社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
COMP99-33 (1999-09)

鳩の巣原理に対する木状導出原理の証明サイズについて

宮崎 修一　　岩間 一雄

京都大学 大学院情報学研究科
〒606-8501 京都市左京区吉田本町
Tel 075-753-5392
Fax 075-753-5379
{shuichi,iwama}@kuis.kyoto-u.ac.jp

あらまし　鳩の巣原理に対する木状導出原理の証明サイズの上下限の改良を行なう．上下限の改良という本来の結果の他に，木状導出原理による証明サイズが一般の導出原理に対するものよりも超多項式的に大きい例を，最も研究のなされている鳩の巣原理に対しても示した．下限の証明には，木状導出原理とSATアルゴリズムであるバックトラック法との等価性を利用した．

キーワード　導出原理，木状導出原理，鳩の巣原理

On the Size of the Tree-Like Resolution for the Pigeonhole Principle

Shuichi Miyazaki　　Kazuo Iwama

Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501
Phone +81-75-753-5392
Fax +81-75-753-5379
{shuichi,iwama}@kuis.kyoto-u.ac.jp

Abstract　We show improved upper and lower bounds of the size of tree-like resolution refutation for the pigeonhole principle. These results imply the super-polynomial separation between the size of tree-like resolution refutation and that of DAG-like resolution refutation. Although a stronger result was already shown for this separation, our aim is to do the separation for the pigeonhole principle, which is one of the most famous CNF formula in this field. We use a novel technique for the proof of a lower bound; we exploit the equivalence between tree-like resolution and the backtracking, which is a well-known algorithm for the Satisfiability.

key words　resolution, tree-like resolution, the pigeonhole principle

1 Introduction

A *proof system* is a nondeterministic procedure to prove the unsatisfiability of CNF formulas, which proceeds by applying (usually simple) rules each of which can be computed in polynomial time. Therefore, if there is a proof system which runs in polynomial time for any formula, then it means NP=coNP [5]. Since this is not likely, it has been constantly a nice research topic to try to find exponential lower bounds for existing proof systems. It should be noted that there are still a number of well-known proof systems for which no exponential lower bounds have been found, such as Frege systems [3].

Resolution is one of the most popular and simplest proof systems. Even so, it took more than two decades before Haken [6] finally obtained its exponential lower bound for the pigeonhole principle. This wonderful settlement of the major open question, however, has even stimulated the similar line of research [1, 4, 8]. The reason is that Haken's lower bound is quite far from tight and this proof, although based on an excellent idea later called bottleneck counting, is not so easy to read.

Tree-like resolution is a restricted resolution whose proof must be given as not a directed acyclic graph (DAG) but a tree. It is a common perception that a tree-like proof system is exponentially weaker than its DAG counterpart. Again, however, it was not easy to prove this presumption for resolution: In [2], Bonet et al. showed that there exists a formula for which tree-like resolution requires $2^{\Omega(n^\epsilon)}$ steps for some ϵ but DAG-like resolution needs only $n^{O(1)}$ ones.

In this paper, we do this separation between tree-like and DAG-like resolutions using the pigeonhole principle that is apparently the most famous and well-studied formula. Our new lower bound for tree-like resolution is $(\frac{n}{4})^{\frac{n}{4}}$ steps for the $n+1$ by n pigeonhole formula. The best previous lower bound is 2^n [4] which is not enough for the (exponential) separation since the best known upper bound of DAG-like resolution is $O(n^3 2^n)$ [4]. Our new lower bound shows that tree-like resolution is exponentially slower than DAG-like resolution for the pigeonhole principle.

Another contribution of this paper is that the new bound is obtained by fully exploiting the relation between resolution and backtracking. The relation itself has long been well recognized in the community, but it was informal and no explicit research results have been reported. This paper is the first one which formally claimed the benefit of using the relation. Our lower bound proof is completely based on backtracking, whose top-down structure, as one can see, makes the argument surprisingly simple and easier to understand.

In Sec.2, we give basic definitions and notations of resolution, backtracking and the pigeonhole principle. We also show the relation between resolution and backtracking. In Sec.3, we prove an $(\frac{n}{4})^{\frac{n}{4}}$ lower bound of tree-like resolution for the pigeonhole principle. In Sec.4, we give an upper bound $O(n^2 2^n)$ of the DAG-like resolution which is slightly better than $O(n^3 2^n)$ proved in [4]. It should be noted that our argument in this paper holds for a generalized pigeonhole principle, called the *weak pigeonhole principle*, which is an m by n ($m > n$) version of the pigeonhole principle. Finally, in Sec.5, we mention future research topics related to this paper.

