COMBINATORICA
Bolyai Society – Springer-Verlag

COMBINATORICA **24** (4) (2004) 585–603

NEAR OPTIMAL SEPARATION OF TREE-LIKE AND GENERAL RESOLUTION

ELI BEN-SASSON*, RUSSELL IMPAGLIAZZO†, AVI WIGDERSON‡

Received January 17, 2000

We present the best known separation between tree-like and general resolution, improving on the recent $\exp(n^{\epsilon})$ separation of [2]. This is done by constructing a natural family of contradictions, of size n, that have $O(n)$-size resolution refutations, but only $\exp(\Omega(n/\log n))$-size tree-like refutations. This result implies that the most commonly used automated theorem procedures, which produce tree-like resolution refutations, will perform badly on some inputs, while other simple procedures, that produce general resolution refutations, will have polynomial run-time on these very same inputs. We show, furthermore that the gap we present is nearly optimal. Specifically, if S (S_T) is the minimal size of a (tree-like) refutation, we prove that $S_T = \exp(O(S \log\log S/\log S))$.

1. Introduction and Main Results

The resolution refutation system is one of the oldest and most commonly methods for proving the unsatisfiability of CNF formulas, and is interesting from a theoretical as well as practical point of view. From the theoretical point of view, this system is fairly simple, as it has a single derivation rule, and all proof-lines are clauses. Still, in spite of its simplicity, we do not understand it fully, and there remain several interesting and important open

Mathematics Subject Classification (2000): 03F20; 68Q17

* This research was supported by Clore Foundation Doctoral Scholarship.

† Research supported by NSF Award CCR-0098197 and USA–Israel BSF Grant 97-00188.

‡ This research was supported by grant number 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities.

problems regarding this system. One such natural question, solved in this paper, is the following:

Question: What is the largest possible gap between the minimal number of lines (=clauses) in a refutation, versus the minimal number of lines in a tree-like refutation?

"Tree-like" means that each non-axiom line is a premise for at most one other line in the refutation.

In order to understand the appeal of resolution to automated theorem proving (ATP) let us recall the ATP problem: Given a contradiction $\mathcal{T}$, try to find a refutation of $\mathcal{T}$ in the most efficient way, in terms of time and space. Let us look at the following couple of natural methods for finding such a refutation.

Recursion: pick a variable x, and recursively try to refute $\mathcal{T}_0$ and $\mathcal{T}_1$, which are the restrictions of $\mathcal{T}$ to $x=0$, ($x=1$ resp.). Such methods are called DLL-Procedures.

Dynamic Programming (also known as Davis–Putnam): Start from the axioms $\mathcal{T}$, arbitrarily derive new clauses $\mathcal{T}'$, using the resolution rule, and set $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$. Repeat this procedure until the empty clause is derived. If one picks a variable x and performs all possible resolutions on it, the method is called a DP-procedure.

Let us briefly describe the pros and cons of these methods, and their natural connection to resolution. The recursive procedure has recursion depth that is bounded by the number of variables, thus it is extremely space-efficient – the space required is at most linear in the input size. Its main flaw is that it corresponds to a *decision-tree* for the problem of finding a clause falsified by an assignment to the variables. It turns out that such a decision-tree is a tree-like resolution refutation of $\mathcal{T}$ (see lemma 7). Thus, the recursive method will be time-costly on inputs that do not have short tree-like resolution refutations.

The dynamic programming method, on the other hand, produces general resolution refutations. Thus, if $\mathcal{T}$ has a short refutation, one might hope to find such a refutation by using such a method. Indeed, we shall see that for certain contradictions, there exist dynamic programming proof search methods that are extremely efficient (polynomial time). The main disadvantage of the method is the large space it might use. If we wish to save all possible resolution consequences we might require space that is exponential in n.

We conclude that understanding the relation between (minimal) general and tree-like refutation size will help us understand the relative efficiency of these two proof search methods. Formally, for a fixed contradiction $\mathcal{T}$ let

$S(\mathcal{T})$ ($S_T(\mathcal{T})$ resp.) be the number of lines in a minimal general (tree-like resp.) refutation of $\mathcal{T}$, and let $w(\mathcal{T}\vdash 0)$ be the size of the largest clause in a refutation of $\mathcal{T}$. We are interested in measuring the gap between S and S_T, and in this paper deliver the following couple of messages. The first message is that the gap can be very large. Let $|\mathcal{T}|$ be the number of clauses in $\mathcal{T}$.

Theorem 1 (Lower Bound). *There exists an infinite family of explicitly constructible contradictions $\{\mathcal{T}_n\}$ such that $|\mathcal{T}_n| = O(n)$, $S(\mathcal{T}_n) = O(n)$, $w(\mathcal{T}_n \vdash 0) = O(1)$ and $S_T(\mathcal{T}) = \exp(\Omega(\frac{n}{\log n}))$.*

It is easy to see that $S_T \leq 2^S$. Our second message is that there is a non-trivial upper bound on the gap, and it is almost tight:

Theorem 2 (Upper Bound). $S_T(\mathcal{T}) = \exp\left(O\left(\frac{S(\mathcal{T})\log\log S(\mathcal{T})}{\log S(\mathcal{T})}\right)\right)$, *for all CNF contradictions $\mathcal{T}$.*

Theorem 1 is proved by giving a construction which associates to every circuit (fan-in 2 DAG) G on n edges a contradiction $\mathcal{T}(G)$ with the following properties.

1. $\mathcal{T}(G)$ has $O(n)$ variables and $O(n)$ clauses.
2. $\mathcal{T}(G)$ has general Resolution refutations of length $O(n)$.
3. $\mathcal{T}(G)$ has Resolution refutations of width $O(1)$ (i.e. all clauses in the refutation have a constant number of variables).
4. Every tree-like Resolution refutation of $\mathcal{T}(G)$ has length at least $\exp(P(G))$, where $P(G)$ is the classical pebbling number of the circuit G.

The pebbling measure of a circuit G is the minimal memory size needed to carry out the computation described in the circuit G, on an input x, (both G and x are given as inputs), assuming each gate output costs one memory unit.

