Partial Gates and the Identification of their Solution Sets

Ryuhei Uehara† Kensei Tsuchida‡

†Center for Information Science, Tokyo Woman's Christian University,
Suginami-ku, Tokyo 167, Japan, uehara@twcu.ac.jp
‡Faculty of Engineering, Toyo University,
Kujirai, Kawagoe-shi, Saitama 350, Japan, kensei@krc.toyo.ac.jp

Key words: Analysis of algorithms, upper/lower bounds of problems, randomized algorithms.

Abstract

A partial gate computes a function of part of the inputs to a circuit. Some of the inputs are connected to the partial gate, and the others are not connected anywhere. For a given partial OR, AND, Parity, or Threshold gate, the problem to check the connectivity in the gate is investigated. The task is to minimize the depth of a decision tree for this problem. When the number of connected inputs is not given, each upper bound coincides exactly with its lower bound. When the number of connected inputs is given and small, each upper bound coincides asymptotically with its lower bound. In the case that the number of connected inputs of a given partial Parity gate is given, the upper bound is given by a Las Vegas algorithm.

1 Introduction

A *partial gate* computes a function of part of the inputs to a circuit. Some of the inputs are connected to the partial gate, and the remaining inputs are not connected (see the next section for a formal definition). We investigate the problem of how to check connectivity in a given partial gate. This problem is a typical decision tree problem which theoretically is of large interest. One might think that this problem is a kind of machine learning of restricted types of propositional formulas, which has been widely investigated (see D. Angluin for a comprehensive reference[1]). However, our problem is different from such machine learning. While the purpose of machine learning is to identify an *unknown* formula, our problem is considered on a given *known* gate. That is, for a given known gate, the purpose of our problem is to identify whether each input of the gate is connected or not.

In this paper, we deal with the problem for a partial OR, AND, Parity, or Threshold gate. Remember that a Threshold gate has one additional parameter; threshold value. Let A be a given partial gate with n inputs. We label each input $1, 2, \cdots, n$. The set of connected inputs is called a *solution set* of A, and is denoted by $S(A)$. That is, each input in $S(A)$ is connected to the gate contained in A, and each input in $\{1, 2, \cdots, n\} - S(A)$ is disconnected. In other words, a given partial gate A computes a function of the inputs in $S(A)$. For example, a partial AND gate A outputs '1' if every input in $S(A)$ is set by '1', and the other inputs have no influence on its output. First, we consider the problem to check connectivity in a given partial gate of n inputs. Next, we consider the problem in which the size of the solution set is given. We also consider the problem for a partial Threshold gate in the case that the size of the solution set and its threshold value are given.

The efficiency of an algorithm for the problem is measured by the number of queries asked. The answer to a query is the output of a given partial gate. More precisely, we make a conjecture $S' \subseteq \{1, 2, \cdots, n\}$ as a query, set '1' to the input i of a given partial gate for each i in S', set '0' to the remainder inputs, and check the output of the gate.

Let n and k denote the number of inputs and the number of the solutions of a given partial gate, respectively. Then the results are summarized as follows: (1) When only n is given, for each partial gate, the upper bound of the problem coincides *exactly* with the lower bound. (2) When n and k are given, for a given partial OR, AND, or Threshold gate, the upper bound coincides *asymptotically* with the lower bound if $k = o(n)$. (3) When n and k are given, for a given partial Parity gate, the upper bound coincides *asymptotically* with the lower bound if

$k = O(n^{1-\epsilon})$ for some $\epsilon > 0$. (4) For a partial Threshold gate, the information for the threshold value has few influence to the results.

The lower bounds are given by the information theory approach. These lower bounds are valid when the algorithm makes no error and has no assumption about the distribution of the inputs. All upper bounds are given by the deterministic algorithms except the algorithm for a partial Parity gate with given n and k. This upper bound is given by a *Las Vegas* algorithm, which uses random bits, and always gives correct answers (see Chapter 1.2 in [3]). As a result, this upper bound contains the expected number of the queries. We remark that the lower bound given by the information theoretic approach is valid for a Las Vegas algorithm.

