A Note on Set Bit Enumeration

Thomas Schwentick
Johannes-Gutenberg Universität Mainz
tick@informatik.mathematik.uni-mainz.de

J. Ramachandran
Ohio State University
ramachan@cis.ohio-state.edu

June 28, 1994

Abstract

An m-set-bit-enumerator for a function f is a source of information that, in response to the query: "How many set bits ('1's) are there in the binary representation of $f(x)$?", computes a list of m numbers, one of which is the correct value. This paper extends previous results on the hardness of set-bit-enumeration for the function classes OptP and #P. Our main theorems state:

> If polynomial time computable n-set-bit-enumerators exist for all OptP functions, then OptP = PF.

and

> If polynomial time computable $n^{\frac{1}{2}-\epsilon}$-set-bit-enumerators exist for all #P functions, then #P = PF.

In both cases n is essentially the length of the function value of the given functions.

1 Introduction

Complexity theory is concerned with the classification of problems based on their *hardness*. While functions computable in polynomial time are considered *feasible*, many other interesting and important functions exist for which no efficient algorithm has yet been discovered, such as computing the permanent of a matrix.

Valiant's #P is a well known class of hard functions. It is the class of functions corresponding to counting the number of accepting computations of polynomial-time nondeterministic machines on some input. Given the difficulty of directly computing such *counting* functions, we examine an important question: Do they have efficiently computable approximations?

Due to limited success in constructing conventional approximation schemes, several alternate notions of approximation have been proposed. We will focus on Cai and Hemachandra's notion of *enumerative counting.*

1.1 Definition. *[CH89] A function g is said to $s(\cdot)$-enumerate a function f if, for all $x \in \Sigma^*$, $g(x)$ is a set of size $s(|x|)$ guaranteed to contain $f(x)$.*

Intuitively an enumerator for a function outputs a short list of values, one of which is guaranteed to be the actual value of the function. We will henceforth call these enumerators *regular* in contrast to a different kind of enumerator defined below.

Amir, Beigel, and Gasarch, and independently Cai and Hemachandra established the difficulty of constructing polynomial time, polynomial sized enumerators for #P by proving the following.

1.2 Theorem. *[ABG90],[CH91] If polynomial time computable $n^{O(1)}$-enumerators exist for all #P functions then #P = PF.*

This is a negative result showing that it is unlikely that polynomial time computable enumerators exist for #P. It is natural to ask if weakening the notion of an enumerator for #P results in feasibly constructable approximations.

In [Ram92] the information obtainable from an enumerator was modified from a set of m candidate values of $f(x)$, to a set of m candidate values for the number of 1's (each 1 is called a *set bit*) in the binary representation of $f(x)$ (denoted by $\#_1(f(x))$). Clearly these *m-set-bit-enumerators* are a much weaker source of information.

More precisely, our definition of a set-bit-enumerator is as follows. We call a function p a bound for f, if p is computable in polynomial time and if the length of $f(x)$ is always less than or equal to $p(x)$.

1.3 Definition. *Let $g : \mathbb{N} \to \mathbb{N}$. A function f is called g-set-bit-enumerable, if for every bound p of f there exists a polynomial-time computable function h such that $h(x)$ is a set of at most $g(p(x))$ numbers, one of which is $\#_1(f(x))$.*

The emphasis in [Ram92] was shifted to weaker function classes as well, especially OptP, first defined by Krentel [Kre88]. It was proved that not all functions in OptP have polynomial time computable set-bit-enumerators giving constant many values, unless OptP already equals PF. Specifically

1.4 Theorem. *[Ram92] If for every function f in* OptP *there is a constant m such that f has polynomial time computable m-set-bit-enumerators then* OptP = PF.

The constant m of the above proof was later elegantly improved by Beigel and Gasarch to $O(\log n)$ (personal communication)[1].

In this paper we further improve this result, showing essentially that there exist OptP functions (with non-constant function value lengths) such that it is not even possible to *exclude one value* for the number of set bits in polynomial time unless P = NP.

