Acta Informatica (2009) 46:155–168
DOI 10.1007/s00236-009-0089-4

ORIGINAL ARTICLE

Random walks for selected boolean implication and equivalence problems

K. Subramani · Hong-Jian Lai · Xiaofeng Gu

Received: 20 May 2008 / Accepted: 7 January 2009 / Published online: 11 February 2009

Abstract This paper is concerned with the design and analysis of a random walk algorithm for the 2CNF implication problem (2CNFI). In 2CNFI, we are given two 2CNF formulas ϕ_1 and ϕ_2 and the goal is to determine whether every assignment that satisfies ϕ_1, also satisfies ϕ_2. The implication problem is clearly coNP-complete for instances of kCNF, $k \geq 3$; however, it can be solved in polynomial time, when $k \leq 2$. The goal of this paper is to provide a Monte Carlo algorithm for 2CNFI with a bounded probability of error. The technique developed for 2CNFI is then extended to derive a randomized, polynomial time algorithm for the problem of checking whether a given 2CNF formula Nae-implies another 2CNF formula.

1 Introduction

This paper is concerned with the design and analysis of a randomized algorithm for the 2CNF implication problem (2CNFI) and its variants under a non-standard model of computation.

The kCNF Implication problem is defined by two kCNF formulas ϕ_1 and ϕ_2 and the goal is to check whether every assignment that satisfies ϕ_1 also satisfies ϕ_2. It is not hard to see that the kCNF implication problem is coNP-complete for $k \geq 3$, since kCNF satisfiability can be inverse reduced to it. However, for $k \leq 2$, the problem can be solved in polynomial

This research has been supported in part by the Air Force Office of Scientific Research under grant FA9550-06-1-0050.

K. Subramani (✉)
LDCSEE, West Virginia University, Morgantown, WV, USA
e-mail: ksmani@csee.wvu.edu

H.-J. Lai · X. Gu
Department of Mathematics, West Virginia University, Morgantown, WV, USA
e-mail: hjlai@math.wvu.edu

X. Gu
e-mail: xgu@math.wvu.edu

time, since in this case, the implication problem can be reduced to the satisfiability problem, which is polynomial time solvable [4].

A problem that is closely related to the CNF satisfiability problem is the CNF Nae-satisfiability problem. In this problem, we are given a kCNF formula and the goal is to find an assignment that satisfies all the clauses, while simultaneously falsifying at least one literal in each clause. It is well-known that the Nae-satisfiability is `NP-complete` for $k \geq 3$, from which it follows that the Nae-implication problem is `coNP-complete` for $k \geq 3$. As with 2CNF satisfiability, the Nae-satisfiability and Nae-implication problems are polynomial time solvable for $k \leq 2$.

We present algorithms for the 2CNF Implication and Nae-implication problems in a non-standard model of computation, in which only restricted forms of queries are permitted. We show that our algorithms solve the implication problems with an error probability of at most $3/4$ and the equivalence problems with an error probability of at most $3/4$.

The principal contributions of this paper are as follows:

(i) A bounded-error randomized algorithm for the 2CNF implication problem,
(ii) A bounded-error randomized algorithm for the 2CNF Nae-implication problem,
(iii) A bounded-error randomized algorithm for the 2CNF equivalence problem, and
(iv) A bounded-error randomized algorithm for the 2CNF Nae-equivalence problem.

The rest of this paper is organized as follows: Sect. 2 provides a formal description of the problem under consideration along with notations and preliminaries that will be in vogue for the rest of the paper. The motivation for our work is detailed in Sect. 3. In Sect. 4, we discuss related work in the literature. We propose a randomized algorithm for 2CNFI in Sect. 5; this section also includes analyses of the expected convergence time and the error probability from the perspective of a new computational model. The techniques developed in Sect. 5 are extended in Sect. 6 to solve the Nae-implication problem in 2CNF formulas. Finally, we conclude in Sect. 7, by summarizing our contributions and outlining problems for future research.

2 Preliminaries and statement of problems

Let $X = \{x_1, x_2, \ldots, x_n\}$ denote a set of n Boolean variables. An assignment to these variables is any mapping $F : X \rightarrow \{\textbf{true}, \textbf{false}\}$. A disjunction of these variables or their negations defines a clause; a conjunction of clauses defines a boolean formula in CNF. A formula is said to be in kCNF, if *every clause has exactly k literals.* A clause is said to be satisfied by an assignment F, if at least one of its literals is **true** under that assignment; a kCNF formula is said to be satisfied by an assignment, if every clause that constitutes that formula is satisfied by that assignment. We use $F \models \phi$ to denote the fact that assignment F satisfies formula ϕ.

Definition 2.1 A formula ϕ_1 is said to be Nae-satisfied by an assignment F (denoted by $F \models_n \phi$), if F satisfies ϕ_1, while simultaneously setting at least one literal in each clause of ϕ_1 to **false**.

We make the following observations:

(i) If $F \models_n \phi_1$, then $F \models \phi_1$, but the converse is not necessarily true.
(ii) If $F \models_n \phi_1$, then $\overline{F} \models_n \phi_1$, where $\overline{F}$ is the assignment in which every variable that is set to **true** in F is set to **false** and vice versa. $\overline{F}$ is called the conjugate of F.

Let ϕ_1 and ϕ_2 denote two Boolean formulas in kCNF over the n Boolean variables $\{x_1, x_2, \ldots, x_n\}$, with ϕ_1 being represented by m clauses and ϕ_2 being represented by m' clauses.

Definition 2.2 We say that ϕ_1 implies ϕ_2, written $\phi_1 \rightarrow \phi_2$, if every assignment that satisfies ϕ_1 also satisfies ϕ_2.

By convention, if ϕ_1 is unsatisfiable, then $\phi_1 \rightarrow \phi_2$ for all ϕ_2. Likewise, if ϕ_2 is unsatisfiable, then $\phi_1 \not\rightarrow \phi_2$, for any *satisfiable* ϕ_1.

