Tree-Resolution complexity of the Weak Pigeon-Hole Principle*

Stefan Dantchev[1]
Dept. of Mathematics and Computer Science, University of Leicester
dantchev@mcs.le.ac.uk
Søren Riis
Dept. of Computer Science, Queen Mary, University of London
smriis@dcs.qmw.ac.uk

Abstract

We show some tight results about the tree-resolution complexity of the Weak Pigeon-Hole Principle, PHP_n^m. We prove that any optimal tree-resolution proof of PHP_n^m is of size $2^{\theta(n \log n)}$, independently from m, even if it is infinity. So far, the lower bound know has been $2^{\Omega(n)}$. We also show that any, not necessarily optimal, regular tree-resolution proof PHP_n^m is bounded by $2^{O(n \log m)}$. To best of our knowledge, this is the first time, worst-case proofs have been considered, and a non-trivial upper bound has been proven. Finally, we discuss and conjecture some refinements of Riis' Complexity Gap theorem for tree-resolution complexity of Second Order Existential ($SO\exists$) sentences of predicate logic.

Keywords Propositional Proof Complexity, Tree-Resolution, Weak Pigeon-Hole Principle
AMS Subject Classification 03F20, 68Q17

1 Introduction

Pigeon-Hole Principle (PHP) is probably the simplest and at the same time the most widely used combinatorial principle. In its classical formulations, it states that there is no *injective* map from a finite m-element set to a finite n-element set if $m > n$. PHP_n^m is very intuitive for the human way of thinking, and it is also easily provable within set theory. This is however not the case for some *propositional proof systems*. In his seminal paper [5], Haken showed that *any resolution proof* of PHP_n^{n+1} is of size $2^{\Omega(n)}$. His proof has been generalised and simplified in [4], [1], [2]. For quite a while, the best known result had been a $2^{\Omega\left(n^2/m\right)}$ lower bound from [4], thus having left the case $m = \Omega\left(n^2/\log n\right)$ as an important open problem in resolution proof complexity. An important step has been done in [8], where a $2^{\Omega(n^{\varepsilon})}$ lower bound on *any regular-resolution proof* of PHP_n^m is proven. Shortly afterwards, the problem has finally been solved by Raz in [11], and further strengthen and improved by Razborov in [12], [13].

*A preleminary version of the paper was presented at the 16th annual IEEE Conference on Computational Complexity, June 2001

[1] Most of the work has been done while working at BRICS (**B**asic **R**esearch **I**n **C**omputer **S**cience, Centre of the Danish National Research Foundation), Dept. of Computer Science, University of Aarhus and visiting Dept. of Computer Science, Queen Mary, University of London

In this paper, we concentrate on *tree-resolution*. Even though it is probably the weakest propositional proof system one could think of, the exact complexity of tree-resolution proofs of PHP_n^m has not been known so far. A $2^{\Omega(n)}$ lower bound was shown in [3], whereas one can construct only a $2^{O(n\log n)}$ tree proof by "unfolding" the $2^{O(n)}$ DAG-resolution proof given in the same paper. A $2^{O(n\log n)}$ lower was proven in [6], but only for *ordinary pigeon-hole principle*, i.e. PHP_n^{n+1}.

The first contribution (section 3) of our paper is closing the gap. We prove a $2^{\Omega(n\log n)}$ *lower bound* on *any tree-resolution proof* of PHP_n^m, independently from m, even if it is infinity. It is *tight* up to a constant factor in the exponent or, in other words, up to *a polynomial transformation*. In order to prove the result, we introduce a new method for proving lower bounds on tree-resolution proofs. It is more general than the existing one (see e.g. [14]). The latter works only for balanced proofs, whereas any tree-like resolution proof of PHP_n^m is highly *unbalanced* as shown in the paper.

The second contribution (section 4) of the paper is considering *the worst tree regular resolution* proofs of PHP_n^m. To best of our knowledge, this is for the first time, *the worst case* proof complexity is considered. We prove an *upper bound* of $2^{O(n\log m)}$, which is non-trivial, as there are mn variables, and one can therefore expect the worst case to be as bad as 2^{mn} (we consider of course only proofs which do not contain vacuous weakening of axioms). This has the following very interesting consequence: Consider $PHP_n^{poly(n)}$, i.e. m is some polynomial in n. The optimal and the worst regular tree-resolution proofs of $PHP_n^{poly(n)}$ are *polynomially related*, and so are any two *random* regular tree-resolution proofs. This also has an implication in automated theorem proving, as it shows that there are natural problems for which any *DLL-based* proof search heuristic is as good as any other.

