Counting Problems and the Inclusion-Exclusion Principle.[☆]

Sriram Sankaranarayanan[a], Huxley Bennett[a]

[a]*Department of Computer Science, University of Colorado, Boulder, CO.*
firstname.lastname@colorado.edu

Abstract

In this paper, we present improved techniques for computing and getting bounds on the cardinality of a union of sets using the inclusion-exclusion principle and Bonferroni inequalities. We organize the terms involved in the inclusion-exclusion sum as a tree, showing that a set inclusion between a parent and its children yields a cancellation, where we may prune an entire subtree. Next, we provide a straightforward extension to the standard Bonferroni inequalities where we obtain upper and lower bounds by pruning the tree of terms at arbitrary odd and even depths, respectively. We conclude by showing how our work can be applied to the problem of counting the number of solutions to a given propositional SAT formula.

1. Introduction

The inclusion-exclusion principle gives a formula for computing the cardinality of the union of a collection of sets: $|\cup_{i=1}^{n} A_i|$. The formula, expressed as an alternating sum, plays an important role in combinatorics and probability. Bonferroni inequalities generalize the inclusion-exclusion principle by showing that truncactions of the sum at odd (even) depths give upper (lower) bounds. The inclusion-exclusion sum includes a term for each subset in the powerset of $\{A_1, \ldots, A_n\}$ and therefore requires exponentially many computations in the worst case.

Work on improving Bonferroni inequalities, either in terms of generalization or reduced computation, is prevelant in the literature [2, 3]. In this paper we obtain a generalization of Bonferroni inequalities, and then give a novel technique for identifying cancellations in the inclusion-exclusion sum, leading to large reductions in computation. Throughout the paper we represent the terms in an inclusion-exclusion sum as a tree, which is useful in giving our results concisely.

[☆]This work was partially supported by the National Science Foundation (NSF) award CNS-1016994.

Our work was motivated by a desire to solve #SAT, the problem of counting the number of models of a propositional formula. Because #SAT belongs to the class of #P-complete problems it is thought to be highly intractible in the worst case. Ever since Valiant introduced counting complexity in his seminal paper [6], determining the best ways to compute exact and approximate solutions to #P-complete problems has intrigued theoreticians. More recently researchers have also been interested in solving these problems because of their applications in artificial intelligence [4] and formal verification.

Gomes et al. provide a comprehensive survey of model counting algorithms and implementations for solving #SAT both exactly and approximately in [4]. These largely build on existing SAT-solving paradigms. Using DPLL is especially prevelant [1], and forms the core of the elementary `CDP` algorithm. As the survey authors point out, both approximate and exact model counters can only handle formulas with several orders of magnitude fewer variables than modern SAT solvers.

To the best of our knowledge, Linial and Nisan in [7] first introduced the idea of using the inclusion-exclusion principle to count the models of a propositional formula, also showing that past depth $O(\sqrt{n})$ an inclusion-exclusion sum converges rapidly. This work was furthered by Iwama [5] and Lonzinskii [8], both of which take advantage of the structure of CNF SAT to analyze the average time complexity required for exactly computing the number of models of a random CNF formula with fixed clause length. However, as [7] points out, the idea of using inclusion-exclusion to solve hard counting problems goes back to [9] in which Ryser uses it in an algorithm for computing the permanent of a matrix before the problem's complexity was formally defined.

2. Inclusion-Exclusion Principle

In this section we present some preliminary ideas on the Inclusion-Exclusion Principle and Bonferroni inequalities.

Given a family $\mathcal{A}$ of $n > 0$ sets $A_1, \ldots, A_n$, the inclusion-exclusion principle provides us with a technique to count the number of elements in their union $A : \bigcup_{i=1}^{n} A_i$:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \textstyle\sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n| \quad .$$

The sum can be written succinctly as

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{S\subseteq[n], S\neq\emptyset} (-1)^{1+|S|} \left|\bigcap_{i\in S} A_i\right| \quad . \tag{1}$$

Given a family $\mathcal{A}$ with n sets, the summation in Equation 1 has 2^n terms.

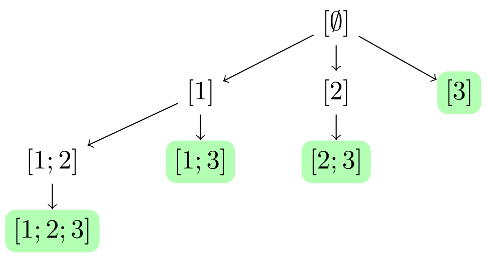

Figure 1: IE tree for a family $\mathcal{A} = \{A_1, A_2, A_3\}$ consisting of three sets. Leaves of the tree are shaded in green.