2 Preliminaries

A *variable* is a logic variable which takes true (1) or false (0). A *literal* is a variable x or its negation $\overline{x}$. A *clause* is a sum of literals and a *CNF formula* is a product of clauses. A *truth assignment* for a CNF formula f is a mapping from the set of variables in f into $\{0,1\}$. If there is no truth assignment that satisfies all the clauses in f, we say that f is *unsatisfiable*.

The pigeonhole principle is a tautology which states that there is no bijection from a set of $n+1$ elements into a set of n elements. PHP_n^{n+1} is a CNF formula that expresses a negation of the pigeonhole principle; hence PHP_n^{n+1} is unsatisfiable. PHP_n^{n+1} consists of $n(n+1)$ variables $x_{i,j}$ $(1 \leq i \leq n+1,\ 1 \leq j \leq n)$. There are two sets of clauses. The first part consists of clauses $(x_{i,1} + x_{i,2} + \cdots + x_{i,n})$ for $1 \leq i \leq n+1$. The

second part consists of clauses $(\overline{x_{i,k}} + \overline{x_{j,k}})$, where $1 \leq k \leq n$ and $1 \leq i < j \leq n+1$. Thus there are totally $(n+1) + \frac{1}{2}(n^2(n+1))$ clauses.

The *resolution* is a proof system for unsatisfiable CNF formulas. It consists of only one rule called an *inference rule*, which infers a clause $(A+B)$ from two clauses $(A+x)$ and $(B+\overline{x})$, where each of A and B denotes a sum of literals such that there is no variable y that appears positively (negatively, resp.) in A and negatively (positively, resp.) in B. We say that the variable x is *deleted* by this inference. A *resolution refutation* for f is a sequence of clauses $C_1, C_2, \cdots, C_t$, where each C_i is a clause in f or a clause inferred from clauses C_j and C_k $(j, k < i)$. The last clause C_t must be the empty clause $(\emptyset)$. The size of a resolution refutation is the number of clauses appeared in the sequence.

A resolution refutation can be represented by a directed acyclic graph. (In this sense, we sometimes use the term *DAG-like resolution* instead of "resolution.") If a graph is restricted to a tree, the refutation is called *tree-like resolution refutation* and we call the directed acyclic graph of the refutation a *resolution refutation tree* (*rrt*). Here we formally define a resolution refutation tree. (We ignore direction of edges because there is no fear of confusion in considering trees.) An rrt for f is a binary tree. Each vertex of an rrt is associated with a clause; we denote the clause associated with the vertex v by $Cl(v)$: For each leaf v, $Cl(v)$ is a clause in f, and for each vertex v other than leaves, $Cl(v)$ is a clause inferred from $Cl(v_1)$ and $Cl(v_2)$ where v_1 and v_2 are v's children. For the root v, $Cl(v)$ is the empty clause $(\emptyset)$. The size of an rrt is the number of vertices in the rrt.

Backtracking (e.g., [7]) is an algorithm that determines whether a given CNF formula is satisfiable or not. Let $f_{x=a}$ (where x is a variable in f and $a \in \{0,1\}$) be the formula obtained by substituting the value a for the variable x. For each step, we choose a variable x and calculate $f_{x=0}$ and $f_{x=1}$ recursively. If f is simplified to 1 at any point, then f is satisfiable, and otherwise, f is unsatisfiable. Here we are interested in only unsatisfiable formulas, so we consider backtracking only for unsatisfiable formulas.

Search of backtracking is also represented by a tree. A *backtracking tree* (*btt*) for an unsatisfiable formula f is a binary tree. Each edge of a btt is labeled with a substitution for a variable satisfying the following three conditions: (i) For each vertex v and its children v_1 and v_2, the labels of edges (v, v_1) and (v, v_2) must be substitutions for the same variable, say, x; one is $x = 0$ and the other is $x = 1$. (ii) For each leaf v and each variable x, x appears at most once in the path from the root to v. (iii) For each vertex v, let $As(v)$ be the (partial) truth assignment obtained by collecting labels of edges in the path from the root to v. Then, for each v, (a) if v is a leaf then f becomes false by $As(v)$, and (b) if v is not a leaf then f does not become false by $As(v)$. The size of a btt is the number of vertices in the btt.