2 Preliminaries

In this section, we give some formal definitions and notations.

Recall that a binary relation *parity* is defined by $0 \oplus 0 = 1 \oplus 1 = 0$, and $0 \oplus 1 = 1 \oplus 0 = 1$, and a *threshold function* T_n^t is defined by $T_n^t(x_1, x_2, \cdots, x_n) = 1$ if and only if $\sum_{i=1}^n x_i \geq t$. For a partial Threshold gate, if the number of connected inputs is less than its threshold value, it always outputs '0', and we can not solve the problem. To exclude this meaningless case, we assume that the number of connected inputs is greater than or equal to its threshold value.

Definition 1 Let n, k and t be integers with $n \geq k \geq t \geq 0$, and i be an injective function such that $0 \leq i(j) \leq n$ for $0 \leq j \leq k$. The *problem* for a partial gate is one of the classes defined by the following:

(1) P-OR(n, k) is the class of all functions $f(x_1, x_2, \cdots, x_n) = x_{i(1)} \vee x_{i(2)} \vee \cdots \vee x_{i(k)}$, and P-OR$(n)$ is the class defined by $\bigcup_{k=0}^n$ P-OR(n, k).

(2) P-AND(n, k) is the class of all functions $f(x_1, x_2, \cdots, x_n) = x_{i(1)} \wedge x_{i(2)} \wedge \cdots \wedge x_{i(k)}$, and P-AND$(n)$ is the class defined by $\bigcup_{k=0}^n$ P-AND(n, k).

(3) P-Parity(n, k) is the class of all functions $f(x_1, x_2, \cdots, x_n) = x_{i(1)} \oplus x_{i(2)} \oplus \cdots \oplus x_{i(k)}$, and P-Parity$(n)$ is the class defined by $\bigcup_{k=0}^n$ P-Parity(n, k).

(4) P-Thr(n, k, t) is the class of all functions $f(x_1, x_2, \cdots, x_n) = T_n^t(x_{i(1)}, x_{i(2)}, \cdots, x_{i(k)})$. P-Thr$(n, k)$ is the class defined by $\cup_{t=0}^k$P-Thr(n, k, t), and P-Thr(n) is the class defined by $\cup_{k=0}^n$P-Thr(n, k).

Let A be a partial gate computing $f(x_1, x_2, \cdots, x_n)$ in some class P. Suppose that A is characterized by an injective function i and an integer k. We call a set $\{i(1), i(2), \cdots, i(k)\}$ a *solution set* of A, and denote by $S(A)$. Each element in $S(A)$ is called a *solution* of A. Each element in $\{0, 1\}^n$ is called a *query*. The *answer* to a query q is $f(q)$.

In this paper, we ask a query by calling a subroutine Query(S') with a parameter $S' \subseteq \{1, 2, \cdots, n\}$. Query$(S')$ returns the answer to a query $q = (q_1, q_2, \cdots, q_n)$, where $q_i = 1$ if and only if $i \in S'$ for each i.

By de Morgan's law, P-OR(n) and P-OR(n, k) can easily be seen as equivalent to P-AND(n) and P-AND(n, k), respectively. That is, the results for a partial OR gate can immediately be reduced to the results for a partial AND gate. Thus, hereafter, we omit the proofs for a partial AND gate.

Let P be the problem for a partial gate. A deterministic algorithm *solves* P in *time* t if it finds $S(A)$ for any A in P asking query at most t times. A Las Vegas algorithm *solves* P in *expected time* t if the expected number of queries to be asked such that the algorithm finds $S(A)$ for any A in P is at most t for any distribution of P.

