Minimum Witnesses for Unsatisfiable 2CNFs

Joshua Buresh-Oppenheim and David Mitchell

Simon Fraser University
jburesho@cs.sfu.ca, mitchell@cs.sfu.ca

Abstract. We consider the problem of finding the smallest proof of unsatisfiability of a 2CNF formula. In particular, we look at Resolution refutations and at minimum unsatisfiable subsets of the clauses of the CNF. We give a characterization of minimum tree-like Resolution refutations that explains why, to find them, it is not sufficient to find shortest paths in the implication graph of the CNF. The characterization allows us to develop an efficient algorithm for finding a smallest tree-like refutation and to show that the size of such a refutation is a good approximation to the size of the smallest general refutation. We also give a polynomial time dynamic programming algorithm for finding a smallest unsatisfiable subset of the clauses of a 2CNF.

1 Introduction

Two important areas of SAT research involve identification of tractable cases, and the study of minimum length proofs for interesting formulas. Resolution is the most studied proof system, in part because it is among the most amenable to analysis, but also because it is closely related to many important algorithms. The two most important tractable cases of SAT, 2-SAT and Horn-SAT, have linear time algorithms that can be used to produce linear-sized Resolution refutations of unsatisfiable formulas. However, for Horn formulas it is not possible even to approximate the minimum refutation size within any constant factor, unless P=NP [1]. Here, we consider the question of finding minimum-size Resolution refutations, both general and tree-like, for 2-SAT.

The linear-time 2-SAT algorithm of [2] is based on the implication graph, a directed graph on the literals of the CNF. It seems plausible that finding a minimum tree-like Resolution refutation would amount to finding shortest paths in the implication graph. This approach is proposed in [3], but is incorrect. Hence, while [3] correctly states that finding a minimum tree-like refutation can be done in polytime, the proof is flawed. We show that a different notion of shortest path is needed, and give an $O(n^2(n+m))$-time algorithm based on BFS. We also show that such a refutation is at most twice as large as the smallest general Resolution refutation and that there are cases where this bound is tight. This contrasts with the above-mentioned inapproximability in the Horn case.

Since 2-SAT is linear time, the formula itself, or any unsatisfiable subset of its clauses, is an efficiently checkable certificate of unsatisfiability. For the question of finding a minimum unsatisfiable subset of a set of 2-clauses, analysis of certain types of paths in the implication graph again allows us to develop a polytime algorithm. This is interesting in light the fact that finding a maximum satisfiable subset of the clauses of a 2CNF is NP-hard, even to approximate. Perhaps surprisingly, a minimum tree-like Resolution refutation of a 2CNF is not necessarily a refutation of a minimum unsatisfiable subformula. This also seems to be the case with minimum general Resolution refutations.

2 Preliminaries and Characterization

Throughout, let $\mathcal{C}$ be a collection of 2-clauses over the variables $\{x_1, ..., x_n\}$. Say $|\mathcal{C}| = m$. As first suggested by [2], $\mathcal{C}$ can be represented as a directed graph $G_{\mathcal{C}}$ on $2n$ nodes, one for each literal. If $(a \vee b) \in \mathcal{C}$ for literals a, b, then the edges $(\bar{a}, b)$ and $(\bar{b}, a)$ appear in $G_{\mathcal{C}}$ (note that literals a and b can be the same). Both of these edges are labelled by the clause $(a \vee b)$. For an edge $e = (a, b)$, let $dual(e)$, the dual edge of e, be the edge $(\bar{b}, \bar{a})$.

Consider a directed path P in $G_{\mathcal{C}}$ (that is, a sequence of not-necessarily-distinct directed edges). Note that in $G_{\mathcal{C}}$ even a simple path may contain two edges with the same clause label. Let $set(P)$ denote the set of clause-labels underlying the edges of P. We define $|P|$, the *size* of the path P, to be $|set(P)|$. In contrast, let $length(P)$ denote the length of P as a sequence. Call a path P *singular* if it does not contain two edges that have the same clause label. For any singular path P, $|P| = length(P)$.

For literals a, b, define $\mathcal{P}_{ab}$ to be the set of all simple, directed paths from a to b in $G_{\mathcal{C}}$. If c is also a literal, let $\mathcal{P}_{abc}$ be the set of all simple, directed paths that start at a, end at c and visit b at some point. Let $P \in \mathcal{P}_{ab}$. We say P is *minimum* if it has minimum size among all paths in $\mathcal{P}_{ab}$.