Proposition 1. Let f be an unsatisfiable CNF formula. If there exists an rrt for f whose size is k, then there exists a btt for f whose size is at most k.

Proof. Let $RRT(f)$ be an rrt for f whose size is k. It is known that a shortest tree-like resolution refutation is regular, i.e., for each path from the root to a leaf, each variable is deleted at most once. Thus we can assume, without loss of generality, that $RRT(f)$ is regular.

Given $RRT(f)$, we assign a label to each edge of $RRT(f)$: Let v be a vertex of $RRT(f)$ and let v_1 and v_2 be children of v. Suppose $Cl(v) = (A+B)$, $Cl(v_1) = (A+x)$ and $Cl(v_2) = (B+\overline{x})$. Then the labels $x = 0$ and $x = 1$ are assigned to edges (v, v_1) and (v, v_2), respectively. We show that this tree is a btt for f.

It is not hard to see that conditions (i) and (ii) for btt are satisfied. It remains to show that the condition (iii) is satisfied. To this aim, we show that, for each v, the clause $Cl(v)$ becomes false by the partial assignment $As(v)$. Recall that, for each leaf v, $Cl(v)$ is a clause in f. So the above statement implies that the (iii)-(a) is satisfied. We show this by induction. For the induction basis, it can be easily checked that the statement is true for the root. Then suppose that the statement is true for a vertex v, i.e., $Cl(v)$

becomes false by $As(v)$. Now we show that the statement is also true for v's children. Let v_1 and v_2 be children of v, and let $Cl(v) = (A+B)$, $Cl(v_1) = (A+x)$ and $Cl(v_2) = (B+\overline{x})$. Then the label of the edge (v, v_1) is $x = 0$. So $As(v_1)$ is $As(v)$ together with $x = 0$. Since $As(v)$ makes $(A+B)$ false, $As(v_1)$ must make $(A+x)$ false. The same argument shows that $As(v_2)$ makes $Cl(v_2)$ false.

Thus we have proved that for each leaf v, $As(v)$ makes f false. If, for an inner vertex v, $As(v)$ makes f false, then we remove v's children so that v becomes a leaf of the tree, by which we can satisfy the condition (iii)-(b). In the above operations, we do not increase the size of the tree. □

Thus to show a lower bound of the size of the tree-like resolution, it suffices to show a lower bound of the size of btts.

3 A Lower Bound

In this section, we prove a lower bound of the tree-like resolution for PHP_n^{n+1}.

Theorem 1. Any tree-like resolution refutation for PHP_n^{n+1} requires $(\frac{n}{4})^{\frac{n}{4}}$ steps.

Proof. Considering Proposition 1, we show that any btt for PHP_n^{n+1} requires $(\frac{n}{4})^{\frac{n}{4}}$ vertices. For simplicity, we consider the case that n is a multiple of 4. Let B be an arbitrary btt for PHP_n^{n+1}. As we have seen before, each vertex of B corresponds to a partial truth assignment. For better exposition, we use an $n+1$ by n array representation to express a partial assignment for PHP_n^{n+1}. Fig.1 shows an example of PHP_4^5. A cell in column i and row j corresponds to the variable $x_{i,j}$. We consider that the value 1 (resp. 0) is assigned to the variable $x_{i,j}$ if the (i,j) entry of the array is 1 (resp. 0). For example, Fig.1 (b) expresses a partial assignment such that $x_{1,4} = x_{2,2} = x_{4,4} = x_{5,3} = 0$, $x_{3,3} = x_{4,1} = 1$. It should be noted that the CNF formula PHP_n^{n+1} becomes false at a vertex v iff (i) $As(v)$ contains a column filled with 0s or (ii) $As(v)$ contains a row in which two 1s exist.