Finally (section 5), we discuss and conjecture some refinements of Riis' Complexity Gap theorem for tree-resolution complexity of Second Order Existential ($SO\exists$) sentences of predicate logic [14], motivated by our results. These conjectures nicely relate tree-resolution gap(s) to a possible general-resolution gap as well as to a characterisation, involving optimal and worst-case tree-resolution proofs.

2 Preliminaries

We first give some definitions. A *literal* is either a propositional variable or the negation of a propositional variable. A *clause* is a set of literals. It is satisfied by a truth assignment if at least one of its literals is true under this assignment. A set of clauses is *satisfiable* if there exists a truth assignment satisfying all the clauses.

As we have already said, by PHP_n^m we denote the claim that there is no *injective map* from a set of size m to a set of size n, where $m > n$. We encode its negation as the following set of clauses

1. $\{p_{i1}, p_{i2}, \ldots p_{in}\}$ for $1 \le i \le m$
2. $\{\overline{p}_{ij}, \overline{p}_{ik}\}$ for $1 \le i \le m$, $1 \le j < k \le n$

We allow m to be infinity. In this case, we have an infinite set of clauses, but all the clauses themselves are finite. Although we consider the *injective PHP*, all the results and proofs from the paper remain valid for the *bijective PHP*, too.

Resolution is a proof system designed to *refute* given set of clauses i.e. to prove that it is unsatisfiable. This is done by means of the resolution rule

$$\frac{C_1 \bigcup \{v\} \quad C_1 \bigcup \{\overline{v}\}}{C_1 \bigcup C_2}.$$

Thus, we can derive a new clause from two other clauses that contain a variable and its negation respectively. The goal is to derive the empty clause from the initial ones. Anywhere we say we *prove* some proposition, we mean that first we take its negation in a clausal form and then resolution is used to refute these clauses.

There is an obvious way to represent every resolution refutation as a directed acyclic graph whose nodes are labelled by clauses. The sources, i.e. the vertices with no incoming edges, are the initial clauses. The only sink, i.e. the vertex with no outgoing edges, is the empty clause. Everywhere in the paper, we say *"the size of a proof"*, we really mean *the number of vertices* in the corresponding graph.

We can now define two important restricted versions of resolution. First one is *tree resolution* when the graph is a tree or, in other words, we are not allowed to reuse any previously derived clauses. The other one is r*egular resolution* when every variable is resolved at most once along any path from a source to the sink.

For an unsatisfiable set of clause, we can consider the following *search problem*: given a truth assignment, find a clause which is falsified under it. There is a close connection between refuting an unsatisfiable set of clauses by some proof system and solving the corresponding search problem within some model of computation. In [7], it is proven that tree-resolution refutations are equivalent to *boolean decision trees*. More precisely, given a refutation of the set of clauses, it can be viewed as a decision tree, solving the search problem and vice versa. The same result holds for regular resolution refutations and *read-once branching programs*. In contrast to these, general resolution proofs are not equivalent to branching programs. As a matter of fact, there is a polynomial-size branching program, solving the search problem corresponding to PHP_n^{n+1} while all resolution refutations are of exponential size.

Everywhere in the paper, we use the equivalence between a tree-resolution proof and a boolean decision tree. All the proofs are, in fact, for decision trees, whereas the results are stated in terms of tree-resolution proofs. We only consider proofs that are regular. This is not a restriction at all as in a decision tree, it does not make sense to query any variable more than once. On the other hand, if we do not set this restriction, we would not be able to prove any upper bounds, as any given proof can be extended by (unbounded) number of "meaningless" applications of the resolution rule. Thus, from now on, every time we say "tree resolution", we really mean "regular tree resolution". As already mentioned we do not allow proofs to contain vacuous weakening of axioms. In terms of decision trees a branch terminates as soon as a contradiction is reached.