Definition 2 When no Las Vegas algorithm solves P in expected time less than t, we say that the *lower bound* of P is t (denoted by Low(P) $= t$). When there exists a deterministic algorithm that solves P in time t, we say that the *upper bound* of P is t (denoted by Up(P) $= t$). When there exists a Las Vegas algorithm that solves P in expected time t, we say that the *expected time of the upper bound* of P is t (denoted by Up(P) $= E(t)$, and Up(P) $= EO(t')$ if $t = O(t')$).

We denote the binary logarithm by log, and the binomial coefficient by $\binom{n}{k}$, which is defined by $\frac{n!}{k!(n-k)!}$.

3 Lower bounds

In this section, to show the lower bounds, we use "information theoretic lower bound" such that at least $\log n$ bits of information are required to distinguish n cases (see Chapter 5.3 in [2]). In other words, we must ask at

least $\log n$ queries to solve a problem of size n, since one query outputs one bit of information. It is easy to see that the information theoretic lower bound is valid for the Las Vegas algorithm, since Las Vegas algorithms do not make errors, and require no assumptions about the distribution of the input. The following lower bounds can be proved by simple counting arguments.

Proposition 1 Let n and k be integers with $n \geq k \geq 0$; $\mathsf{P}(n,k)$ be one of $\mathsf{P\text{-}OR}(n,k)$, $\mathsf{P\text{-}AND}(n,k)$, and $\mathsf{P\text{-}Parity}(n,k)$; and $\mathsf{P}(n)$ be one of $\mathsf{P\text{-}OR}(n)$, $\mathsf{P\text{-}AND}(n)$, and $\mathsf{P\text{-}Parity}(n)$. Then, (1) $\textsc{Low}(\mathsf{P}(n,0)) = 0$, (2) $\textsc{Low}(\mathsf{P}(n,k)) = \lceil \log \binom{n}{k} \rceil \geq \lceil k \log \frac{n}{k} \rceil$ for $k > 0$, and (3) $\textsc{Low}(\mathsf{P}(n)) = n$.

Theorem 2 Let n, k, and t be integers with $n \geq k \geq t \geq 0$. Then,

(1) $\textsc{Low}(\mathsf{P\text{-}Thr}(n,k,0)) = \textsc{Low}(\mathsf{P\text{-}Thr}(n,0)) = \textsc{Low}(\mathsf{P\text{-}Thr}(0)) = 0$,

(2) $\textsc{Low}(\mathsf{P\text{-}Thr}(n,k,t)) = \lceil \log \binom{n}{k} \rceil \geq \lceil k \log \frac{n}{k} \rceil$ for $t > 0$,

(3) $\textsc{Low}(\mathsf{P\text{-}Thr}(n,k)) = \lceil \log \left(k \binom{n}{k} + 1\right) \rceil \geq \lceil \log k + k \log \frac{n}{k} \rceil$ for $k > 0$, and

(4) $\textsc{Low}(\mathsf{P\text{-}Thr}(n)) = n - 1 + \lceil \log(n+1) \rceil$ for $n > 0$.

Proof. Using their definitions, we have (1) and (2) immediately.

(3) For any partial Threshold gate in $\mathsf{P\text{-}Thr}(n,k)$ with $k > 0$, it has a threshold value t with $k \geq t \geq 0$. Considering the case $t = 0$, we have $|\mathsf{P\text{-}Thr}(n,k)| = \sum_{t=0}^{k} |\mathsf{P\text{-}Thr}(n,k,t)| = 1 + \sum_{t=1}^{k} \binom{n}{k} = k \binom{n}{k} + 1$.