Here we give some notations. Let A be a partial assignment. If there is a 1 in the ith column of A, then the column i is called a *dead column* of A and $DC(A)$ denotes the set of dead columns of A. Similarly, if there is a 1 in the jth row of A, the row j is called a *dead row* of A and $DR(A)$ is the set of dead rows of A. For an assignment A, a 0 in the (i,j) entry of A is called a *bad* 0 if $i \notin DC(A)$ and $j \notin DR(A)$, and is called a *good* 0 otherwise. (The reason why we use terms "bad" and "good" will be seen later. Bad 0s make it difficult to count the number of vertices in B in our analysis.) $\#BZ(A)$ denotes the number of bad 0s of A. A variable $x_{i,j}$ is called an *active* variable for A if (i,j) entry of A is blank and $i \notin DC(A)$ and $j \notin DR(A)$. For example, let A_0 be the assignment described in Fig.1 (b). Then $DC(A_0) = \{3,4\}$, $DR(A_0) = \{1,3\}$, and $\#BZ(A_0) = 2$ (0s assigned to $x_{1,4}$ and $x_{2,2}$). $x_{2,4}$, for example, is an active variable for A_0. Let v be a vertex in B and v_0 and v_1 be its children. Suppose that labels of edges (v, v_0) and (v, v_1) are $x = 0$ and $x = 1$, respectively. Then we write $Var(v) = x$, namely, $Var(v)$ denotes the variable selected for substitution at the vertex v. We call v_0 a *false-child* of v and write $F(v) = v_0$. Similarly v_1 is called a *true-child* of v and we write $T(v) = v_1$.

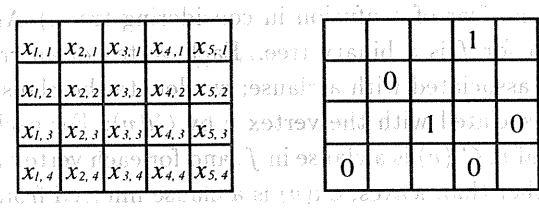

Figure 1: An array representation of a partial assignment for PHP_4^5

We want to show a lower bound of the number of vertices in B. To this end, we construct a tree S from B. Before showing how to construct S, we show some properties of S: The set of vertices of S is a subset of the set of vertices of B, so the number of vertices of S gives a lower bound of the number of vertices of B. Each vertex of S except for leaves has exactly $\frac{n}{4}$ children or exactly one child. The height of S is $\frac{n}{2}$; more precisely,

the length of any path from the root to a leaf is exactly $\frac{n}{2}$.

Now we show how to construct S. The root of S is the root of B. For each vertex v of S, we select v's children in the following way: Let $CH(v)$ be the set of v's children in the tree S we are going to construct. $CH(v)$ is initially empty and vertices are added to $CH(v)$ one by one by tracing the tree B down from the vertex v. We look at vertices $T(v)$, $T(F(v))$, $T(F(F(v)))(= T(F^2(v)))$, $\cdots$, $T(F^l(v))$, $\cdots$ in this order. We add $T(F^l(v))$ $(l \geq 0)$ to $CH(v)$ if $Var(F^l(v))$ is an active variable for $As(F^l(v))$. Fig.2 illustrates how to trace the tree B in constructing the tree S. In this example, $T(F^2(v))$ is "skipped" because $Var(F^2(v))$ is not an active variable. We stop adding vertices when $|CH(v)|$ becomes $\frac{n}{4}$. Thus v has exactly $\frac{n}{4}$ children.

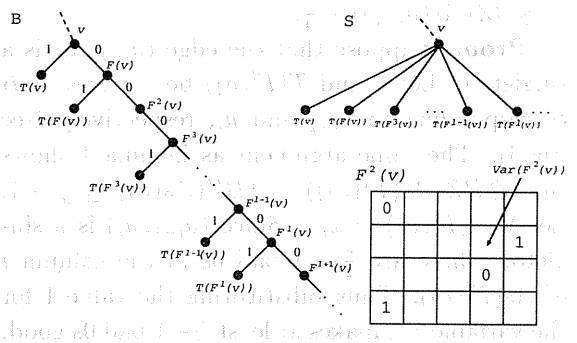

Figure 2: A part of S constructed from B

However, there is one exceptional condition to stop adding vertices to $CH(v)$ even if $|CH(v)|$ is less than $\frac{n}{4}$; when the number of bad 0s in some column reaches $\frac{n}{2}$, we stop adding vertices to $CH(v)$. More formally, let us consider a vertex $F^l(v)$. Suppose that each column of $As(F^l(v))$ contains at most $\frac{n}{2}-1$ bad 0s. Suppose that $Var(F^l(v))$ is $x_{i,j}$ where the column i of $As(F^l(v))$ contains exactly $\frac{n}{2}-1$ bad 0s and $j \notin DR(As(F^l(v)))$. (See Fig.3 for an example of the case that $n = 12$. There are eight 0s in the column i. Among them, five 0s are bad 0s.) Then $As(F^{l+1}(v))$ contains $\frac{n}{2}$ bad 0s in the column i so we do not look for vertices any more, namely, $T(F^l(v))$ is the last vertex added to $CH(v)$. (Note that $T(F^l(v))$ is always selected since $x_{i,j}$ is an active variable for $As(F^l(v))$.) In this case, $|CH(v)|$ may be less than $\frac{n}{4}$. If so, we adopt only the last vertex as a child of v, i.e., only $T(F^l(v))$ is a child of v in S. Thus, in this case, v has only one child. It should be noted that, in the tree S, every assignment corresponding to the vertex in the depth i contains exactly i 1s. We continue this procedure until the length of every path from the root to a leaf becomes $\frac{n}{2}$.

