8 Appendix

8.1 Proofs for Theoretical Analysis

Lemma 1. *Given a formula F and set of sampling variables P, the tree returned by* PCompile(F, P) *is a d-DNNF* dag *which represents the set of satisfying assignments of the formula F projected over the set of sampling variables P.*

Proof. Consider a d-DNNF tree for the formula F in which decisions are either taken from the set of sampling variables or non sampling variables only when no sampling variables are left. This tree can be constructed by following the compilation routine of component caching and clause learning based d-DNNF compiler DSHARP [43]. Now, consider any path in such a d-DNNF. It corresponds to a satisfying assignment for the formula F. Consider its trace in the d-DNNF till decisions are taken on sampling variables. At this point, moving further deep in the tree can only lead to further decisions or assignments on non-sampling variables. Therefore, if we cut the edge connecting the rest of the tree to d-DNNF on non-sampling variables, we would have a partial satisfying assignment on this path. Further, ignoring non-sampling variables encountered upto this point, we would have the projection of the satisfying assignment on sampling variables. Now, observe that this can be done for all the satisfying assignments on this path, essentially projecting them to a unique partial satisfying assignment. Now, we claim that d-DNNF on non-sampling variables would only be present if the residual formula is satisfiable. We know that d-DNNF for a formula F can only be constructed if and only if there exists a satisfying assignment. Therefore, checking for satisfiability, has same effect as that of cutting the edges to d-DNNF on non-sampling variables. If the residual formula is found as unsatisfiable, then it would simply return as it does in DSHARP without existence of d-DNNF on non-sampling variables.

Lemma 2. *Every node t in the d-DNNF* dag *returned by* WAnnotate *is annotated by $W(T(t))$, where $T(t)$ is the set of all the partial assignments corresponding to the subtree rooted at t.*

Proof. The proof proceeds by induction on construction of d-DNNF dag. The base case is trivial as for d-DNNF dag of size 1, i.e. literal, the weighted model count for this partial assignment is its weight. The base cases holds trivially true at all the leaves. Since the internal nodes in d-DNNF are only constituted by AND and OR nodes, by proving that weighted model count holds at each internal node, we effectively prove that we get weighted model count at the root of the d-DNNF. The induction holds at each AND and OR node as explained below:

OR: The property of determinism implies that the OR node accumulates mutually disjoint solutions from its children. Therefore the total weights of the partially satisfying assignments at the OR node can be given by the sum of weights of both of its children.

AND: The property of decomposability implies that all the partial assignments of AND node can be given by the concatenating partial assignments of the children of the AND node. The weight of any partial assignment given by AND node will be the product of weights of the corresponding partial assignments given by each of its children. Therefore, the sum of weights of all such satisfying assignments of the AND node can be given by the product of weighted model count of its children which essentially represents the sum of weights of partial satisfying assignments for each of its children.

Theorem 1. *For a given F, P and s,* SampleList *is the list of samples generated by* WAPS*. Let* SampleList$[i]$ *indicate the sample at the i^{th} index of the list. Then for each $y \in R_{F\downarrow P}$, $\forall i \in [s]$, we have*

$$\mathsf{Pr}\left[y = \mathsf{SampleList}[i]\right] = \frac{W(y)}{W(R_{F\downarrow P})}$$

Proof. Lemma 1 shows that the d-DNNF returned by PCompile represents the set of satisfying assignments of the formula F projected over the set of sampling variables P. Lemma 2 shows that d-DNNF returned by WAnnotate is annotated with $W(T(t))$ where $T(t)$ is a set of all the partial assignments given by subtree rooted at corresponding node t. To complete the proof we focus on Subroutine Sampler. The proof proceeds by induction on construction of d-DNNF dag. The base case is trivial as for dag of height one, i.e. literal, there is only one solution, so the base case holds true for all the leaf nodes. Since all the internal nodes of d-DNNF contains only AND and OR nodes we complete the induction step by proving both cases as follows:

OR: The property of determinism ensures disjoint solutions; hence, it boils down to the problem of selecting a model uniformly at random from two buckets (children nodes) having weighted model count w_l and w_r. To do the same, Bernoulli trail is performed with probability $p = \frac{w_l}{w_l + w_r}$, and partial assignment for solution is taken from respective bucket. In order to draw s samples, s Bernoulli trials are simulated by Binomial distribution by given weights and randomly permuting the union of the solution lists returned from all the children.