An important technique, we use to prove lower bounds on proofs, is considering a proof as a *Prover-Adversary game*. It is first introduced in [10] and developed further in [9] for general resolution. For tree resolution, it can be simplified, as done in [14]. Adversary claims that there is a satisfying assignment. Prover's task is to expose him. In order to do that, Prover asks questions about variables according to a decision tree, she holds. Clearly, there is no way for Adversary to win the game. His task is therefore to enforce a big enough subtree, contained in Prover's decision tree. If he has a strategy, enforcing that, no matter what strategy Prover uses, we have a lower bound on the tree-resolution refutations of the given set of clauses.

The only Adversary's strategy, used so far, essentially shows that there are certain number of branching points in any decision tree. It implies the existence of *a big balanced subtree* of a certain hight, thus proving an exponential in the hight lower bound. Unfortunately, this technique does not work for *unbalanced decision trees*. PHP_n^m tree-resolution refutations is such an example as we shall see in the next section. In order to tackle these, we introduce new, more general method for proving lower bounds. It requires defining a function on the nodes of the decision tree. The value of the function at any node should be a lower bound of the size of the subtree rooted by that node. Thus the function value on the root is a lower bound on the size of any decision tree solving the given search problem.

3 Optimal proofs

We first construct a $2^{O(n \log n)}$ tree-resolution proof (in fact, boolean decision tree, as we have already mentioned), and we prove the corresponding lower bound.

Here we fix some notations that we will use in both this and the next section. We denote the bigger, m-element set by M, and the other, n-element set by N. We consider M and N as the two parts of the complete bipartite graph $K_{m,n}$, and then there is 1-1 correspondence between the edges of the graph and variables p. Thus we can speak about a partial matching in $K_{m,n}$ instead of a partial function form M to N. All the queries/questions, from the decision tree, are about the edges. We can however say that a question is about a vertex, too if the corresponding edge is incident to that vertex.

Upper bound

The sketch of the construction is as follows. Obviously, Prover can restrict herself to the first $n+1$ elements of M. She asks consecutively all the questions about the first element from M, namely p_{11}, p_{12}, $\dots$ p_{1n}. If all the answers are "no", a contradiction is found. Otherwise, suppose p_{1j} is the first question with a positive answer. Prover then asks all the remaining questions about the j-th element of N, namely p_{2j}, p_{3j}, $\dots$ $p_{n+1,n}$. If at least one answer is "yes", a contradiction is found. If not, we can safely remove the first element from M and the j-th element from N, and then look for a contradiction on a PHP^{m-1}_{n-1} instance.

The boolean decision tree is given on the figure 1 below. The internal nodes are labelled

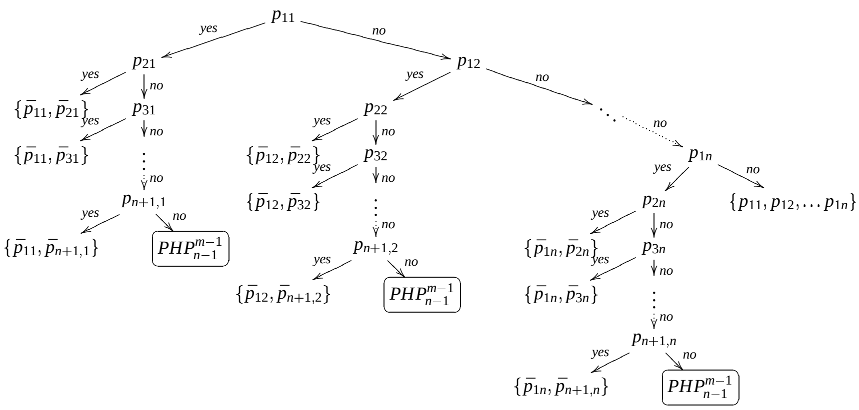

Figure 1: An optimal decision tree for PHP^m_n

with the queried variables, and the edges are marked with the corresponding answer. Every external node (leave) is labelled by the found contradiction, i.e. a clause falsified under the (partial) truth assignment corresponding to the path from the root to this node. The nodes marked by PHP^{m-1}_{n-1} are, in fact, subtrees.

What remains is to estimate the size. The decision tree for PHP^m_n consists of n copies of the decision tree for PHP^{m-1}_{n-1} plus a quadratic in n overhead. More precisely

$$S(n) = \begin{cases} nS(n-1) + 2n^2 + n + 1 & \text{if } n > 1 \\ 5 & \text{if } n = 1 \end{cases},$$

where $S(n)$ is the size of the decision tree for PHP^m_n.

