \{ x, y, z \} \).

B. PROB Parameters

In the PROB structure, each decision node \( n_d \) has two parameters \( \theta_{\text{Lo}(n_d)} \) and \( \theta_{\text{Hi}(n_d)} \), associated with the two branches...
Given a weight function, the determinism of probabilities of each child node. At each decision node, there are only two possible outcomes on the joint probability of a given node. At each conjunction node, it is straightforward to calculate the joint probabilities at every literal set of the child nodes of $n$. One can view the decomposability and determinism properties introduced later. Let $x_i$ be $\text{Var}(n_d)$. Given a weight function $W$:

$$\theta_{\text{Lo}(n_d)} = \frac{W(\neg x_i)}{W(\neg x_i) + W(x_i)} \quad \theta_{\text{Hi}(n_d)} = \frac{W(x_i)}{W(\neg x_i) + W(x_i)}$$

C. PROB Properties

The PROB structure has important properties such as determinism and decomposability. In addition to the determinism and decomposability properties, we ensure that PROBs used in this work have the smoothness property through a smoothing process (Algorithm 1).

**Property 1** (Determinism). For every decision node $n_d$, the set of satisfying assignments represented by $\text{Hi}(n_d)$ and $\text{Lo}(n_d)$ are logically disjoint.

**Property 2** (Decomposability). For every conjunction node $n_c$, $\text{VarSet}(c_i) \cap \text{VarSet}(c_j) = \emptyset$ for all $c_i$ and $c_j$ where $c_i, c_j \in \text{Child}(n_c)$ and $c_i \neq c_j$.

**Property 3** (Smoothness). For every decision node $n_d$, $\text{VarSet}(\text{Hi}(n_d)) = \text{VarSet}(\text{Lo}(n_d))$.

D. Joint Probability Calculation with PROB

In Section III-B, we mention that one can view the branch parameters as the probability of choosing between the positive and negative literal of a decision node. Notice that because of the decomposability and determinism properties of PROB, it is straightforward to calculate the joint probabilities at every given node. At each conjunction node $n_c$, since the variable sets of the child nodes of $n_c$ are disjoint by decomposability, the joint probability of $n_c$ is simply the product of joint probabilities of each child node. At each decision node $n_d$, there are only two possible outcomes on $\text{Var}(n_d)$ - positive literal $\text{Var}(n_d)$ or negative literal $\neg\text{Var}(n_d)$. By determinism property, the joint probability is the sum of the two possible scenarios. Formally, the calculations for joint probabilities $P'$ at each node in PROB are as follows:

$$P' \text{ of } \land\text{-node } n_c = \prod_{c \in \text{Child}(n_c)} P'(c) \quad \text{(EQ1)}$$

$$P' \text{ of decision-node } n_d = \theta_{\text{Lo}(n_d)} \times P'(\text{Lo}(n_d)) + \theta_{\text{Hi}(n_d)} \times P'(\text{Hi}(n_d)) \quad \text{(EQ2)}$$

For true node $n$, $P'(n) = 1$ because it represents satisfying assignments when reached. In contrast $P'(n) = 0$ when $n$ is a false node as it represents non-satisfying assignments. In Proposition 2, we show that weighted sampling is equivalent to sampling according to joint probabilities of satisfying assignments of a PROB.

IV. INC - SAMPLING FROM PROB

In this section, we introduce INC - a bottom-up algorithm for weighted sampling on PROB. We first describe INC for drawing one sample and subsequently describe how to extend INC to draw $k$ samples at once. We also provide proof of correctness that INC is indeed performing weighted sampling. As a side note, samples are drawn with replacement, in line with the existing state-of-the-art weighted sampler [6].

A. Preprocessing PROB

In the main sampling algorithm (Algorithm 2) to be introduced later in this section, the input is a smooth PROB. As a preprocessing step, we introduce Smooth algorithm that takes in a PROB $\psi$ and performs smoothing.