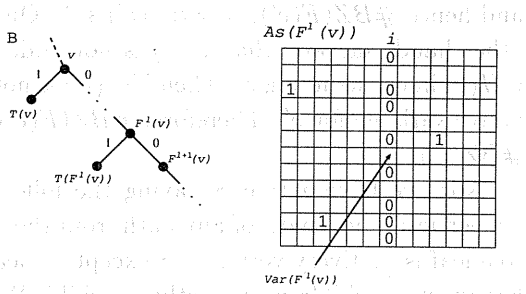

Figure 3: A condition to stop tracing B

We then show that it is possible to construct such S from any B. To see this, it suffices to show that we can always select (at least one) child for each vertex until the height of S becomes $\frac{n}{2}$. There are three cases that we do not (or cannot) add vertices to $CH(v)$ any more: (1) $|CH(v)|$ becomes $\frac{n}{4}$. (2) There arises a column which contains $\frac{n}{2}$ bad 0s. (3) We reach a leaf of B and so, there is no more vertices to trace in the tree B. In the case (1), v has $\frac{n}{4}$ children and in the case (2), v has one child, so in both cases v has at least one child. To consider the case (3), recall the definition of backtracking tree: For any leaf u of btt, $As(u)$ makes the CNF formula false. For $As(u)$ to make PHP_n^{n+1} false, $As(u)$ must have a row containing two 1s or a column full of 0s. The former case does not happen in S because we have skipped such vertices in constructing S. The latter case does not happen in the following reason: Recall that once the number of bad 0s in some column reaches $\frac{n}{2}$, we stop tracing the tree B. So, as long as we trace B to construct S, we never visit an assignment such that the number of bad 0s in a column exceeds $\frac{n}{2}-1$. Also, recall that the number of 1s in $As(v)$ is at most $\frac{n}{2}$ since v's depth in S is at most $\frac{n}{2}$.

So the number of good 0s in a column is at most $\frac{n}{2}$. Hence the number of 0s in each column is at most $n-1$, and so, any column never becomes filled with 0s.

Now let us consider the following observation which helps to prove lemmas below:

Observation 1. Consider a vertex v in B and let $v' = F^l(v)$ (see Fig.4). Suppose first that $T(v')$ is added to $CH(v)$ in constructing S. Then $Var(v')$ must be an active variable for v', and hence $\#BZ(F(v')) = \#BZ(v') + 1$. On the other hand, suppose that $T(v')$ is not added to $CH(v)$ in constructing S. Then $Var(v')$ is not an active variable for v'. Therefore, $\#BZ(F(v')) = \#BZ(v')$.

Now we have a tree S having the following properties: The length of any path from the root to a leaf is $\frac{n}{2}$. Every vertex in S except for leaves has exactly $\frac{n}{4}$ children or exactly one child. When a vertex v has one child, we call the edge between v and its child a *singleton*.

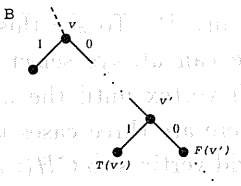

Figure 4: An example for Observation 1

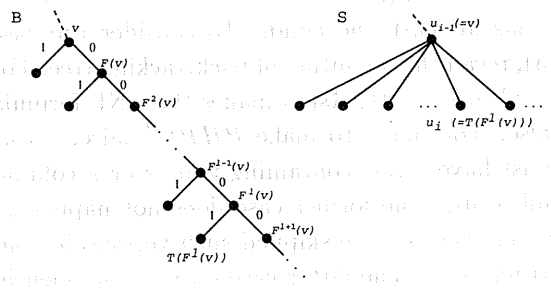

Figure 5: An example for Lemma 1

Lemma 1. Consider an arbitrary path $P = u_0u_1u_2\cdots u_{\frac{n}{2}}$ in S, where u_0 is the root and $u_{\frac{n}{2}}$ is a leaf. For each i, if (u_{i-1}, u_i) is not a singleton, $\#BZ(As(u_i))$
$\leq \#BZ(As(u_{i-1})) + \frac{n}{4} - 1$.