Proposition 1 ([2]). *If $\mathcal{C}$ is unsatisfiable, then there is a variable x such that there is a path from x to $\bar{x}$ and a path from $\bar{x}$ to x in $G_{\mathcal{C}}$. Furthermore, for any Resolution derivation of the clause $(\bar{a} \vee b)$ ($\bar{a}$ and b need not be distinct) there must a path $P \in \mathcal{P}_{ab}$ whose labels are contained in the axioms of this derivation.*

Let $P \in \mathcal{P}_{ab}$. Let $IR(P)$ be the Input Resolution derivation that starts by resolving the clauses labelling the first two edges in P and then proceeds by resolving the latest derived clause with the clause labelling the next edge in the sequence P. This is a derivation of either $(\bar{a} \vee b)$ or simply (b). It is not hard to see that the size of the derivation $IR(P)$ is $2 \cdot length(P) - 1$.

For a path $P = (e_1, ..., e_k) \in \mathcal{P}_{ab}$, let $dual(P) \in \mathcal{P}_{\bar{b}\bar{a}}$ be the path ($dual(e_k)$, ..., $dual(e_1)$). Let $suf(P)$ be the maximal singular suffix of P (as a sequence). Similarly, let $pre(P)$ be the maximal singular prefix of P. For a simple path $P \in \mathcal{P}_{ab\bar{b}}$, let $extend(P)$ be the following path in $\mathcal{P}_{a\bar{a}}$: let P' be the portion of P that starts at a and ends at b. Then $extend(P)$ is the sequence P concatenated with the sequence $dual(P')$. If $P \in \mathcal{P}_{a\bar{a}b}$, then $extend(P) \in \mathcal{P}_{\bar{b}b}$ is defined similarly.

Proposition 2. *Let x be a literal and let $P \in \mathcal{P}_{x\bar{x}}$. There is some literal a (possibly equal to x) such that $suf(P) \in \mathcal{P}_{a\bar{a}\bar{x}}$, $pre(P) \in \mathcal{P}_{xa\bar{a}}$. If P is minimum, then $extend(suf(P))$ and $extend(pre(P))$ are minimum.*

Lemma 1. *Assume a clause* (a)*, for some literal* a*, has a Resolution derivation from* $\mathcal{C}$*. Then the size of the smallest Resolution derivation of* (a) *is* $2\ell - 1$*, where* $\ell = \min_{P \in \mathcal{P}_{\bar{a}a}} |P|$*. Moreover, if* P *is the minimum such path, then* $IR(suf(P))$ *is a smallest derivation.*

Proof. We first show that there is an input derivation of size at most $2\ell - 1$. Let P be a minimum path from $\bar{a}$ to a. Then $length(suf(P)) = |suf(P)| = \ell$ and, by Proposition 2, there is some b such that $suf(P) \in \mathcal{P}_{b\bar{b}a}$. Let P' be the prefix of $suf(P)$ that ends at literal $\bar{b}$. Then $IR(P')$ is a derivation of the singleton clause $(\bar{b})$ and $IR(suf(P))$ is a derivation of (a). This derivation has size $2 \cdot length(suf(P)) - 1 = 2\ell - 1$.

To see that any Resolution derivation of (a) has size at least $2\ell - 1$, assume otherwise. Any Resolution derivation that uses k axioms has size at least $2k - 1$, so (a) is derivable from $\ell' < \ell$ axioms of $\mathcal{C}$. These axioms cannot form a path from $\bar{a}$ to a by minimality, so (a) cannot be derived from them by Proposition 1.

3 Finding Minimum Tree-Like Refutations

Lemma 1 gives us the size of a minimum tree-like Resolution refutation of any contradictory $\mathcal{C}$ and suggests a way to find one. Let $size_{gen}(\mathcal{C})$ ($size_{tree}(\mathcal{C})$) be the size of a smallest general (tree-like) Resolution refutation of $\mathcal{C}$. Then, $size_{tree}(\mathcal{C})$ is $2 \min_{i \in [n]} \left(\min_{P \in \mathcal{P}_{x_i \bar{x}_i}} |P| + \min_{P \in \mathcal{P}_{\bar{x}_i x_i}} |P| \right) - 1$. That is, any minimum tree-like refutation of $\mathcal{C}$ consists of minimum derivations of x_i and $\bar{x}_i$, for some x_i, plus the empty clause. Such derivations of x_i and $\bar{x}_i$ come from input derivations along the suffix of minimum paths from x_i to $\bar{x}_i$ and vice versa. We search for such suffixes by doing BFS from x_i, avoiding already-used clause labels, until either we reach $\bar{x}_i$ or, for some literal y, both y and $\bar{y}$ are visited along the same path (the latter case constitutes the prefix of a minimum path in $\mathcal{P}_{x_i \bar{x}_i}$, which defines the suffix to be used in the minimum derivation of $\bar{x}_i$).