The Smooth algorithm processes the nodes in the input PROB $\psi$ in a bottom-up manner while keeping track of $\text{VarSet}(n)$ for every node $n$ in $\psi$ using a map $\kappa$. True and false nodes have $\emptyset$ as they are leaf nodes and do not represent any variables. At each conjunction node, its variable set is the union of variable sets of its child nodes.

The smoothing happens at decision node $n$ in $\psi$ when $\text{VarSet}(\text{Lo}(n))$ and $\text{VarSet}(\text{Hi}(n))$ do not contain the same set of variables as shown by lines 8 and 16 of Algorithm 1. In the smoothing process, a new conjunction node $(\text{lcNode for } \text{Lo}(n)$ and $\text{rcNode for } \text{Hi}(n))$ is created to replace the corresponding child of $n$, with the original child node now set as a child of the conjunction node. Additionally, for each of the missing variables $v$, a decision node representing $v$ is created and added as a child of the respective conjunction node. The decision nodes created during smoothing have both their lo-child and hi-child set to the true node. To reduce memory footprint, we check if there exists the same decision node before creating it in the checkMakeTrueDecisionNode function.

As an example, we refer to $\psi_2$ in Figure 2. It is obvious that $\psi_2$ is not smooth, because $\text{VarSet}(\text{Lo}(n_1)) = \{y\}$ and $\text{VarSet}(\text{Hi}(n_1)) = \{z\}$. In the smoothing process, we replace $\text{Lo}(n_1)$ with a new conjunction node $n_2$ and add a decision node $n_4$ representing missing variable $z$, with both child set to true node $n_9$. We repeat the steps for $\text{Hi}(n_1)$ to arrive at PROB $\psi_1$ in Figure 1.
processes each of the remaining nodes in bottom-up order from the space of satisfying assignments we can ignore FalseNode in the same bottom-up pass. Since we want to sample in a bottom-up manner while integrating the process is illustrated by Algorithm 2.

For node $n$ of $\psi$'s nodes and edges leading to them, as shown by line 3. As hideFalseNode when applied to $\psi_1$ would remove node $n_5$ and the edges immediately leading to it. Next, INC processes each of the remaining nodes in bottom-up order while keeping two caches - $\omega$ to store the partial samples from each node, $\varphi$ to store the joint probability at each node.

**Algorithm 1** Smooth - returns a smoothed PROB

**Input:** PROB $\psi$

**Output:** smooth PROB

1. $\kappa \leftarrow \text{initMap}()$
2. for node $n$ of $\psi$ in bottom-up order do
3.  if $n$ is true node or false node then
4.      $\kappa[n] \leftarrow 0$
5.  else if $n$ is $\land$-node then
6.      $\kappa[n] \leftarrow \text{unionVarSet}(\text{Child}(n), \kappa)$
7.  else
8.    if $\kappa[\text{Hi}(n)] - \kappa[\text{Lo}(n)] \neq 0$ then
9.        lset $\leftarrow \kappa[\text{Hi}(n)] - \kappa[\text{Lo}(n)]$
10.       lcNode $\leftarrow \text{new \land-node}()$
11.       lcNode.addChild(Lo($n$))
12.       for $\varphi$ in lset do
13.           dNode $\leftarrow \text{checkMakeTrueDecisionNode}(\varphi)$
14.           lcNode.addChild(dNode)
15.       rcNode $\leftarrow \text{new \land-node}()$
16.       rcNode.addChild(Hi($n$))
17.       for $\varphi$ in rset do
18.           dNode $\leftarrow \text{checkMakeTrueDecisionNode}(\varphi)$
19.           rcNode.addChild(dNode)
20.       $\kappa[n] \leftarrow \text{Var}(n) \cup \text{unionVarSet}([\text{Hi}(n), \text{Lo}(n)])$
21. return $\psi$