Definition 2.3 We say that ϕ_1 Nae-implies ϕ_2, written as $\phi_1 \rightarrow_n \phi_2$, if every assignment that Nae-satisfies ϕ_1 also Nae-satisfies ϕ_2.

By convention, if ϕ_1 is Nae-unsatisfiable, then $\phi_1 \rightarrow_n \phi_2$ for all ϕ_2. Likewise, if ϕ_2 is Nae-unsatisfiable, then $\phi_1 \not\rightarrow_n \phi_2$, for any *Nae-satisfiable* ϕ_1.

It is important to note that for a given pair of formulas ϕ_1 and ϕ_2, we can have $\phi_1 \rightarrow_n \phi_2$ but $\phi_1 \not\rightarrow \phi_2$. For instance, let $\phi_1 = (x_1, x_2)$ and let $\phi_2 = (\bar{x_1}, \bar{x_2})$. It is clear that $\phi_1 \not\rightarrow \phi_2$, since the assignment ($x_1 =$ **true**, $x_2 =$ **true**) satisfies ϕ_1 but falsifies ϕ_2. On the other hand, every assignment which Nae-satisfies ϕ_1 also Nae-satisfies ϕ_2 and vice versa.

However, the converse is not possible, i.e., if ϕ_1 implies ϕ_2, then it must Nae-imply ϕ_2 as well.

Theorem 2.1 $(\phi_1 \rightarrow \phi_2) \rightarrow (\phi_1 \rightarrow_n \phi_2)$

Proof Let $\phi_1 \rightarrow \phi_2$ and assume that $\phi_1 \not\rightarrow_n \phi_2$. Then there exists an assignment F, such that $F \models_n \phi_1$, but $F \not\models_n \phi_2$.

However, $F \models \phi_1$ (since Nae-satisfaction implies satisfaction) and hence $F \models \phi_2$. Inasmuch as $F \models \phi_2$, but does not Nae-satisfy it, there is at least one clause (say C) in ϕ_2 in which all the literals are set to **true** under F.

Let $\overline{F}$ denote the conjugate of F. Since $F \models_n \phi_1$, it follows that $\overline{F} \models_n \phi_1$ and hence $\overline{F} \models \phi_1$. By the hypothesis, $\overline{F} \models \phi_2$.

However, all the literals in C must be set to **false** under $\overline{F}$ and hence $\overline{F} \not\models \phi_2$.

The above contradiction resulted out of the assumption that $F \not\models_n \phi_2$; the theorem follows. □

Definition 2.4 An algorithm $\mathcal{A}$, for a decision problem $\mathcal{D}$, is said to be a Monte Carlo algorithm with negative-sided error (MNE) if,

(i) For all "yes"-instances of $\mathcal{D}$, the probability that $\mathcal{A}$ errs is 0, and,
(ii) For all "no"-instances of $\mathcal{D}$, the probability that $\mathcal{A}$ errs is at most a fixed constant.

Assume that an MNE $\mathcal{A}$ is provided with an input instance, say x. If x is a "yes"-instance, then as per Definition 2.4, $\mathbf{Pr}(\mathcal{A}\ accepts\ x) = 1$. Likewise, if x is a "no"-instance, $\mathbf{Pr}(\mathcal{A}\ accepts\ x) \leq c$, where c is some fixed constant, $0 \leq c < 1$. It follows that when $\mathcal{A}$ rejects x, we can conclude that x is a "no"-instance with probability 1. However, if $\mathcal{A}$ accepts x, it is possible that x is a "no"-instance and $\mathcal{A}$ has erred, with the probability of $\mathcal{A}$ making an error, being at most c.

The algorithms discussed in Sects. 5 and 6 for the implication and equivalence problems are Monte Carlo algorithms with negative-sided error. In contrast, the random walk algorithm discussed in [3] for 2CNF satisfiability, is a Monte Carlo algorithm with positive-sided error.

2.1 Model of computation

Unlike the traditional RAM model of computation, in which the control unit has access to the formulas *in toto*, we consider a query-based form of access for the control unit. The control unit is permitted to query two independent oracles O_1 and O_2, with respect to a current assignment F. Oracle O_1 informs the control unit whether F satisfies ϕ_1 and if not returns a clause which is not satisfied by F. Likewise, oracle O_2 informs the control unit whether F falsifies ϕ_2 and if not returns some clause of ϕ_2. We also assume that the storage available to the control unit is limited, i.e., $O(1)$ units. Thus the control unit cannot build ϕ_1 or ϕ_2 using a collection of satisfiability queries.

When the control unit presents O_1 with an assignment that does not satisfy ϕ_1, O_1 returns a falsified clause uniformly at random from the set of falsified clauses. Likewise, when O_2 is presented with an assignment that satisfies ϕ_2, it (O_2) returns a clause from ϕ_2 that is chosen uniformly at random. The asymmetry between the values returned by the two oracles results because our search focusses on an assignment that simultaneously satisfies ϕ_1 and falsifies ϕ_2.

2.2 Assignment verification

Let $\vec{\mathbf{x}}$ denote a particular assignment to X; verifying that $\vec{\mathbf{x}} \models \phi_1$ is clearly a linear time operation.

Let ϕ_1 denote a satisfiable 2CNF formula. Let $\vec{\mathbf{r}}$ denote a partial assignment in which precisely two variables (say x_1 and x_2) are set to **false**. We can check whether $\vec{\mathbf{r}}$ can be completed to a *satisfying*, total assignment (denoted by $\vec{\mathbf{r}} \models_{(x_1=\textbf{false},x_2=\textbf{false})} \phi_1$) as follows:

(i) Each clause containing $\bar{x_1}$ or $\bar{x_2}$ is trivially satisfied.
(ii) If a clause contains both x_1 and x_2, then $\vec{\mathbf{r}}$ cannot be completed into a total, satisfying assignment.
(iii) A clause containing either x_1 or x_2 becomes a unit clause, which forces the remaining literal to **true**.