(4) By (3), we find that $|\mathsf{P\text{-}Thr}(n)| = \sum_{k=0}^{n} |\mathsf{P\text{-}Thr}(n,k)| = \sum_{k=1}^{n} \left(k \binom{n}{k} + 1\right) + 1 = n + 1 + \sum_{k=0}^{n} k \binom{n}{k} = n + 1 + n2^{n-1} = \left(n + \frac{n+1}{2^{n-1}}\right) 2^{n-1}$. It is not difficult to see that $\lceil \log \left(n + \frac{n+1}{2^{n-1}}\right) 2^{n-1} \rceil = n - 1 + \lceil \log(n+1) \rceil$ for any integer $n > 0$. ∎

4 Upper bounds

Proposition 3 The upper bounds of the following classes are equal to 0; $\mathsf{P\text{-}OR}(0)$, $\mathsf{P\text{-}AND}(0)$, $\mathsf{P\text{-}Parity}(0)$, $\mathsf{P\text{-}Thr}(0)$, $\mathsf{P\text{-}OR}(n,0)$, $\mathsf{P\text{-}AND}(n,0)$, $\mathsf{P\text{-}Parity}(n,0)$, $\mathsf{P\text{-}Thr}(n,0)$, and $\mathsf{P\text{-}Thr}(n,k,0)$.

First, we show $\textsc{Up}(\mathsf{P\text{-}OR}(n))$, $\textsc{Up}(\mathsf{P\text{-}AND}(n))$, $\textsc{Up}(\mathsf{P\text{-}Parity}(n))$, and $\textsc{Up}(\mathsf{P\text{-}Thr}(n))$.

Proposition 4 $\textsc{Up}(\mathsf{P\text{-}OR}(n)) = \textsc{Up}(\mathsf{P\text{-}AND}(n)) = \textsc{Up}(\mathsf{P\text{-}Parity}(n)) = n$.

Proof. Asking n queries, $\textsc{Query}(\{1\})$, $\textsc{Query}(\{2\})$, $\cdots$, $\textsc{Query}(\{n\})$, we find that $\textsc{Up}(\mathsf{P\text{-}OR}(n)) = n$. It is easy to see that the algorithm also solves $\mathsf{P\text{-}Parity}(n)$ with n queries. ∎

Theorem 5 $\textsc{Up}(\mathsf{P\text{-}Thr}(n)) = n - 1 + \lceil \log(n+1) \rceil$.

Proof. Let k and t be the number of solutions and the threshold value of the given partial Threshold gate, respectively. Using the binary search, the first phase of our algorithm finds the minimum s such that $\textsc{Query}(\{1,2,\cdots,s\}) = 1$. When $s = 1$, asking more one query $\textsc{Query}(\emptyset)$, the algorithm distinguishes the case $t = 0$ from the case that $t = 1$ and the input 1 is a solution. The algorithm halts if $t = 0$. We assume $t > 0$. Then the algorithm has got a solution s and a set $\{1,2,\cdots,s\}$ containing t solutions. For each $j = 1,2,\cdots,s-1$, we can regard the second phase to determine whether j is a solution or not as $\mathsf{P\text{-}AND}(s-1)$ using $\textsc{Query}(\{1,2,\cdots,s\} - \{j\})$ instead of $\textsc{Query}(\{1,2,\cdots,s-1\} - \{j\})$. On the other hand, for each $j = s+1, s+2, \cdots, n$, we can regard the third phase to determine whether j is a solution or not as $\mathsf{P\text{-}OR}(n-s)$ using $\textsc{Query}(\{1,2,\cdots,s-1\} \cup \{j\})$ instead of $\textsc{Query}(\{j\})$. Thus, the second and third phases can be implemented using the algorithms produced in the proof of Proposition 4. The correctness of the algorithm is trivial.

Now, let us estimate the time complexity of the algorithm. We can implement the first phase as the binary search on a set of $n+1$ elements, concluding the case $t = 0$. Thus, the first phase asks $\lceil \log(n+1) \rceil$ queries. Clearly, the second and third phases ask $s-1$ and $n-s$ queries, respectively. Thus, the algorithm solves $\mathsf{P\text{-}Thr}(n)$ with at most $n - 1 + \lceil \log(n+1) \rceil$ queries. ∎

Remark that the algorithm in the proof of Theorem 5 does not explicitly use the value of t.