**Algorithm 2** INC - returns a satisfying assignment based on PROB $\psi$ parameters

**Input:** smooth PROB $\psi$

**Output:** a sampled satisfying assignment

1. cache $\omega \leftarrow \text{initCache}()$
2. joint prob cache $\varphi \leftarrow \text{initCache}()$
3. $\varphi' \leftarrow \text{hideFalseNode(}\psi\text{)}$
4. for node $n$ of $\psi'$ in bottom-up order do
5.  if $n$ is true node then
6.      $\omega[n] \leftarrow \emptyset$
7.  else
8.      $\varphi[n] \leftarrow 1$
9. else if $n$ is $\land$-node then
10.     $\omega[n] \leftarrow \text{unionChild}(\text{Child}(n), \omega)$
11.     $\varphi[n] \leftarrow \prod_{c \in \text{Child}(n)} \varphi[c]$
12. else
13.     $p_{lo} \leftarrow \theta_{\text{Lo}(n)} \times \varphi[\text{Lo}(n)]$
14.     $p_{hi} \leftarrow \theta_{\text{Hi}(n)} \times \varphi[\text{Hi}(n)]$
15.     $p_{\text{joint}} \leftarrow p_{lo} + p_{hi}$
16.     $r \leftarrow x \sim \text{binomial}(1, \frac{p_{hi}}{p_{\text{joint}}})$
17. if $r$ is 1 then
18.     $\omega[n] \leftarrow \varphi[\text{Hi}(n)] \cup \text{Var}(n)$
19. else
20.     $\omega[n] \leftarrow \omega[\text{Lo}(n)] \cup \neg \text{Var}(n)$
21. return $\omega[\text{rootnode}(\psi)]$

INC starts with $\emptyset$ at the true node since there is no associated variable. At each conjunction node, INC takes the union of the child nodes in line 9. Using $n2$ in Figure 1 as an example, if sample drawn at $n4$ is $\omega[n4] = \{ \neg z \}$ and at $n5$ is $\omega[n5] = \{ y \}$, then unionChild($\text{Child}(n2), \omega) = \{ y, \neg z \}$. At each decision node $n$, a decision on Var($n$) is sampled from lines 16 to 20. We first calculate the joint probabilities, $p_{lo}$ and $p_{hi}$ of choosing $\neg \text{Var}(n)$ and choosing Var($n$). Subsequently, we sample decision on Var($n$) using a binomial distribution in line 16 with the probability of success being the joint probability of choosing Var($n$). After processing all nodes, the sampled assignment is the output at root node of $\psi$.

**Extending INC to $k$ samples:** It is straightforward to extend the single sample INC shown in Algorithm 2 to draw $k$ samples in a single pass, where $k$ is a user-specified number. At each node, we have to store a list of $k$ independent copies of partial assignments drawn in $\omega$. At each conjunction node $n_c$, we perform the same union process in line 9 of Algorithm 2 for child outputs in the same indices of the respective lists in $\omega$. More specifically, if $n_c$ has child nodes $c_x$ and $c_y$, the outputs of index $i$ are combined to get the output of $n_c$ at index $i$. This process is performed for all indices from 1 to $k$. At each decision node $n_d$, we now draw $k$ independent samples instead of a single sample from the binomial distribution as shown in line 16. The sampling step in lines 16 to 20 are performed independently for the $k$ random numbers. There is no change necessary for the calculation of joint probabilities.

Fig. 2: A PROB $\psi_2$ representing Boolean formula $F = (x \lor y) \land (\neg x \lor \neg z)$, branch parameters are omitted

B. Sampling Algorithm

INC takes a PROB $\psi$ representing Boolean formula $F$ and draws a sample from the space of satisfying assignments of $F$, the process is illustrated by Algorithm 2. INC performs sampling in a bottom-up manner while integrating the annotation process in the same bottom-up pass. Since we want to sample from the space of satisfying assignments we can ignore false nodes in $\psi$ entirely by considering a sub-DAG that excludes false nodes and edges leading to them, as shown by line 3. As an example, hideFalseNode when applied to $\psi_1$ would remove node $n_5$ and the edges immediately leading to it. Next, INC processes each of the remaining nodes in bottom-up order while keeping two caches - $\omega$ to store the partial samples from each node, $\varphi$ to store the joint probability at each node.
in Algorithm 2 as there is no change in literal weights.