The above algorithm can be repeated till a contradiction is obtained in the form of setting a variable to both **true** and **false** or there is a subset ϕ_1' of ϕ_1 with no variables assigned. Inasmuch as ϕ_1 is satisfiable, so is ϕ_1' and hence we can conclude that $\vec{\mathbf{r}}$ can be completed into a satisfying total assignment. It is important to note that unit resolution in a 2CNF formula is a linear-time operation and hence the process of verifying whether a partial assignment can be completed into a satisfying total assignment takes linear time.

3 Motivation

There are three principal motivating factors for our work, viz.,

(i) Search through verification—In [1], a randomized algorithm was presented for the problem of computing the Minimum Spanning Tree (MST) of an undirected, weighted graph; this algorithm runs in linear time in the expected case and makes only verification calls, as opposed to the Kruskal approach of dynamically testing whether an edge creates cycles. In a similar way, we use only verification calls to compute an assignment that proves that one formula does not imply the other.
(ii) Learning with zero knowledge—In a seminal paper, Valiant [10] discusses the learnability of kCNF formulas in the Probably Approximate Correct (PAC) model. Within

that model, the oracle provides both positive and negative examples and the goal is to capture the concept approximately with high probability. In their paper, the concept is a kCNF formula; in this paper, the concept is 2CNF implication and the goal is to learn an assignment that disproves the implication with bounded error, using negative examples only. It must be noted that we do not learn the actual implication, only the assignment of interest; accordingly, our protocol needs $O(1)$ space.

(iii) Extending the limits of the random walk approach—The literature includes a number of papers that use random walk algorithms to solve constraint satisfaction problems. To the best of our knowledge, this paper represents the first work of its kind to *explicitly* consider implication through the framework of random walks.

4 Related work

The random walk approach for 2CNF satisfiability was first proposed by Papadimitriou [3]. In its essence, the technique consists of the following protocol:

Function SATISFIABILITY- TESTING(ϕ)
1: Start with an arbitrary assignment $\vec{\mathbf{x}}$ to the variables.
2: **if** ($\vec{\mathbf{x}} \models \phi$) **then**
3: **return**("ϕ is satisfiable").
4: **end if**
5: $count = 0$.
6: **while** (the current assignment is not satisfying) **and** ($count \leq 2 \cdot n^2$) **do**
7: Pick an unsatisfied clause.
8: Uniformly at random flip the value assigned to one of its two literals (variables).
9: $count = count + 1$.
10: **if** ($\vec{\mathbf{x}} \models \phi$) **then**
11: **return**("ϕ is satisfiable").
12: **end if**
13: **end while**
14: **return**("ϕ is probably unsatisfiable").

Algorithm 1: Papadimitrious's randomized algorithm for 2CNF Satisfiability

Inasmuch as with probability at least one-half, the protocol chooses the "correct" literal to flip, it can be easily modeled as a one dimensional random walk with one absorbing barrier and one reflecting barrier. As per the theory of random walks, on a satisfiable 2CNF instance, the protocol finds a satisfying assignment in n^2 iterations (expected), where n is the number of variables in the 2CNF formula. Applying Markov's inequality [2], we can conclude that the probability of not finding a satisfiable assignment in $2 \cdot n^2$ iterations is less than one-half.

Note that Algorithm 1 is a Monte Carlo algorithm with positive-sided error in that when it says ϕ is satisfiable, there is no error. However, if it states that ϕ is unsatisfiable, it is possible that ϕ is satisfiable and Algorithm 1 has erred, with the probability of error being at most one-half.

A variant of Algorithm 1 can actually be used to solve the implication problem, if ϕ_1 is in CNF and ϕ_2 is in DNF. Consider the protocol represented by Algorithm 2.

Theorem 4.1 *Assume that $\phi_1 \not\rightarrow \phi_2$. The probability that Algorithm 2 does not discover an assignment falsifying the implication is less than one-half.*

Function CNF- DNF- IMPLICATION- TESTING(ϕ_1, ϕ_2)
1: {We reiterate that ϕ_1 is 2CNF and ϕ_2 is 2DNF.}
2: Start with an arbitrary assignment to the variables, say $\vec{\mathbf{x}}$.
3: {We focus on an assignment that satisfies ϕ_1 and falsifies ϕ_2. Note that if $\phi_1 \not\rightarrow \phi_2$, then such an assignment must exist.}
4: **if** $((\vec{\mathbf{x}} \models \phi_1)$ **and** $(\vec{\mathbf{x}} \not\models \phi_2))$ **then**
5: **return**($\phi_1 \not\rightarrow \phi_2$)
6: **end if**
7: $count = 0$.
8: **while** $((\vec{\mathbf{x}} \not\models \phi_1)$ **or** $(\vec{\mathbf{x}} \models \phi_2))$ **and** $(count \leq 2 \cdot n^2)$ **do**
9: **if** $(\vec{\mathbf{x}} \not\models \phi_1)$ **then**
10: Pick an unsatisfied clause in ϕ_1.
11: Uniformly at random flip the value assigned to one of its two literals (variables).
12: $count = count + 1$.
13: **else**
14: **if** $(\vec{\mathbf{x}} \models \phi_2)$ **then**
15: Pick any satisfied conjunct in ϕ_2.
16: Uniformly at random flip the value assigned to one of its two literals (variables).
17: $count = count + 1$.
18: **else**
19: **return**("$\phi_1 \not\rightarrow \phi_2$.")
20: **end if**
21: **end if**
22: **if** $((\vec{\mathbf{x}} \models \phi_1)$ **and** $(\vec{\mathbf{x}} \not\models \phi_2))$ **then**
23: **return**("$\phi_1 \not\rightarrow \phi_2$.")
24: **end if**
25: **end while**
26: **return**("ϕ_1 probably implies ϕ_2.")