The construction of the contradictions is motivated by a special case of it for the Pyramid graph of [2], which was in turn motivated by [17]. We show that the pebbling measure of the graph G is an exponential lower bound on the tree-like size of refuting $\mathcal{T}(G)$. The connection of pebbling to tree-like size allows us to use graphs that have a high pebbling measure. Specifically, [10] explicitly construct for every n a graph G_n of size $O(n)$ satisfying $P(G_n) = \Omega(n/\log n)$. This, combined with (1) and the upper bound (2), gives a truly exponential separation between general and tree-like Resolution systems (the previous best bound being the recent $\exp(\sqrt{n})$ separation of [2]. Finally, (3) means that for these contradictions, the natural "restricted-width" dynamic programming algorithm [7], that searches

for a minimal width refutation, has polynomial time, which is exponentially faster than any recursive method.

Theorem 2 uses techniques from [10], who showed that if a Boolean function has a circuit of size S then it has a circuit of depth $S/\log S$. We use similar techniques to show that one can "cut" any general refutation roughly in half, by removing some of the middle lines. Then we construct a small tree-like refutation, depending on the relative size of the "middle part" that was removed.

The paper is organized as follows. After presenting the essential definitions in section 2, we start by proving the lower bound (section 3), followed by the upper bound (section 4). We end with a discussion of the applications of these bounds to automated theorem proving (section 5).

2. Definitions

2.1. General

x will denote a Boolean variable, ranging over $\{0,1\}$. Throughout this paper we shall identify 1 with $True$ and 0 with $False$. A *literal* over x is either x (denoted also as x^1) or $\bar{x}$ (denoted also as x^0). A *clause* is a set of literals, and a CNF formula is a set of clauses. A clause is viewed as the Boolean function that is the disjunction of its literals. It is customary to write a clause $\{\ell_1,\ldots,\ell_t\}$ as $\ell_1 \vee \ldots \vee \ell_t$. Analogously, a CNF formula is viewed as the Boolean function that is the conjunction of its clauses. It is customary to write a CNF $\{C_1,\ldots,C_m\}$ as $C_1 \wedge \ldots \wedge C_m$. We say that a variable x *appears* in a clause C (denoted $x \in C$) if a literal over x is an element of C. A CNF formula $\mathcal{T}$ is said to be *satisfiable* if there exists some assignment to the Boolean variables, that sets it to 1. Otherwise the formula is said to be *unsatisfiable*. Let $\mathcal{T} = \{C_1, C_2 \ldots C_m\}$ be a CNF formula over n variables, a Resolution *derivation* π of a clause A from $\mathcal{T}$ is a sequence of clauses $\pi = (D_1, D_2 \ldots D_S)$ such that the last clause is A and each line D_i is either some initial clause $C_j \in \mathcal{T}$ or is derived from previous lines using one of the following rules.

Resolution Rule: If C, D are clauses and x is a variable such that $x \in C$ and $\bar{x} \in D$, we say C, D are resolvable on x and their resolvent is $C \cup D - \{x, \bar{x}\}$. C, D are called the *assumptions* of the derivation and $C \cup D - \{x, \bar{x}\}$ is the *consequence*.

Weakening Rule: If C, D are clauses then $C \cup D$ is a weakening of C. C is the assumption and $C \cup D$ is the consequence.

A resolution *refutation*[1] of $\mathcal{T}$ is a resolution derivation of the empty clause 0 from $\mathcal{T}$. The following theorem is the fundamental property of the resolution proof system. A CNF $\mathcal{T}$ *implies* a clause C (denoted $\mathcal{T} \models C$) if all assignments that fix $\mathcal{T}$ to 1 also fix C to 1.

Theorem 3 (Implicational Completeness). *For any CNF $\mathcal{T}$ and any clause C, $\mathcal{T} \models C$ iff there is a resolution derivation of C from $\mathcal{T}$. In particular, $\mathcal{T}$ is unsatisfiable iff it has a resolution refutation.*

A *circuit* is a Directed Acyclic Graph, in which each vertex has fan-in 2 or 0. A vertex with fan-in 0 is called a *source*, and a vertex with fan-out 0 is called a *target*. All non-source vertices are called *internal vertices*.

A derivation π yields a circuit G_π with $|\pi|$ vertices. Each vertex of G_π is labeled by a clause of π, and for derivation steps edges are added from the vertices labeled by the assumptions to the vertex labeled by the consequence. A derivation π is called *tree-like* if G_π is a tree; a tree-like derivation may have multiple sources labeled by the same initial clause in $\mathcal{T}$. The *size* of the derivation π is the number of lines (clauses) in it, denoted S_π. $S(\mathcal{T})$ ($S_T(\mathcal{T})$) is the *minimal size* of a (tree-like) refutation of $\mathcal{T}$. Notice that since every tree-like derivation is also a general derivation, we have $S(\mathcal{T}) \leq S_T(\mathcal{T})$. On the other hand, it is not hard to see that any general derivation of size S can be converted to a tree-like derivation of size at most 2^S, giving the trivial upper bound $S_T(\mathcal{T}) \leq 2^{S(\mathcal{T})}$.

2.2. Restrictions

For C a clause, x a variable and $a \in \{0,1\}$, the *unit restriction* of C setting x to a is:

$$C|_{x=a} \stackrel{\text{def}}{=} \begin{cases} C & \text{if } x \text{ does not appear in } C, \\ 1 & \text{if the } \textit{literal } x^a \text{ appears in } C, \\ C \setminus \{x^{1-a}\} & \text{otherwise.} \end{cases}$$

A *restriction* is a set of distinct unit restrictions. Namely, a restriction ρ of size k is set of ordered pairs $\rho = \{(x_{i_1}, a_1), \ldots, (x_{i_k}, a_k)\}$, where $i_j \neq i_{j'}$ for all distinct $j, j' \in \{1, \ldots, k\}$, and $a_j \in \{0,1\}$. For ρ a restriction of size k, we define

$$C|_\rho \stackrel{\text{def}}{=} C|_{x_{i_1}=a_1}|_{x_{i_2}=a_2} \cdots |_{x_{i_k}=a_k}$$

[1] Throughout this paper we will only discuss derivations and refutations in the Resolution proof system. Hence a *derivation* is always a *resolution derivation* and a *refutation* is always a *resolution refutation*.