Proof. Consider u_{i-1} and u_i in the path P. Let v be the vertex in the btt B corresponding to u_{i-1}. Then there is some l such that $T(F^l(v))$ corresponds to u_i (see Fig.5). By Observation 1, $\#BZ(As(F^l(v))) - \#BZ(As(v))$ is equal to the number of vertices added to $CH(v)$ among $T(v), T(F(v)), T(F^2(v)), \cdots, T(F^{l-1}(v))$. So $\#BZ(As(F^l(v))) - \#BZ(As(v)) \leq \frac{n}{4} - 1$. Note that $As(T(F^l(v)))$ is the result of adding one 1 to some active variable of $As(F^l(v))$, so $\#BZ(As(T(F^l(v)))) \leq \#BZ(As(F^l(v)))$. Hence $\#BZ(As(T(F^l(v)))) \leq \#BZ(As(v)) + \frac{n}{4} - 1$, namely, $\#BZ(As(u_i)) \leq \#BZ(As(u_{i-1})) + \frac{n}{4} - 1$. □

Lemma 2. Consider an arbitrary path $P = u_0u_1u_2\cdots u_{\frac{n}{2}}$ in S, where u_0 is the root and $u_{\frac{n}{2}}$ is a leaf. For each i, if (u_{i-1}, u_i) is a singleton, then $\#BZ(As(u_i))$
$\leq \#BZ(As(u_{i-1})) - \frac{n}{4}$.

Proof. Suppose that the edge (u_{i-1}, u_i) is a singleton. Let v and $T(F^l(v))$ be vertices in B corresponding to u_{i-1} and u_i, respectively (see Fig.6). The same argument as Lemma 1 shows that $\#BZ(As(F^l(v))) - \#BZ(As(v)) \leq \frac{n}{4} - 1$. Let $Var(F^l(v)) = x_{i,j}$. Since (u_{i-1}, u_i) is a singleton, there are $\frac{n}{2} - 1$ bad 0s in the column i of $As(F^l(v))$. Thus substituting the value 1 for the variable $x_{i,j}$ makes at least $\frac{n}{2} - 1$ bad 0s good, namely, $\#BZ(As(T(F^l))) \leq \#BZ(As(F^l(v))) - (\frac{n}{2} - 1)$. Hence $\#BZ(As(T(F^l(v)))) \leq \#BZ(As(v)) - \frac{n}{4}$, namely, $\#BZ(As(u_i)) \leq \#BZ(As(u_{i-1})) - \frac{n}{4}$. □

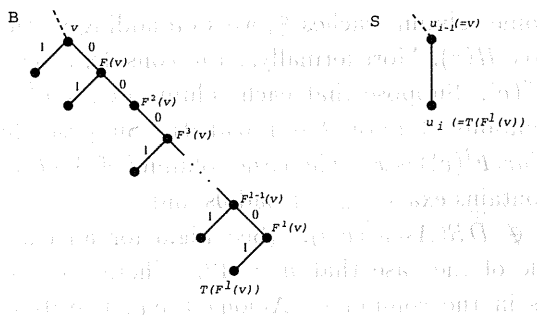

Figure 6: An example for Lemma 2

Lemma 3. Consider an arbitrary path $P =$

$u_0u_1u_2\cdots u_{\frac{n}{2}}$ in S, where u_0 is the root and $u_{\frac{n}{2}}$ is a leaf. The number of singletons in P is at most $\frac{n}{4}$.

Proof. Suppose contradictly that there exist more than $\frac{n}{4}$ singletons in the path P. We count the number of bad 0s of assignments along P. At the root, $\#BZ(As(u_0)) = 0$. Going down the path from the root u_0 to the leaf $u_{\frac{n}{2}}$, the number of bad 0s increases at most $(\frac{n}{4}-1)(\frac{n}{4}-1) < \frac{n^2}{16}$ by Lemma 1, and decreases at least $\frac{n}{4}(\frac{n}{4}+1) > \frac{n^2}{16}$ by Lemma 2. This is a contradiction because the number of bad 0s becomes negative at the leaf. This completes the proof. □

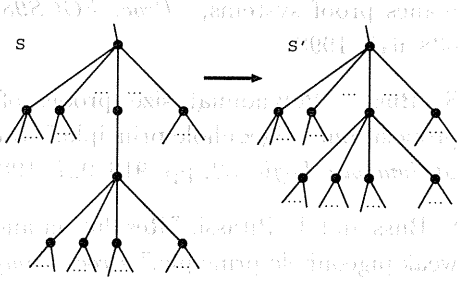

Figure 7: Shrinking S to obtain S'