Algorithm 2: Random walk for 2CNF-2DNF implication

Proof In proof, we mimic the argument used in [4]. Assume that $\phi_1 \not\rightarrow \phi_2$ and focus on the unique assignment $\hat{T}$ that simultaneously satisfies ϕ_1 and falsifies ϕ_2. Let T denote the current assignment; assume that this assignment differs from $\hat{T}$ on i variables.

Assume that T falsifies ϕ_1. In this case, Steps (10)–(12) of Algorithm 2 are executed and in Step (10) an unsatisfied clause, say C, is picked. Observe that T has set at least one of the two literals in C incorrectly with respect to $\hat{T}$. Hence choosing one of them uniformly at random and flipping the same, moves T closer to $\hat{T}$ by one variable, with probability at least one-half. Likewise, it could also move T away from $\hat{T}$ by one variable, with probability at most one-half.

Assume that T satisfies both ϕ_1 and ϕ_2. In this case, Steps (15)–(17) of Algorithm 2 are executed. Note that under $\hat{T}$, every conjunct is false. Thus, at least one of the two literals in each satisfied conjunct has been incorrectly set under T. Consequently, choosing one of the two literals associated with a satisfied conjunct, uniformly at random, and flipping it, permits T to move closer to $\hat{T}$ by one variable, with probability at least one-half. Likewise, T moves away from $\hat{T}$ by one variable, with probability at most one-half.

It is now straightforward to see that Algorithm 2 can be modeled as a one-dimensional random walk with a reflecting barrier (T differs from $\hat{T}$ on all n variables) and an absorbing barrier (T and $\hat{T}$ coincide). The theorem follows [5]. □

Corollary 4.1 *Algorithm 2 is a Monte Carlo algorithm with negative-sided error for the 2CNF-2DNF implication problem. If the antecedent implies the consequent, there is no error. In the event that the antecedent does not imply the consequent, the probability that an error is made is at most one-half. Alternatively, if Algorithm 2 states that the antecedent does*

not imply the consequent, there is no error. However, if Algorithm 2 states that the antecedent implies the consequent, the probability of error is at most one-half.

Proof Follows from the arguments above. □

The arguments used in the discussions above should also make it clear why the random walk approach cannot be directly used for 2CNF implication, when ϕ_2 is in CNF form. In this case, when ϕ_2 is satisfied, we cannot simply pick a satisfied clause and flip its literals. Recall that a 2CNF formula can be falsified with a single unsatisfied clause and hence picking any satisfied clause from the set of all satisfied clauses may not be helpful, since *both* the literals corresponding to the picked clause *may have been correctly set under* $\hat{T}$.

Applications of the random walk strategy to harder versions of satisfiability with detailed implementation profiles are described in [11]. In particular, they consider the efficacy of biasing the process of selecting the clause which is satisfied in the next round. [8] extended the ideas presented by Papadimitriou [3] to derive a randomized algorithm for the Q2SAT problem. Local search through random walks has also been studied for harder Satisfiability problems [6].

5 Randomized implication algorithms

Algorithm 3 is a randomized implication algorithm for testing whether ϕ_1 implies ϕ_2, where both ϕ_1 and ϕ_2 are 2CNF formulas on the same n variables. We assume that ϕ_1 has m clauses and ϕ_2 has m' clauses.

Function CNF- CNF- IMPLICATION- TESTING(ϕ_1, ϕ_2)
1: {We reiterate that both ϕ_1 and ϕ_2 are 2CNF formulas.}
2: Verify the satisfiability of ϕ_1 using the random walk technique described in Algorithm 1.
3: **if** (ϕ_1 is unsatisfiable) **then**
4: **return** $(\phi_1 \rightarrow \phi_2)$.
5: **end if**
6: $count = 0$.
7: Let $\vec{\mathbf{x}}_{count}$ denote an arbitrary assignment that satisfies ϕ_1.
8: **if** $(\vec{\mathbf{x}}_{count} \not\models \phi_2)$ **then**
9: **return**("$\phi_1 \not\rightarrow \phi_2$")
10: **end if**
11: **while** $(count \leq m')$ **do**
12: $count = count + 1$.
13: Uniformly at random pick a clause C from ϕ_2.
14: Let x_r and x_s denote the two literals in C.
15: Without loss of generality, assume that $x_r =$ **true** and $x_s =$ **false**.
16: Construct the *partial* assignment $\vec{\mathbf{x}}_{count}$ in which $x_r =$ **false** and $x_s =$ **false**. {This ensures that $\vec{\mathbf{x}}_{count} \not\models_{(x_r=\textbf{false},x_s=\textbf{false})} C$ and hence $\vec{\mathbf{x}}_{count} \not\models_{(x_r=\textbf{false},x_s=\textbf{false})} \phi_2$.}
17: **if** $(\vec{\mathbf{x}}_{count} \models_{(x_r=\textbf{false},x_s=\textbf{false})} \phi_1)$ **then**
18: **return**("$\phi_1 \not\rightarrow \phi_2$.")
19: **else**
20: $\vec{\mathbf{x}}_{count} = \vec{\mathbf{x}}_{count-1}$.
21: Note that $\vec{\mathbf{x}}_{count}$ is now a total assignment that satisfies ϕ_2.
22: **end if**
23: **end while**
24: **return**("ϕ_1 probably implies ϕ_2.")

Algorithm 3: Randomized algorithm for checking whether $\phi_1 \rightarrow \phi_2$

5.1 Analysis

Let $V(m,n)$ denote the time required by oracle O_1 to verify that a given assignment $\vec{\mathbf{x}} \models \phi_1$. Likewise, $V(m',n)$ denotes the time required by oracle O_2 to verify that a given assignment $\vec{\mathbf{x}} \models \phi_2$. From the discussion in Sect. 2.2, $V(m,n) = O(m+n)$ and $V(m',n) = O(m'+n)$.