Similarly, $\mathcal{T}|_\rho \stackrel{\text{def}}{=} \{C|_\rho : C \in \mathcal{T}\}$. For $\pi = (C_1, \ldots, C_s)$ a derivation of C_s from $\mathcal{T}$ and ρ a restriction, let $\pi|_\rho = (C'_1, \ldots, C'_s)$ be the restriction of π on ρ, defined inductively by:

$$C'_i = \begin{cases} C_i|_\rho & C_i \in \mathcal{T}, \\ C'_{j_1} \vee C'_{j_2} & C_i \text{ was derived from } C_{j_1} \vee y \text{ and } C_{j_2} \vee \bar{y} \text{ via one resolution step, for } j_1 < j_2 < i, \\ C'_j \vee A|_\rho & C_i = C_j \vee A \text{ via the weakening rule, for } j < i. \end{cases}$$

The consequence of resolving a clause B with 1 is defined to be B. We shall assume w.l.o.g. that $\pi|_\rho$ does not contain the clause 1, by removing all such clauses from $\pi|_\rho$.

2.3. Width

Definition 4 (Clause width). The *width* of a clause C, denoted $w(C)$, is the number of literals appearing in it. The *width* of a set of clauses is the *maximal width* of a clause in the set, i.e. $w(\mathcal{T}) = \max_{C \in \mathcal{T}}\{w(C)\}$.

Definition 5 (Proof width). The width of *deriving* a clause A from the formula $\mathcal{T}$, denoted $w(\mathcal{T} \vdash A)$ is $\min_\pi\{w(\pi)\}$, where the minimum is taken over all derivations π of A from $\mathcal{T}$. We also use the notation $\mathcal{T} \vdash_w A$ to mean that A can be derived from $\mathcal{T}$ in width w. We will be mainly interested in the width of refutations, namely in $w(\mathcal{T} \vdash 0)$.

2.4. Decision Trees

Let $\mathcal{T}$ be a CNF formula. A *search problem* for $\mathcal{T}$ is the following: given an assignment α to the variables, find a clause $C \in \mathcal{T}$ such that $C(\alpha) = 0$, if there is such a clause, and otherwise (i.e. $\mathcal{T}(\alpha) = 1$) answer 1.

Definition 6 (Decision Trees for *CNF* Search Problems). A Decision Tree is a binary tree, with internal vertices labeled by variables, edges labeled by 0 or 1, and leaves labeled with the possible outputs. Every assignment to the variables defines a path through the tree in the natural way, and the label at the end of the path is said to be the output of the decision tree on that assignment.

We say that D is a Decision Tree for the Search problem for $\mathcal{T}$ if it correctly solves it on every input.

For $\mathcal{T}$ a CNF formula, let $S_D(\mathcal{T})$ denote the minimal size of a Decision Tree solving the CNF Search Problem for $\mathcal{T}$.

Decision trees for CNF search problems are closely related to tree-like resolution, as the following lemma shows.

Lemma 7. *For $\mathcal{T}$ an unsatisfiable CNF, $S_T(\mathcal{T}) = S_D(\mathcal{T})$*

Proof. The Tree of the resolution refutation is a decision tree, where each internal vertex is labeled by the variable resolved upon at that step. Hence $S_T(\mathcal{T}) \leq S_D(\mathcal{T})$.

For the opposite direction, we claim that given a decision tree, we can derive from it a tree like refutation without increasing its size. Notice that if $\mathcal{T}$ is unsatisfiable, every leaf is labeled by a clause, since 1 is not a legitimate answer. We look at two leaves labeled C_i, C_j, with their father v labeled x. If x does not appear in one of the two clauses (w.l.o.g. C_i), then we may label v with C_i, erase its sons, and make the tree smaller. Otherwise, w.l.o.g. x must appear in C_i in positive form and in C_j in negative form. In this case we may label v with the consequence of resolving C_i, C_j on x. Continuing in this way up through the decision tree we conclude that $S_T(\mathcal{T}) \geq S_D(\mathcal{T})$. ∎

2.5. Pebbling

The pebbling measure of a circuit is, intuitively, the space needed for simulating the computation of the circuit on a Turing machine. For a thorough introduction to results regarding pebbling, see the game excellent survey [18]. In this section we briefly state the essential definitions and facts that will be used later on in our discussion.

Definition 8 (Pebbling). Let G a circuit, and let S and T be subsets of the vertices such all sources of G are included in S and the target is included in T. S is called the set of *starting points* and T is the set of *terminal points* for the following 1-player game. At any point in the game, some vertices of G will have pebbles on them (1 pebble per vertex), while the remaining vertices will not. A *configuration* is a subset of vertices, comprising just those vertices that have pebbles on them. The rules of the pebble game are as follows.

1. At any time, a pebble may be placed on any vertex in S.
2. If all immediate predecessors of a vertex have a pebble on them, a pebble may be placed on that vertex.
3. A pebble may be removed from any vertex.
4. If a pebble is placed on a vertex in T, the player wins and the game ends.

A *legal* pebbling of T on G from S is a sequence of configurations, the first being the empty set, the last containing some element of T, and in which each configuration follows from the previous configuration by one of the rules. The number of pebbles used in such a pebbling is the maximum number of pebbles in any configuration. The pebbling price of T on G from S, denoted $P_G(S,T)$, is the minimal number of pebbles needed in any legal pebbling of T from S on G.

Notice that $P_G(S,T) = 1$ iff S and T intersect, and that $P_G(S,T) = P_G(S,\{t\})$ for some $t \in T$. We remark that in this paper all vertices of the DAGs considered have fan-in either 0 or 2.

2.6. A game for proving lower bounds on tree-like resolution

Lower bounds for size of tree-like resolution proofs can be given in terms of a 2-player game; this description is due to [19]. Any small tree-like proof will give a good strategy for one of the players, so a good strategy for the other player yields a corresponding lower bound on proofs.

