JOURNAL OF ALGORITHMS 10, 429–448 (1989)

Monte-Carlo Approximation Algorithms for Enumeration Problems

RICHARD M. KARP*

Computer Science Division,
University of California, Berkeley, California 94720

MICHAEL LUBY†

Computer Science Department,
University of Toronto, Toronto, Ontario, Canada M5S 1A4

AND

NEAL MADRAS‡

Department of Mathematics,
York University, North York, Ontario, Canada

Received August 4, 1987; revised May 13, 1988

We develop polynomial time Monte-Carlo algorithms which produce good approximate solutions to enumeration problems for which it is known that the computation of the exact solution is very hard. We start by developing a Monte-Carlo approximation algorithm for the DNF counting problem, which is the problem of counting the number of satisfying truth assignments to a formula in disjunctive normal form. The input to the algorithm is the formula and two parameters ϵ and δ. The algorithm produces an estimate which is between $1 - \epsilon$ and $1 + \epsilon$ times the number of satisfying truth assignments with probability at least $1 - \delta$. The running time of the algorithm is linear in the length of the formula times $1/\epsilon^2$ times $\ln(1/\delta)$. On the other hand, the problem of computing the exact answer for the DNF counting problem is known to be #P-complete, which implies that there is no polynomial time algorithm for the exact solution if P $\neq$ NP. This paper improves and gives new applications of some of the work previously reported. Variants of an ϵ, δ approximation algorithm for the DNF counting problem have been highly tailored to be especially efficient for the network reliability problems to which they are applied. In this paper the emphasis is on the development and analysis of a much more efficient ϵ, δ approximation algorithm for the DNF counting problem.

*Research partially supported by the International Computer Science Institute, Berkeley, California, 94704 and NSF grant DCR-8411954.

†Research partially supported by the International Computer Science Institute, Berkeley, California, 94704 and NSERC of Canada grant A8092.

‡Research partially supported by a NSERC of Canada grant.

0196-6774/89 $3.00

The running time of the algorithm presented here substantially improves the running time of versions of this algorithm given previously. We give a new application of the algorithm to a problem which is relevant to physical chemistry and statistical physics. The resulting ε, δ approximation algorithm is substantially faster than the fastest known deterministic solution for the problem.

1. Introduction

A classic example of an enumeration problem is the DNF counting problem. The input is a propositional formula F, which is the disjunction of m clauses $C_1, \ldots, C_m$, where each clause is the conjunction of a subset of literals defined with respect to n boolean variables $X_1, \ldots, X_n$. The output is the number of truth assignments over the n variables which satisfy the formula F; this quantity is denoted by $\#F$. The DNF counting problem is a #P-complete problem [Va], which implies that if NP $\neq$ P then there is no polynomial time algorithm for the DNF counting problem.

We consider the problem of approximating $\#F$. An ε, δ approximation algorithm for the DNF problem is a Monte-Carlo algorithm which on every input formula F, $\varepsilon > 0$, $\delta > 0$, outputs a number $\tilde{Y}$ such that

$$\Pr\left[(1-\varepsilon)\#F \leq \tilde{Y} \leq (1+\varepsilon)\#F\right] \geq 1-\delta.$$

The goal is to design an approximation algorithm for the DNF counting problem which runs in time polynomial in the length of F.

The first few algorithms we develop can be thought of as variants of the following abstract Monte-Carlo algorithm. We have a finite set of known size U and an efficient method for randomly choosing elements from U such that each $u \in U$ is equally likely to be chosen. We have a function f defined on U such that $f(u)$ is either 0 or 1 at each $u \in U$ and we have an efficient method for computing $f(u)$ given u. Our goal is to estimate $|G|$, where G is the subset of U at which the function f takes on the value 1. One trial of the Monte-Carlo algorithm consists of randomly choosing $u \in U$, computing $f(u)$, and setting $Y \leftarrow |U| \cdot f(u)$. The Monte-Carlo algorithm consists of running N trials, where Y_i is the value of Y from the ith trial. The output of the algorithm is $\tilde{Y} \leftarrow \sum_{i=1}^{N} Y_i/N$.

It is easy to verify that $E[Y] = |G|$, and consequently $E[\tilde{Y}] = |G|$. The question we address here is how large does N have to be to guarantee that the Monte-Carlo algorithm is an ε, δ approximation algorithm. The following theorem provides the answer.

Zero–One Estimator Theorem. *Let $\mu = |G|/|U|$. Let $\varepsilon \leq 2$. If $N \geq (1/\mu) \cdot (4\ln(2/\delta)/\varepsilon^2)$ then the Monte-Carlo algorithm described above is an ε, δ approximation algorithm.*

Proof. Follows using standard techniques from an inequality due to Bernstein cited in Rényi [Re]. A proof is supplied in the Appendix. □

The theorem provides an upper bound on the number of trials sufficient to guarantee an ε, δ approximation algorithm. It is not hard to prove that this upper bound is within a constant multiplicative factor of the lower bound on the number of trials necessary for the Monte-Carlo algorithm to be an ε, δ approximation algorithm. To obtain an ε, δ approximation algorithm for the DNF counting problem which runs in polynomial time it is sufficient to design the trial of the Monte-Carlo algorithm to have the following properties. The input to the algorithm is a DNF formula F. Polynomial means polynomial in the length of F.

1. In polynomial time we can compute $|U|$ and we can randomly choose members from U with the uniform probability distribution.

2. The function f is computable in polynomial time.

3. We can compute in polynomial time an upper bound B on $|U|/|G|$ such that the value of B is polynomial.

If we can achieve these goals, then the polynomial time ε, δ approximation algorithm consists of computing B and running $N = B \cdot 4\ln(2/\delta)/\varepsilon^2$ trials.