2.1. Inclusion-Exclusion Tree

We will now present a simple organizational device that views the terms in the inclusion exclusion principle summation from Equation 1 as nodes of a tree. This tree will be called the Inclusion-Exclusion (IE) Tree.

Definition 2.1 (IE Tree)**.** *Given a family $\mathcal{A}$ of $n > 0$ sets $A_1, \ldots, A_n$, the IE tree corresponding to the family has node for each subset $S \subseteq [n]$.*

1. *The root of the IE tree is denoted by $[\emptyset]$. Each non-root node is written as $S :\ [i_1; \cdots ; i_d]$, such that $1 \leq i_1 < i_2 < \cdots < i_d \leq n$.*
2. *For each node $S : [i_1; \cdots ; i_d]$, we have an edge from S to a node of the form $S' : [i_1; \cdots ; i_d; i_{d+1}]$. For convenience we will write S' as $[S; i_{d+1}]$. Note that, by our node naming convention, $i_{d+1} > i_d$.*
3. *For each node $S : [i_1; \cdots ; i_d]$, we assume that its children are ordered from left to right using a* lexicographic ordering *as*

$$S_1 : [S; i_d + 1],\ S_2 : [S; i_d + 2],\ \ldots\ , S_l : [S; n],\ \textit{wherein } l = n - i_d\,.$$

Each node S in the IE tree is associated with the term

$$t(S) = (-1)^{1+|S|} |\cap_{i \in S} A_i|\,,$$

in Eq. (1). By convention, we set $t([\emptyset]) = 0$. For a given non-root node $S \neq \emptyset$, $t(S) = 0$ if and only if $\cap_{i\in S} A_i = \emptyset$. Furthermore, all nodes in the subtree rooted at S will also lead to an empty intersection.

Figure 1 illustrates an example IE tree for a family $\mathcal{A}$ with three sets.

For each node S in the IE tree, let $\mathsf{subtree}(()S)$ denote the subtree rooted at S (including S and all its proper descendents in the tree) and let $\mathsf{subtree}^\dagger(S) = \mathsf{subtree}(S) - \{S\}$. We extend the valuation function to subtrees as the sum over all the nodes in the subtree:

$$t(\mathsf{subtree}(S)) = \sum_{S' \in \mathsf{subtree}(S)} t(S') \ \text{ and } \ t(\mathsf{subtree}^\dagger(S)) = t(\mathsf{subtree}(S)) - t(S)\,.$$

Algorithm 1: Algorithm for computing cardinality using simple IE tree exploration.

```
Data: Family of sets $\mathcal{A} = \{A_1, \ldots, A_n\}$
Result: $|\bigcup_{i=1}^{n} A_i|$
$S \leftarrow \emptyset$ ;
queue $\leftarrow \{S\}$;
sum $\leftarrow 0$;
while not $\mathsf{isEmpty}$(queue) do
    $S \leftarrow \mathsf{head}$(queue) ;
    sum $\leftarrow$ sum $+ t(S)$;
    if $S = \emptyset \vee t(S) > 0$ then
        queue $\leftarrow$ queue $\cup$ $\mathsf{Children}(S)$;
return sum;
```

Given an IE tree, the overall cardinality of the union is given by $t(\mathsf{subtree}([\emptyset]))$. Algorithm 1 shows the basic IE tree exploration for computing the cardinality of a family $\mathcal{A}$. The algorithm uses a queue structure to store the set of unexplored nodes. At each step, a node is taken out of the queue and its contribution added to the summation. Next, the children of the node are added to the queue if it is empty or its contribution to the summation is strictly positive. This step has the effect of pruning nodes and their subtrees whenever the intersection of the corresponding sets is empty.

We will now observe a key observation that arises from the structure of the IE tree. Let $S = [i_1; \ldots; i_d]$ be a non-root node (i.e, $d > 0$) of the IE tree with $k > 0$ children $T_1 : [S; i_d + 1], \ldots, T_k : [S; n]$, where $n = i_d + k$. Recall that the contribution of the proper subtree rooted at S is given by

$$t(\mathsf{subtree}^{\dagger}(S)) = \sum_{j=1}^{k} t(\mathsf{subtree}([S; i_d + j]))$$

Lemma 2.1. *The following identity holds:*

$$(-1)^d t(\mathsf{subtree}^{\dagger}(S)) = \left| \bigcup_{j=i_d+1}^{n} (\mathsf{Intersect}(S) \cap A_j) \right| ,$$

wherein $\mathsf{Intersect}(S) = \bigcap_{i \in S} A_i$.