As per the discussion above, Steps (1) through (4) can be accomplished in $2 \cdot n^2 \cdot V(m,n)$ time. Step (8) takes $V(m',n)$ time. Finally, observe that the **while** loop takes $m' \cdot V(m,n)$ time, since constructing the total assignment from a partial assignment (Step (17)) is equivalent to assignment verification, as per Sect. 2.2. Thus, the total time taken by Algorithm 3 is $O(n^2 \cdot V(m,n) + V(m',n) + m' \cdot V(m,n))$.

5.2 Correctness

We need the following technical lemma before analyzing the correctness of Algorithm 3.

Lemma 5.1 *Given two CNF formulas ϕ_1 and ϕ_2, $\phi_1 \rightarrow \phi_2$ if and only if $\phi_1 \rightarrow C_i$, for every clause C_i in ϕ_2.*

Proof If $\phi_1 \rightarrow \phi_2$, then clearly $\phi_1 \rightarrow C_i$, for each clause C_i in ϕ_2.

To see the converse, we use the tautology that $[(P \rightarrow A) \wedge (P \rightarrow B)] \rightarrow (P \rightarrow (A \wedge B))$ and conclude that if $\phi_1 \rightarrow C_i$, for each clause C_i in ϕ_2, then $\phi_1 \rightarrow \phi_2$. □

Observe that Algorithm 3 is a Monte Carlo algorithm with negative-sided error. If ϕ_1 does imply ϕ_2, there is no error since there is no assignment that simultaneously satisfies ϕ_1 and falsifies ϕ_2. On the other hand, if ϕ_1 does not imply ϕ_2, it is possible that Algorithm 3 makes an error. Alternatively, when Algorithm 3 states that $\phi_1 \not\rightarrow \phi_2$, the probability of error is 0. However, if it states that $\phi_1 \rightarrow \phi_2$, it could be the case that $\phi_1 \not\rightarrow \phi_2$ and Algorithm 3 has erred. We now proceed to bound the probability that Algorithm 3 has erred.

Assume that $\phi_1 \not\rightarrow \phi_2$.

There are two ways in which Algorithm 3 can state that $\phi_1 \rightarrow \phi_2$:

(i) Algorithm 3 declares that ϕ_1 is unsatisfiable. We denote this event by A; note that $\mathbf{Pr}(A) < \frac{1}{2}$.

(ii) Algorithm 3 declares that ϕ_1 is satisfiable, but does not find the witness assignment that simultaneously satisfies ϕ_1 and falsifies ϕ_2. Since $\phi_1 \not\rightarrow \phi_2$, as per Lemma 5.1, there must be a clause C in ϕ_2 such that $\phi_1 \not\rightarrow C$. We denote by B, the event that Algorithm 3 could not find the clause C such that $\phi_1 \not\rightarrow C$. The probability that this clause is not picked in the **while** loop is at most $(1 - \frac{1}{m'})^{m'} \leq e^{-1} < \frac{1}{2}$, i.e., $\mathbf{Pr}(B) < \frac{1}{2}$.

Observe that the failure of Algorithm 3 can be characterized by the event $A \cup (A^c \cap B)$, where A^c denotes the complement of the event A.

Now,

$$\begin{aligned}
\mathbf{Pr}(A \cup (A^c \cap B)) &= \mathbf{Pr}(A) + \mathbf{Pr}(A^c \cap B), \text{ mutual exclusivity} \\
&= \mathbf{Pr}(A) + \mathbf{Pr}(A^c) \cdot \mathbf{Pr}(B), \text{ since B and A are independent} \\
&= \mathbf{Pr}(A) + (1 - \mathbf{Pr}(A)) \cdot \mathbf{Pr}(B) \\
&\leq \frac{3}{4}
\end{aligned}$$

Thus, we have established that the probability that Algorithm 3 errs is at most 3/4.

Corollary 5.1 *Algorithm 3 is a Monte Carlo algorithm for the 2CNF implication problem with negative-sided error; the probability that it errs on an input instance is at most* 3/4.

Proof Follows from the discussion above. □

5.3 Extension to 2CNF equivalence

We now consider the equivalence problem: Does $\phi_1 \leftrightarrow \phi_2$?

We know that $(\phi_1 \leftrightarrow \phi_2) \leftrightarrow [(\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)]$.

Consider the protocol represented by Algorithm 4.

```
Function EQUIVALENCE- TESTING(φ1, φ2)
1: if (CNF- CNF- IMPLICATION- TESTING(φ1, φ2)) then
2:    {If Algorithm 3 states that φ1 → φ2}
3:    if (CNF- CNF- IMPLICATION- TESTING(φ2, φ1)) then
4:       {If Algorithm 3 states that φ2 → φ1}
5:       return("φ1 is probably equivalent to φ2").
6:    else
7:       return("φ1 ↛ φ2").
8:    end if
9: else
10:    return("φ1 ↛ φ2").
11: end if
```

Algorithm 4: A Monte Carlo algorithm for 2CNF equivalence

Algorithm 4 first queries Algorithm 3 as to whether $\phi_1 \rightarrow \phi_2$. If the answer is negative, it states that $\phi_1 \not\leftrightarrow \phi_2$. If the answer is affirmative, it queries Algorithm 3 as to whether $\phi_2 \rightarrow \phi_1$. If the answer is affirmative, it states that $\phi_1 \leftrightarrow \phi_2$; otherwise, it states that $\phi_1 \not\leftrightarrow \phi_2$. It is to be noted that the queries are *independent*.

If $\phi_1 \leftrightarrow \phi_2$, there is no error since there is no assignment that simultaneously satisfies ϕ_1 and falsifies ϕ_2 or simultaneously satisfies ϕ_2 and falsifies ϕ_1. On the other hand, if $\phi_1 \not\leftrightarrow \phi_2$, Algorithm 4 will err, if Algorithm 3 states that $\phi_1 \rightarrow \phi_2$ and $\phi_2 \rightarrow \phi_1$. We now proceed to bound the probability that Algorithm 4 has erred.