2. A Naive Monte-Carlo Algorithm

In this section we present a simple but ineffective algorithm for estimating $\#F$. The reason for presenting this algorithm is to further emphasize the importance of designing the trial carefully. Let U be the set of all possible truth assignments for the n variables. Let f be the function which evaluates to 1 on the set of truth assignments which satisfy F. Thus, G is the set of truth assignments which satisfy F and the quantity we are trying to estimate is $|G|$. It is easy to verify that the first two properties described in the previous section are true with this definition of U and f. The difficulty is the third property, i.e., the polynomial time computation of an upper bound B on $|U|/|G|$ such that the value of B is polynomial. The best upper bound B that can be computed in general is exponentially large in the length of F. The difficulty is not just in the computation of B, because there are easy examples of DNF formulas F such that $|U|/|G|$ is exponentially large in the length of F. This commmonly occurs, for instance, in network reliability problems, which can be thought of as DNF counting problems (see Section 6). Thus, the algorithm presented here is not only theoretically an exponential time algorithm but also for typical problems encountered in

practice it would have to be run for an exponential amount of time to produce a good estimate.

3. The Coverage Algorithm for the Union of Sets Problem

The DNF counting problem can be thought of as a special case of the union of sets problem described below, which we apply to other problems later in the paper. The input to the union of sets problem is a description of m sets, $D_1, \ldots, D_m$. The aim is to compute the cardinality of $D = \bigcup_{i=1}^{m} D_i$. Below we present an ε, δ approximation algorithm for estimating $|D|$, which is the basis of a more efficient algorithm presented in Section 5. To implement the algorithm, we make the following assumptions:

1. For all i between 1 and m, $|D_i|$ can be easily computed.

2. For all i between 1 and m, we can randomly choose an element $s \in D_i$ such that the probability of choosing each such s is $1/|D_i|$.

3. Given any $s \in D$ and any i between 1 and m, it is easy to decide whether or not $s \in D_i$.

The following is a description of one trial of the algorithm. Let U be the direct sum of the D_i, i.e., $U = D_1 \oplus \cdots \oplus D_m$. An element in U is represented by a pair (s, i), where i is between 1 and m and $s \in D_i$. Thus, $|U| = \sum_{i=1}^{m} |D_i|$. For each $s \in D$, define the coverage set to be

$$\operatorname{cov}(s) = \{(s, i) : (s, i) \in U\}.$$

There is one element in $\operatorname{cov}(s)$ for each set D_i containing s. The coverage sets define a partition on U, where the size of each coverage set is at most m and the number of coverage sets is exactly $|D|$. We define $f(s, i) = 1$ if i is the smallest index such that $s \in D_i$ and $f(s, i) = 0$ otherwise. Let G be the subset of U for which f takes on the value of 1. Since, for each $s \in D$, f takes on the value 1 for exactly one element of $\operatorname{cov}(s)$, $|G| = |D|$.

A trial proceeds as described in Section 1: choose (s, i) at random from U (uniformly) and set $Y \leftarrow |U| \cdot f(s, i)$. In more detail:

1. Randomly choose $i \in \{1, \ldots, m\}$ such that i is chosen with probability $|D_i|/|U|$.

2. Randomly choose $s \in D_i$ such that s is chosen with probability $1/|D_i|$.

Note. Steps 1 and 2 randomly choose $(s, i) \in U$ with probability $1/|U|$.

3. Compute $f(s, i)$; $Y \leftarrow f(s, i) \cdot |U|$.

This completes the abstract description of one trial of the coverage algo-

rithm for the union of sets problem. In the next section we present an implementation of this algorithm for the DNF counting problem. The advantage of the coverage algorithm over the naive algorithm is that $m \geq 1/\mu$ (where $\mu = |G|/|U|$), and thus, applying the Zero–One Estimator Theorem, $N = m \cdot 4\ln(2/\delta)/\varepsilon^2$ trials suffice for an ε, δ approximation algorithm.

4. Implementation of the Coverage Algorithm for the DNF Counting Problem

In this section we present an ε, δ approximation algorithm for the DNF counting problem based on the coverage algorithm for the union of sets presented in Section 3. The running time of the algorithm is $O[nm^2 \cdot \ln(1/\delta)/\varepsilon^2]$. In Section 6 we present a more efficient ε, δ approximation algorithm for the DNF counting problem.

For $i = 1, \ldots, m$, let D_i be the set of truth assignments which satisfy clause C_i. Let $D = \bigcup_{i=1}^{m} D_i$. It is easy to verify that $\#F = |D|$. There is one element in $\text{cov}(s)$ for each clause which s satisfies. To make the abstract description more concrete, we provide the following example.

Example 1. Let $F = C_1 \vee C_2 \vee C_3$, where $C_1 = X_1 \wedge X_2$, $C_2 = \bar{X}_1$ and $C_3 = \bar{X}_3$. Consider the following matrix. The rows of the matrix correspond to the elements of D and the columns correspond to the m clauses. A box in entry (s, i) in this matrix indicates that $(s, i) \in U$. The set of boxes in row s correspond to the elements in $\text{cov}(s)$. For each $(s, i) \in U$, if $f(s, i) = 1$ then the corresponding box is filled in and if $f(s, i) = 0$ then the corresponding box is left blank (Fig. 1).

To simplify the discussion of the implementation, we assume a simple representation of the formula F which is not as compact as it could be. We

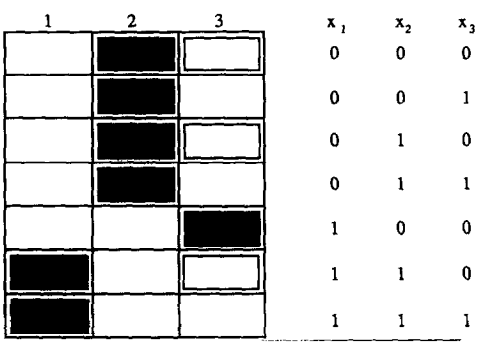


Figure 1

leave it to the reader to derive a more compact representation when eac clause contains a small number of the possible literals. F is represented b an m by n matrix, where the i, j entry in the matrix is 1 if literal X_j appears in clause i, 0 if literal $\bar{X}_j$ appears in clause i, and empty otherwise

The preprocessing for the coverage algorithm consists of the following For $i = 1, \ldots, m$, compute $|D_i| = 2^{n - \#\text{ literals in } C_i}$. Let A be an integer arra of length $m + 1$ such that $A_0 = 0$ and $A_i = \sum_{j=1}^{i} |D_j|$. Then $|U| = A_m$ The total time for the preprocessing step is $O(mn)$.