It is now easy to prove by induction that $S(n) \leq 6(n+1)!$. Finally, an application of Stirling's approximation of the factorial gives the desired upper bound.

Lower Bound

The main idea in our proof is to define a function on the nodes of the decision tree. The value of the function at any node should be a lower bound of the size of the subtree rooted by that node. After having done that, it suffices to compute the function value on the root. The result is a lower bound on the size of any decision tree, solving the search problem for PHP_n^m.

We assume, w.l.o.g., that n is even. W.l.o.g. we can also assume that Prover's decision tree is *read-once*, i.e. along every path any question is asked at most once. Now, we can explain Adversary's strategy.

An important concept, we introduce here, are *counters*. A counter is attached to every vertex in M which is not matched yet to any vertex in N. In addition, there is one special counter that will be explained later on. Initially all the counters are set to zero. During the game, every counter is an upper bound of the number of vertices in N that are "forbidden" for the corresponding vertex in M. When some counter reaches the value n, Adversary "gives up", although it might be possible to continue the game a few more rounds.

We can now classify all the questions that can appear in the decision tree and show how to maintain the counters. Let k be the size of the partial matching obtained so far, i.e. the number of "yes" answers along the path from the root to the current node. There are three kinds of queries:

1. *Free-choice.* Neither of the two vertices involved is in the current partial matching and the counter of the vertex from M is less than $\frac{n}{2}+k$. Adversary chooses either "yes" or "no" answer with some probability. The actual probability does not matter, the important point is that the free choice forces Prover to branch the decision tree at that point. If the answer is "no", only the counter of the element form M increases by one. If the answer is "yes" this counter is cancelled, i.e. not maintained any more, but the counters of all the other elements in M are increased by one.

2. *Critical.* Neither of the two vertices involved is in the current partial matching but the counter of the vertex from M is equal to $\frac{n}{2}+k$. Adversary answers "yes", he current counter is cancelled, and the counters of all the other elements in M are increased by one.

3. *Forced.* Some of the vertices involved (or both) is already in the matching. Adversary answers "no" and does not change any of the counters attached to the elements in M. He however increases by one the special counter, which counts the forced questions.

First of all, it is easy to see that for a given element in M, its counter is an upper bound on the number of elements in N that cannot be matched to that element. There are also some other simple observations to be made. First one says that Adversary always "survives" certain number of rounds.

Lemma 1 *A contradiction can be found only when some counter reaches the value n. In this case, at least $\frac{n}{2}$ "yes" answers must be present on the path from the root to the current node.*

Proof A simple induction on k proves the following assertion: All the counters are bounded from above by $\frac{n}{2}+k$ along any path from a node, where the partial matching is of size k, to the node, where that size becomes $k+1$. The lemma then follows.□

The next lemma shows that there must be a very long branch in any decision tree. Together with the main result, it implies that every such tree is unbalanced.

Lemma 2 *In every decision tree for PHP_n^m, there is a path of length $\Omega\left(n^2\right)$.*

Proof Consider the path, where Adversary answers "no" to every free-choice question. It is now easy to observe that when k-th critical questions asked, the corresponding vertex from M has a counter value equal to $\frac{n}{2}+k-1$. That counter has been increased $k-1$ times because of the previous $k-1$ critical question. The remaining $\frac{n}{2}$ increases are result of "no" answers to free-choice question about the corresponding vertex. Thus, along the particular path, we consider, any "yes" answer is preceded by $\frac{n}{2}$ negative answers about the same vertex.

The lemma 1 claims that every path contains at least $\frac{n}{2}$ "yes" answers. Therefore our path contains at least $\frac{n^2}{4}$ "no" answers.$\square$

We can now prove the main result.

Theorem 1 *Every tree-resolution proof of PHP_n^m is of size $2^{\Omega(n\log n)}$.*

Proof First we define an appropriate function as it has been explained in the beginning of the section.

Let us denote by k the size of the partial matching at the current node u, i.e. the number of "yes" answers along the path from the root to u. Let us also sort the $m-k$ unmatched vertices from M in decreasing order of their counters, and denote the values of the counters themselves by $p_1 \geq p_2 \geq \ldots \geq p_{m-k}$. The forced question counter is denoted by p_0. The value of the function at the node is then defined by

$$f(u) = \prod_{i=1}^{\frac{n}{2}-k} q_i, \qquad \text{where} \quad q_i = \begin{cases} \frac{n}{2}+k-i-p_i & \text{if it is positive} \\ 1 & \text{elsewhere} \end{cases}$$

On the root, r, we have $f(r) = \left(\frac{n}{2}-1\right)!$, so that $f(r) = 2^{\Omega(n\log n)}$. It only remains to prove that at any node the function value is a lower bound for the size of the subtree rooted by the node.