**Incremental sampling:** Given a Boolean formula \( F \) and weight function \( W \), INC performs incremental sampling with the sampling process shown in Figure 3. In the initial round, INC compiles \( F \) and \( W \) into a PROB \( \psi \) and performs sampling. Subsequent rounds involve applying a new set of weights \( W \) to \( \psi \), typically generated based on existing samples by the controller [10], and performing weighted sampling according to the updated weights. The number of sampling rounds is determined by the controller component, whose logic varies according to application.

In the equations above, \( Q \) refers to the corresponding log joint probabilities in EQ1 and EQ2. In the experiments section, we detail the runtime advantages of using log computations compared to arbitrary precision math computations.

**Dynamic Annotation:** In existing state-of-the-art weighted sampler WAPS, sampling is performed in two passes - the first pass performs annotation and the second pass samples assignments according to the joint probabilities. In INC, we combine the two passes into a single bottom-up pass performing annotation dynamically while sampling at each node.

### D. Theoretical Analysis

**Proposition 1.** Branch parameters of any decision node \( n_d \) are correct sampling probabilities, i.e. \( W(x_i) : W(\neg x_i) = \theta_{\text{Hi}}(x_i) : \theta_{\text{Lo}}(x_i) \) where \( \text{Var}(n_d) = x_i \).

**Proof.**

\[
\frac{W(x_i)}{W(\neg x_i)} = \frac{W(x_i) + W(\neg x_i)}{W(x_i) + W(\neg x_i)} = \frac{\theta_{\text{Hi}}(x_i)}{\theta_{\text{Lo}}(x_i)}
\]

We start with the ratio of literal weights of \( x \), multiply both numerator and denominator by \( W(x_i) + W(\neg x_i) \) and arrive at the ratio of branch parameters of \( n_d \). Notice that only the ratio matters for sampling correctness and not the absolute value of weights.

**Remark 1.** Let \( n_d \) be an arbitrary decision node in PROB \( \psi \). When performing sampling according to a weight function \( W \), \( \theta_{\text{Lo}}(n_d) \) is the probability of picking \( \neg \text{Var}(n_d) \) and \( \theta_{\text{Hi}}(n_d) \) is that of \( \text{Var}(n_d) \). The determinism property states that the choice of either literal is disjoint at each decision node.

**Proposition 2.** INC samples an assignment \( \tau \) from PROB \( \psi \) with probability \( \frac{1}{N} \prod_{l \in \tau} W(l) \), where \( N \) is a normalization factor.

**Proof.** The proof consists of two parts, one for \( \wedge \)-node and another for decision node.

**\( \wedge \)-node:** Let \( n_c \) be an arbitrary conjunction node in PROB \( \psi \). Recall that by decomposability property, \( \forall c_i, c_j \in \text{Child}(n_c) \) and \( c_i \neq c_j \), \( \text{VarSet}(c_j) \cap \text{VarSet}(c_j) = \emptyset \). As such an arbitrary variable \( x_i \in \text{VarSet}(n_c) \) only belongs to the variable set of one child node \( c_i \in \text{Child}(n_c) \). Therefore, assignment of \( x_i \) can be sampled independent of \( x_j \) where \( x_j \in \text{VarSet}(c_j) \), \( \forall c_j \neq c_i \). Let \( \tau_{c_i} \) be partial assignment for child node \( c_i \in \text{Child}(n_c) \). Notice that each partial assignment \( \tau_{c_i} \) is sampled independently of others as there are no overlapping variables, hence their joint probability is simply the product of their individual probabilities. This agrees with the weight of an assignment being the product of its components, up to a normalization factor.

**Decision node:** Let \( n_d \) be an arbitrary decision node in PROB \( \psi \) and \( x_d \in \text{Var}(n_d) \). At \( n_d \), we sample an assignment of \( x_d \) based on the parameters \( \theta_{\text{Lo}}(x_d) \) and \( \theta_{\text{Hi}}(x_d) \), which are probabilities of literal assignment by Proposition 1. By Proposition 1, one can see that the assignment of \( x_d \) is sampled...
correctly according to \( W \). As the sampling process at \( n_d \) is independent of its child nodes by the determinism property, the joint probability of sampled assignment of \( x_d \) and the output partial assignment from the corresponding child node would be the product of their probabilities. Notice that the joint probability aligns with the definition of weight of an assignment being the product of the weight of its literals, up to a normalization factor.