We run the algorithm for $N = m \cdot 4\ln(2/\delta)/\varepsilon^2$ trials. A trial of th algorithm consists of the following steps:

1. Randomly choose $i \in \{1, \ldots, m\}$ such that i is chosen with proba bility $|D_i|/|U|$. This is done by randomly choosing a number r between 1 and $|U|$ and using binary search to find the entry in A such that $A_{i-1} < r \leq A_i$.

2. Randomly choose $s \in D_i$ such that s is chosen with probability $1/|D_i|$. This is done by setting the truth assignments for the variables which appear in C_i to satisfy clause C_i and then choosing the truth values for the variables that do not appear in C_i randomly to be "true" or "false" each with probability $\frac{1}{2}$.

(Note. Steps 1 and 2 randomly choose $(s, i) \in U$ with probability $1/|U|$.)

3. Compute $f(s, i)$. Let $j = \min\{l: s \in D_l\}$. Checking whether $s \in D_l$ consists of checking whether s satisfies C_l. The value of j is computed by indexing sequentially through the clauses. Then, $f(s, i) = 1$ iff $i = j$.

4. $Y = f(s, i) \cdot |U|$.

The total time for steps 1 and 2 of a trial is $O(n)$. The bottleneck in terms of time is step 3, the computation of $f(s, i)$. For each clause C_l, the time to check if s satisfies C_l is $O(n)$. The total number of clauses which are checked in the worst case is m. Thus, the total time for step 3 is $O(mn)$. The total time for preprocessing plus all trials of the algorithm is $O(nm^2 \cdot \ln(1/\delta)/\varepsilon^2)$.

5. Self-adjusting Coverage Algorithm for the Union of Sets Problem

The most costly step in terms of time for the coverage algorithm is the computation of the function f. In this section we give a different way of computing f which at first glance does not seem to improve the efficiency of the algorithm, but together with another modification substantially reduces the running time of the algorithm. In the description of the

coverage algorithm in Section 3, we note that $E[Y] = |D|$ if f is any function which satisfies, for each $s \in D$,

$$\sum_{(s,i)\in \mathrm{cov}(s)} f(s,i) = 1.$$

We first describe a variant of the coverage algorithm for the union of sets problem, using random variables in place of f. For each $s \in D$, $f'(s)$ is a random variable such that $E[f'(s)] = 1/|\mathrm{cov}(s)|$. A trial consists of randomly choosing $(s, i) \in U$, choosing a random value for $f'(s)$ and setting the estimator Y to $|U| \cdot f'(s)$. It can be easily verified that $E[Y] = |D|$.

For each $s \in D$, not only is $f'(s)$ a random variable, but also the amount of time to compute $f'(s)$ is a random variable $t(s)$. Given $s \in D$, $f'(s)$ is computed as follows:

computing $f'(s)$
$t(s) \leftarrow 0$
repeat
    $t(s) \leftarrow t(s) + 1$
    randomly choose $j \in \{1, \ldots, m\}$ with probability $1/m$
until $s \in D_j$.
$f'(s) \leftarrow t(s)/m$

At each iteration of the repeat loop, the probability that $s \in D_j$ is $|\mathrm{cov}(s)|/m$. Thus, $t(s)$ is a geometric random variable and $E[t(s)] = m/|\mathrm{cov}(s)|$. Thus, $E[f'(s)] = 1/|\mathrm{cov}(s)|$.

When we use this implementation of a trial of the coverage algorithm, the time per trial is dominated by the time spent executing the repeat loop in the computation of f'. In the rest of the discussion, one step is defined to be a single iteration of this repeat loop. The total running time of the algorithm is then the number of steps times the time it takes to execute a step. An upper bound on the time to execute a step of the algorithm is the time to randomly choose $(s, i) \in U$ (this is executed once for each trial of the algorithm, but an upper bound on the number of trials is the total number of steps) plus the time to randomly choose $j \in \{1, \ldots, m\}$ plus the time to determine if $s \in D_j$.

Let $\mu = |D|/|U|$ and let t be the random variable which is defined to be the number of steps in a trial. Then,

$$E[t] = \sum_{(s,i)\in U} \frac{E[t(s)]}{|U|} = \frac{m \cdot |D|}{|U|} = m \cdot \mu.$$

An upper bound on $E[t]$ is m. We can prove a theorem analogous to the Zero–One Estimator Theorem which states that this variant of the coverage algorithm is an ε, δ approximation algorithm when the number of trials is $O[m \cdot \ln(1/\delta)/\varepsilon^2]$. However, this does not improve the total running time of the algorithm since the upper bound on the expected number of steps per

trial is m. For instance, if this variant of the coverage algorithm is used for the DNF counting problem, the total running time of the entire algorithm is $O[nm^2 \cdot \ln(1/\delta)/\varepsilon^2]$, which is no better than the time for the original coverage algorithm.

There is a nice tradeoff which we could take advantage of if we could compute μ. Let c be a constant and let

$$N(\mu) = \frac{c \cdot \ln(1/\delta)}{\mu \cdot \varepsilon^2}.$$

For the coverage algorithm we showed that $N(\mu)$ trials are sufficient to guarantee an ε, δ approximation algorithm for a small value of c. Let $T(\mu)$ be the random variable which is the total number of steps completed by the variant of the coverage algorithm when we execute $N(\mu)$ trials. Then,

$$E[T(\mu)] = N(\mu) \cdot E[t] = \frac{cm \cdot \ln(1/\delta)}{\varepsilon^2}$$

and thus the expected number of steps for this algorithm is m times less than the upper bound on the number of steps for the variant of the coverage algorithm described above. The problem is that we cannot implement this algorithm because it requires the computation of μ which is the quantity we are trying to estimate.

We overcome our inability to implement the algorithm which depends on the computation of μ by the following trick. Let $T = cm \ln(2/\delta)/\varepsilon^2$, where c is a constant to be specified later. We run the algorithm for T steps and let the number of trials completed during these T steps be a random variable N_T. Let TOTAL $\leq T$ be the total number of steps completed in the first N_T trials. The average value of the estimates from the N_T trials is exactly $\tilde{Y} = (\text{TOTAL} \cdot |U|)/(m \cdot N_T)$. Another very similar estimator of $|D|$ is $\tilde{Y}' = (T \cdot |U|)/(m \cdot N_T)$. In the Appendix we prove that for an appropriate choice of c the algorithm is an ε, δ approximation algorithm when using $\tilde{Y}$ or $\tilde{Y}'$ as the estimator. It turns out that the proof for $\tilde{Y}'$ is cleaner and simpler than the proof for $\tilde{Y}$. The intuition for why the algorithm is an ε, δ approximation algorithm when using estimator $\tilde{Y}$ follows. The expected value of N_T is approximately $T/E[t] = c \ln(2/\delta)/\mu\varepsilon^2 = N(\mu)$. Thus, the intuition is that by fixing T, the number of trials completed by the algorithm self-adjusts so that with high probability enough trials are run to guarantee an ε, δ approximation algorithm.