Proof. Proof follows by application of the inclusion exclusion principle to the term on the RHS of the identity and matching up each resulting term with a node in $\mathsf{subtree}^{\dagger}(S)$. Specifically, each term in the inclusion exclusion sum for the RHS will be of the form

$$(-1^{l+1})|\mathsf{Intersect}(S) \cap A_{j_1} \cap \cdots A_{j_l}|, \text{ werein}, j_1, \ldots, j_l > i_d \,.$$

Such a term will be mapped bijectively onto the term $(-1)^d t([S; A_{j_1}; \ldots; A_{j_l}])$ involved in the LHS summation, wherein $[S; A_{j_1}; \cdots; A_{j_l}] \in \mathsf{subtree}^\dagger(S)$. It remains to verify that the terms themselves are identical:

$$\begin{array}{rcl} (-1)^d t([S; A_{j_1}; \ldots; A_{j_l}]) & = & (-1)^d (-1)^{d+l+1} |\mathsf{Intersect}(S) \cap A_{j_1} \cap \cdots \cap A_{j_l} \\ & = & (-1^{l+1}) |\mathsf{Intersect}(S) \cap A_{j_1} \cap \cdots A_{j_l}| \end{array}$$

Therefore, the identity is verified. □

2.2. Subsumption Pruning

We now augment the basic tree search by a technique to prune based on *subsumed sets*. As observed in [2], many terms in the inclusion-exclusion sum have equal magnitude but opposite sign. Our technique exploits a symmetry that often arises in the inclusion-exclusion sum which allows us to cancel the contribution of an entire subtree.

The idea behind subsumption pruning is based on the following idea:

We always have that $|t(S)| \geq \left|t(\mathsf{subtree}^\dagger(S))\right|$ by Lemma 2.1. When S is contained in the union of the sets introduced by its children we also have $|t(S)| \leq \left|t(\mathsf{subtree}^\dagger(S))\right|$, so $t(S) = -t(\mathsf{subtree}^\dagger(S))$ and $t(\mathsf{subtree}(S)) = 0$.

We now formally define subsumption of a node by its children and then prove the main result of this section.

Definition 2.2 (Subsumption)**.** *We say that a node S is* subsumed *by its children*
Children$(S) = \{T_1 : [S; A_{i_1}], \ldots, T_k : [S; A_{i_k}]\}$ *if*

$$\mathsf{Intersect}(S) \subseteq \bigcup_{j \in \{i_1, \ldots, i_k\}} A_j \qquad (2)$$

Remark 2.1. *A simple instance of subsumption occurs when for a child T_j of a node S, we have $\mathsf{Intersect}(S) = \mathsf{Intersect}(T_j)$.*

We now prove the main result that shows that if subsumption occurs at a node S then the contribution of the entire subtree rooted at S vanishes.

Theorem 2.1. *Let S be a node in the IE tree that is subsumed by its children. It follows that $t(\mathsf{subtree}(S)) = 0$.*

Proof. By Lemma 2.1 we have that

$$(-1)^{|S|} t(\mathsf{subtree}^\dagger(S)) = \left| \bigcup_{T_j \in \mathsf{Children}(S)} \mathsf{Intersect}(T_j) \right|$$

Combining this with the definition of subsumption and noting that S has the opposite sign of $t(\mathsf{subtree}^\dagger(S))$ we have that

$$\begin{array}{rcl} t(\mathsf{subtree}(S)) & = & t(S) + t(\mathsf{subtree}^\dagger(S)) \\ & = & (-1)^{|S|} \left|\mathsf{Intersect}(S)\right| \ + \ (-1)^{1+|S|} \left| \bigcup_{j=i_d+1}^{n} (\mathsf{Intersect}(S) \cap A_j) \right| \\ & = & (-1)^{|S|} \left|\mathsf{Intersect}(S)\right| \ + \ (-1)^{1+|S|} \left| \mathsf{Intersect}(S) \cap \bigcup_{j=i_d+1}^{n} (A_j) \right| \\ & = & (-1)^{|S|} \left|\mathsf{Intersect}(S)\right| \ + \ (-1)^{1+|S|} \left|\mathsf{Intersect}(S)\right| \\ & = & 0 \end{array}$$