The algorithm proceeds as follows. For each literal x, perform a modified BFS starting at x, except: **(1)** Whenever y is reached from x, store a list $L_1(y)$ of all clause-labels on the path from x to y and a list $L_2(y)$ of all literals on the path from x to y; **(2)** If $\bar{y}$ appears in $L_2(y)$, set $path(x, \bar{x})$ to $extend(path(x, y))$. Terminate BFS at this point; Otherwise, **(3)** when continuing from y, avoid all edges labelled with clauses in $L_1(y)$. When BFS is completed for each literal, find a literal x such that $|path(x, \bar{x})| + |path(\bar{x}, x)|$ is minimum. The tree-like refutation is $IR(suf(path(x, \bar{x})))$, $IR(suf(path(\bar{x}, x)))$ and the empty clause.

BFS, runs in time $O(n+m)$; Doing it for each literal takes time $O(n(n+m))$. Adding the time to check lists L_1 and L_2, the algorithm takes time $O(n^2(n+m))$.

Theorem 1. *For any contradictory 2CNF* $\mathcal{C}$*,* $size_{tree}(\mathcal{C}) < 2\ size_{gen}(\mathcal{C})$*.*

Proof. Let π be the minimum General Resolution refutation of $\mathcal{C}$. Assume π ends by resolving variable x with $\bar{x}$. Assume, wlog, that the minimum Resolution derivation of x is at least as big as the minimum derivation of $\bar{x}$, and let ℓ be the size of this derivation. Clearly $size(\pi) \geq \ell$ since π contains a derivation of

x. In fact, $size(\pi) \geq \ell + 1$ since π also contains the empty clause (which is not used in the derivation of x). On the other hand, there is a tree-like refutation of size at most $2\ell + 1$: use the minimum derivations of x and $\bar{x}$, which are tree-like by Lemma 1, and then resolve the two.

Hence, the algorithm for finding the shortest tree-like refutation is an efficient 2-approximation for computing $size_{gen}$. In fact, this algorithm cannot do better than a 2-approximation in the worst-case.

Theorem 2. *For any $\epsilon > 0$, there exists a contradictory 2CNF $\mathcal{C}_n$ such that $size_{tree}(\mathcal{C}) \geq (2 - \epsilon) \cdot size_{gen}(\mathcal{C})$.*

Proof. Choose n such that $2\epsilon n \geq 9$. $\mathcal{C}$ will be a formula over $n+1$ variables $\{a, x_1, \ldots, x_n\}$ with the following clauses: $(\bar{a} \vee x_1), \{(\bar{x}_i \vee x_{i+1})\}_{i=1}^{n-1}, (\bar{x}_n \vee \bar{a}), (a \vee x_1), (\bar{x}_n \vee a)$. It is not hard to verify that $\forall y,\ P \in \mathcal{P}_{y\bar{y}},\ P' \in \mathcal{P}_{\bar{y}y},\ |P| + |P'| \geq 2n + 2$. Any refutation must consist of a derivation of y, a derivation of $\bar{y}$ and the empty clause, for some variable y. By Lemma 1, the size of a derivation for y plus the size of a derivation for $\bar{y}$ must be at least $2(2n+2) - 2 = 4n + 2$, so any tree-like refutation has size at least $4n + 3$.

On the other hand, there is a general Resolution refutation that proceeds as follows: derive the clause $(\bar{x}_1 \vee x_n)$ using an input derivation of size $2(n-1) - 1 = 2n - 3$. Using also $(\bar{a} \vee x_1)$ and $(\bar{x}_n \vee \bar{a})$, derive $\bar{a}$. Likewise, using $(a \vee \bar{x}_n)$ and $(x_1 \vee a)$ and the already-derived $(\bar{x}_1 \vee x_n)$, derive a. Finally derive the empty clause. This derivation has size $2n - 3 + 4 + 4 + 1 = 2n + 6$. Certainly $4n + 3 \geq (2 - \epsilon)(2n + 6)$.