SELF-ADJUSTING COVERAGE ALGORITHM THEOREM I. *When* $\varepsilon < 1$ *and* $T = (8 \cdot (1 + \varepsilon) \cdot m \ln(2/\delta))/\varepsilon^2$, *the self-adjusting coverage algorithm is an* ε, δ *approximation algorithm when estimator* $\tilde{Y}'$ *is used.*

Proof. In the Appendix. □

SELF-ADJUSTING COVERAGE ALGORITHM THEOREM II. *When* $\varepsilon < 1$ *and* $T = (8 \cdot (1 + \varepsilon) \cdot m \ln(3/\delta))/(1 - \varepsilon^2/8)\varepsilon^2$, *the self-adjusting coverage algorithm is an* ε, δ *approximation algorithm when estimator* $\tilde{Y}$ *is used.*

Proof. In the Appendix. □

A related but weaker theorem is stated in [KL1, Lu1]. The theorem stated here substantially improves the theorem stated there; here the running time is proportional to $\ln(1/\delta)$ whereas there the running time is proportional to $1/\delta$. The estimators $\tilde{Y}$ and $\tilde{Y}'$ are not necessarily unbiased estimators of $|D|$ because of the stopping rule.

SELF-ADJUSTING COVERAGE ALGORITHM.

gtime ← 0 {*gtime* counts the global number of steps executed}
TOTAL ← 0
$N_T \leftarrow 0$
T is set as specified in Theorem I or Theorem II
trial:
  randomly choose $(s, i) \in U$ with probability $1/|U|$ {as before}
  $t(s) \leftarrow 0$
  step:
    $t(s) \leftarrow t(s) + 1$
    gtime ← *gtime* + 1
    If *gtime* > T **then go to finish**
    randomly choose $j \in \{1, \ldots, m\}$ with probability $1/m$
    If not $s \in D_j$ **then go to step**
  TOTAL ← *gtime*
  $N_T \leftarrow N_T + 1$
  $f'(s) \leftarrow t(s)/m$
  $Y_{N_T} \leftarrow |U| \cdot f'(s)$
  Go to trial
finish:
$\tilde{Y} \leftarrow \sum_{i=1}^{N_T} Y_i/N_T$
(Equivalently, $\tilde{Y} \leftarrow$ (TOTAL · $|U|$)/$(m \cdot N_T)$)
$\tilde{Y}' \leftarrow T \cdot |U|/m \cdot N_T$

6. APPLICATIONS TO DNF COUNTING AND RELIABILITY PROBLEMS

The implementation details and timing analysis of the self-adjusting coverage algorithm applied to the DNF counting problem are as follows. The preprocessing is exactly the same as for the coverage algorithm presented in Section 4. The time per step is dominated by the time to determine if truth assignment s satisfies clause C_j; this can be executed in $O(n)$ time. Thus, the total running time of the self-adjusting coverage algorithm for the DNF counting problem is $O[nm \cdot \ln(2/\delta)/\varepsilon^2]$.

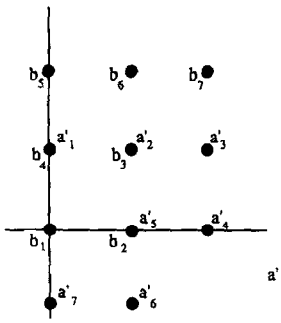
An important generalization of the DNF counting problem is the DNF probability problem. The input for the DNF probability problem is the

same as for the DNF counting problem together with a set of probabilities $\{p_1, \ldots, p_n\}$. We define a probability distribution on the set of 2^n truth assignments for the variables as follows. For $i = 1, \ldots, n$, the value of variable X_i is "true" with probability p_i and "false" with probability $1 - p_i$ independently of the values of the other variables. We are interested in computing $\Pr[F]$, which is the probability that a truth assignment randomly chosen according to the probability distribution satisfies the formula F. The DNF counting problem is the special case of the DNF probability problem when all the variable probabilities are $\frac{1}{2}$, in which case $\Pr[F] = \#F/2^n$. Both the coverage algorithm and the self-adjusting coverage algorithm for the DNF counting problem can be easily modified for the DNF probability problem such that the resulting algorithms are ε, δ approximation algorithms. The number of steps sufficient to produce an ε, δ approximation algorithm using estimators $\tilde{Y}$ and $\tilde{Y}'$ remains unchanged from before. The running time of the self-adjusting coverage algorithm for the DNF probability problem is linear in the size of the input times $\ln(1/\delta)/\varepsilon^2$. A brief description of these algorithms for the DNF probability problem is given in [KL1, Lu1].

Many network reliability problems can be expressed as instances of the DNF probability problem. For example, the 2-terminal reliability problem can be reduced to the DNF probability problem by listing all the cut sets in the graph [KL2, Lu2]. This reduction can take exponential time in the size of the input, because m is the number of cut sets in the graph and can be exponential in the size of the input graph. In [KL2, Lu2], ε, δ approximation algorithms for the 2-terminal reliability problem (and the multi-terminal reliability problem, which is a generalization of the 2-terminal reliability problem) are developed when the input graph G is planar. These algorithms avoid listing the cut sets. The algorithms are shown to run in polynomial time when the probabilities of the edges are sufficiently small, which is the hard case for the naive Monte-Carlo algorithm. [Va, PB] give evidence that there is no polynomial time algorithm for computing the exact value of μ even in this case.