Then we "shrink" the tree S by deleting all singletons from S. Let S' be the resulting tree (see Fig.7). Each vertex of S' has exactly $\frac{n}{4}$ children and the length of every path from the root to a leaf is at least $\frac{n}{4}$ by Lemma 3. So there are at least $(\frac{n}{4})^{\frac{n}{4}}$ vertices in S', and hence there are at least this number of vertices in B. □

4 An Upper Bound

It is known that the size of a DAG-like resolution refutation for PHP_n^{n+1} is $O(n^32^n)$[4]. Here we show a slightly better upper bound which is obtained by the similar argument as [4].

Theorem 2. There is a DAG-like resolution refutation for PHP_n^{n+1} whose size is $O(n^22^n)$.

Proof. Let Q and R be subsets of $\{1,2,\cdots, n+1\}$ and $\{1,2,\cdots,n\}$, respectively. Then we denote by $P_{Q,R}$ the sum of positive literals $x_{i,j}$, where $i \in Q$ and $j \in R$. Let $[i,j]$ denote the set $\{i, i+1,\cdots, j-1, j\}$. Let $P_{Q,R}$ denote the sum of positive literals $x_{i,j}$, where $i \in Q$ and $j \in R$.

We first show a rough sketch of the refutation and then describe it in detail. The 0th level is the single clause $P_{\{1\},[1,n]}$. The first level consists of n clauses $P_{[1,2],R^{(n-1)}}$ for all sets $R^{(n-1)} \subset [1,n]$ of size $n-1$. The second level consists of ${}_nC_{n-2}$ clauses $P_{[1,3],R^{(n-2)}}$ for all sets $R^{(n-2)} \subset [1,n]$ of size $n-2$. Generally speaking, the ith level consists of ${}_nC_{n-i}$ clauses $P_{[1,i+1],R^{(n-i)}}$ for all $R^{(n-i)} \subset [1,n]$ of size $n-i$. At the $(n-1)$th level, we have ${}_nC_1 = n$ clauses $P_{[1,n],\{1\}}, P_{[1,n],\{2\}}, \cdots, P_{[1,n],\{n\}}$. Finally, at the nth level, we have the empty clause. We call the clauses described here *main clauses*. Note that there are $\Sigma_{i=0}^{i=n}({}_nC_i) = 2^n$ main clauses. Fig.8 shows an example of the case when $n = 4$. A "+" sign in the (i,j) entry means the existence of the literal $x_{i,j}$ in that clause.

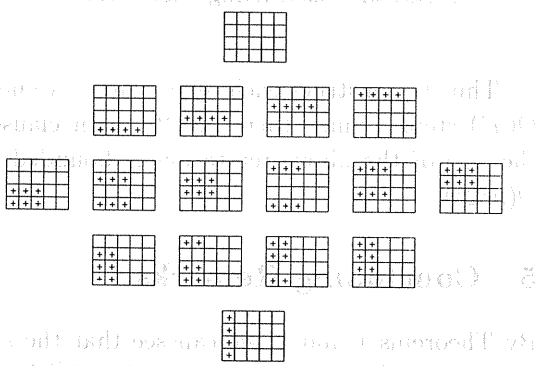

Figure 8: Main clauses of the refutation

Then we describe the detail of the refutation. Each clause at the ith level is obtained by using i clauses of the $(i-1)$th level and some initial clauses. To construct a clause $P_{[1,i+1],\{j_1,j_2,\cdots,j_{n-i}\}}$ in the ith level, we use i clauses $P_{[1,i],\{j_1,j_2,\cdots,j_{n-i},k\}}$ for all $k \notin \{j_1, j_2, \cdots, j_{n-i}\}$. First, for each k, we construct a clause $P_{[1,i],\{j_1,j_2,\cdots,j_{n-i}\}} \cup \overline{x_{i+1,k}}$ using the clause $P_{[1,i],\{j_1,j_2,\cdots,j_{n-i},k\}}$ of the $(i-1)$th level and i clauses $(\overline{x_{1,k}}+\overline{x_{i+1,k}})(\overline{x_{2,k}}+\overline{x_{i+1,k}})\cdots(\overline{x_{i,k}}+\overline{x_{i+1,k}})$. Then we construct a target clause $P_{[1,i+1],\{j_1,j_2,\cdots,j_{n-i}\}}$ by using those i clauses $P_{[1,i],\{j_1,j_2,\cdots,j_{n-i}\}} \cup \overline{x_{i+1,k}}$ and the initial clause $P_{\{i+1\},[1,n]}$. Fig.9 illustrates an example of deriving $P_{[1,3],\{1,4\}}$ in the second level from clauses $P_{[1,2],\{1,2,4\}}$ and $P_{[1,2],\{1,3,4\}}$ in the first level. Similarly as the "+" sign, a "−" sign in the (i,j) entry means the existence of the literal $\overline{x_{i,j}}$ in