Algorithm 2: Improved algorithm for computing cardinality using IE tree exploration and subsumption pruning.

```
Data: Family of sets $\mathcal{A} = \{A_1, \ldots, A_n\}$
Result: $|\bigcup_{i=1}^{n} A_i|$
$S \leftarrow \emptyset$ ;
queue $\leftarrow \{S\}$;
sum $\leftarrow 0$;
while not isEmpty(queue) do
    $S \leftarrow$ head(queue) ;
    if not checkSubsumption($S$) then
        sum $\leftarrow$ sum $+ t(S)$;
        if $S = \emptyset \vee t(S) > 0$ then
            queue $\leftarrow$ queue $\cup$ Children($S$);
return sum;
```

□

The idea of subsumption can be used to potentially improve the running time of the IE tree exploration algorithm (Algorithm 1) by checking if a subsumption occurs at each node and backtracking whenever subsumption is encountered.

Algorithm 2 shows the use of a subsumption check to limit the number of nodes explored during a search. For each node $S : [i_1; \cdots; i_d]$, the function checkSubsumption(S) checks that $\bigcap_{i \in S} A_i \subseteq \bigcup_{i_d < j \leq n} A_j$.

3. Bonferroni Inequalities

In this section, we generalize Bonferroni inequalities using the IE tree to yield upper and lower bounds on the cardinalities of the union of a family of sets.

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a family of sets. For convenience, let B_k denote the summation in Equation (1) restricted to subsets of size at most k:

$$B_k = \sum_{S \subseteq [n], 1 \leq |S| \leq k} (-1)^{1+|S|} \left| \bigcap_{i \in S} A_i \right| \tag{3}$$

The Bonferroni inequalities state that

$$\left| \bigcup_{i=1}^{n} A_i \right| \leq B_{2l+1}, \text{ and } \left| \bigcup_{i=1}^{n} A_i \right| \geq B_{2l}, \quad 1 \leq l \leq \left\lfloor \frac{n}{2} \right\rfloor . \tag{4}$$

In other words, the truncated summation B_k for odd values of k yields an upper bound to the summation whereas B_k for even values of k yields a lower bound.

In general, the Bonferroni inequalities may be interpreted in the light of the IE tree search framework as follows:

1. For a given depth cutoff $1 \leq d \leq n$, let $\mathbb{T}_d$ denote the set of IE tree obtained by removing all nodes of the full IE tree of depth strictly greater than d.

2. If the cutoff depth d is odd then the sum of terms in the truncated tree form an upper bound of the cardinality of the union
$$\sum_{S \in \mathbb{T}_d} t(S) \;\geq\; \left| \bigcup_{i=1}^{n} A_i \right| .$$

3. Similarly for even cutoff depths, the sum of terms in the truncated tree forms a lower bound:
$$\sum_{S \in \mathbb{T}_d} t(S) \;\leq\; \left| \bigcup_{i=1}^{n} A_i \right| .$$

Note that all the cutoffs have to occur at the same depth d. Using the ideas developed so far, we extend the standard Bonferroni bounds by allowing entire subtrees $\mathsf{subtree}^{\dagger}(S)$ to be cut off at different depths, provided all the cutoffs are either at odd depths, or all the cutoffs are at even depths.

Definition 3.1 (Cutoff IE Tree)**.** *An odd (even) depth cutoff IE tree $\mathbb{T}$ is obtained by selecting a set of non-root nodes $\{S_1, \ldots, S_K\}$ such that*

1. *All the nodes S_i, $1 \leq i \leq K$, occur at odd (even) depths.*

2. *The tree $\mathbb{T}$ is obtained by removing the nodes in the subtrees $\mathsf{subtree}^{\dagger}(S_i)$ from the original IE tree.*

Figure 2 illustrates three different cutoff trees for the IE tree from Figure 1. Note that the trees (A) and (B) in the figure yield bounds corresponding the the classic Bonferroni inequalities, whereas the tree (C) has one node at depth 1 whose children are cut off whereas other nodes at the same depth retain their children.

Let us define $t(\mathbb{T})$ for a cutoff IE tree as the sum over all the terms that occur in the tree, i.e, $t(\mathbb{T}) = \sum_{S \in \mathbb{T}} t(S)$.