1.5 Theorem. *If polynomial time computable n-set-bit-enumerators exist for all* OptP *functions then* OptP = PF *(hence* P = NP*).*

Note that the n in the above theorem is the length bound $p(x)$ of the function value, not the length of x.

Because for every function f, bounded by some p, the number of ones in $f(x)$ is an element of $\{0, \ldots, p(x)\}$, every function is trivially $(n+1)$-set-bit-enumerable. Therefore the above result is somehow optimal.

Furthermore we resolve the primary open problem stated in [Ram92] in a strong manner, showing that, unless #P = PF, set bit enumerators for #P-functions must, in general, give at least about $\sqrt{n}$ values.

1.6 Theorem. *If polynomial time computable $n^{\frac{1}{2}-\epsilon}$-set-bit-enumerators exist for all* #P *functions then* #P = PF.

Our paper is organised as follows.

- In Section 2 we define basic notations,
- In Section 3, we outline the general method of the algorithms. The structure is similar to the "expand-prune" algorithms of [CH89],
- In Section 4 we show theorem 1.5 and discuss how the proof technique used there translates to other function classes,
- In Section 5, we show how to construct a function's value, through queries to a 1-set-bit-enumerator,
- In Section 6 we state and prove our main theorem about the infeasiblity of set-bit-enumerating #P functions,
- In Section 7 we finally discuss our results and state some open problems.

[1]Here, as in the original definition of [Ram92] n is the length of the input. However for constant set-bit-enumerators, and hence for the results of [Ram92], both definitions ecoincide.

2 Definitions and Notations

In this section we present definitions and notation for the functions and classes considered. We use standard notation for languages and function classes as in [BDG88].

We fix a tupling function which maps a tuple $(x_1, \ldots x_k)$ of strings to a natural number $\langle x_1, \ldots, x_k \rangle$, where $k > 0$ is arbitrary, in a way such that $|\langle x_1, \ldots, x_k \rangle|$ depends only on $\sum_{i=1}^{k} |x_i|$, and both the encoding and decoding are computable in polynomial time. Where possible we omit brackets such that, for example, $g(x, \langle y_0, \ldots, y_k \rangle)$ is written as $g(x, y_0, \ldots, y_k)$.

As mentioned before we call an integer-valued function p a bounding function for function f, if p is computable in polynomial time and $|f(x)| \leq p(x)$ for every x. For a fixed bound p we will view $f(x)$ as a $p(x)$-bit number. For given x we will write p instead of $p(x)$ to simplify notation.

A function f belongs to #P if there exists a nondeterministic polynomial-time machine whose count of the number of accepting paths on input x is $f(x)$. A function f belongs to OptP if there exists a nondeterministic polynomial-time transducer whose largest output value on input x equals $f(x)$.

We denote the value of the i-th bit in the binary representation of x, where the low order bit is counted as bit number 0, with $\mathrm{BIT}(x, i)$ (i.e., $\mathrm{BIT}(2^i, i) = 1$). We define $\#_1(x)$ to be the number of ones in the binary representation of x.

3 The method in general

The method we use to obtain our results about OptP and #P is in both cases essentially identical and is derived from similar methods in [CH89]. Therefore it seems to be useful to describe our method first in general.

Let $\mathcal{F}$ denote the function class under examination. We show:

- if set bit enumerators of a given size exist for all functions in $\mathcal{F}$, all functions in $\mathcal{F}$ also possess polynomial time computable regular enumerators,
- if $\mathcal{F}$ has polynomial time computable regular enumerators, then some collapse occurs (for example $\mathcal{F} = \mathrm{PF}$).

In the case OptP and #P only the first step needs to be proved because for both classes the second implication is already established. In the case of OptP functions, this follows by modifying an OptP function f to produce another OptP function g such that $g(x) = \langle f(x), y \rangle$, where y is a witness to f producing $f(x)$. Thus, given a polynomial time regular enumerator for g, it can be verified in polynomial time whether a given list of values is valid and which of the values is maximum. In the case of #P, the proof is more complicated, the reader is refered to [CH91].

Now we sketch the method for the first implication.