AND: The property of decomposability allows us to construct a sample selected uniformly according to weights by sampling a solution segment from each of its children and appending them.

Lemma 3. *For a given F and P, let* $\mathrm{fol}(\rho)$ *be the event of following a path ρ and $N(\rho)$ be a function that returns the end node of the path ρ, then*

$$\mathsf{Pr}\left[\mathrm{fol}(\rho)\right] = \frac{W(T(N(\rho))) \times c_\rho}{W(R_{F\downarrow P})}$$

where c_ρ is the product of weight of all the OR nodes' siblings encountered in the path ρ from root to $N(\rho)$ and $T(N(\rho))$ is the set of all the partial satisfying assignments represented by subtree rooted at t.

Proof. To argue about the probability of visiting a node, we can argue about the probability of going through a path from root to that node. Without loss of generality we can assume that the root node of d-DNNF is an OR node. Let $\rho = (O_1 \rightarrow A_1 \rightarrow \cdots t_{2n})$ be a particular path from root node to t_{2n} where $t_{2n} = A_n$ and $\rho_i = (O_1 \rightarrow A_1 \rightarrow \cdots t_i)$. Let $N(\rho)$ be a function that returns the end node of the path ρ. For any node t, let $type(t)$ be a function which returns the type of node t which can be AND or OR. Now, let $\mathsf{Pr}\left[\text{fol}(\rho_i)\right]$ be the probability of the event of following the path ρ_i, then:

$$\mathsf{Pr}\left[\text{fol}(\rho_i) \mid \text{fol}(\rho_{i-1})\right] = \begin{cases} 1, & \text{if } type(N(\rho_{i-1})) = \text{AND} \\ \frac{W(N(\rho_i))}{W(N(\rho_{i-1}))}, & \text{if } type(N(\rho_{i-1})) = \text{OR} \end{cases}$$

Note that $N(\rho_{2n}) = A_n,\ N(\rho_{2n-1}) = O_n,\ N(\rho_{2n-2}) = A_{n-1}$ and $N(\rho_{2n-3}) = O_{n-1}$. Therefore,

$$\begin{aligned} \mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_{2n-2})\right] &= \mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_{2n-1})\right] \times \mathsf{Pr}\left[\text{fol}(\rho_{2n-1}) \mid \text{fol}(\rho_{2n-2})\right] \\ &= \frac{W(N(\rho_{2n}))}{W(N(\rho_{2n-1}))} \times 1 \end{aligned}$$

$$\begin{aligned} \mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_{2n-3})\right] &= \mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_{2n-2})\right] \times \mathsf{Pr}\left[\text{fol}(\rho_{2n-2}) \mid \text{fol}(\rho_{2n-3})\right] \\ &= \frac{W(N(\rho_{2n}))}{W(N(\rho_{2n-1}))} \times 1 \times \frac{W(N(\rho_{2n-2}))}{W(N(\rho_{2n-3}))} \\ &= \frac{W(A_n)}{W(O_n)} \times 1 \times \frac{W(A_{n-1})}{W(O_{n-1})} \end{aligned}$$

Now,

$$W(A_{n-1}) = W(O_n) \times \prod_{i \in Siblings(O_n)} W(i)$$

Using the above results,

$$\begin{aligned} \mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_{2n-3})\right] &= \frac{W(A_n)}{W(O_n)} \times \frac{W(O_n) \times \prod_{i \in Siblings(O_n)} W(i)}{W(O_{n-1})} \\ &= W(A_n) \times \frac{\prod_{i \in Siblings(O_n)} W(i)}{W(O_{n-1})} \end{aligned}$$

Now, applying the above method recursively, we get

$$\mathsf{Pr}\left[\text{fol}(\rho_{2n}) \mid \text{fol}(\rho_1)\right] = \frac{W(A_n) \times \prod_{i \in [1,n]} \prod_{j \in Siblings(O_i)} W(j)}{W(R_{F \downarrow P})}$$

Since, we always visit the root node to fetch any samples, $\mathsf{Pr}\left[fol(\rho_1)\right] = 1$. Therefore,

$$\begin{aligned}\mathsf{Pr}\left[\text{fol}(\rho_{2n})\right] &= \frac{W(A_n) \times \prod_{i\in[1,n]} \prod_{j\in Siblings(O_i)} W(j)}{W(R_{F\downarrow P})} \\ &= \frac{W(T(t)) \times c}{W(R_{F\downarrow P})}\end{aligned}$$

where $c = \prod_{i\in[1,n]} \prod_{j\in Siblings(O_i)} W(j)$ i.e the product of weight of all the OR nodes' siblings encountered in a path from root to t_{2n}.