Now, we turn to the case when the number of solutions is given. Second, we show more efficient algorithm for $\mathsf{P\text{-}OR}(n,k)$ and $\mathsf{P\text{-}AND}(n,k)$.

Theorem 6 For any n and k with $n \geq k > 0$, $\textsc{Up}(\mathsf{P\text{-}OR}(n,k))$ and $\textsc{Up}(\mathsf{P\text{-}AND}(n,k))$ are at most $k\lceil \log \frac{n}{k} \rceil + 2k - 1$.

Proof. The top level structure of the deterministic algorithm for solving P-OR(n,k) first initialize the set T by $\emptyset$, and call the subroutine FindSol with setting $\{1,2,\cdots,n\}$ as a parameter. Let S be the set given to FindSol as a parameter. FindSol first checks whether $|S|=1$ or not. If $|S|=1$, FindSol updates T by $T\cup S$ and returns. Otherwise, FindSol splits S into S_1 and S_2 such that $|S_1|=\lfloor\frac{1}{2}|S|\rfloor$ and $|S_2|=\lceil\frac{1}{2}|S|\rceil$, and it recursively calls FindSol(S_i) if Query$(S_i)=1$ for each $i=1,2$. Finally, the algorithm outputs T as the solution set. Since the set given to FindSol as a parameter contains at least one solution, the correctness of the algorithm is trivial.

Now, let us estimate the time complexity of the the algorithm. Let S' be the set given to FindSol as a parameter. We denote by $S(S')$ the set contains all solutions in S'. When FindSol splits S' into S'_1 and S'_2, there exists a real number p with $0\le p\le 1$ such that $|S(S'_1)|=p\,|S(S')|$ and $|S(S'_2)|=(1-p)\,|S(S')|$. We can regard this process as reducing the problem from P-OR$(|S'|,|S(S')|)$ to P-OR$(\lfloor\frac{1}{2}|S'|\rfloor,p\,|S(S')|)$ and P-OR$(\lceil\frac{1}{2}|S'|\rceil,(1-p)\,|S(S')|)$. Through this process, by Proposition 1, the number of possible solution sets is reduced from $\binom{|S'|}{|S(S')|}$ to $\binom{\lfloor\frac{1}{2}|S'|\rfloor}{p|S(S')|}\binom{\lceil\frac{1}{2}|S'|\rceil}{(1-p)|S(S')|}$.

Here, for any integers n and k with $n\ge k\ge 0$, it is not difficult to see that $\binom{n}{pk}\binom{n}{(1-p)k}$ has its peak when $p=\frac{1}{2}$ for $0\le p\le 1$. This together with the fact $\binom{n}{k}=\binom{n}{n-k}$ implies that $\binom{\frac{1}{2}n}{pk}\binom{\frac{1}{2}n}{(1-p)k}$ has its peak when $p=\frac{1}{2}$ for any integers n and k and any real number p such that $n\ge k\ge 0$ and $\frac{1}{2}n\ge pk,(1-p)k\ge 0$.

This observation leads us to the fact that the following case is the worst case: In splitting of FindSol, $||S(S_1)|-|S(S_2)||\le 1$ always holds. We estimate the time complexity of the problem in this worst case. Let k' be the integer with $k'=\lceil\log k\rceil$. Then each internal node of depth k' of the binary search tree of the algorithm is the root of a subtree which contains at most one solution. For each subtree containing one solution, the algorithm will find the solution by the binary search. The algorithm asks no more queries for each subtree containing no solutions. Thus, the number of queries is at most $\sum_{i=0}^{k'-1}2^i+k\log\lceil\frac{n}{2^{k'}}\rceil\le 2^{k'}-1+k\lceil\log\frac{n}{k}\rceil\le 2k-1+k\lceil\log\frac{n}{k}\rceil$. ∎

Remark that the algorithm in the proof of Theorem 6 does not explicitly use the value of k.