Let $f \in \mathcal{F}$, let $q : I\!N \to I\!N$ be the function that gives the number of values of the set-bit-enumerator, let $x \in \Sigma^*$ and let $n \geq |f(x)|$, the length of the binary representation of $f(x)$. The following algorithm computes a set $X = X_n$ of strings with the properties (1) $f(x) \in X$ and (2) $|X| \leq q(n)$.

```
Algorithm ComputeRegularEnumeration
input:  x,n such that |f(x)| ≤ n
output:  X with properties (1) and (2)
X_0 := {λ}
FOR i := 1 TO n DO
   Expand:  X_i := ⋃_{z∈X_{i-1}} {z0, z1}
   Prune :  IF |X_i| > q(n) THEN
      Reduce X_i carefully s.t.  X_i contains
      a prefix of f(x) and |X_i| ≤ q(n)
```

We maintain "X_i contains a prefix of $f(x)$" as a loop invariant.

It is easy to verify, that ComputeRegularEnumeration has the following properties:

- Before entering the loop first, X contains a prefix of length 0 of $f(x)$.
- If X contains the prefix of length $i-1$ of $f(x)$ before entering the loop for the i^{th} time, then X contains the prefix of length i after expansion, in line `Expand:` $X_i := \bigcup_{x \in X_{i-1}} \{x0, x1\}$.
- If the reduction in the pruning step does not remove the prefix of length i of $f(x)$ then X contains it after the i^{th} pass of the loop.
- If the time needed for the pruning step is polynomial in $|x|+n$ then the running time of ComputeRegularEnumeration is polynomial in $|x|+n$ too.

The idea of the reduction step is as follows[2] : We define a function $h \in \mathcal{F}$ which gets x and $q(n)+1$ different strings $y_0, \ldots, y_{q(n)}$ of equal length such that there exist pairwise different natural numbers $k_0, \ldots, k_{q(n)}$ with the property

$$y_i \text{ is prefix of } f(x) \implies \#_1(h(x, y_0, \ldots, y_{q(n)})) = k_i.$$

Thus the number of ones in the function value of h determines uniquely the correct prefix of $f(x)$.

[2]In the case of #P the situation will be a little more complicated as there the function h needs more information as input. It is the size of these additional inputs that prohibits us from matching the optimal weakening of a set-bit-enumerator achieved for OptP.

By assumption there exists a polynomial time computable set bit enumerator for h, which will give us a list of at most $q(n)$ possible values for the number of ones in the function value of h. This allows us to remove one or more strings from X which cannot be prefixes of $f(x)$.

As expansion can blow up the size of any X_i to at most $2q(n)$, at most $q(n)$ iterations of the above procedure are needed (each supplying a different set of prefixes to h), in the prune step.

The main challenge will always be to find an appropriate function h. The difficulty arises from the requirement that h has to be a member of $\mathcal{F}$.

4 Set bit enumeration for OptP

In this section we will establish the following result.

4.1 Theorem. *If every function in* OptP *is n-set-bit-enumerable, then* OptP $= PF$.

There is one remark in order. The theorem as stated is trivial, because one could e.g. look at the characteristic function of SAT, where a n-set-bit-enumerator becomes a 1-set-bit-enumerator. But the theorem remains true (and becomes less trivial) if we restrict its range to all "non-constant" functions in OptP. To make things precise, we may take n-set-bit-enumerable as an abbreviation for $\max(\log|x|, n)$-set-bit-enumerable.

Proof. Let MaxSat be the function which maps every Boolean formula in n variables to its lexicographic maximal satisfying assignment, if any exists, and to 0^n otherwise. It is well-known and easy to see that MaxSat $\in$ OptP. We show that, under the hypothesis of the theorem, MaxSat has polynomial-time regular enumerators, which immediately implies OptP = PF.

Let $p(F)$ be the number of variables in F. p is a bounding function for MaxSat. As outlined in section 3 we have to find a function h in OptP which "picks" the correct value from a list of potential prefixes of MaxSat(F).