4 Finding Minimum Unsatisfiable Subformulas

Any unsatisfiable subformula of $\mathcal{C}$ must have a variable x for which there is a path from x to $\bar{x}$ and a path from $\bar{x}$ to x in $G_{\mathcal{C}}$. However, each of these paths might use the same clause twice and the two paths may share clauses. Therefore, we are searching for the set of clauses that comprise the paths that minimize the expression $\min_x \min_{P_1 \in \mathcal{P}_{x\bar{x}}, P_2 \in \mathcal{P}_{\bar{x}x}} |set(P_1) \cup set(P_2)|$. Call two such paths *joint-minimum.* Define the *cost* of any two paths P_1 and P_2 to be $|set(P_1) \cup set(P_2)|$.

Proposition 2 states that if P is minimum path, then $extend(suf(P))$ is minimum. We can say a similar thing about joint-minimum paths: If P_1 and P_2 are joint-minimum, then $extend(suf(P_1))$ and $extend(suf(P_2))$ are joint-minimum, and $cost(suf(P_1), suf(P_2)) = cost(P_1, P_2)$. Therefore, we need to find not-necessarily distinct literals x, a, b and singular paths $P_1 \in \mathcal{P}_{a\bar{a}\bar{x}}$ and $P_2 \in \mathcal{P}_{b\bar{b}x}$ of minimum cost.

A *segment* of a path is a consecutive subsequence of the path's sequence. For two singular paths P_1 and P_2, a *shared segment* is a maximal common segment. A *dual shared segment* of P_1 with respect to P_2 is a maximal segment t of P_1 such that $dual(t)$ is a segment of P_2. For two disjoint segments s and t of P, say $s \prec_P t$ if s appears before t in P.

Consider the following properties of two paths P_1 and P_2.

Property I: Let $s_1 \prec_{P_1} \cdots \prec_{P_1} s_k$ be the shared segments of P_1 and P_2. Then $s_k \prec_{P_2} \cdots \prec_{P_2} s_1$.
Property II: Let $t_1 \prec_{P_1} \cdots \prec_{P_1} t_\ell$ be the dual shared segments of P_1 with respect to P_2. Then $dual(t_1) \prec_{P_2} \cdots \prec_{P_2} dual(t_\ell)$.
Property III: Let $s_1 \prec_{P_1} \cdots \prec_{P_1} s_k$ be the shared segments of P_1 and P_2 and let $t_1 \prec_{P_1} \cdots \prec_{P_1} t_\ell$ be the dual shared segments of P_1 with respect to P_2. For any i, j, $t_i \prec_{P_1} s_j$ if and only if $dual(t_i) \prec_{P_2} s_j$.

Lemma 2. *There are joint-minimum paths P_1 and P_2 such that $suf(P_1)$ and $suf(P_2)$ satisfy Properties I-III.*

Proof. Consider Property I. If $suf(P_1)$ and $suf(P_2)$ violate the property, then there is some $i < j$ such that $s_i \prec_{P_2} s_j$. Let P_1' be the segment of P_1 starting at the beginning of s_i and ending at the end of s_j. Likewise, let P_2' be the segment of P_2 that starts at the beginning of s_i and ends at the end of s_j. Assume, wlog, that $length(P_1') \le length(P_2')$. Let P_2'' be the path P_2 with P_2' replaced by P_1'. Certainly P_1 and P_2'' are still joint-minimum. Property II follows in the same way by looking at P_1 and $dual(P_2)$.

Consider Property III. If $suf(P_1)$ and $suf(P_2)$ violate the property, then there is some i, j such that, wlog, $t_i \prec_{P_1} s_j$, but $s_j \prec_{P_2} dual(t_i)$. Let a, b be the endpoints of t_i. Then there is a cycle that includes a and $\bar{a}$ that uses a strict subset of the edges of P_1 and P_2.

The algorithm will search for the suffixes guaranteed by Lemma 2. More generally, given two pairs of endpoints (and possibly two intermediate points), we will find a pair of (not necessarily singular) paths P_1 and P_2 that obey Properties I-III, that have the specified endpoints (and perhaps intermediate points) and that have minimum cost over all such pairs of singular paths. The fact that P_1 and P_2 themselves may not be singular is not a problem since they will achieve the same optimum that singular paths achieve.