Let $\mathcal{T}$ be an inconsistent set of clauses. Consider the following game between two players, the Prover, and the Delayer. In each round, the Prover chooses a variable to be assigned a value. Then the Delayer chooses one of 0, 1, or $*$. If 0 or 1 is chosen, no points are scored, the variable is set to the chosen bit, and the next round begins. If $*$ is chosen, then the Delayer scores one point, but the Prover then can choose the value for the variable. The game ends when one of the clauses in $\mathcal{T}$ is forced to false by the assigned values, i.e., when all the literals in the clause are assigned false.

Lemma 9. *If $\mathcal{T}$ has a tree-like resolution refutation of size S, then the Prover has a strategy where the Delayer can win at most $\lceil \log S \rceil$ points.*

Proof. Fix a proof of size S. The Prover will maintain the following invariant after each round: If the Delayer has scored t points, then the partial assignment will falsify a clause in the proof, and the sub-tree rooted at this clause will be of size at most $S/2^t$. In particular, this will show the claimed bound on the number of points the Delayer can score.

Let C be the clause from the invariant at the previous round. If C is a leaf, then the game halts, since a clause in $\mathcal{T}$ has been falsified. Otherwise, the Prover picks the variable x resolved on to derive C from C_0 and C_1 in the proof. If the Delayer assigns the variable a value, at least one of the two clauses C_0, C_1 is falsified, and this is the new clause for the invariant. If the Delayer chooses $*$, and scores a point, then the Prover chooses the value to

falsify the clause with the smaller sub-tree. This is at most half the size of the sub-tree for the current clause, so the invariant is maintained. ∎

Corollary 10. *If the Delayer has a strategy which always scores r points, then $S_T(\mathcal{T}) \geq 2^r$.*

3. Lower Bound

In this section we prove the lower bound of theorem 1, by explicitly presenting a family of contradictions that achieves the lower bound. The family of contradictions is a generalization of [17] and [2]. For these contradictions we expose a connection between pebbling and tree-like Resolution. We start by defining these contradictions.

Definition 11 (Pebbling Contradictions). Let G be a circuit, and S and T subsets of vertices. Associate a pair of Boolean variables $x(v)_0, x(v)_1$ with every vertex $v \in V(G)$. $Peb_{G,S,T}$, the *Pebbling Contradiction* of G is the Boolean formula consisting of the following clauses:

Source Axioms: $x(s)_0 \vee x(s)_1$ for each $s \in S$.
Target Axioms: $\bar{x}(t)_0$ and $\bar{x}(t)_1$ for each $t \in T$.
Pebbling Axioms: $(x(u_1)_a \wedge x(u_2)_b) \rightarrow (x(v)_0 \vee x(v)_1)$ for u_1, u_2 the two predecessors of v, and all $a, b \in \{0,1\}$. This is equivalent to the clause $\bar{x}(u_1)_a \vee \bar{x}(u_2)_b \vee x(v)_0 \vee x(v)_1$.

Notice that $Peb_{G,S,T}$ is a 4-CNF over $2|V|$ variables, with $O(|V|)$ clauses. Since by definition all sources of G appear in S, and the target of G appears in T, it is not hard to see that $Peb_{G,S,T}$ is unsatisfiable. Furthermore, it has a short, constant width resolution refutation.

Lemma 12. *For G a circuit, $S(Peb_{G,S,T}) = O(|V|)$ and $w(Peb_{G,S,T} \vdash 0) \leq 6$.*

Proof. Fix a topological sort of G. In order of this sort, we derive $x(v)_0 \vee x(v)_1$ for each $v \in V$. If v has no predecessors, $v \in S$, so this is an axiom. If v has two predecessors, u_1 and u_2, we have inductively derived $x(u_1)_0 \vee x(u_1)_1$, and $x(u_2)_0 \vee x(u_2)_1$. Together with the four Pebbling Axioms for v, these imply the clause $x(v)_0 \vee x(v)_1$. By the implicational completeness of Resolution (theorem 3) there is a derivation of $x(v)_0 \vee x(v)_1$ from the above mentioned clauses. This derivation has constant width and size, since it involves at most 6 variables. Hence, starting from the Source Axioms, one can infer $x(t)_0 \vee x(t)_1$, for a target $t \in T$, in constant width and size $O(|V|)$. Then, resolving with the Target Axioms, one derives 0. ∎

The following theorem is the main result presented in this section, and it immediately implies theorem 1.

Theorem 13. $S_T(Peb_{G,S,T}) = 2^{\Omega(P_G(S,T))}$.

Proof [Theorem 1]. Lemma 12 shows that for *any* circuit G, the pebbling contradiction based on G has a linear size, constant width, general resolution refutation. [10] present an infinite family of explicitly constructible circuits $\{G_n\}$, with $|V(G)| = n$ and $P_G(S,T) = \Omega(n/\log n)$. Taking the Pebbling contradictions of these graphs, theorem 13 implies a lower bound of $2^{\Omega(n/\log n)}$ on the tree-like refutation size, completing the proof of theorem 1. ∎

The rest of this section is devoted to the proof of theorem 13. We use Corollary 10; we give a strategy for the Delayer that achieves at least $P_G(S,T) - 3$ points on $Peb_{G,S,T}$.

The strategy is as follows: Set $S' = S, T' = T$. On each round, the Prover proposes a variable $x(v)_i$.

The Delayer's response is as follows:

Case 1. If $v \in T'$, assign the variable 0.
Case 2. If $v \in S'$ assign the variable 1.
Case 3. If $v \notin S' \cup T'$, and if $P_G(S', T' \cup \{v\}) = P_G(S', T')$, assign the variable 0 and add v to T'.
Case 4. If $v \notin S' \cup T'$, and if $P_G(S', T' \cup \{v\}) < P_G(S', T')$, respond $*$ (letting the Prover assign a value) and add v to S'.

We prove that the strategy above earns the Delayer at least $P_G(S,T) - 3$ points by showing that $P_G(S', T')$ only decreases by the number of points earned, and is at most 3 at the end of the game.

Lemma 14. *When the game terminates,* $P_G(S', T') \leq 3$.