7. An Application to Counting the Number of Distinct Translates

Let $a = \{a_1, \ldots, a_m\}$ be a set of m distinct points with integer coordinates in the plane. For $i = 1, \ldots, m$ let $(a_i(x), a_i(y))$ be the (x, y) coordinates of a_i. Similarly, let b be a set of m distinct points in the plane with integer coordinates (this can easily be modified to the case when b is a set of n points, where $n \neq m$; it can also easily be modified to dimensions greater than two). A **translate** $a' = \{a'_1, \ldots, a'_m\}$ of a is a set of m points in the plane such that for some pair of integers $(\Delta x, \Delta y)$, for all $i =$



index	a-coor x	a-coor y	b-coor x	b-coor y	b-sort x	b-sort y
1	0	0	0	0	0	0
2	1	0	1	0	0	1
3	2	0	1	1	0	2
4	2	-1	0	1	1	0
5	1	-1	0	2	1	1
6	1	-2	1	2	1	2
7	0	-2	2	2	2	2

a' = g(2,3) cov(a') = {(1,4), (2,3), (5,2)}

FIGURE 2

$1, \ldots, m$, $a_i'(x) = a_i(x) + \Delta x$ and $a_i'(y) = a_i(y) + \Delta y$. We can represent a translate in two different but equivalent ways; either as the vector points a' just described or by the pair of integers $(\Delta x, \Delta y)$ just described.

Let D be the set of distinct translates a' of a such that a' and b have at least one coincident point. We are interested in computing $|D|$ when m is large. For simplicity, we assume that each integer coordinate can be stored in one word of memory. The best deterministic algorithm which we know for computing $|D|$ is described below and has running time $O(m^2 \log m)$. In this section we show how to apply the self-adjusting coverage algorithm to obtain an ε, δ approximation algorithm to estimate $|D|$ which has running time $O[m \cdot \log m \cdot \ln(1/\delta)/\varepsilon^2]$. Let

$$U = \{(i, j): i = 1, \ldots, m \text{ and } j = 1, \ldots, m\}.$$

We define $g: U \to D$ such that for all $(i, j) \in U$, $g(i, j)$ is the translate (in $(\Delta x, \Delta y)$ representation) $a' \in D$ of a such that a_i' and b_j coincide, i.e.,

$$g(i, j) = \big(b_j(x) - a_i(x), b_j(y) - a_i(y)\big).$$

Let $D_i = \{g(i, j): j = 1, \ldots, m\}$. For each i between 1 and m, D_i is a set of m distinct translates and $D = \bigcup_{i=1}^{m} D_i$. Thus, $m \le |D| \le m^2$.

The input to the problem consists of two arrays a–coor, b–coor each of length m, where the ith entry contains the x and y coordinates of point a_i, b_i, respectively. $|D|$ can be computed in $O(m^2 \log m)$ time by first computing $g(i, j)$ for all $(i, j) \in U$ and then sorting these values to remove duplicates.

EXAMPLE 2. The (x, y) coordinates of a and b are stored in arrays a–coor and b–coor as shown in Fig. 2, respectively. The figure shows translate a' of a, where a' is the translate represented by $g(2, 3)$; $g(2, 3)$ is in sets D_1, D_2 and D_5. Also shown is an auxiliary array b–sort, which contains the coordinates of the points in b sorted first by x coordinate and then by y coordinate.

The preprocessing for the self-adjusting coverage algorithm for estimating $|D|$ consists of forming a new array b–sort of length m, which is exactly the same as b–coor except that the points are sorted first by x coordinate and then by y coordinate. The time for the preprocessing is $O(m \cdot \log m)$. Figure 2 shows the b–sort array for a particular a and b.

The body of the self-adjusting coverage algorithm to estimate $|D|$ is essentially the same as the self-adjusting coverage algorithm presented in Section 5, with the following implementation:

1. Randomly choose $(i, j) \in U$ with probability $1/m^2$. Then, $g(i, j) \in D$.

2. Randomly choose $k \in \{1, \ldots, m\}$ with probability $1/m$.

3. Determine if $g(i, j) \in D_k$. Let a' be the translate $g(i, j)$ of a. We first compute the coordinates of a'_k,

$$a'_k(x) = a_k(x) + \big(b_j(x) - a_i(x)\big)$$

and

$$a'_k(y) = a_k(y) + \big(b_j(y) - a_i(y)\big),$$

using the a–coor and b–coor arrays, and then search for a'_k in the b–sort array using binary search. In Example 2, if $i = 2$, $j = 3$, $k = 5$,

$$a'_5(x) = 1 + (1 - 1) = 1$$

and

$$a'_5(y) = -1 + (1 - 0) = 0$$

In this case, $a'_5 \in b$ and thus $g(2, 3) \in D_5$.

The total time for steps 1–3 per execution is $O(\log m)$. The analysis for the number of steps sufficient to guarantee an ε, δ approximation algorithm is exactly the same as the analysis for the self-adjusting coverage algorithm for the union of sets problem. The running time of the algorithm per step is $O(\log m)$ and the total running time of the entire algorithm is $O[m \log m \ln(1/\delta)/\varepsilon^2]$. Thus, the running time of the algorithm is linear in the size of the input times $\log m$ times $\ln(1/\delta)/\varepsilon^2$. This algorithm can be easily generalized to the case of sets of points in a space of dimension greater than two, such that the running time of the algorithm remains linear in the size of the input times $\log m$ times $\ln(1/\delta)/\varepsilon^2$.

An application of the algorithm for estimating the number of distinct translates occurs in the literature of "self-avoiding walks." A self-avoiding walk $a = \{a_1, \ldots, a_m\}$ is a sequence of m distinct points with integer coordinates in the plane such that for $i = 1, \ldots, m - 1$, a_{i+1} is at distance one from a_i. In Fig. 2, both a and b are self-avoiding walks with 7 points along each walk. Self-avoiding walks in higher dimensions are defined in an

analogous way. The statistical properties of self-avoiding walks are of interest in physical chemistry (as models of long-chain polymer molecules) and in statistical physics (as a simple model of critical phenomena).

Two interesting quantities are c_n, the number of self-avoiding walks with n points which start at the origin, and $c_{m,n}$, the number of pairs (a, b) of self-avoiding walks such that b is a walk with n points which starts at the origin and a is a walk with m points which starts anywhere such that at least one point in b coincides with some point in a. Also of interest is the ratio $c_{m,n}/c_m \cdot c_n$, which can be estimated by a Monte-Carlo algorithm, where one trial of the algorithm consists of the following:

1. Choose a random self-avoiding walk a of length m and a random self-avoiding walk b of length n such that both a and b start at the origin.