Since we do not consider the \textit{false} node and treat it as having 0 probability, we always sample from satisfying assignments by starting at the \textit{true} node in bottom-up ordering. Reconciling the sampling process at the two types of nodes, it is obvious that any combination of decision and \( \land \)-nodes encountered in the sampling process would agree with a given weight function \( W \) up to a normalization factor \( 1/N \). In fact, \( N = \sum_{\tau_i \in S} W(\tau_i) \) where \( S \) is the set of satisfying assignments of Boolean formula \( F \) that \( \psi \) represents. As mentioned in Proposition 1 proof, normalization factors do not affect the correctness of sampling according to \( W \), and we have shown that \( \text{INC} \) performs weighted sampling correctly under multiplicative weight functions.

**Remark 2.** From the proof of Proposition 2, the determinism and decomposability property is important to ensure the correctness of \( \text{INC} \). The smoothness property is important to ensure that the sampled assignment by \( \text{INC} \) is complete. For formula \( F = (x \lor y) \land (\neg x \lor \neg z) \), an assignment \( \tau_1 \) sampled from a non-smooth \( \text{PROB} \) could be \( \{x, \neg z\} \). Notice that \( \tau_1 \) is missing assignment for variable \( y \). By performing smoothing, we will be able to sample a complete assignment of all variables in the Boolean formula as both child nodes of each decision node \( \alpha \) have the same \( \text{VarSet}(\cdot) \).

V. EXPERIMENTS

We implement \( \text{INC} \) in Python 3.7.10, using NumPy 1.15 and Toposort package. In our experiments, we make use of an off-the-shelf KC diagram compiler, KCBox [23]. In the later parts of this section, we performed additional comparisons against an implementation of \( \text{INC} \) using the Gmpy2 arbitrary precision math package (\text{INC}_{\text{GP}}) to determine the impact of log-space annotation computations.

Our benchmark suite consists of instances arising from a wide range of real-world applications such as DQMR networks, bit-blasted versions of SMT-LIB (SMT) benchmarks, ISCAS89 circuits, and configurable systems [6], [10]. For incremental updates, we rely on the weight generation mechanism proposed in the context of prior applications of incremental sampling [10]. In particular, new weights are generated based on the samples from the previous rounds, resulting in the need to recompute joint probabilities in each round. Keeping in line with prior work, we perform 10 rounds (R1-R10) of incremental weighted sampling and 100 samples drawn in each round. The experiments were conducted with a timeout of 3600 seconds on clusters with Intel Xeon Platinum 8272CL processors.

In this section, we detail the extensive experiments conducted to understand \( \text{INC} \)'s runtime behavior and to compare it with the existing state-of-the-art weighted sampler \text{WAPS} [6] in incremental weighted sampling tasks. We chose \text{WAPS} as it has been shown to achieve significant runtime improvement over other samplers, and accordingly has emerged as a sampler of the choice for practical applications [10]. In particular, our empirical evaluation sought to answer the following questions:

**RQ 1** How does \( \text{INC} \)'s incremental weighted sampling runtime performance compare to current state-of-the-art?

**RQ 2** How does using \( \text{PROB} \) affect runtime performance?

**RQ 3** How does log-space calculations impact runtime performance?

**RQ 4** Does \( \text{INC} \) correctly perform weighted sampling?

**RQ 1: Incremental Sampling Performance:** The scatter plot of incremental sampling runtime comparison is shown in Figure 4, with Figure 4a showing runtime comparison for the first round (R1) and Figure 4b showing runtime comparison over 10 rounds. The vertical axes represent the runtime of \( \text{INC} \) and the horizontal axes represent that of \text{WAPS}. In the experiments, \( \text{INC} \) completed 650 out of 896 benchmarks whereas \text{WAPS} completed 674. \( \text{INC} \) completed 21 benchmarks that \text{WAPS} timed out and similarly, \text{WAPS} completed 45 benchmarks that \( \text{INC} \) timed out. In the experiments, \( \text{INC} \) achieved a median speedup of \( 1.69 \times \) over \text{WAPS}.