Proof. Note that, if any variable $x(v)_i$ is set to 1, it happened either in Case 2 or Case 4, and in either case $v \in S'$. Similarly, if both $x(v)_0$ and $x(v)_1$ are assigned 0, the first to be assigned was either via Case 1 or Case 3, and so $v \in T'$. (If the first to be assigned fell under Case 4, v would be put in S', and the second variable would have been assigned 1 via Case 2.)

So a source axiom will never be violated for $s \in S \subseteq S'$, because both variables will not be set to 0, and similarly, a target axiom will never be violated for $t \in T \subseteq T'$. So at the end of the game, a a pebbling axiom is violated, say for node v with predecessors u and u'. To be violated, both $x(v)_0$ and $x(v)_1$ must be set to 0, so $v \in T'$. Also, at least one of $x(u)_0$ and $x(u)_1$ must be set to 1, so $u \in S'$. Similarly, $u' \in S'$. So to pebble T' from S' using three pebbles, simply place a pebble on both u and u', and then on t. ∎

Lemma 15. *For any* $v \in V$ *and any sets* S, T, $P_G(S,T) \leq \max\{P_G(S, T \cup \{v\}), P_G(S \cup \{v\}, T) + 1\}$.

Proof. One way to pebble T from S is to first pebble $T \cup \{v\}$ from S, using $P_G(S, T \cup \{v\})$ pebbles. If the result is a pebble in T, stop, otherwise the final configuration has a pebble on v. Keeping that pebble on v, remove the other pebbles, then simulate a pebbling of T from $S \cup \{v\}$. This second stage uses a total of $P_G(S \cup \{v\}, T) + 1$ pebbles. ∎

Lemma 16. *After any round, if the Delayer has scored p points, $P_G(S', T') \geq P_G(S, T) - p$.*

Proof. At the beginning of the game, both sides are $P_G(S, T)$. Note that the only case when $P_G(S', T')$ changes is in Case 4. In this case, $P_G(S', T' \cup \{v\}) < P_G(S', T')$ at the beginning of the round. By Lemma 15, this means $P_G(S' \cup \{v\}, T') \geq P_G(S', T') - 1$. Since in Case 4, the Delayer scores a point, and v is added to S', this preserves the invariant. ∎

Corollary 17. *Using the strategy described, the Delayer scores at least $P_G(S, T) - 3$ points.*

Corollary 18. *Any tree-like resolution proof of $Peb_{G,S,T}$ has size $\Omega(2^{Peb_G(S,T)})$.*

4. Upper Bound

In this section we prove that the gap presented in the previous section cannot be "too large". Specifically, we prove theorem 2. This shows that the lower bound stated in theorem 1 is nearly optimal.

The theorem will be proved by explicitly constructing a small size tree like refutation of $\mathcal{T}$ given a small size general refutation. We shall limit our attention only to "hard" contradictions, for which the gap between tree-like and general size is maximal, and bound from above the refutation size for these inputs.

In order to prove theorem 2, we shall work with a different size measure, hereby defined. This measure, inspired by [15,10], counts the number of internal edges in the graph of a minimal refutation, and is closely connected to the standard size measure S defined above. For technical reasons we will restrict ourselves to proofs π such that G_π has maximal fan-out 2. This is not a big limitation, because any proof π can be converted to a proof π' such that $G_{\pi'}$ has fan-out 2, where the size of π' is at most twice the size of π. (If C is used as an assumption in k derivation steps, make k copies of C using the weakening rule).

Definition 19 (Internal Size). An *internal vertex* of a DAG is any non-source vertex. An *internal edge* in a DAG connects two internal vertices. For G a DAG, $e(G)$ is the number of internal edges in G. For $\mathcal{T}$ a CNF contradiction, define the internal size of refuting $\mathcal{T}$ to be:

$$e(\mathcal{T}) \stackrel{\text{def}}{=} \min\{e(G_\pi) : \pi \text{ is a refutation of } \mathcal{T} \text{ and } G_\pi \text{ has maximal fan-out 2}\}.$$

We define $f(e)$ to be the maximal tree-like size of refuting $\mathcal{T}$, given that $\mathcal{T}$ has a refutation with internal size e. Formally:

$$f(e) \stackrel{\text{def}}{=} \max\{S_T(\mathcal{T}) : e(\mathcal{T}) \leq e\};$$

$\mathcal{T}$ is called *e-maximal* if $e(\mathcal{T}) = e$, $S_T(\mathcal{T}) = f(e)$, and removing any clause from $\mathcal{T}$ enlarges e or makes $\mathcal{T}$ satisfiable.

We start by proving two lemmas that will be used in the proof of the main theorem.

Lemma 20. $f(e+1) \leq 2f(e)$.

Proof [Lemma 20]. Let $\mathcal{T}$ be an $e+1$-maximal contradiction, and let π be a refutation of $\mathcal{T}$ having $e+1$ internal edges. Let C be a clause of π that is a consequence of resolving two axioms $A, B \in \mathcal{T}$, and is involved in at least one internal edge. (The proof has at least one internal edge. Repeatedly backing from the current node to any internal predecessor, starting at the source of this edge, we eventually get to a minimally internal vertex, which must be labeled by such a C.) Let $\mathcal{T}' = \mathcal{T} \cup \{C\}$. $\mathcal{T}'$ has a tree like refutation T' of size at most $f(e)$, since deleting the derivation of C from π gives a refutation of $\mathcal{T}'$ with $\leq e$ internal edges. Adding the derivation $\frac{A \quad B}{C}$ to T' whenever C appears as an axiom in T', will be a refutation of $\mathcal{T}$ which is at most double the size of T'. ∎

Let $\binom{n}{\leq m} \stackrel{\text{def}}{=} \sum_{i=0}^{m} \binom{n}{i}$. For $m > n$ let $\binom{n}{\leq m} \stackrel{\text{def}}{=} 2^n$.