Third, we show an efficient Las Vegas algorithm that solves P-Parity(n,k).

Theorem 7 For any n and k with $n\ge k>0$, Up(P-Parity(n,k)) is at most $E(k)+k\lceil\log n\rceil$.

Proof. The top level structure of the Las Vegas algorithm for solving P-Parity(n,k) finds one solution k times using the binary search with a partial *parity* gate. We consider when the binary search works using a parity gate. When a set S contains an odd number of solutions, for any splitting of S, one of the two sets will contain an odd number of solutions. Thus, the binary search with a parity gate works for the set containing an odd number of solutions. The heart of the algorithm is, for a given set S containing an *even* number of solutions, the subroutine PickUp which returns a subset S' containing an *odd* number of solutions. PickUp(S) repeatedly do the following until Query$(S')=1$; it initializes S' by an empty set, and randomly picks up each element from S to S'. That is, in each repetition, each element in S is set into S' with probability $\frac{1}{2}$. Clearly, PickUp(S) returns a subset S' containing an odd number of solutions.

Now, let us estimate the time complexity of the algorithms. First, the number of iterations of the while-loop in the top level structure is k. In each iteration, the parity of the number of the solutions can be determined by the value of k. Thus, PickUp is called $\lfloor\frac{k}{2}\rfloor$ times (without asking query). Next we consider the expected number that PickUp(S) calls Query(S'). Since S contains an even number of solutions, the probability of both S' and $S-S'$ containing an odd number of solutions is the same as if they contain an even number of solutions. Thus, S' contains an odd number of solutions with probability $\frac{1}{2}$. Hence, the expected number that PickUp(S) calls Query(S') is equal to $\sum_{i=1}^{\infty}i\cdot\left(\frac{1}{2}\right)^i=2$. Finally, since $|S'|\le|S|$ holds, each binary search asks at most $\lceil\log n\rceil$ queries. Thus the algorithm solves P-Parity(n,k) in $E(2\lfloor\frac{k}{2}\rfloor)+k\lceil\log n\rceil$ times. ∎

Finally, we construct an efficient algorithm solving both P-Thr(n,k) and P-Thr(n,k,t).

Theorem 8 Let n, k, and t be integers with $n\ge k\ge t>0$. Then the upper bounds of P-Thr(n,k,t) and P-Thr(n,k) are at most $3k-5+\lceil\log(n+1)\rceil+(k-1)\log\frac{n-1}{k-1}$.

Proof. We first consider Up(P-Thr(n,k,t)). The top level structure of the deterministic algorithm for P-Thr(n,k,t) is the same as the algorithm in the proof of Theorem 5. The first phase is the same. After the first phase, the algorithm knows n,k,t, and s such that s is a solution and a set $\{1,2,\cdots,s\}$ contains t solutions. Thus, the algorithm can use the algorithms for P-AND$(s-1,t-1)$ and P-OR$(n-s,k-t)$ shown in the proof of Theorem 6 to implement the second and third phases, respectively. Thus, Up(P-Thr(n,k,t)) is at most $\lceil\log(n+1)\rceil+(t-1)\lceil\log\frac{s-1}{t-1}\rceil+(k-t)\lceil\log\frac{n-s}{k-t}\rceil+2k-4$ for some s with $t\le s\le n$. Now,