Theorem 3.1. *The following hold for cutoff trees:*

1. *If $\mathbb{T}$ is an odd depth cutoff tree then $t(\mathbb{T}) \geq |\bigcup_{i=1}^{n} A_i|$.*

2. *If $\mathbb{T}$ is an even depth cutoff tree then $t(\mathbb{T}) \leq |\bigcup_{i=1}^{n} A_i|$.*

Proof. For any node S in the full IE tree that occurs at depth $d > 0$, we observe by Lemma 2.1 that
$$t(\mathsf{subtree}^{\dagger}(S)) = (-1)^d \, |C_s| \ ,$$
for some set C_s. If the depth d is odd then $t(\mathsf{subtree}^{\dagger}(S)) \leq 0$ and likewise, if the depth d is even, then $t(\mathsf{subtree}^{\dagger}(S)) \geq 0$.

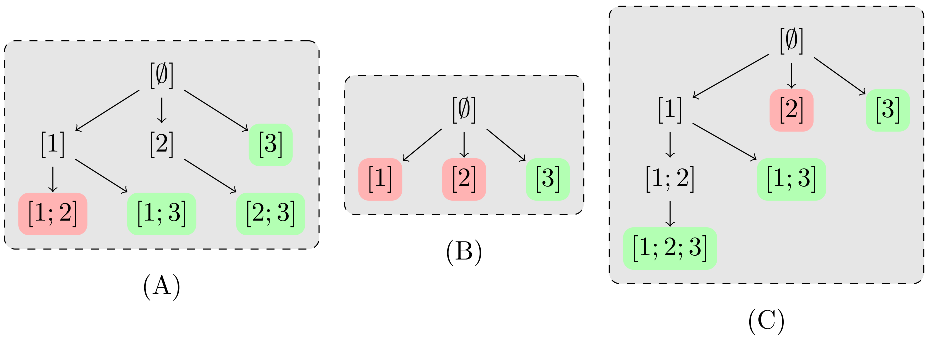

Figure 2: Three examples of depth cutoff trees for the IE tree from Figure 1. Leaves of the tree are shaded green and nodes whose children are cutoff are shaded red. (A) Subtree of node $[1; 2]$ at depth 2 is cut off, yielding a lower bound to the overall summation, (B) Subtrees of nodes $[1], [2]$ at depths 1 are cutoff yielding an upper bound and (C) Subtree of node $[2]$ at depth 1 is cut off yielding an upper bound. Notice that (A) and (B) are legal cutoff trees using established Bonferroni inequalities, while (C) requires our generalization.

Let $\mathbb{T}$ be an odd depth cutoff tree where cutoffs are carried out at nodes $S_1, \ldots, S_K$. Without loss of generality, we may assume that no S_i in the set is an ancestor or a descendent of some other node S_j. Using the inclusion-exclusion principle, we observe that

$$t(\mathbb{T}) = \left| \bigcup_{i=1}^{n} A_i \right| - \sum_{i=1}^{K} t(\mathsf{subtree}^{\dagger}(S_i)) \,.$$

However, since each node S_i occurs at an odd depth, we have $\sum_{i=1}^{K} t(\mathsf{subtree}^{\dagger}(S_i)) \leq 0$. Therefore, we conclude that

$$t(\mathbb{T}) \geq \left| \bigcup_{i=1}^{n} A_i \right| \,.$$

The proof for an even depth cutoff tree is very similar. □

Remark 3.1. *The odd (even) depth cutoff trees induced by subsumption give upper (lower) bounds.*

4. Application: Propositional Model Counting

We now present an algorithm for the #SAT problem using inclusion-exclusion principle, following Iwama [5] and Lozinskii [8]. The #SAT problem seeks the number of satisfying solutions to a given Boolean formula in the conjunctive normal form (CNF).

Let φ be a k-CNF formula consisting of variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. Each clause $\mathsf{lits}(C_i) : \{\ell_{(1,k)}, \ldots, \ell_{(i,k)}\}$ is a set of k literals, each

Algorithm 3: Basic algorithm for #UNSAT using inclusion-exclusion.

```
Input: $\varphi$: CNF SAT formula with variables $x_1, \ldots, x_n$ and clauses
       $C_1, \ldots, C_m$
Output: $\#\textsc{unsat}(\varphi)$
begin
    count := 0
    for (each subset $S \subseteq \{C_1, \ldots, C_m\}, S \neq \emptyset$) do
        Compute $\mathsf{lits}(S)$
        if ($\mathsf{lits}(S)$ has no interfering literals) then
            count := count + $(-1)^{1+|S|} 2^{n-|\mathsf{lits} S|}$
end
```

literal of the form $\ell_i : x_j$ or $\ell_i : \neg x_j$. Using the inclusion-exclusion principle, we count the number $N_U : \#\textsc{unsat}(\varphi)$ of solutions that do not satisfy φ. Given N_U, we may compute the number of satisfying solutions as $2^n - N_U$. We now present the basic approach for expressing the count of the number of satisfying solutions to φ using the inclusion-exclusion principle.