Lemma 21. *Given a CNF formula with m clauses and n variables, there is a decision tree solving the CNF search problem, with size at most $\binom{n}{\leq m}$.*

Proof [Lemma 21]. Consider the following strategy for the CNF search problem. In every step, we have a partial assignment ρ , we query a variable, and add its value to ρ. If at the beginning of a step, ρ has forced any clause to be false, we stop and output that clause. If ρ has forced all clauses to be true, we stop and output "true". Otherwise, we choose the first clause whose value is not forced, and query the first variable in that clause whose value is not assigned.

In each step of the above strategy, there is one value for the query that satisfies the current clause. Along any path of the decision tree, the satisfying value can be chosen at most m times before all clauses are forced to true, and the decision tree terminates. Thus, a path in the tree can be described by a sequence of n bits with at most m 1's, where 1 means, assign the satisfying value, and 0, assign the other value. Therefore, there are at most $\binom{n}{\leq m}$ such paths. ∎

Next, we wish to show that any G_π can be split "in half", into two derivations, each having roughly half the number of internal edges appearing in the original G_π.

Definition 22 (Topological Partition). Let $v_1 \dots v_S$ be a topological ordering of the vertices of a circuit G. For $0 \leq i \leq S$, let $V_0(i) = \{v_1, \dots, v_i\}$, and let $V_1(i) = \{v_{i+1}, \dots, v_S\}$. Let $G_0(i)$ ($G_1(i)$ resp.) be the subgraph of G induced by $V_0(i)$ ($V_1(i)$ resp.). Let $e_0(i)$ ($e_1(i)$ resp.) be the number of internal edges in $G_0(i)$ ($G_1(i)$ resp.). Let M_i be the set of *internal* vertices of $V_0(i)$ that are connected to vertices in $V_1(i)$, and define $m_i = |M_i|$.

An internal vertex of $G_0(i)$ is also an internal vertex in G, thus M_i is a subset of the internal vertices of G. Notice that $\pi = (C_1, \dots, C_S)$ is a topological ordering of G_π. If $V_0(i), V_1(i)$ is a topological partition of G_π, then $V_0(i)$ is a resolution derivation from $\mathcal{T}$, and $V_1(i)$ is a resolution refutation of $\mathcal{T} \cup M_i$.

Lemma 23 (Existence of an Equal Topological Partition). *[10] For $v_1 \dots v_S$ a topological ordering of the vertices of a single-target circuit G with maximal fan-out 2, there exists a partition of G such that $|e_0(i) - e_1(i)| \leq 2$. Such a partition is called* equal, *and for an equal partition, $e_0(i) < \frac{e - m_i}{2} + 2$.*

Proof. $e_0(0) = 0$, $e_0(S) = e$ and $e_0(i+1) \leq e_0(i) + 2$, since adding the vertex v_{i+1} to $V_0(i)$ increases the number of internal edges in $V_0(i)$ by at most 2 (G has maximal fan-in 2). The fan-out of G is at most 2, so $e_1(i+1) \geq e_1(i) - 4$, since removing v_{i+1} from $V_1(i)$ can transform at most 2 internal vertices of $G_1(i)$ into source vertices of $G_1(i+1)$, and each of these two new sources has fan-out at most 2, meaning at most 4 internal edges are removed from $G_1(i)$ when v_{i+1} is removed. Hence, there exists an i such that $|e_0(i) - e_1(i)| \leq 3$.

G has a single target, so every vertex in M_i is connected to at least one vertex in $V_1(i)$. Hence $e(G) \geq e_0(i) + e_1(i) + m_i$, and $e_0(i) \leq \frac{e - m_i + 3}{2}$. ∎

We are ready to prove the main theorem of this section:

Theorem 24. *There exists a constant $c > 0$, such that for all integers $k \geq 4$, $f(k2^k) \leq 2^{c \cdot 2^k \cdot \log k}$.*

Proof. By induction on k. Let $c \geq 8$ be large enough that the claim is true for all $k < 4$, and assume the claim holds for all values smaller than some fixed $k \geq 4$. Let $e = k2^k$. We assume $\mathcal{T}$ is a CNF in n variables which has a refutation with at most e internal edges, and show how to construct a tree-like refutation of size $2^{c \cdot 2^k \cdot \log k}$.

First, if $n > e$, there is some variable x never resolved on in the proof. Deleting x and any clause containing x from the proof makes it only smaller, and keeps it a refutation. So without loss of generality, we assume $n \leq e$.

Let $\pi = (C_1, C_2, \ldots, C_S)$ be a refutation of $\mathcal{T}$, such that G_π has e internal edges. Let π_0, π_1 be the equal partitioning of π at some $1 \leq i \leq S$, denote $e_0 = e_0(i), e_1 = e_1(i)$ and $m = |M_i|$.

Given the partition, we present two alternative methods for constructing tree-like refutations of $\mathcal{T}$. We choose our method according to the size of m:

$m \geq 2^k$ – The Brute Force method. Each clause $C \in M_i$ was derived by a derivation with at most $e_0 \leq \frac{e-m}{2} + 2 \leq (k-1)2^{k-1} + 2$ internal edges (by lemma 23), hence it has a tree like derivation of size at most $f((k-1)2^{k-1}+2) \leq 4f((k-1)2^{k-1})$ (by lemma 20). We replace π_1 with a tree-like derivation, that, similarly, has size at most $4f((k-1)2^{k-1})$, and then replace each axiom $C \in M_i$ by the tree-like derivation of size at most $4f((k-1)2^{k-1})$. The total size of this derivation is at most $16f^2((k-1)2^{k-1})$. Using induction we get:

$$16f^2\left((k-1)2^{k-1}\right) \leq 2^{4+c2^k \log(k-1)} \leq 2^{c2^k \log k}.$$

The last inequality is true for $c \geq 8$ and $k \geq 4$.