2. Compute $|D|$, which is the number of distinct translates of a which intersect b in at least one point. The estimate of the ratio is $Y \leftarrow |D|$.

Step 1 is difficult to perform quickly, especially if it is required that all self-avoiding walks are chosen independently; no polynomial time algorithm is known which does this. However, in practice there is a way of generating the next a and b in time $O(m + n)$ and the new walks have the property that they are "approximately" independent of the previous a and b [MS]. The best known algorithm for step 2, computing $|D|$, is $O(m^2 \log m)$ when $m = n$, as described above. We advocate instead using the above ε, δ approximation algorithm to estimate $|D|$ in step 2, which has running time $O[m \cdot \log m \cdot \ln(1/\delta)/\varepsilon^2]$.

8. Other Applications and Summary

This paper substantially improves the analysis given in [KL1, Lu1] for the self-adjusting coverage algorithm for DNF counting. We also provide a new application to a problem of interest in physical chemistry and statistical physics. We imagine that many other applications of these techniques will be found to other problems in the future.

In this paper, ε, δ approximation algorithms are presented for the DNF counting problem. We know that unless RP = NP that there is no $0, \delta$ approximation algorithm for DNF counting which runs in time polynomial in $1/\delta$ times the length of the formula. However, it is not out of the question that there is a $\varepsilon, 0$ approximation algorithm for DNF counting which runs in time polynomial in $1/\varepsilon$ times the length of the formula. In fact, [AW] provide an $\varepsilon, 0$ algorithm for the DNF counting problem when the number of literals per clause is constant. The question is open when the number of literals per clause is not restricted.

APPENDIX: PROOF OF THEOREMS

We prove the Zero–One Estimator Theorem stated in Section 1 and the Self-adjusting Coverage Algorithm Theorem stated in Section 5. We first state some lemmas needed for the proof of both theorems.

LEMMA 1. *Let $Z, Z_1, Z_2, \ldots$ be independently and identically distributed random variables and let c and d be constants. Then,*

$$E[e^{d(Z_1 + \cdots Z_k) - c}] = E[e^{dZ}]^k \cdot e^{-c}.$$

LEMMA 2. *For any random variable Z and any constant $d \geq 0$,*

$$\Pr[Z \geq 0] \leq E[e^{dZ}].$$

Proof. Define the random variable Z' to be one if $Z \geq 0$ and zero otherwise. Then, $Z' \leq e^{dZ}$ and thus $E[Z'] \leq E[e^{dZ}]$. The lemma follows because $E[Z'] = \Pr[Z \geq 0]$. □

The next few lemmas and corollaries are used in the proof of the Zero–One Estimator theorem. Let $Y, Y_1, Y_2, \ldots$ be independently and identically distributed zero–one valued random variables such that $E[Y] = \mu$.

LEMMA 3. *Let $d \leq 1$ be a constant. Then,*

$$E[e^{dY}] \leq e^{\mu d(1+d)}.$$

Proof. $E[e^{dY}] = \mu e^d + (1 - \mu) \leq 1 - \mu + \mu(1 + d + d^2) = 1 + \mu d(1 + d) \leq e^{\mu d(1+d)}$. □

COROLLARY 4. *For $\varepsilon \leq 2$,*

$$\Pr\left[\sum_{i=1}^{N} Y_i > (1 + \varepsilon)\mu N\right] \leq e^{-\mu\varepsilon^2 N/4}.$$

Proof. From Lemmas 1 and 2,

$$\Pr\left[\sum_{i=1}^{N} Y_i > (1 + \varepsilon)\mu N\right] \leq E[e^{d(Y-(1+\varepsilon)\mu)}]^N.$$

From Lemma 1,

$$E[e^{d(Y-(1+\varepsilon)\mu)}] = E[e^{dY}] \cdot e^{-d\mu(1+\varepsilon)}.$$

Using Lemma 3 and letting $d = \varepsilon/2$,

$$E[e^{d(Y-(1+\varepsilon)\mu)}] \leq e^{-\mu\varepsilon^2/4}.$$

The corollary follows. □

LEMMA 5. *Let $d \leq 1$ be a constant. Then,*

$$E[e^{-dY}] \leq e^{-\mu d(1-d/2)}.$$

Proof. $E[e^{-dY}] = \mu e^{-d} + (1-\mu) \leq 1 - \mu + \mu(1 - d + d^2/2) = 1 - \mu d(1 - d/2) \leq e^{-\mu d(1-d/2)}$. □

COROLLARY 6. *For $\varepsilon \leq 2$,*

$$\Pr\left[\sum_{i=1}^{N} Y_i < (1-\varepsilon)\mu N\right] \leq e^{-\mu\varepsilon^2 N/4}.$$

Proof. From Lemmas 1 and 2,

$$\Pr\left[\sum_{i=1}^{N} Y_i < (1-\varepsilon)\mu N\right] \leq E[e^{d((1-\varepsilon)\mu - Y)}]^N.$$

From Lemma 1,

$$E[e^{d((1-\varepsilon)\mu - Y)}] = E[e^{-dY}] \cdot e^{d\mu(1-\varepsilon)}.$$

Using Lemma 5 and letting $d = \varepsilon/2$,

$$E[e^{d((1-\varepsilon)\mu - Y)}] \leq e^{-\mu\varepsilon^2/4}.$$

The corollary follows. □

ZERO-ONE ESTIMATOR THEOREM. *When $\varepsilon \leq 2$ and $N = 4/(\varepsilon^2\mu) \cdot \ln(2/\delta)$ then*

$$\Pr\left[(1-\varepsilon)\mu \leq \sum_{i=1}^{N} Y_i/N \leq (1+\varepsilon)\mu\right] \geq 1 - \delta.$$

Proof. It is sufficient to show

$$\Pr\left[(1-\varepsilon)\mu > \sum_{i=1}^{N} Y_i/N\right] + \Pr\left[\sum_{i=1}^{N} Y_i/N > (1-\varepsilon)\mu\right] \leq \delta.$$