Algorithm 5 Top Down pass for sampling s samples

```
1: function Sampler(t, s)
2:     SampleList ← ∅
3:     if label(t) = OR then
4:         if Weight(t) = 0 then
5:             p ← 0
6:         else
7:             p ← Weight(t.left) / Weight(t)
8:         b ~ Binomial(s, p)
9:         SampleList_0 ← Sampler(t.left, b)
10:        SampleList_1 ← Sampler(t.right, s − b)
11:        SampleList ← Shuffle(SampleList_0 ∪ SampleList_1)
12:        return SampleList
13:    else if label(t) = AND then
14:        for c ∈ children(t) do
15:            SampleList_c ← Sampler(c, s)
16:            SampleList ← Stitch(SampleList, SampleList_c)
17:        return SampleList
18:    else                                          ▷ label(t) == Literal
19:        for i = 1 to s do
20:            Append(val(t), SampleList)
21:        return SampleList
```

8.2 Sampling

Algorithm 5 takes the annotated d-DNNF dag and the required number of samples s and returns a list L of s samples conforming to the distribution of their weights as governed by weight function $W(\cdot)$ given to WAnnotate. The Algorithm for Sampling is very similar to the sampling procedure in our previous work [49] except that we take the annotated weight of the node instead of annotated count in the previous work as the probability for Bernoulli trials. The description of Sampler procedure as per Algorithm 5 is given as follows:

OR (lines 4–8) : We perform s Bernoulli trials with probability p proportional to the weights of each of the OR node's children to determine the number of samples to draw from them following the weighted distribution. We then recursively draw samples from children as per the overall result of trials simulated via Binomial distribution and shuffle the drawn samples to ensure that the samples are randomly permuted.

AND (lines 13–17) : Samples are recursively drawn from each of the children of AND node and then concatenated with each other using the subroutine Stitch(line 16)

Literal (lines 18–21) : We sample the literal s times for each sample. Function $val(t)$ gives us the literal for literal node.

8.3 Example of projected compilation

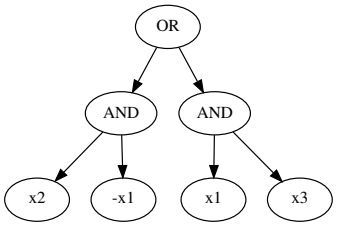

Fig. 6: The projected d-DNNF compilation of example 2

We illustrate PCompile procedure on following example formula F:

Example 2. $F = \{\{x_1, x_2\}, \{\neg x_3, \neg x_5, x_6\}, \{\neg x_2, x_4, \neg x_1\}, \{x_3, \neg x_6, \neg x_1\}, \{x_6, x_5, \neg x_1, x_3\}, \{x_3, x_6, \neg x_5, \neg x_1\}\}$

Let $P = \{x_1, x_2, x_3\}$ be the set of sampling variables and $Vars(F) = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ be the set of all variables in the formula F. We represent F in clausal form, following steps are then performed for compilation of formula F to a d-DNNF on P:

1. We decide on x_1 which splits the problem into two branches. $C_1 = \{\{x_2\}, \{\neg x_3, \neg x_5, x_6\}\}$, $C_2 = \{\{\neg x_3, \neg x_5, x_6\}, \{\neg x_2, x_4\}, \{x_3, \neg x_6\}, \{x_6, x_5, x_3\}, \{x_3, x_6, \neg x_5\}\}$ as per the positive and negative assignment to x_1 respectively.
2. Then, while solving C_1, x_2 gets implied by unit propagation and we represent it as a child of AND node. Then we make a decision on x_3 to split the formula into two components. Since both of them turn out to be satisfiable, it is implied that both the assignments of x_3 are possible in this configuration. Hence, we don't have to construct a node for x_3.