We call a sequence $y_0, \ldots, y_n$ OK, if $y_i < y_{i+1}$ and $|y_i| = |y_{i+1}|$ for every $i < n$.
Now we define

$$h(F, y_0, \ldots, y_n) := \begin{cases} 1^i 0^{n-i} & \text{if } y_0, \ldots, y_n \text{ is OK and } y_i \text{ is the lexico-} \\ & \text{graphic maximal string in } y_0, \ldots, y_n \text{ that} \\ & \text{can be extended to a satisfying assignment} \\ & \text{of } F, \\ 0^n & \text{otherwise.} \end{cases}$$

We note that p', defined by $p'(F, y_0, \ldots, y_n) := p(F)$, bounds h.

Let M be a nondeterministic transducer that computes MaxSat in an OptP fashion. We describe a nondeterministic transducer whose lexicographic maximal output equals $h(F, y_0, \ldots, y_n)$:

- Check if $y_0, \ldots, y_n$ is OK. Otherwise output 0^n.
- Guess an output path of M on input F.
- If M outputs a value y on this path, check if any of the y_i is a prefix of y. If so, output $1^i 0^{n-i}$. Otherwise output 0^n.

This shows that h is in OptP. On the other hand, if $y_0, \ldots, y_n$ contains a prefix y_i of MaxSat(F), then $h(F, y_0, \ldots, y_n) = 1^i 0^{n-i}$, thus $\#_1(h(F, y_0, \ldots, y_n)) = i$.

Therefore we can conclude that the algorithm ComputeRegularEnumeration will compute a regular enumeration of MaxSat(F). □

The above proof doesn't make very much use of specific properties of the class OptP. In fact for many function classes like NPSV, PF^{NP}, the membership of f implies the membership of h in the class. It follows that for those classes the existence of set bit enumerators implies the existence of regular enumerators. But for those classes no general collapse result like [CH91] is known.

On the other hand it is well-known that the result of [CH89] can be easily extended to those classes. Therefore, if $\mathcal{F}$ is one of the mentioned classes, if for any $\epsilon > 0$ every function in $\mathcal{F}$ has a $\min(|x|^{1-\epsilon}, n)$-set-bit-enumerator then $\mathcal{F} = \mathrm{PF}$.

5 1-set-bit-enumerators for $\#P$

It was already shown in [Ram92] that if every #P-function has a 1-set-bit-enumerator, then #P = PF. In this section we show this result again with a simplified proof.

This proof makes use of the following observation: If the i^{th} bit of y (denoted by $\mathrm{BIT}(y, i)$) is zero, then $y + 2^i$ has one more set bit than y. On the other hand, if the i^{th} bit of y is a one, then $y + 2^i$ has at most as many ones as y. More precisely

5.1 Proposition. *Let k be the exact number of ones contiguosly preceeding the i-th bit of y (i.e. $BIT(y, i+j) = 1$ for $j \in \{1, \ldots, k\}$ and $BIT(y, i+k+1) = 0$). Then*

$$\begin{array}{lll} (1) & BIT(y,i) = 0 \iff \#_1(y + 2^i) = \#_1(y) + 1, \text{ and} \\ (2) & BIT(y,i) = 1 \iff \#_1(y + 2^i) = \#_1(y) - k \end{array}$$

5.2 Theorem. *[Ram92] If every #P-function has a 1-set-bit-enumerator, then* #P = PF.

Proof. Let $f \in \#\mathrm{P}$ with bound p. Then the function g defined by $g(x, y) := f(x) + y$ is also a #P-function. The following algorithm computes $f(x)$ in polynomial time given an algorithm which computes $\#_1(g(x))$ in polynomial time.

```
y := 0
FOR i := 0 TO p(x)
    IF #_1(g(x, 2^i)) ≠ #_1(g(x, 0)) + 1 THEN
        y := y + 2^i
```

The correctness of this algorithm follows from Proposition 5.1. □

The following theorem shows that one doesn't even need to know $\#_1(f(x))$ exactly. Knowing the parity of $\#_1(f(x))$ suffices. To get this we only have to set bit $i+1$ to zero when we compute bit i.