This follows using Corollaries 4 and 6 substituting in the value for N. □

We now introduce some notation and prove some lemmas and corollaries which are used in the proof of the Self-adjusting Coverage Algorithm Theorem. For $k = 1, \ldots, m$, let

$$R_k = \{s \in D : |\mathrm{cov}(s)| = k\}$$

and let $r_k = |R_k|$. Then, $\sum_{k=1}^{m} r_k = |D|$ and $\sum_{k=1}^{m} k r_k = |U|$. The probability that in a trial the element of D chosen is in R_k is $kr_k/|U|$. For all $s \in R_k$, the distribution of the random variable $t(s)$ depends only on k. We

define a random variable τ_k such that for all $s \in R_k$, τ_k has the same distribution as $t(s)$. The random variable τ_k has a geometric distribution with mean m/k,

$$\Pr[\tau_k = j] = \frac{k(1 - k/m)^{j-1}}{m}.$$

Let $d = \lambda/m$ for some constant $\lambda \leq \frac{1}{2}$. Then,

$$E[e^{d\tau_k}] = \sum_{j=1}^{\infty} \frac{e^{dj}k(1 - k/m)^{j-1}}{m} = \frac{e^d k}{m(1 - (1 - k/m)e^d)}.$$

The series converges because $(1 - k/m)e^d < 1$ for $d \leq 1/2m$. Recall that random variable t is the number of steps in a trial of the self-adjusting coverage algorithm. As before, let $\mu = |D|/|U|$. As shown in Section 5, $E[t] = m\mu$. In the remainder of the Appendix, $t_1, t_2, \ldots$ are independent and identically distributed random variables with the same distribution as t, where t_i is the number of steps in trial i. Let $S_l = \sum_{i=1}^{l} t_i$ be the step at which the lth trial is completed and let N_i be the number of trials completed by time step i. Then $N_i < l$ if and only if $S_l > i$.

LEMMA 7. *Let* $0 \leq \lambda \leq \frac{1}{2}$ *and let* $d = \lambda/m$. *Then*

$$E[e^{dt}] \leq e^{(\lambda + 2\lambda^2)\mu} = e^{(md + 2(md)^2)\mu}.$$

Proof.

$$E[e^{dt}] \leq \sum_{k=1}^{m} \frac{kr_k E[e^{d\tau_k}]}{|U|} = \frac{1}{|U|} \sum_{k=1}^{m} \frac{k^2 r_k e^d}{m(1 - (1 - k/m)e^d)}$$

$$= \frac{1}{|U|} \sum_{k=1}^{m} \frac{k^2 r_k}{m(e^{-d} - 1 + k/m)}.$$

From $1 - d \leq e^{-d}$ we obtain $k/m - d \leq e^{-d} - 1 + k/m$. Using these two facts in sequence,

$$\frac{k}{m(e^{-d} - 1 + k/m)} \leq 1 + \frac{d}{e^{-d} - 1 + k/m} \leq 1 + \frac{d}{k/m - d}.$$

Thus,

$$E[e^{dt}] \leq 1 + \frac{1}{|U|} \sum_{k=1}^{m} \frac{kr_k d}{k/m - d}.$$

Because $d = \lambda/m$ and $k \geq 1$, $k/m - d \geq k(1 - \lambda)/m$. Using these facts,

$$E[e^{dt}] \leq 1 + \frac{1}{|U|} \sum_{k=1}^{m} \frac{r_k \lambda}{1 - \lambda}.$$

Using $1/(1 - \lambda) \leq 1 + 2\lambda$, we obtain

$$E[e^{dt}] \leq 1 + \frac{|D|\lambda(1 + 2\lambda)}{|U|} = 1 + \mu(\lambda + 2\lambda^2) \leq e^{(\lambda + 2\lambda^2)\mu}. \quad \square$$

COROLLARY 8. *Let* $\varepsilon \leq 2$. *Then*

$$\Pr[S_l > (1 + \varepsilon)m\mu l] \leq e^{-\mu\varepsilon^2 l/8}.$$

Proof. Let $t' = t - (1 + \varepsilon)m\mu$ and let $t_i' = t_i - (1 + \varepsilon)m\mu$. Using Lemmas 1 and 2,

$$\Pr[S_l > (1 + \varepsilon)m\mu l] = \Pr\left[\sum_{i=1}^{l} t_i' > 0\right] \leq E[e^{dt'}]^l.$$

Using Lemma 1,

$$E[e^{dt'}] = E[e^{dt}] \cdot e^{-(1+\varepsilon)m\mu d}.$$

Using Lemma 7,

$$E[e^{dt}] \leq e^{(md + 2(md)^2)\mu}.$$

Thus,

$$E[e^{dt'}] \leq e^{m\mu d(2md - \varepsilon)}.$$

Letting $d = \varepsilon/4m$ yields

$$E[e^{dt'}]^l \leq e^{-\mu\varepsilon^2 l/8}. \quad \square$$

LEMMA 9. *Let* $0 \leq \lambda \leq \frac{1}{2}$ *and let* $d = \lambda/m$. *Then*

$$E[e^{-dt}] \leq e^{-(\lambda - \lambda^2)\mu} = e^{-(md - (md)^2)\mu}.$$

Proof. (similar to the proof of Lemma 7).

$$E[e^{-dt}] \leq 1 - \frac{1}{|U|} \sum_{k=1}^{m} \frac{k r_k d}{k/m + d}.$$

Because $d = \lambda/m$ and $k \geq 1$, $k/m + d \leq k(1 + \lambda)/m$. Using these facts,

$$E[e^{-dt}] \leq 1 - \frac{1}{|U|} \sum_{k=1}^{m} \frac{r_k \lambda}{1 + \lambda}.$$

Using $1/(1+\lambda) \geq 1-\lambda$, we obtain

$$E[e^{-dt}] \leq 1 - \frac{|D|\lambda(1-\lambda)}{|U|} = 1 - \mu(\lambda - \lambda^2) \leq e^{-(\lambda-\lambda^2)\mu}. \quad \square$$

COROLLARY 10. *Let* $\varepsilon \leq 2$. *Then*

$$\Pr[S_l < (1-\varepsilon)m\mu l] \leq e^{-\mu\varepsilon^2 l/8}.$$

Proof. Let $t' = -t + (1-\varepsilon)m\mu$ and let $t_i' = -t_i + (1-\varepsilon)m\mu$. Using Lemmas 1 and 2,

$$\Pr[S_l < (1-\varepsilon)m\mu l] = \Pr\left[\sum_{i=1}^{l} t_i' > 0\right] \leq E[e^{dt'}]^l.$$