The proof is by induction on the tuples of the form

$$\left(p_1, p_2, \ldots, p_{\frac{n}{2}-k}, p_0 + \sum_{i=\frac{n}{2}-k+1}^{m-k} p_i\right).$$

We order them as follows. The shorter a tuple, the smaller it is. If two tuples have equal length, the lexicographically *bigger* one is the smaller. Clearly, this ordering makes the induction work from the leaves to the root of the decision tree, as the tuple on any node is strictly bigger than the tuples on its successors in the tree.

The basis case is then $k = \frac{n}{2}$, where $f(u) = 1$, as the product is empty. Obviously, the function value at the node is a lower bound of the corresponding subtree, no matter what the only element of the tuple is.

To prove the induction steep, we need to consider all possible kind of questions that can appear at the current node u.

1. Forced. We consider the "no" branch only. Denoting its root (the "no" successor of u) by v, we have $f(u) = f(v)$, as only p_0 increases by one when going from u to v and f does not depend from p_0. By the induction hypothesis, we are done.

2. Critical. W.l.o.g. we assume that the question is about the element, having p_1 as a counter. It is so, because a critical question always involves the biggest counter (Even if there are many counters with the biggest value $\frac{n}{2}+k$, we can always consider p_1, as two elements, having the same counter value are indistinguishable to Adversary's strategy). We consider the "yes" branch only. Denoting the "yes" successor of u by v, we have again $f(u) = f(v)$. That is the case, because all the counters $p_2, \ldots, p_{\frac{n}{2}-k}$ increase by one when going from u to v, but so does k, therefore the contributions $q_2, \ldots, q_{\frac{n}{2}-k}$ do not change. q_1 vanishes at v, but its value at u is one, as $p_1 = \frac{n}{2}+k$. By the induction hypothesis, we are done.

3. Free-choice. There are three sub-cases:

   (a) The index involved, j, is greater than $\frac{n}{2}-k$. W.l.o.g. we can also assume $p_{\frac{n}{2}-k} > p_j$ since if they were equal Adversary could behave as the question were about $\frac{n}{2}-k$-th element (again, any two vertices having the same counter value are indistinguishable to Adversary's strategy). The "no" answer then does not change anything except the last element of the tuple, but f does not depend on it. So, $f(u) = f(v)$, where v is the "no" successor of u. By the induction hypothesis, we are done.

   (b) The index involved, j, is between 1 and $\frac{n}{2}-k$, but the contribution, q_j, of that element to the function f is one. That is similar to the previous sub-case, as the "no" answer leaves the value of f unchanged when going from from u to to its "no" successor v.

   (c) The index j is between 1 and $\frac{n}{2}-k$ and the contribution, q_j, of that element to the function f is greater than one. This is the only non-trivial case, in the sense that we need consider both subtrees of the current node u. Note that if there are many counters, having the same value equal to p_j, w.l.o.g. we can think that j is the minimum such index, so that the "no" answer does not change the order of the counters.
   The "no" subtree gives the tuple

$$\left(p_1,\dots p_{j-1}, p_j+1, p_{j+1}\dots, p_{\frac{n}{2}-k}, p_0 + \sum_{i=\frac{n}{2}-k+1}^{m-k} p_i\right)$$

   and the value

$$f(v) = (q_j - 1) \prod_{\substack{i=1 \\ i\neq j}}^{\frac{n}{2}-k} q_i.$$

   The "yes" subtree gives

$$\left(p_1+1,\dots p_{j-1}+1, p_{j+1}+1\dots, p_{\frac{n}{2}-k}+1, m-\frac{n}{2}+p_0+\sum_{i=\frac{n}{2}-k+1}^{m-k} p_i\right)$$

   and the value

$$f(w) = \prod_{\substack{i=1 \\ i\neq j}}^{\frac{n}{2}-k} q_i.$$

   The induction hypothesis then applies to both subtrees, so the size of the current subtree is at least

$$1 + f(v) + f(w) = 1 + f(u) > f(u).$$

This completes the proof.□

4 Worst case proofs

We first construct a $2^{O(n\log m)}$ boolean decision tree for PHP_n^m which is a lower bound for the worst-case regular tree -resolution proofs. We also show the same upper bound, i.e. any such proof cannot be worse than that. It is very important to now note that "worst case", in our context, has *a completely different meaning* than the usual one, used in Complexity Theory or Analysis of Algorithms.