The algorithm uses dynamic programming based on the following idea. The reason joint-minimum paths P_1 and P_2 may not each be of minimum length is that, while longer, they benefit by sharing more clauses. If we demand that P_1 and P_2 have a shared segment with specified endpoints, then that segment should be as short as possible; likewise, for any segment of, say, P_1 with specified endpoints that is guaranteed not to overlap any shared segment. By doing this, we isolate segments of P_1 and P_2 that we can locally optimize and then concentrate on the remainder of the paths.

We will compute a table $A[(a_1, b_1, c_1), (a_2, b_2, c_2), k, \ell]$ which stores the minimum of $cost(P_1, P_2)$ over all paths $P_1 \in \mathcal{P}_{a_1 b_1 c_1}$ and $P_2 \in \mathcal{P}_{a_2 b_2 c_2}$ such that: **(1)** We recognize at most k shared segments between P_1 and P_2; **(2)** We recognize at most ℓ dual shared segments of P_1 with respect to P_2; and **(3)** P_1, P_2 obey Properties I-III. By "recognizing" k shared segments, we mean that if there are more shared segments, their lengths are added twice to the cost of P_1 and P_2, with no benefit from sharing. If we omit b_1, respectively b_2, as a parameter in $A[\,]$, then P_1, respectively P_2, comes from $\mathcal{P}_{a_1 c_1}$.

To begin, for all literals a, b, set $B[a, b]$ to the length of a shortest path in $\mathcal{P}_{ab}$. Likewise, set $B[a, b, c]$ to the length of a shortest path in $\mathcal{P}_{abc}$. For all $a_1, b_1, c_1, a_2, b_2, c_2$, set $A[(a_1, b_1, c_1), (a_2, b_2, c_2), 0, 0]$ equal to $B[a_1, b_1, c_1] + B[a_2, b_2, c_2]$. Set $A[((a_1, c_1), (a_2, c_2), 0, 0]$ to $B[a_1, c_1] + B[a_2, c_2]$.

To compute a general entry in A where ℓ is nonzero, let P_1 and P_2 be the paths that achieve the minimum corresponding to the entry in question. By Properties II and III, there are two cases. **(1)** The first shared segment of any kind in P_1 (in order of appearance) is a dual shared segment t_1 and $dual(t_1)$ is the first shared segment of any kind in P_2. **(2)** The last shared segment of any kind in P_1 is a dual shared segment t_k and $dual(t_k)$ is the last shared segment of any kind in P_2.

Suppose we are in Case 1 (Case 2 is similar). We try placing b_1 before, in, or after t_1 in P_1 (likewise for $b_2, dual(t_1), P_2$) and we try all endpoints for t_1. For example, in the case where we try placing b_1 before t_1 in P_1 and b_2 before $dual(t_1)$ in P_2, we take the minimum over all literals u, v, of

$$B[a_1, b_1, u] + B[a_2, b_2, \bar{v}] + B[u, v] + A[(v, c_1), (\bar{u}, c_2), k, \ell - 1].$$

Then we assign A the minimum over all nine placements of b_1 and b_2. Finally, if $A[(a_1, b_1, c_1), (a_2, b_2, c_2), k, \ell - 1]$ is less than the calculated value, we replace the current entry with that.

If $\ell = 0$ and k is nonzero, we proceed similarly except that the first shared segment in P_1 is the last shared segment in P_2 by Property I. Therefore (placing b_1 before s_1 and b_2 before s_k), we take the minimum over all u, v of

$$B[a_1, b_1, u] + B[v, c_2] + B[u, v] + A[(v, c_1), (a_2, b_2, u), k - 1, 0].$$

Again, minimize over all b_1 and b_2, then check $A[(a_1, b_1, c_1), (a_2, b_2, c_2), k - 1, 0]$.

The size of the joint minimum paths will finally be stored in $A[(a, \bar{a}, x), (b, \bar{b}, \bar{x}), n, n]$ for some literals a, b, x; we simply find the smallest such entry. We can recover the actual set of edges comprising these paths using the standard dynamic-programming technique of remembering which other entries of A were used to compute the current entry. The algorithm is clearly polynomial time, since there are polynomially-many entries in A and each one is computed as the minimum of polynomially-many expressions.

References

1. A. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimal propositional proof length is NP-hard to linearly approximate. In *Proc., MFCS'98*, pages 176–184, 1998. (Also LNCS 1450).
2. Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, March 1979.
3. K. Subramani. Optimal length tree-like resolution refutations for 2SAT formulas. *ACM Transactions on Computational Logic*, 5(2):316–320, April 2004.