Further results are shown in Table I. Observe that for runtime taken for R1 (column 3), \text{WAPS} is faster and takes around \( 0.44 \times \) of \( \text{INC} \)'s runtime in the median case. However, \( \text{INC} \) takes the lead in runtime performance when we examine the total time taken for the incremental rounds R2 to R10 (column 4). For incremental rounds, \text{WAPS} always took longer than \( \text{INC} \), in the median case \text{WAPS} took 4.48 times longer than \( \text{INC} \). We compare the average incremental round runtime with the first round runtime for both samplers in columns 1 and 2. In the median case, an incremental round for \text{WAPS} takes 67% of the time for R1 whereas an incremental round for \( \text{INC} \) only requires 5.9% of the time R1 takes. We show the per round runtime for 5 benchmarks in Table II to further illustrate \( \text{INC} \)'s runtime advantage over \text{WAPS} for incremental sampling rounds, even though both tools reuse the respective KC diagram compiled in R1. This set of results highlights \( \text{INC} \)'s superior performance over \text{WAPS} in the handling of incremental sampling settings. \( \text{INC} \)'s advantage in incremental sampling rounds led to better overall runtime performance than \text{WAPS} in 75% of evaluations. The runtime advantage of \( \text{INC} \) would be more obvious in applications requiring more than 10 rounds of samples.

Therefore, we conducted sampling experiments for 20 rounds to substantiate our claims that \( \text{INC} \) will have a larger runtime lead over \text{WAPS} with more rounds. Both samplers are given the same 3600s timeout as before and are to draw 100 samples per round, for 20 rounds. The number of completed benchmarks is shown in Table III. In the 20 sampling round setting, \( \text{INC} \) completed 649 out of 896 benchmarks, timing out on 1 additional benchmark compared to 10 sampling round setting. In comparison, \text{WAPS} completed 596 of 896 benchmarks, timing out on 78 additional benchmarks than in
Fig. 4: Runtime comparisons between INC and state-of-the-art weighted sampler WAPS

<table>
<thead>
<tr>
<th>Statistic</th>
<th>WAPS MEAN(R2 to R10)</th>
<th>INC MEAN(R2 to R10)</th>
<th>WAPS R1</th>
<th>INC R1</th>
<th>WAPS SUM(R2 to R10)</th>
<th>INC SUM(R2 to R10)</th>
<th>WAPS Total</th>
<th>INC Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.74</td>
<td>0.064</td>
<td>1.03</td>
<td>15.66</td>
<td>6.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std</td>
<td>0.24</td>
<td>0.040</td>
<td>1.47</td>
<td>26.42</td>
<td>10.73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>0.67</td>
<td>0.059</td>
<td>0.44</td>
<td>4.48</td>
<td>1.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>1.25</td>
<td>0.188</td>
<td>10.65</td>
<td>172.66</td>
<td>73.96</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE I: Incremental weighted sampling runtime ratio statistics for WAPS and INC (Numerators and denominators refer to the corresponding runtimes)

RQ 2: PROB Performance Impacts: We now focus on the analysis of the impact of using PROB compared to d-DNNF in the design of a weighted sampler. We analyzed the size of both PROB and d-DNNF across the benchmarks that both tools managed to compile and show the results in Table IV. From Table IV, PROB is always smaller than the corresponding d-DNNF. Additionally, PROB is at median 4.64× smaller than the corresponding d-DNNF, and that for PROB is an order of magnitude smaller for at least 25% of the benchmarks. As such, PROB emerges as the clear choice of knowledge compilation diagram used in INC, owing to its succinctness which leads to fast incremental sampling runtimes.