Using Lemma 1,

$$E[e^{dt'}] = E[e^{-dt}] \cdot e^{(1-\varepsilon)m\mu d}.$$

Using Lemma 9,

$$E[e^{-dt}] \leq e^{-(md-(md)^2)\mu}.$$

Thus,

$$E[e^{dt'}] \leq e^{m\mu d(md-\varepsilon)}.$$

Letting $d = \varepsilon/4m$ yields

$$E[e^{dt'}]^l \leq e^{-\mu\varepsilon^2 l/8}. \quad \square$$

SELF-ADJUSTING COVERAGE ALGORITHM THEOREM I. *When* $\varepsilon < 1$ *and* $T = 8(1+\varepsilon)m\ln(2/\delta)/\varepsilon^2$ *then the self-adjusting coverage algorithm for the union of sets problem is an* ε, δ *approximation algorithm when estimator* $Y = \tilde{Y}'$ *is used to estimate* $|D|$.

Proof. Let

$$k_1 = \frac{8(1+\varepsilon)\ln(2/\delta)}{\mu\varepsilon^2(1+\varepsilon)}$$

and let

$$k_2 = \frac{8(1+\varepsilon)\ln(2/\delta)}{\mu\varepsilon^2(1-\varepsilon)}.$$

Thus, $T = k_1 m\mu(1 + \varepsilon)$ and $T = k_2 m\mu(1 - \varepsilon)$. If $k_1 \le N_T \le k_2$ then

$$\frac{T}{k_2} \le \frac{T}{N_T} \le \frac{T}{k_1}$$

and thus

$$\frac{T \cdot |U|}{mk_2} \le Y \le \frac{T \cdot |U|}{mk_1}$$

and consequently

$$|D| \cdot (1 - \varepsilon) \le Y \le |D| \cdot (1 + \varepsilon).$$

Thus, it is sufficient to show that

$$\Pr[N_T > k_2] + \Pr[N_T < k_1] \le \delta.$$

From Corollary 8,

$$\Pr[N_T < k_1] = \Pr\left[S_{k_1} > T\right] = \Pr\left[S_{k_1} > k_1 m\mu(1 + \varepsilon)\right] \le \delta/2.$$

From Corollary 10,

$$\begin{aligned}\Pr[N_T > k_2] &\le \Pr[N_T \ge k_2] = \Pr\left[S_{k_2} \le T\right] \\ &= \Pr\left[S_{k_2} \le k_2 m\mu(1 - \varepsilon)\right] \le \delta/2. \ \square\end{aligned}$$

SELF-ADJUSTING COVERAGE ALGORITHM THEOREM II. *When* $\varepsilon < 1$ *and* $T = 8(1 + \varepsilon)m \ln(3/\delta)/(1 - \varepsilon^2/8)\varepsilon^2$ *then the self-adjusting coverage algorithm for the union of sets problem is an* ε, δ *approximation algorithm when estimator* $Y = \tilde{Y}$ *is used to estimate* $|D|$.

Proof. The proof is similar to the proof of Theorem I. Let $gap = m \ln(3/\delta)$ and let $T' = T - gap$. Since $gap \le (\varepsilon^2/8) \cdot T$, $T' \ge (1 - \varepsilon^2/8)T$. Proceedings as in Theorem I, let

$$k_1 = \frac{T}{m\mu(1 + \varepsilon)}$$

and let

$$k_2 = \frac{T(1 - \varepsilon^2/8)}{m\mu(1 - \varepsilon)}$$

If $T' < S_{N_T} \le T$ and $k_1 \le N_T \le k_2$ then

$$\frac{T'}{k_2} \le \frac{S_{N_T}}{N_T} \le \frac{T}{k_1}$$

and consequently

$$|D| \cdot (1 - \varepsilon) \leq Y \leq |D| \cdot (1 + \varepsilon).$$

Certainly $S_{N_T} \leq T$. Thus, it is sufficient to show that

$$\Pr[S_{N_T} \leq T'] + \Pr[N_T > k_2] + \Pr[N_T < k_1] \leq \delta.$$

Corollaries 8 and 10 can be used similarly to the way they were used in the proof of Theorem I to show that $\Pr[N_T > k_2] \leq \delta/3$ and that $\Pr[N_T < k_1] \leq \delta/3$. To bound $\Pr[S_{N_T} \leq T']$ above, we note that $S_{N_T} \leq T'$ can occur only if the trial in progress at step T' does not complete by step T. In the worst case, the number of steps to complete the trial in progress at time T' has the same distribution as τ_1. Thus,

$$\Pr[S_{N_T} \leq T'] \leq \Pr[\tau_1 > gap] \leq \sum_{j=gap+1}^{\infty} \frac{(1 - 1/m)^{j-1}}{m}$$

$$= (1 - 1/m)^{gap} \leq e^{-gap/m} = \frac{\delta}{3}. \quad \square$$

References

[AW] M. Ajtai and A. Wigderson, Deterministic simulation of probabilistic constant depth circuits, in "Proceedings 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 11–19.

[KL1] R. M. Karp and M. G. Luby, Monte-Carlo algorithms for enumeration and reliability problems, in "Proceedings 24th IEEE Foundations of Computer Science Symposium, 1983, pp. 56–64.

[KL2] R. M. Karp and M. G. Luby, Monte-Carlo algorithms for planar multiterminal network reliability problems, *J. Complexity* **1** (1985), 45–64.

[Lu1] M. G. Luby, "Monte-Carlo Methods for Estimating System Reliability," Report UCB/CSD 84/168, Computer Science Division, University of California, Berkeley, 1983.

[Lu2] M. G. Luby, "Monte-Carlo Algorithms for Planar Multiterminal Network Reliability Problems, Ph.D. thesis, Computer Science Division, University of California, Berkeley, 1983.

[MS] N. Madras and A. Sokal, Highly efficient Monte-Carlo algorithm for the self-avoiding walk, working paper, 1987.

[PB] J. S. Provan and M. O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, working paper, 1981.

[Re] A. Rényi, "Probability Theory, North-Holland, Amsterdam, 1970.

[Va] L. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8** (1979), 410–421.