$(t-1)\log\frac{s-1}{t-1}+(k-t)\log\frac{n-s}{k-t}=\log\left(\frac{s-1}{t-1}\right)^{(t-1)}\left(\frac{n-s}{k-t}\right)^{(k-t)}=\log\left(\frac{n-1}{k-1}\right)^{(k-1)}\left(\left(\frac{a}{b}\right)^{b}\left(\frac{1-a}{1-b}\right)^{(1-b)}\right)^{(k-1)}$, where $a=\frac{s-1}{n-1}$ and $b=\frac{t-1}{k-1}$. It is not difficult to see that $\left(\frac{a}{b}\right)^{b}\left(\frac{1-a}{1-b}\right)^{(1-b)}\leq 1$ for any $0\leq a,b\leq 1$. Thus, $(t-1)\log\frac{s-1}{t-1}+(k-t)\log\frac{n-s}{k-t}\leq(k-1)\log\frac{n-1}{k-1}$. Hence, we have $\textsc{Up}(\mathsf{P\text{-}Thr}(n,k,t))\leq\lceil\log(n+1)\rceil+(t-1)\lceil\log\frac{s-1}{t-1}\rceil+(k-t)\lceil\log\frac{n-s}{k-t}\rceil+2k-4\leq\lceil\log(n+1)\rceil+(t-1)\log\frac{s-1}{t-1}+(k-t)\log\frac{n-s}{k-t}+3k-5\leq\lceil\log(n+1)\rceil+(k-1)\log\frac{n-1}{k-1}+3k-5$.

As remarked after the proofs of Theorem 5 and 6, the first phase does not require the value of t, and the second and third phases do not require the number of the solutions. Thus the algorithm also solves $\mathsf{P\text{-}Thr}(n,k)$. ∎

The following corollary is the summary of Sections ?? and ??.

Corollary 9 (1) Let $\mathsf{P}(n)$ be one of $\mathsf{P\text{-}OR}(n)$, $\mathsf{P\text{-}AND}(n)$, $\mathsf{P\text{-}Parity}(n)$, and $\mathsf{P\text{-}Thr}(n)$. Then $\textsc{Up}(\mathsf{P}(n))=\textsc{Low}(\mathsf{P}(n))$. (2) Let $\mathsf{P}(n,k)$ be one of $\mathsf{P\text{-}OR}(n,k)$, $\mathsf{P\text{-}AND}(n,k)$, and $\mathsf{P\text{-}Thr}(n,k)$. Then $\textsc{Up}(\mathsf{P}(n,k))$ coincides asymptotically with $\textsc{Low}(\mathsf{P}(n,k))$ if $k=o(n)$. (3) $\textsc{Up}(\mathsf{P\text{-}Parity}(n,k))$ coincides asymptotically with $\textsc{Low}(\mathsf{P\text{-}Parity}(n,k))$ if $k=O(n^{1-\epsilon})$ for some $\epsilon>0$. (4) $\textsc{Up}(\mathsf{P\text{-}Thr}(n,k,t))$ coincides asymptotically with $\textsc{Low}(\mathsf{P\text{-}Thr}(n,k,t))$ if $k=o(n)$.

5 Concluding Remarks

There are two problems that still need to be solved: The first problem is whether there exists a deterministic algorithm solving $\mathsf{P\text{-}Parity}(n,k)$ in $O(k\log n)$ time. We can solve this problem by constructing a deterministic algorithm that picks up a set containing an odd number of solutions in $O(\log n)$ time. Remember that our Las Vegas algorithm achieves this using random bits in constant expected time. The second problem is whether the bounds can be improved for large k. Ingo Wegener noticed the following facts [4]: $\mathsf{P\text{-}Parity}(n,k)$ and $\mathsf{P\text{-}Parity}(n,n-k)$ are equivalent, and $\mathsf{P\text{-}Parity}(n,2)$ has $3\lceil\log n\rceil$ upper bound given by a deterministic algorithm. These facts give a partial answer to the problems.

Acknowledgment

The authors would like to thank Professor Etsuro Moriya for his comments on this problem, and Professor Masao Hara and Professor Sei'ich Tani for their helpful suggestions. The first author wishes to thank Professor Takumi Kasai, Professor Seinosuke Toda, and Professor Zhi-Zhong Chen for their helpful advice during the course of this work. The first author also thanks to Professor Misao Nagayama for her comments on earlier drafts of this paper.

References

[1] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

[2] D.E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing Company, 1973. Sorting and Searching.

[3] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge, 1995. 1995.

[4] I. Wegener. Personal communication.