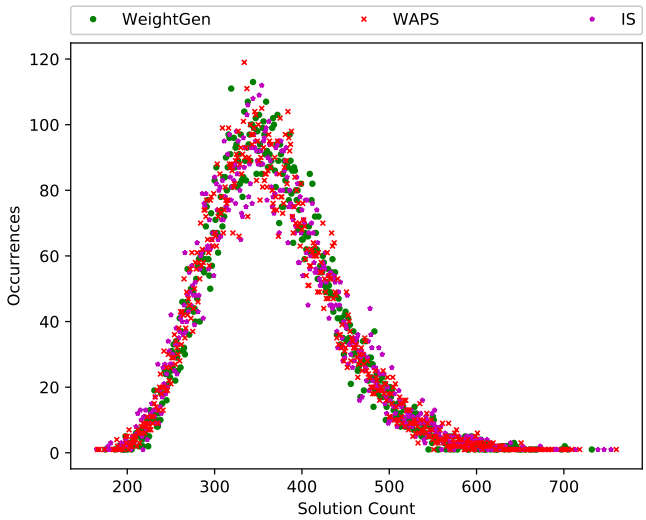

Fig. 7: Distribution comparison between IS, WAPS and WeightGen on case110 with 6×10^6 samples.

3. Now, to solve C_2, we make decision on x_3 and find that the component generated on giving negative assignment to x_3 is unsatisfiable. Thus implying positive assignment for x_3 which is represented by child of AND node. This results in the residual formula $C_{2,2} = \{\{\neg x_2, x_4\}, \{\neg x_5, x_6\}\}$
4. For solving $C_{2,2}$, we decide on x_2 to split the formula into two components that turn out to be satisfiable, thus implying that both the assignments of x_2 are possible in this configuration. Hence, we don't have to construct a node for x_2.

Figure 6 represents the d-DNNF generated of example 2 on $P = \{x_1, x_2, x_3\}$.

8.4 Distribution comparison

To measure the accuracy of WAPS, we implemented an *ideal weighted and projected sampler* (similar to [10]), henceforth called IS: given a CNF formula F, IS first generates all the satisfying assignments, then compute their weights and uses these weights to sample the ideal distribution. We generated a large number N (6×10^6) of samples using WAPS, WeightGen and IS. In each case, the number of times various samples were generated was recorded, giving a distribution of the counts. Figure 7 shows the distributions generated by WAPS, WeightGen and IS for one of benchmark (case110) which has $16,384$ solutions. In the figure, each point (x, y) indicates that x distinct models are generated y times. Since the number of samples is much larger than the number of models, each model occurred multiple times in the list of samples. We computed the frequency of

Table 4: Runtime (in sec.) of WAPS to generate samples while conditioning on arbitrary y% variables

Benchmark	Vars	Clauses	WAPS(Compilation and A+S on y% conditioned vars)					
			Compile	$y=1$	$y=2$	$y=3$	$y=4$	$y=5$
s27_new_15_7	17	43	0.01	0.29	0.27	NS	0.28	NS
or-50-20-2	100	250	6.77	112.32	111.56	109.17	111.01	110.63
or-50-5-1	100	250	8.91	154.21	153.62	154.86	153.47	152.91
or-50-20-3	100	250	13.47	184.78	187.13	185.68	185.59	185.02
or-50-10-6	100	250	15.36	342.58	339.97	341.09	339.52	337.56
or-50-20-7	100	250	23.93	384.90	381.29	383.07	380.76	380.94

generation of individual model and grouped the models that had the same frequency. Figure 7 shows that the distribution generated by WAPS is practically indistinguishable from IS.

8.5 Effect of conditioning on variables

We evaluated the performance of WAPS for sampling when the formula is constrained by conditions, i.e. partial assignments on some of its variables apart from the usual CNF clauses. In particular, we experimented by randomly choosing 1%, 2%, 3%, 4% and 5% of all variables 10 times and assigned arbitrary signs to chosen variables. This resulted in 50 instances for each problem with different conditions and we sampled 1000 samples from each instance. For each instance, let the set of conditions be represented by a set of literals L. We observed an improvement in average runtime as more and more variables get constrained since, the paths which lead to assignments not conforming to the given condition get pruned in the annotation phase. This happens, as some nodes in d-DNNF are attributed with 0 weight. This occurs when all the partial assignments represented by the subtree rooted at a node in d-DNNF becomes inconsistent with the added conditions. Table 4 represent the results of conditioned sampling on representative set of benchmarks. The first column represents the benchmark name. Second, third and fourth columns represent the number of variables, clauses and d-DNNF compilation time. The next five columns represent the time taken by WAPS when the formula from benchmark is conditioned on 1%, 2%, 3%, 4% and 5% of all variables. We report the average time over 10 runs for each case. The acronym NS for y% conditioned variables means that no samples could be drawn for atleast one such instance as it became unsatisfiable by the additional conditions.