Assume that $\phi_1 \not\leftrightarrow \phi_2$ and consider the following three mutually exclusive cases:

(i) $\phi_1 \not\rightarrow \phi_2$ and $\phi_2 \not\rightarrow \phi_1$. In this case, the probability that Algorithm 3 states that $\phi_1 \rightarrow \phi_2$ and $\phi_2 \rightarrow \phi_1$ is at most $\frac{3}{4} \cdot \frac{3}{4} = \frac{9}{16}$. It follows that the probability that Algorithm 4 has erred is at most $\frac{9}{16}$.

(ii) $\phi_1 \not\rightarrow \phi_2$ and $\phi_2 \rightarrow \phi_1$. Observe that in this case, the probability that Algorithm 4 errs is precisely the probability that Algorithm 3 errs on the query $\phi_1 \rightarrow \phi_2$, which as discussed before is at most $\frac{3}{4}$.

(iii) $\phi_1 \rightarrow \phi_2$ and $\phi_2 \not\rightarrow \phi_1$. Arguing in identical fashion as Case (ii), we conclude that the probability that Algorithm 4 errs is at most $\frac{3}{4}$.

Hence, regardless of the input, if $\phi_1 \not\leftrightarrow \phi_2$, the probability that Algorithm 4 errs is at most 3/4.

Corollary 5.2 *Algorithm 4 is a Monte Carlo algorithm for the 2CNF equivalence problem with negative-sided error; the probability that it errs on an input instance is at most* 3/4.

Proof Follows from the discussion above. □

6 Nae-implication and Nae-equivalence

In [9], we showed that random walks can be used to determine the Nae-satisfiability of a 2CNF formula; the algorithm itself is not too different from the algorithm in [3]. However, we were able to exploit the presence of conjugate pairs of Nae-satisfying assignments for a Nae-satisfiable formula to derive sharper convergence bounds. The bounds used in this paper can be found in [7], which is a recent revision of [9].

6.1 Assignment Nae-verification

Let $\vec{\mathbf{x}}$ denote a particular assignment to X; verifying that $\vec{\mathbf{x}} \models_n \phi_1$ is a clearly a linear time operation.

Let ϕ_1 denote a satisfiable 2CNF formula; let $\vec{\mathbf{r}}$ denote a partial assignment in which precisely two variables (say x_1 and x_2) are set to **false**. We can check whether $\vec{\mathbf{r}}$ can be completed to a *Nae-satisfying*, total assignment (denoted by $\vec{\mathbf{r}} \models_{n(x_1=\textbf{false},x_2=\textbf{false})} \phi_1$) as follows:

(i) If a clause contains both x_1 and x_2, then $\vec{\mathbf{r}}$ cannot be completed into a total, Nae-satisfying assignment.
(ii) If a clause contains both $\bar{x_1}$ and $\bar{x_2}$, then $\vec{\mathbf{r}}$ cannot be completed into a total, Nae-satisfying assignment.
(iii) If a clause contains both x_1 and $\bar{x_2}$, or, both $\bar{x_1}$ and x_2, then the clause is Nae-satisfied.
(iv) A clause containing either x_1 or x_2 becomes a unit clause, which forces the remaining literal to **true**.
(v) A clause containing either $\bar{x_1}$ or $\bar{x_2}$ becomes a unit clause, which forces the remaining literal to **false**.

The above algorithm can be repeated till a contradiction is obtained in the form of setting a variable to both **true** and **false** or there is a subset ϕ_1' of ϕ_1 with no variables assigned. Inasmuch as ϕ_1 is Nae-satisfiable, so is ϕ_1' and hence we can conclude that $\vec{\mathbf{r}}$ can be completed into a Nae-satisfying total assignment. It is important to note that the above procedure can be implemented in linear time, i.e., the process of verifying whether a partial assignment can be completed into a Nae-satisfying total assignment takes linear time.

Similarly, we can check whether $\vec{\mathbf{r}} \models_{n(x_1=\textbf{true},x_2=\textbf{true})} \phi_1$ in linear time.

6.2 Analysis

Let $V(m, n)$ denote the time required by oracle O_1 to verify that a given assignment $\vec{\mathbf{x}} \models_n \phi_1$. Likewise, $V(m', n)$ denotes the time required by oracle O_2 to verify that a given assignment $\vec{\mathbf{x}} \models_n \phi_2$. From the discussion in Sect. 6.1, $V(m, n) = O(m+n)$ and $V(m', n) = O(m'+n)$.

As per the discussion in [7], Steps (1) through (4) can be accomplished in $\frac{5}{4} \cdot n^2 \cdot V(m, n)$ time. Step (7) takes $V(m', n)$ time. Finally, observe that the **while** loop takes $2 \cdot m' \cdot V(m, n)$ time, since constructing the total assignment from a partial assignment (Steps (15) and (19)) is equivalent to assignment Nae-verification, as per Sect. 6.1. Thus, the total time taken by Algorithm 5 is $O(n^2 \cdot V(m, n) + V(m', n) + m' \cdot V(m, n))$.

6.3 Correctness

As was the case with Algorithm 3, we need the following technical lemma before analyzing the correctness of Algorithm 5.