We define $\oplus_1(f)$, the set-bit-parity-function of f, by $\oplus_1(f)(x) = 1$, if $\#_1(f(x))$ is odd and $\oplus_1(f)(x) = 0$ otherwise.

5.3 Theorem. *If for every $f \in \#\mathrm{P}$ the set-bit-parity-function of f can be computed in polynomial time, then $\#\mathrm{P} = \mathrm{PF}$.*

Proof. Let $f \in \#\mathrm{P}$ with bound p. Again we define the function $g \in \#\mathrm{P}$ by $g(x, y) := f(x) + y$. By hypothesis $\oplus_1(g) \in \mathrm{PF}$. Therefore the following algorithm computes $y = f(x)$ in polynomial time.

```
y := 0
FOR i := p(x) − 1 DOWNTO 0 DO
    IF BIT(y, i + 1) = 0
        THEN z := 0
        ELSE z := 2^(i+1)
    IF ⊕1(g)(x, z) = ⊕1(g)(x, z + 2^i) THEN
        y := y + 2^i
```

To verify the correctness of the algorithm, we note that the choice of z assures that $f(x)+z$ has 0 at bit position $i+1$, and that the i^{th} bit of $f(x)+z$ matches the i^{th} bit of $f(x)$. □

It is not hard to see that we can replace $\oplus_1$ in the above lemma with any predicate on the natural numbers which has the property that there exists a polynomial p such that for every n, between n and $p(n)$ at least one but not all numbers fulfil the predicate (e.g. "is a prime").

6 $n^{\frac{1}{2}-\epsilon}$-set-bit-enumerators for $\#P$

In this section we are going to prove the following result.

6.1 Theorem. *Let $\epsilon > 0$. If every $\#\mathrm{P}$-function is $n^{\frac{1}{2}-\epsilon}$-set-bit-enumerable then $\#\mathrm{P} = \mathrm{PF}$.*

In principle we use the algorithm described in section 3 to show that under the given hypothesis $\#\mathrm{P}$ has regular enumerators.

In the case of OptP we made use of the fact that, given a function $f \in \mathrm{OptP}$, we can construct an OptP function h which, on input $x, y_0, \ldots, y_k$, gives the index i of the prefix of $f(x)$. Unfortunately $\#\mathrm{P}$ is not known to have such strong closure properties.

We therefore extend our notion of a prefix. The *i-prefix* of $f(x)$ is a pair (y, z), where $z = \#_1(f(x))$ and y is the number which has zeroes at positions $0, \dots, i-1$ and equals $f(x)$ at the other positions. More formally: $y = (f(x)\mathrm{div}2^i) \times 2^i$.

Given a bound p for f we call the vector $\langle \#_1(f(x)), \#_1(f(x)+2^{p-1}), \dots, \#_1(f(x)+2^i)\rangle$ the *i-set-bit-vector of* $f(x)$. Note that the i-prefix and the i-set-bit-vector of $f(x)$ can easily be computed from each other.

We will construct a #P-function h, such that, on input x and a tuple of possible i-prefixes $y_0, \dots, y_k$, including the correct one, the number of ones in $h(x, y_0, \dots, y_k)$ allows us to compute the correct i-set-bit-vector of $f(x)$. We need the following lemma which tells us how to encode the number of ones of several function values of a #P function into one single function value.

6.2 Lemma. *Let $g \in$ #P. Then there is a $h \in$ #P such that for every $k \in \mathbb{N}$ and natural numbers $b_0, \dots, b_k$ it holds that*

$$\#_1(h(x, 1^{b_0}, \dots, 1^{b_k})) = \sum_{i=0}^{k} b_i \times \#_1(g(x, 1^i)).$$

Proof.
Let p be a bound for g, set $c_i := \sum_{j=0}^{i-1} b_j$ and define

$$h(x, 1^{b_0}, \dots, 1^{b_k}) := \sum_{i=0}^{k}(\sum_{j=0}^{b_i-1} g(x, 1^i)2^{jp})2^{pc_i}.$$