Lower bound

The sketch of the construction is as follows. Prover ask all the questions about the first element from N, namely p_{11}, p_{21}, $\ldots$ p_{m1}. If all the answers are "no", we can remove the first element from N, and thus get an PHP^m_{n-1}instance. Otherwise, suppose p_{i1} is the first question with a positive answer. Prover then asks all the remaining questions about the first element of N, namely p_{i+11}, p_{i+21}, $\ldots$ p_{m1}. If at least one answer is "yes", a contradiction is found. If not, we can safely remove the first element from N and the i-th element from M, and then look for a contradiction on a PHP^{m-1}_{n-1} instance.

The boolean decision tree is given on the figure 2 below.

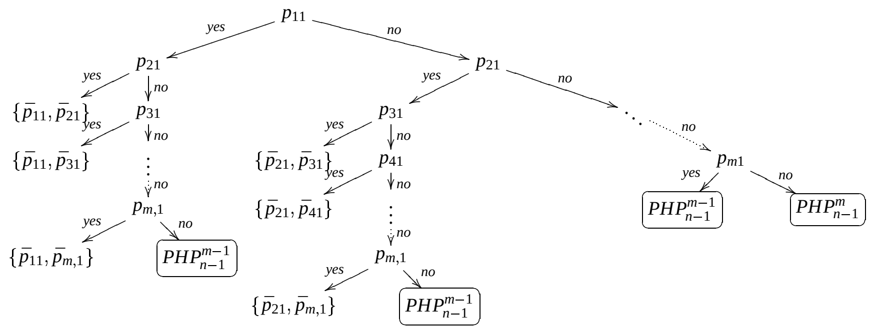

Figure 2: A worst-case decision tree for PHP^m_n

What remains is to estimate the size. The decision tree for PHP^m_n consists of m copies of the decision tree for PHP^{m-1}_{n-1}, one decision tree for PHP^m_{n-1} plus a quadratic in m overhead. More precisely

$$S(m,n) = \begin{cases} mS(m-1,n-1) + S(m,n-1) + m^2 & \text{if } n > 1 \\ 5 & \text{if } n = 1 \end{cases},$$

where $S(m,n)$ denotes the size of the decision tree for PHP^m_n.

We have

$$S(m,n) > mS(m-1,n-1) > m(m-1)S(m-2,n-2) >$$
$$\ldots \prod_{i=0}^{\lceil \frac{n}{2} \rceil - 1} (m-i) S\left(m - \left\lceil \frac{n}{2} \right\rceil, \left\lfloor \frac{n}{2} \right\rfloor\right).$$

Therefore, for every $m > n > 2$, we get

$$S(m,n) > 5\left(m - \left\lceil \frac{n}{2} \right\rceil\right)^{\lceil \frac{n}{2} \rceil} = 2^{\Omega(n \log m)}.$$

Upper bound

The main idea is the same as in the proof of the lower bound on the optimal refutation. This time however, we introduce the counters to the elements of the set N. Every counter p_j equals to m minus the number of questions about the j-th element of N that have already been asked. In other words, the counter contains *exactly* the number of possible questions about the element to be asked in the future. There is also one global counter p_0 that is the sum of all the counters p_j, $1 \leq j \leq n$.

We can now prove the main result of this section.

Theorem 2 *Every regular tree-resolution proof of PHP^m_n is of size $2^{O(n \log m)}$.*

Proof Again we define an appropriate function on the nodes of the read-once decision tree. At any node the value of the function will be an upper bound on the size of the subtree rooted at that node.

Let us denote by u the current node, and by P, $P \subseteq N$, the set of all the vertices from N that are not yet matched to any vertex in M. The function f is the defined as

$$f(u) = 2(p_0+1)\prod_{j\in P}(p_j+1) - 1.$$

At the root of the tree, r, we have $f(r) = 2(mn+1)(m+1)^n - 1$, so that $f(r) = 2^{O(n\log m)}$. It only remains to prove that at any node the function value is an upper bound for the size of the subtree rooted by the node.