Function NAE- IMPLICATION TESTING(ϕ_1, ϕ_2)
1: Verify the Nae-satisfiability of ϕ_1 using the algorithm in [7].
2: **if** (ϕ_1 is Nae-unsatisfiable) **then**
3: **return** ($\phi_1 \rightarrow_n \phi_2$).
4: **end if**
5: $count = 0$.
6: Let $\vec{\mathbf{x}}_{count}$ denote an arbitrary assignment, that Nae-satisfies ϕ_1.
7: **if** ($\vec{\mathbf{x}}_{count} \not\models_n \phi_2$) **then**
8: **return**("$\phi_1 \not\rightarrow_n \phi_2$")
9: **end if**
10: **while** ($count \leq m'$) **do**
11: Uniformly at random pick a clause C from ϕ_2.
12: $count = count + 1$.
13: Let x_r and x_s denote the two literals in C.
14: Without loss of generality, assume that $x_r =$ **true** and $x_s =$ **false**.
15: Construct the *partial* assignment $\vec{\mathbf{x}}_{count}$ in which $x_r =$ **false** and $x_s =$ **false**. {This ensures that $\vec{\mathbf{x}}_{count} \not\models_{n(x_r=\textbf{false},x_s=\textbf{false})} C$ and hence $\vec{\mathbf{x}}_{count} \not\models_{n(x_r=\textbf{false},x_s=\textbf{false})} \phi_2$.}
16: **if** ($\vec{\mathbf{x}}_{count} \models_{n(x_r=\textbf{false},x_s=\textbf{false})} \phi_1$) **then**
17: **return**("$\phi_1 \not\rightarrow_n \phi_2$.")
18: **else**
19: Construct the *partial* assignment $\vec{\mathbf{x}}_{count}$ in which $x_r =$ **true** and $x_s =$ **true**. {This ensures that $\vec{\mathbf{x}}_{count} \not\models_{n(x_r=\textbf{true},x_s=\textbf{true})} C$ and hence $\vec{\mathbf{x}}_{count} \not\models_{n(x_r=\textbf{true},x_s=\textbf{true})} \phi_2$.}
20: **if** ($\vec{\mathbf{x}}_{count} \models_{n(x_r=\textbf{true},x_s=\textbf{true})} \phi_1$) **then**
21: **return**("$\phi_1 \not\rightarrow_n \phi_2$.")
22: **else**
23: $\vec{\mathbf{x}}_{count} = \vec{\mathbf{x}}_{count-1}$.
24: Note that $\vec{\mathbf{x}}_{count}$ is now a total assignment that Nae-satisfies ϕ_2.
25: **end if**
26: **end if**
27: **end while**
28: **return**("ϕ_1 probably Nae-implies ϕ_2.")

Algorithm 5: Randomized algorithm for checking whether $\phi_1 \rightarrow_n \phi_2$

Lemma 6.1 *Given two CNF formulas ϕ_1 and ϕ_2, $\phi_1 \rightarrow_n \phi_2$ if and only if $\phi_1 \rightarrow_n C_i$, for every clause C_i in ϕ_2.*

Proof If $\phi_1 \rightarrow_n \phi_2$, then clearly $\phi_1 \rightarrow_n C_i$ for each clause C_i in ϕ_2.

To see the converse, we use the tautology that $[(P \rightarrow_n A) \wedge (P \rightarrow_n B)] \rightarrow (P \rightarrow_n (A \wedge B))$ and conclude that if $\phi_1 \rightarrow_n C_i$, for each clause C_i in ϕ_2, then $\phi_1 \rightarrow_n \phi_2$. □

Observe that Algorithm 5 is a Monte Carlo algorithm with negative-sided error. If ϕ_1 does Nae-imply ϕ_2, there is no error since there is no assignment that simultaneously Nae-satisfies ϕ_1 and Nae-falsifies ϕ_2. On the other hand, if ϕ_1 does not Nae-imply ϕ_2, it is possible that Algorithm 5 makes an error. Alternatively, when Algorithm 5 states that $\phi_1 \not\rightarrow_n \phi_2$, the probability of error is 0. However, if it states that $\phi_1 \rightarrow_n \phi_2$, it could be the case that $\phi_1 \not\rightarrow_n \phi_2$ and Algorithm 5 has erred. We now proceed to bound the probability that Algorithm 5 has erred.

Assume that $\phi_1 \not\rightarrow_n \phi_2$.

There are two ways in which Algorithm 5 can state that $\phi_1 \rightarrow_n \phi_2$:

(i) Algorithm 5 declares that ϕ_1 is Nae-unsatisfiable. We denote this event by A; note that $\mathbf{Pr}(A) < \frac{1}{24}$ (see [7]).
(ii) Algorithm 5 declares that ϕ_1 is Nae-satisfiable, but does not find the witness assignment that simultaneously Nae-satisfies ϕ_1 and Nae-falsifies ϕ_2. Since $\phi_1 \not\rightarrow_n \phi_2$, as per Lemma 6.1, there must be a clause C in ϕ_2 such that $\phi_1 \not\rightarrow_n C$. We denote by B, the

event that Algorithm 5 could not find the clause C such that $\phi_1 \not\rightarrow_n C$. The probability that this clause is not picked in the **while** loop is at most $(1 - \frac{1}{m'})^{m'} \leq e^{-1} < \frac{1}{2}$, i.e., $\mathbf{Pr}(B) < \frac{1}{2}$.

Observe that the failure of Algorithm 5 can be characterized by the event $A \cup (A^c \cap B)$, where A^c denotes the complement of the event A.

Now,

$$\begin{aligned}
\mathbf{Pr}(A \cup (A^c \cap B)) &= \mathbf{Pr}(A) + \mathbf{Pr}(A^c \cap B), \text{ mutual exclusivity} \\
&= \mathbf{Pr}(A) + \mathbf{Pr}(A^c) \cdot \mathbf{Pr}(B), \text{ since B and A are independent} \\
&= \mathbf{Pr}(A) + (1 - \mathbf{Pr}(A)) \cdot \mathbf{Pr}(B) \\
&\leq \frac{1}{24} + \frac{23}{24} \cdot \frac{1}{2} \\
&= \frac{25}{48}
\end{aligned}$$

Thus, we have established that the probability that Algorithm 5 errs is at most 25/48.

Corollary 6.1 *Algorithm 5 is a Monte Carlo algorithm for the 2CNF Nae-implication problem with negative-sided error; the probability that it errs on an input instance is at most* 25/48.

Proof Follows from the discussion above. □

6.4 Extension to Nae-equivalence

Now we consider the Nae-equivalence problem: Does $\phi_1 \leftrightarrow_n \phi_2$?