$m < 2^k$ – The Intelligent method. By lemma 21, there is a decision tree D of size at most $\binom{n}{\leq m}$ solving the CNF search problem over M_i (M_i may be satisfiable). Look at a leaf v of D. If v is labeled $C \in M_i$, using lemma 23 we claim C can be derived from $\mathcal{T}$ by a tree like derivation of size at most $f(\frac{e-m}{2}+2) \leq 4f(\frac{e-m}{2})$ (the previous inequality uses lemma 20 again), and we plug into v this "small" tree-like derivation. Otherwise v must be labeled 1. This means that the restriction ρ defined by the path leading to v, satisfies M_i, and hence $\pi_1|_\rho$ is a refutation of $\mathcal{T}|_\rho$ with at most $\frac{e-m}{2}+2$ internal edges (lemma 23). Hence, there is a tree like refutation of $\mathcal{T}|_\rho$ of size at most $4f(\frac{e-m}{2})$ (lemma 20). This can be transformed into a derivation from $\mathcal{T}$ of C_ρ, the clause which contains all literals set to 0 by ρ, without increasing its size. Plugging the tree-like derivation of C_ρ into v, for every v labeled 1 in D, transforms D into a tree-like refutation of $\mathcal{T}$. We need only show that the new tree-like refutation is not too large.

Its size is bounded by:

$$\leq \binom{n}{\leq m} \cdot 4f\left(\frac{k2^k - m}{2}\right). \tag{1}$$

$n \leq e+1$, so

$$\leq \binom{k2^k+1}{\leq m} \cdot 2f\left(\frac{k2^k - m}{2}\right). \tag{2}$$

$k \geq 4$ and $m < 2^k$, so

$$\leq 2^{k+1}\binom{k2^k}{m} \cdot f\left(\frac{k2^k - 2^k}{2} + \frac{2^k - m}{2}\right). \tag{3}$$

By lemma 20

$$\leq 2^{k+1}\binom{k2^k}{m} \cdot 2^{\frac{2^k-m}{2}} f\left((k-1)2^{k-1}\right), \tag{4}$$

$$\leq 2^{k+1}\binom{k2^k}{2^k} f\left((k-1)2^{k-1}\right), \tag{5}$$

(because $\binom{r}{m}2^{\frac{-m}{2}}$ is monotonically increasing in m for $m \leq r/4$, and $m < 2^k \leq k2^k/4$, we can substitute $m = 2^k$ to upper bound the quantity);

$$\leq 2^{k+1}(4k)^{2^k} \cdot 2^{c \cdot 2^{k-1} \cdot \log k - 1}, \tag{6}$$

$$\leq 2^{k+1+2 \cdot 2^k + 2^k \cdot (\log k) + (1/2)c(2^k \log k)}, \tag{7}$$

$$= 2^{k+1+2^k \cdot (2+\log k(1+c/2))} \leq 2^{c2^k \cdot \log k}. \tag{8}$$

The last inequality follows for $k \geq 4$ and $c \geq 8$. ∎

We are now set to prove theorem 2.

Proof [Theorem 2]. Let $\mathcal{T}$ be a contradiction, and π be its minimal size refutation, with $S(\pi) = S(\mathcal{T}) = S$. Let π' be the minimal size refutation for which $G_{\pi'}$ has maximal fan-out 2. By our discussion in the beginning of this section, $|\pi'| \leq 2S$. Hence $e \stackrel{\text{def}}{=} e(G_{\pi'}) \leq 4 \cdot S$, because at most two internal edges enter a vertex of $G_{\pi'}$. Set $k \stackrel{\text{def}}{=} \lceil \log \frac{2e}{\log e} \rceil$. Clearly $e \leq k2^k$, and recall that $f(e)$ is monotonically increasing, hence:

$$S_T(\mathcal{T}) \leq f(e) \leq f(k2^k) = 2^{O\left(\frac{S \log\log S}{\log S}\right)}. \quad \blacksquare$$

5. Applications to Automated Theorem Proving

Some of the most extensively used and investigated methods for proving unsatisfiability of CNF formulas, are called Davis–Putnam procedures. Actually, these procedures are derived from a system devised by Davis, Logemann and Loveland [13], and hence we will refer to them as *DLL Procedures*. A DLL procedure relies on choosing a variable x (called the *splitter*), and trying to refute $\mathcal{T}|_{x=T}$ and $\mathcal{T}|_{x=F}$ recursively.

If $\mathcal{T}$ is unsatisfiable, $DLL(\mathcal{T})$ terminates providing a tree-like resolution refutation of $\mathcal{T}$. The DLL procedure is actually a family of algorithms; an algorithm in the family is specified by a rule that determines the choice of splitter at each recursive step.

A different method is to seek a *minimal width* refutation. Algorithms for finding such refutations are well-known in the AI community ([24]), but were given additional theoretical motivation by [7]. (See also [4]). One algorithm to do such a search can be described as follows:

```
A (T)
  fix w=0
  Repeat {
        If 0∈T end
        Else {
             increase w
             repeatedly derive from T all clauses of
             width ≤w and add to T
             }
        }
```

Algorithm A has running time bounded by $n^{O(w(\mathcal{T}\vdash 0))}$, because this is the maximal number of different clauses that will be encountered. A simple consequence of this observation, is that the pebbling contradictions provide concrete examples for which algorithm A exponentially outperforms any DLL-procedure.

Theorem 25. *Let DLL be any implementation of a DLL procedure. There exists an infinite family of unsatisfiable CNF formulas $\mathcal{T}$ such that $Time(DLL(\mathcal{T}))$ is exponentially larger than $Time(A(\mathcal{T}))$, i.e.,* $\exp(Time^{\Omega(1)}Time(A(\mathcal{T})))$.

Proof. We use the notation of section 3. Take $\mathcal{T}=Peb_{G,S,T}$ a circuit G with high pebbling measure $P_G(S,T)=|V|/\log|V|$. By lemma 12

$$Time(A(\mathcal{T})) = |V|^{O(1)}.$$

By theorem 1, any tree-like refutation of $\mathcal{T}$, must have size at least

$$2^{\Omega\left(\frac{|V|}{\log|V|}\right)}.$$

Since the run time of any DLL procedure is bounded from below by the size of its output, which is a tree-like refutation, the theorem follows. ∎

6. Open Problems

One result of this paper is that for all CNFs

$$S_T \leq \exp\left(\frac{S \log\log S}{\log S}\right).$$

Another result is that for some CNFs

$$\exp\left(\frac{S}{\log S}\right) \leq S_T\,.$$

Can the gap between the two bounds be closed?