RQ 3: Log-space Computation Performance Impacts: In the design of INC, we utilized log-space computations to perform annotation computations as opposed to naively using arbitrary precision math libraries. In order to analyze the impact of this design choice, we implemented a version of INC where the dynamic annotation computations are performed using arbitrary precision math in a similar manner as WAPS. We refer to the arbitrary precision math version of INC as INCAP. As an ablation study, we compare the runtime of both implementations across all the benchmarks and show the comparison in Table V. The statistics shown is for the ratio of INCAP runtime to INC runtime, a value of 1.12 means that INCAP takes 1.12 × that of INC for the corresponding statistics.

The results in Table V highlight the runtime advantages of our decision to use log-space computations over arbitrary precision computations. INC has faster runtime than INCAP in majority of the benchmarks. INC displayed a minimum of 0.70×, a median of 1.12× and a max of 1.89× speedup over INCAP. Furthermore, INCAP timed out on 2 more benchmarks compared to INC. It is worth emphasizing that log-space
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Tool</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>Total</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>or-50-5-5-UC-10 (100, 253)</td>
<td>WAPS</td>
<td>56.6</td>
<td>56.3</td>
<td>52.5</td>
<td>59.4</td>
<td>52.5</td>
<td>53.6</td>
<td>59.4</td>
<td>53.2</td>
<td>53.4</td>
<td>61.7</td>
<td>558.6</td>
<td>1.0×</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>1461.3</td>
<td>7.6</td>
<td>8.4</td>
<td>8.4</td>
<td>8.5</td>
<td>8.5</td>
<td>8.4</td>
<td>8.5</td>
<td>8.5</td>
<td>1536.3</td>
<td></td>
<td>0.4×</td>
</tr>
<tr>
<td>or-100-20-9-UC-30 (200, 528)</td>
<td>WAPS</td>
<td>73.0</td>
<td>69.1</td>
<td>66.7</td>
<td>76.0</td>
<td>66.5</td>
<td>66.9</td>
<td>76.6</td>
<td>66.0</td>
<td>66.9</td>
<td>78.6</td>
<td>706.1</td>
<td>1.0×</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>269.5</td>
<td>4.7</td>
<td>4.8</td>
<td>4.8</td>
<td>4.9</td>
<td>5.1</td>
<td>4.8</td>
<td>4.8</td>
<td>5.1</td>
<td>313.4</td>
<td></td>
<td>2.3×</td>
</tr>
<tr>
<td>s953a_15_7 (602, 1657)</td>
<td>WAPS</td>
<td>1.7</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.3</td>
<td>11.9</td>
<td>1.0×</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>4.9</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>11.5</td>
<td>1.0×</td>
</tr>
<tr>
<td>h8max (1202, 3072)</td>
<td>WAPS</td>
<td>90.3</td>
<td>104.2</td>
<td>92.4</td>
<td>116.0</td>
<td>94.3</td>
<td>94.1</td>
<td>112.9</td>
<td>92.9</td>
<td>94.4</td>
<td>120.4</td>
<td>1011.9</td>
<td>1.0×</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>34.1</td>
<td>2.1</td>
<td>2.2</td>
<td>2.4</td>
<td>2.3</td>
<td>2.4</td>
<td>2.4</td>
<td>2.4</td>
<td>2.4</td>
<td>2.4</td>
<td>55.7</td>
<td>18.2×</td>
</tr>
<tr>
<td>innovator (1256, 50452)</td>
<td>WAPS</td>
<td>195.5</td>
<td>221.9</td>
<td>201.3</td>
<td>244.4</td>
<td>200.1</td>
<td>206.7</td>
<td>247.2</td>
<td>202.0</td>
<td>202.9</td>
<td>257.4</td>
<td>2179.3</td>
<td>1.0×</td>
</tr>
<tr>
<td>INC</td>
<td></td>
<td>195.5</td>
<td>221.9</td>
<td>201.3</td>
<td>244.4</td>
<td>200.1</td>
<td>206.7</td>
<td>247.2</td>
<td>202.0</td>
<td>202.9</td>
<td>257.4</td>
<td>2179.3</td>
<td>1.0×</td>
</tr>
</tbody>
</table>

Table II: Runtime (seconds) breakdowns for each of ten rounds (R1-R10) between WAPS and INC for benchmarks of different sizes e.g. ‘h8max’ benchmark consists of 1202 variables and 3072 clauses.