Let A_i denote the set of all valuations to $x_1, \ldots, x_n$ that do not satisfy the clause C_i. If C_i has k distinct literals, we note that $|A_i| = 2^{n-k}$. This is because, there is exactly one (partial) truth assignment to the literals in C_i that does not satisfy C_i. Such a partial truth assignment fixes the value of k variables in the original formula, allowing $n-k$ variables to be assigned arbitrarily, giving a total of 2^{n-k} such assignments. Therefore, the number of dis-satisfying solutions for C_i is given by 2^{n-k} assuming that all literals in C_i are distinct.

Similarly, consider a subset $S \subseteq \{C_1, \ldots, C_m\}$ of the clauses appearing in the formula φ. Let $\mathsf{lits}(S)$ be the union of all the literals that appear in each of the clauses in S, $\mathsf{lits}(S) = \bigcup_{C_j \in S} \mathsf{lits}(C_j)$.

Definition 4.1 (Interfering Clauses)**.** *Given two clauses C_i, C_j in φ, we say that C_i* interferes *with C_j iff C_i contains a literal ℓ and C_j contains its negation $\neg\ell$. A set of clauses $S \subseteq \{C_1, \ldots, C_m\}$ is interfering iff there are two interfering clauses in S.*

Using inclusion-exclusion bounds to compute the number of dis-satisfying solutions for a given CNF SAT formula φ.

$$N_U = \sum_{S \subseteq \{C_1, \ldots, C_m\}} t(S) \quad \text{where} \quad t(S) = \left\{ \begin{array}{ll} 0 & \text{if } \exists\, j,\ \{x_j, \neg x_j\} \subseteq \mathsf{lits}(S) \\ \left((-1)^{|S|+1} \cdot 2^{n-|\mathsf{lits}(S)|}\right) & \text{otherwise} \end{array} \right. ,$$

Likewise, using Bonferroni inequalities, it is possible to estimate upper and lower bounds for the number of dis-satisfying solutions.

Algorithm 3 shows the basic algorithm for #SAT. We note that since S must range over $\sum_{i=1}^{m} \binom{m}{i} = 2^m - 1$ clause sets that the running time of the algorithm is exponential in m. By limiting the summation in Algorithm 3 to clause sets of size up to B, we may obtain an upper bound or a lower bound depending on whether B is odd or even.

Subsumption Checking We now present the idea of subsumption checking to prune away further terms from consideration in Algorithm 3.

Lemma 4.1. *The improved #UNSAT algorithm has worst case $O(n)$ running time.*

Notice that the naive algorithm 3 requires worst case $O(m)$ time.

Proof. Because φ has only n variables any sequence of $n+1$ clauses must $\square$

Proof. Suppose we have a clause-set S with non-interfering literals $\mathsf{lits}(S)$. By subsumption and the conflict rule if adding a new clause C to S introduces no new literals, or if $\mathsf{lits}(C \cup S)$ is conflicting then we prune $\mathsf{subtree}(()C)$. Then because φ has only n variables adding an $n+1$th clause to S will induce either a subsumption or conflict.

If adding a new clause C to S introduces no new literals or C is subsumed or $\mathsf{lits}(C \cup S)$ is conflicting then we terminate.

If a new clause introduces no new literals then it is subsumed. Furthermore, if a clause contains an interfering literal then it is pruned. However a set of non-interfering literals is of size at most n, so its

$\square$

References

[1] E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *J. Artificial Intelligence Research*, 10:457–477, 1999.

[2] K. Dohmen. *Improved Bonferroni Inequalities via Abstract Tubes: Inequalities and identities of the Inclusion-Exclusion Type.* Lecture Notes in Mathematics. Springer–Verlag, 2003.

[3] J. Galambos. Bonferroni inequalities. *The Annals of Probability*, 5(4):pp. 577–581, 1977.

[4] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*, chapter 20. IOS Press, 2008.

[5] K. Iwama. CNF-satisfiability test by counting and polynomial average time. *SIAM Journal on Computing*, 18(2):385–391, 1989.

[6] L.G. and Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189 – 201, 1979.

[7] N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

[8] E. L. Lozinskii. Counting propositional models. *Information Processing Letters*, 41:327–332, 1992.

[9] H. Ryser. *Combinatorial mathematics*. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley New York, 1963.