The proof is by induction on the global counter p_0 .

The basis case is then $p_0 = 0$, so that all other p's are zeros and therefore $f(u) = 1$. In this case all variables have already been queried, as there are no possible questions left. Therefore a contradiction has already been found and $f(u) = 1$ is an upper bound.

To prove the induction steep, we consider the following two cases.

1. The question at the current node, u, is about the i-th element from N, and $i \notin P$. This means that element has already been matched to some element in M, so that the current question is forced. Therefore, the "yes" subtree consists of a single vertex, labelled by the contradiction found. Let us denote by v the "no" successor of u. The induction hypothesis applies at v, as p_0 decreases by one there, so the size of any subtree rooted at u is at most

$$2 + f(v) = 2 + 2p_0\prod_{j\in P}(p_j+1) - 1 \le 2(p_0+1)\prod_{j\in P}(p_j+1) - 1 = f(u).$$

2. The question at the current node, u, is about the i-th element from N, and $i \in P$. The induction hypothesis then applies to both "yes" and "no" successors of u. Denoting them by v and w respectively, we have that the size of any subtree rooted at u is at most

$$1 + f(v) + f(w) =$$
$$1 + 2p_0\prod_{j\in P\setminus\{i\}}(p_j+1) - 1 + 2p_0 p_i\prod_{j\in P\setminus\{i\}}(p_j+1) - 1 =$$
$$2p_0\prod_{j\in P}(p_j+1) - 1 < 2(p_0+1)\prod_{j\in P}(p_j+1) - 1 = f(u).$$

This completes the proof.□

5 Link to Complexity Gap theorem

In this section, we discuss a possible strengthening of Riis' complexity gap theorem for tree resolution. We first state the theorem and conjecture that it can be extended to show a gap not only between $\theta(\text{poly}(n))$ and $2^{\theta(n)}$ but also from $2^{\theta(n)}$ to $2^{\theta(n\log n)}$. We also conjecture the existence of a gap for *general resolution* and its connection with the gap for tree resolution. Let us also mention that there is no complexity gap above $2^{\theta(n\log n)}$, and, moreover, there are uniform, i.e. $SO\exists$-generated, tautologies having highly non-uniform, *fluctuating*, tree-resolution refutations. The proofs of these are however not included in the present paper as they are out of its scope.

Let us first state the complexity gap theorem itself. We give here a formulation which is slightly different than, but essentially equivalent to the original one [14].

Theorem 3 *(Complexity Gap)*

We are given a second order existential ($SO\exists$) sentence ψ *of predicate logic that* fails in all finite models *(in the original formulation first order sentence is used, but the existential quantification over function or/and relation symbols is assumed implicitly). There is a procedure which translates the sequence of sentences* $A_n :='' \psi$ *has a model of size* n'' *into an unsatisfiable set* $C_{\psi,n}$ *of clauses. The sequence* $C_{\psi,n}$ *is uniformly generated (in the sense of [15])and its size is bound by a polynomial in n. The complexity gap theorem states that either 1 or 2 holds:*

1. *The sequence* $C_{\psi,n}$ *have polynomial size in n tree-resolution refutations.*
2. *There exists a positive constant a such that for every n each tree-resolution refutation of* $C_{\psi,n}$ *has to contain at least* 2^{an} *clauses.*

Furthermore, 2 holds if and only if ψ *has an infinite model.*

Thus the gap is between polynomial and exponential size proofs and shows that no super-polynomial (e.g. $2^{\theta(\log^p n)}$ for some $p > 1$) and sub-exponential (e.g. $2^{\theta(n^\varepsilon)}$ for some $0 < \varepsilon < 1$) optimal proofs can appear.

Let us now consider the following encoding of (the negation of) PHP_n^{n+1} as a $SO\exists$ sentence (given also in [14])

$$\exists f \; ((\forall x,y \; (x = y) \vee (f(x) \neq f(y))) \wedge (\exists c \forall x \, f(x) \neq c)) \, .$$

The complexity gap theorem gives only a $2^{\Omega(n)}$ lower bound, whereas we have shown that its real complexity is $2^{\theta(n \log n)}$. We have also shown that any, not necessary optimal, proof of PHP_n^{n+1} is of that size.