that clause.

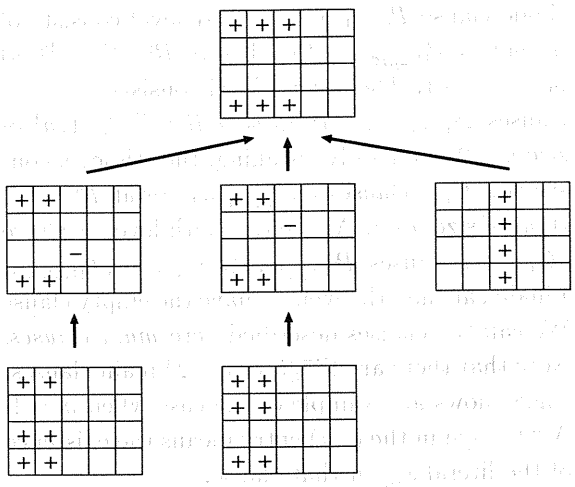

Figure 9: Constructing a main clause

Thus to construct each main clause, we need $O(n^2)$ steps. Since there are 2^n main clauses, the size of the above refutation is bounded by $O(n^2 2^n)$. □

5 Concluding Remarks

By Theorems 1 and 2, we can see that the size of a shortest tree-like resolution refutation is not bounded by any polynomial in the size of a shortest DAG-like resolution refutation. An interesting future research is to find a set of formulas that separates tree-like and DAG-like resolutions in the rate of 2^{cn} for some constant c, which is an obvious improvement of the result in [2].

Another research topic is to find a tight bound of the size of the tree-like resolution for the pigeonhole principle. Its upper bound is obtained from Theorem 2.

Corollary 1. There is a tree-like resolution refutation for PHP_n^{n+1} whose size is $O(n^3 n!)$.

Proof. This is obtained by reforming the directed acyclic graph of the refutation obtained in Theorem 2 into a tree in a trivial manner. For main clauses, we have one level-n clause, n level-$(n-1)$ clauses, $n(n-1)$ level-$(n-2)$ clauses and so on. Generally speaking, we have $n(n-1)\cdots(n-i+1)$ level-i clauses. Thus we have $\sum_{i=0}^{i=n} n(n-1)\cdots(n-i+1) \leq n \cdot n!$ main clauses. Each main clause is constructed in $O(n^2)$ steps and hence the size of the refutation is $O(n^3 n!)$. □

References

[1] P. Beame and T. Pitassi, "Simplified and improved resolution lower bounds," *Proc. FOCS96*, pp. 274–282, 1996.

[2] M. L. Bonet, J. L. Esteban, N. Galesi and J. Johannsen, "Exponential separations between restricted resolution and cutting planes proof systems," *Proc. FOCS98*, pp. 638–647, 1998.

[3] S. Buss, "Polynomial size proofs of the propositional pigeonhole principle," *Journal of Symbolic Logic*, 52, pp. 916–927, 1987.

[4] S. Buss and T. Pitassi, "Resolution and the weak pigeonhole principle," *Proc. Computer Science Logic*, pp.149–156, 1997.

[5] S. A. Cook and R. A. Reckhow, "The relative efficiency of propositional proof systems," *J. Symbolic Logic*, 44(1), pp. 36–50, 1979.

[6] A. Haken, "The intractability of resolution," *Theoretical Computer Science*, 39, pp. 297–308, 1985.

[7] P. Purdom, "A survey of average time analysis of satisfiability algorithms," *Journal of Information Processing*, 13(4), pp.449–455, 1990.

[8] A. A. Razborov, A. Wigderson and A. Yao, "Read-Once Branching programs, rectangular proofs of the pigeonhole principle and the transversal calculus," *Proc. STOC97*, pp. 739–748, 1997.