We know that $(\phi_1 \leftrightarrow_n \phi_2) \leftrightarrow [(\phi_1 \rightarrow_n \phi_2) \wedge (\phi_2 \rightarrow_n \phi_1)]$.

Consider the protocol represented by Algorithm 6.

```
Function NAE- EQUIVALENCE- TESTING(φ1, φ2)
1: if (CNF- CNF- NAE- IMPLICATION- TESTING(φ1, φ2)) then
2:    {If Algorithm 5 states that φ1 →n φ2}
3:    if (CNF- CNF- NAE- IMPLICATION- TESTING(φ2, φ1)) then
4:       {If Algorithm 5 states that φ2 →n φ1}
5:       return("φ1 is probably Nae-equivalent to φ2").
6:    else
7:       return("φ1 ↛n φ2").
8:    end if
9: else
10:    return("φ1 ↛n φ2").
11: end if
```

Algorithm 6: Randomized algorithm for 2CNF Nae-Equivalence

Algorithm 6 first queries Algorithm 5 as to whether $\phi_1 \rightarrow_n \phi_2$. If the answer is negative, it states that $\phi_1 \not\rightarrow_n \phi_2$. If the answer is affirmative, it queries Algorithm 5 as to whether $\phi_2 \rightarrow_n \phi_1$. If the answer is affirmative, it states that $\phi_1 \leftrightarrow_n \phi_2$; otherwise, it states that $\phi_1 \not\leftrightarrow_n \phi_2$. It is to be noted that the queries are *independent*.

If $\phi_1 \leftrightarrow_n \phi_2$, there is no error since there is no assignment that simultaneously Nae-satisfies ϕ_1 and Nae-falsifies ϕ_2 or simultaneously Nae-satisfies ϕ_2 and Nae-falsifies ϕ_1. On

the other hand, when $\phi_1 \not\rightarrow_n \phi_2$, Algorithm 6 will err, if Algorithm 5 states that $\phi_1 \rightarrow_n \phi_2$ and $\phi_2 \rightarrow_n \phi_1$. We now proceed to bound the probability that Algorithm 6 has erred.

Assume that $\phi_1 \not\rightarrow_n \phi_2$ and consider the following three mutually exclusive cases:

(i) $\phi_1 \not\rightarrow_n \phi_2$ and $\phi_2 \not\rightarrow_n \phi_1$. In this case, the probability that Algorithm 5 states that $\phi_1 \rightarrow_n \phi_2$ and $\phi_2 \rightarrow_n \phi_1$ is at most $(25/48)(25/48) = (625/2304)$. It follows that the probability that Algorithm 6 has erred is at most $625/2304$.

(ii) $\phi_1 \not\rightarrow_n \phi_2$ and $\phi_2 \rightarrow_n \phi_1$. Observe that in this case, the probability that Algorithm 6 errs is precisely the probability that Algorithm 5 errs on the query $\phi_1 \rightarrow_n \phi_2$, which as discussed before is at most $25/48$.

(iii) $\phi_1 \rightarrow_n \phi_2$ and $\phi_2 \not\rightarrow_n \phi_1$. Arguing in identical fashion as Case (ii), we conclude that the probability that Algorithm 6 errs is at most $25/48$.

Hence, regardless of the input, if $\phi_1 \not\rightarrow_n \phi_2$, the probability that Algorithm 6 errs is at most $25/48$.

Corollary 6.2 *Algorithm 6 is a Monte Carlo algorithm for the 2CNF Nae-equivalence problem with negative-sided error; the probability that it errs on an input instance is at most* $25/48$.

Proof Follows from the discussion above. □

7 Conclusion

In this paper, we designed and analyzed a random walk algorithm for the 2CNF implication and Nae-implication problems. The existence of randomized algorithms for these problems is not surprising, since both these problems are known to be in P. However, our analysis was conducted in a non-standard model of computation, which finds applications in learning theory. It is important to reiterate the fact that our algorithms are zero-knowledge, in that no attempt is made to construct the clausal formulas.

The following open questions have arisen as a consequence of our research in this paper:

(i) Can we reduce the number of random bits in our protocols? Both theoreticians and practitioners of randomized algorithms are aware of the difficulties associated with obtaining a source of unbiased random bits. Our protocols require $\Omega(n^2)$ random bits; it would be interesting to see if the required number of random bits can be reduced to $O(n \cdot \log n)$ with a bounded loss of accuracy?

(ii) How do our algorithms perform on practical instances? An empirical study of our algorithms would be very instructive from the practical perspective. To wit, random walks and their variants have been used to solve hard instance of the CNF satisfiability problem with reasonable success [11].

References

1. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. J. ACM **42**(2), 321–328 (1995)
2. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
3. Papadimitriou, C.H.: On selecting a satisfying truth assignment. In: IEEE, editor, Proceedings: 32nd annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, October 1–4, 1991, pp. 163–169, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1991. IEEE Computer Society Press

4. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, New York (1994)
5. Ross, S.M.: Probability Models, 7th edn. Academic Press, Inc., London (2000)
6. Schöning, U.: New algorithms for k-SAT based on the local search principle. In: MFCS: Symposium on Mathematical Foundations of Computer Science (2001)
7. Subramani, K., Gu, X.: Absorbing random walks and the nae2sat problem. Fundamenta Informatica. (2008, Submitted)
8. Subramani, K.: Cascading random walks. Int. J. Found. Comput. Sci. (IJFCS) **16**(3), 599–622 (2005)
9. Subramani, K.: Absorbing random walks and the nae2sat problem. In: Preparata, F., et al. (eds.) Proceedings of the 2nd Annual International Frontiers of Algorithmics Workshop. Lecture Notes in Computer Science, vol. 5059, pp. 89–100. Springer, Heidelberg (2008)
10. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)
11. Wei, W., Selman, B.: Accelerating random walks. In: The Eighth International Conference on Constraint Programming (CP), LNCS, pp. 216–232 (2002)