<table>
<thead>
<tr>
<th>Number of rounds</th>
<th>WAPS</th>
<th>INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>674</td>
<td>650</td>
</tr>
<tr>
<td>20</td>
<td>596</td>
<td>649</td>
</tr>
</tbody>
</table>

Table III: Number of completed benchmarks within 3600s, for 10 and 20 round settings

<table>
<thead>
<tr>
<th>Statistic</th>
<th>WAPS KC size</th>
<th>INC KC size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>18.92</td>
<td></td>
</tr>
<tr>
<td>Std</td>
<td>81.19</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>4.64</td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>1734.08</td>
<td></td>
</tr>
</tbody>
</table>

Table IV: Statistics for number of nodes in d-DNNF (WAPS KC diagram) over that of smoothed PROB (INC KC diagram).

Figure 5 shows the distributions of samples drawn by INC and WAPS for ‘case110’ benchmark. A point \((x, y)\) on the plot represents \(y\) number of unique solutions that were sampled \(x\) times in the sampling process by the respective samplers. The almost perfect match between the weighted samples drawn by INC and WAPS, coupled with our theoretical analysis in Section IV-D, substantiates our claim INC’s correctness in performing weighted sampling. Additionally, it also shows that INC can be a functional replacement for existing state-of-the-art sampler WAPS, given that both have theoretical guarantees.

Discussion: We demonstrated the runtime performance advantages of INC and the two main contributing factors - a choice of succinct knowledge compilation form and dynamic log-space annotation. INC takes longer than WAPS for single-round sampling, mainly because WAPS takes less time for KC diagram compilation than INC, leading to WAPS being faster in single-round sampling. In the incremental sampling setting, the compilation costs of KC diagrams are amortized, and since INC is substantially better at handling incremental updates, it thus took the overall runtime lead from WAPS.
in the majority of the benchmarks. Extrapolating the trend, it is most likely that INC would have a larger runtime lead over WAPS for applications requiring more than 10 sampling rounds. The runtime breakdown demonstrates that INC is able to amortize the compilation time over the incremental sampling rounds, with subsequent rounds being much faster than WAPS. In summary, we show that INC is substantially better at incremental sampling than existing state-of-the-art.

VI. CONCLUSION AND FUTURE WORK

In conclusion, we introduced a bottom-up weighted sampler, INC, that is optimized for incremental weighted sampling. By exploiting the succinct structure of PROB and log-space computations, INC demonstrated superior runtime performance in a series of extensive benchmarks when compared to the current state-of-the-art weighted sampler WAPS. The improved runtime performance, coupled with correctness guarantees, makes a strong case for the wide adoption of INC in future applications.

For future work, a natural step would be to seek further runtime improvements for PROB compilation since INC takes longer than SOTA for the initial sampling round, due to slower compilation. Another extension would be to investigate the design of a partial annotation algorithm to reduce computations when only a small portion of the weights have been updated. It would also be of interest if we could store partial sampled assignments at each node as a succinct sketch to reduce memory footprint, for instance we could store each unique assignment and its count.

ACKNOWLEDGEMENT

We sincerely thank Yong Lai for the insightful discussions. Suwei Yang is supported by the Grab-NUS AI Lab, a joint collaboration between GrabTaxi Holdings Pte. Ltd., National University of Singapore, and the Industrial Postgraduate Program (Grant: S18-1198-IPP-II) funded by the Economic Development Board of Singapore. Kuldeep S. Meel is supported in part by National Research Foundation Singapore under its NRF Fellowship Programme(NRF-NRFFAI1-2019-0004), Ministry of Education Singapore Tier 2 grant (MOE-T2EP20121-0011), and Ministry of Education Singapore Tier 1 Grant (R-252-000-B59-114).

REFERENCES