Hopcroft, Paul and Valiant proved that any circuit with n vertices can be pebbled with $O(n/\log n)$ pebbles [15]. Thus, one cannot hope to find "harder" graphs whose pebbling measure will match our upper bound. The analysis we perform is locally tight, so it seems that using our techniques one cannot obtain better results. Finally, we do not have good intuition as to which of the two bounds (if any) is the right one.

References

[1] R. Aharoni and N. Linial: Minimal Non-Two-Colorable Hypergraphs and Minimal Unsatisfiable Formulas, *J. of Combinatorial Theory, Series A* **43(2)** (1986), 196–204.

[2] M. L. Bonet, J. L. Esteban, N. Galesi and J. Johannsen: Exponential Separations between Restricted Resolution and Cutting Planes Proof Systems, in 39*th Symposium on Foundations of Computer Science* (FOCS 1998) pp. 638–647.

[3] P. Beame, R. Karp, T. Pitassi and M. Saks: On the Complexity of Unsatisfiability Proofs for Random k-CNF Formulas, Preliminary version in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 561–571, Dallas, TX, May 1998. Final version in *SIAM Journal on Computing* **31(4)** (2002), 1048–1075.

[4] P. Beame and T. Pitassi: Simplified and Improved resolution lower bounds, Preliminary version in *37th Annual Symposium on Foundations of Computer Science*, pp. 274–282, Burlington, VT, October 1996, IEEE. Final version in *SIAM Journal on Computing* **31(4)** (2002), 1048–1075.

[5] S. Buss and T. Pitassi: Resolution and the Weak Pigeonhole Principle, in *Springer-Verlag Lecture Notes in Computer Science*. (Publications of selected papers presented at Proceedings from Computer Science Logic '97).

[6] S. R. Buss and Gy. Turán: Resolution Proof of generalized Pigeonhole principles, *Theoretical Comp. Sci.* **62(3)** (1988), 311–317.

[7] E. Ben-Sasson and A. Wigderson: Short Proofs are Narrow – Resolution made Simple, *Journal of the ACM* **48(2)** (2001), 149–169. Preliminary version in *STOC99*.

[8] S. A. Cook: An Observation on Time-Storage Trade-Off, *J. of Comp. and Sys. Sci.* **9** (1974), 308–316.

[9] M. Clegg, J. Edmonds and R. Impagliazzo: Using the Groebner Basis algorithm to find proofs of unsatisfiability, In *Proceedings of the 28th ACM symposium on Theory of Computing*, 1996, pp. 174–183.

[10] J. R. Celoni, W. J. Paul and R. E. Tarjan: Space Bounds for a Game on Graphs, *Math. Systems Theory* **10** (1977), 239–251.

[11] S. A. Cook and R. Reckhow: The relative efficiency of propositional proof systems, *J. of Symbolic Logic* **44** (1979), 36–50.

[12] V. Chvátal and E. Szemerédi: Many Hard Examples for Resolution, *J. of the ACM* **35(4)** (1988), 759–768.

[13] M. Davis, G. Logemann and D. Loveland: A Machine program for theorem proving, *Communications of the ACM* **5** (1962), 394–397.

[14] A. Haken: The Intractability of Resolution, *Theoretical Computer Science* **39** (1985), 297–308.

[15] J. E. Hopcroft, W. Paul and L. Valiant: On Time vs. Space, *J. of the ACM* **24** (1977), 332–337.

[16] R. Impagliazzo, P. Pudlák and J. Sgall: Lower Bounds for the Polynomial-Calculus and the Groebner Basis Algorithm, *Computational Complexity* **8(2)** (1999), 127–144. Electronically at *Electronic Colloquium on Computational Complexity* Reports Series 1997, Available at http://www.eccc.uni-trier.de/eccc/, Technical Report TR97-042.

[17] R. Raz and P. McKenzie: Separation of the Monotone NC Hierarchy, *Combinatorica* **19(3)** (1999), 403–435. Preliminary version in *Proceedings of the 38th FOCS* 1997, pp. 234–243.

[18] N. Pippenger: Pebbling, *Technical Report*, IBM Watson Research Center.

[19] P. Pudlák and R. Impagliazzo: Lower bounds for DLL algorithms for k-SAT, *Proceedings of SODA 2000*, pp. 128–136.

[20] A. A. Razborov: Unprovability of Lower Bounds on Circuit Size in Certain Fragments of Bounded Arithmetic, *Izvestia of the RAN* **59(1)** (1995), 201–222.

[21] A. A. Razborov: Lower Bounds for the Polynomial Calculus, *Computational Complexity* **7(4)** (1998), 291–324.

[22] A. A. Razborov and S. Rudich: Natural Proofs, *Journal of Computer and System Sciences* **55(1)** (1997), 24–35. Preliminary version in *Proc. of the 26th STOC* 1994, pp. 204–213.

[23] A. A. Razborov, A. Wigderson and A. Yao: Read-Once Branching programs, rectangular proofs of the pigeonhole principle and the transversal calculus, Proc. of the 29th STOC 1997, pp. 739–748.

[24] B. Selman: personal communication, 1995.

[25] G. S. Tseitin: On the Complexity of Derivation in Propositional Calculus, in *Studies in Constructive Mathematics and Mathematical Logic*, Part 2, Consultants Bureau, New-York–London, 1968, pp. 115–125.

[26] A. Urquhart: Hard Examples for Resolution, *J. of the ACM* **34(1)** (1987), 209–219.

[27] A. Urquhart: The Complexity of Propositional Proofs, *The Bulletin of Symbolic Logic* **1(4)** (1995), 425–467.

Eli Ben-Sasson

Division of Engineering
and Applied Sciences
Harvard University
and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA
USA
eli@eecs.harvard.edu

Avi Wigderson

School of Mathematics
Institute for Advanced Study
Princeton, NJ
USA
and
Hebrew University
Jerusalem
Israel
avi@ias.edu

Russell Impagliazzo

Computer Science
& Engineering Department
University of California, San Diego
Mail Code 0114
9500 Gilman Drive
La Jolla, CA 92093
USA
russell@cs.ucsd.edu