Another example we consider is *the Minimum Element Principle*, saying that a (partial) order always has a minimal element. Its negation can be encoded as

$$\exists L \; ((\forall x \neg L(x,x)) \wedge (\forall x,y,z \; (L(x,y) \wedge L(y,z)) \to L(x,z)) \wedge (\forall x \exists y \, L(y,x))) \, .$$

Here $L(x,y)$ stands for $x < y$. It can be easily shown (the lower bound also follows from theorem 3) that the optimal tree-resolution proof of the minimum element principle, MEP_n (n is the number of elements), is $2^{\theta(n)}$. On the other hand, one can construct a proof which is as bad as possible, i.e. of size $2^{\theta(n^2)}$. There is also a *short, i.e. polynomial size, general resolution* proof of MEP_n.

These two examples motivate the following two conjectures. The first one states that there is a second gap, while the second gives a characterisation of both gaps in terms of optimal and worst-case tree-resolution refutation. It also relates the gaps for tree- and general resolution.

Conjecture 1 *Given a* $SO\exists$ *sentence* φ *of predicate logic that* fails in all finite and infinite models, *and denote its translation (the same as in the theorem 3) into propositional logic by* $C_{\psi,n}$*. Then either 1, 2 or 3 holds:*

1. *The sequence* $C_{\psi,n}$ *have polynomial size in n tree-resolution refutations.*
2. *There is a refutation of* $C_{\psi,n}$ *of size* 2^{an} *for some positive constant a.*
3. *There is a positive constant b such that for every n each tree-resolution refutation of* $C_{\psi,n}$ *has to contain at least* $2^{bn\log n}$ *clauses.*

Conjecture 2 *Under the assumptions of the previous conjecture:*

In the second case $C_{\psi,n}$ *has both a polynomial size general resolution proof and a worst-case tree-resolution proof, significantly worse than the optimal one, i.e. of size* $2^{\Omega(n^2)}$.

In the the third case any general resolution proof of $C_{\psi,n}$ *is of size* $2^{\Omega(n)}$*, and any tree-resolution proof is polynomially related to the optimal one.*

References

[1] P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 37th annual IEEE symposium on Foundation Of Computer Science*, pages 274–282, 1996.

[2] E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2), March 2001. A preleminary version appears at the 31st STOC, 1999.

[3] S. Buss and T. Pitassi. Resolution and the weak pigeonhole principle. In *Computer science logic (Aarhus, 1997)*, pages 149–156. Springer, 1998.

[4] S.R. Buss and G. Turán. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62:311–317, 1988.

[5] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[6] K. Iwama and S. Miyazaki. Tree-like resolution is superpolynomially slower than DAG-like resolution for the pigeonhole principle. In *Algorithms and computation (Chennai, 1999)*, Lecture Notes in Computer Science, pages 133–142. 1999.

[7] J. Krajíĉek. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, 1995.

[8] T. Pitassi and R. Raz. Regular resolution lower bounds for the weak pigeonhole principle. In *Proceedings of the 33rd annual ACM Symposium on Theory Of Computing*, pages 347–355, 2001.

[9] P. Pudlák. Proofs as games. *American Mathematical Monthly*, pages 541–550, June-July 2000.

[10] P. Pudlák and S.R. Buss. How to lie without being (easily) convicted and the lengths of proofs in propositional calculus. In *Computer Science Logic'94*, volume 993 of *Lecture Notes in Computer Science*, pages 151–162, 1995.

[11] R. Raz. Resolution lower bounds for the weak pigeonhole principle. Technical Report 21, Electronic Colloquium on Computational Complexity, 2001. Avaliable at http://www.eccc.uni-trier.de/eccc/.

[12] A. Razborov. Improved resolution lower bounds for the weak pigeonhole principle. Technical Report 55, Electronic Colloquium on Computational Complexity, 2001. Avaliable at http://www.eccc.uni-trier.de/eccc/.

[13] A. Razborov. Resolution lower bounds for the weak functional pigeonhole principle. Technical Report 75, Electronic Colloquium on Computational Complexity, 2001. Avaliable at http://www.eccc.uni-trier.de/eccc/.

[14] S. Riis. A complexity gap for tree-resolution. *Computational Complexity*, 10:179–209, 2001.

[15] S.M. Riis and M. Sitharam. Generating hard tautologies using predicate logic and the symmetric group. *Logic Journal of the IGPL*, 8(6):787–795, 2000.