It is easily verified that $h \in$ #P. The binary representation of $h(x, 1^{b_0}, \dots, 1^{b_k})$ consists of b_0 copies of $g(x, 1^0)$, b_1 copies of $f(x, 1)$ and so on. Note especially that the length of h is bounded by $p\sum_{j=0}^{k} b_i$. □

Taking $k := p - i$, $b_j := p^{j-1}$, $g(x, 0) := f(x)$ and $g(x, i) := f(x) + 2^{p-i}$ for $i > 0$ we get a #P-function which encodes the i-set-bit-vector of $f(x)$ in an easy-to-decode-manner. But the size of the inputs 1^{b_j} is not polynomially bounded. On the other hand the unary encoding of the b_j is crucial to make h a #P-function.

Fortunately we do not really need as much information as the i-set-bit-vector gives. It is enough to find *small* coefficients b_i that separate the given y_j from each other. The following lemma tells us how to do this.

6.3 Lemma. *Let $w^{(1)} = (w_1^{(1)}, \ldots, w_m^{(1)}), w^{(2)}, \ldots, w^{(k)}$ be pairwise different vectors of natural numbers. Then there is a vector $b = (b_1, \ldots, b_m)$ with $b_i \leq k^2 + 1$, for every i, such that the inner products $b \times w^{(i)}$ and $b \times w^{(j)}$ are different for $i \neq j$.*

Proof. We look at the subvectors $w^{(i)}(s) = (w_1^{(i)}, \ldots, w_s^{(i)})$ and show by induction on s that there are $b_1, \ldots, b_s$ such that for every i, j and every s it holds

- $w^{(i)}(s) = w^{(j)}(s)$ or
- $b(s) \times w^{(i)}(s) \neq b(s) \times w^{(j)}(s)$,

where $b(s) = (b_1, \ldots, b_s)$.

$\underline{s = 1:}$ Choose $b_1 := 1$.

$\underline{s > 1:}$ Let $i, j \leq k$. We distinguish three cases:

1. $\underline{w^{(i)}(s) = w^{(j)}(s)}$
In this case there is nothing to show.

2. $\underline{w^{(i)}(s) \neq w^{(j)}(s) \text{ and } w^{(i)}(s-1) = w^{(j)}(s-1)}$
In this case every choice of $b_s \neq 0$ will lead us to $b(s) \times w^{(i)}(s) \neq b(s) \times w^{(j)}(s)$.

3. $\underline{w^{(i)}(s) \neq w^{(j)}(s) \text{ and } w^{(i)}(s-1) \neq w^{(j)}(s-1)}$By induction hypothesis we have

$$b(s-1) \times w^{(i)}(s-1) \neq b(s-1) \times w^{(j)}(s-1)$$

and there is at most one c_{ij} such that

$$b(s-1) \times w^{(i)}(s-1) + c_{ij} w_s^{(i)} = b(s-1) \times w^{(j)}(s-1) + c_{ij} w_s^{(j)}.$$

So we can choose $b_s \neq 0$ minimal such that $b_s \neq c_{ij}$ for all $i, j \leq k$. This is obviously possible with $b_s \leq k^2 + 1$. $\square$

Furthermore the b_i are computable in polynomial time in $|x|$ given that k is polynomially bounded in $|x|$ (and the lengths of the entries $w_i^{(j)}$ are poly-bounded in $|x|$).

Proof. of Theorem 6.1 Let $f \in \#\mathrm{P}$ with bound p, let $x \in \Sigma^*$ and let $\epsilon > 0$. We choose $t > \frac{2-3\epsilon}{2\epsilon}$ and $q = p(x)^t$, the number of prefixes we will maintain[3].
Set

$$g(x, 1^i) := \begin{cases} f(x) & \text{if } i = 0 \\ f(x) + 2^{p-i} & \text{if } i > 0 \end{cases}$$

Let h be the function defined in Lemma 6.2. Assume there is a $n^{\frac{1}{2}-\epsilon}$-set-bit enumerator E for h. The algorithm to compute a regular enumeration for f works as follows.

[3]Unlike stated in section 3 here q is not the size of the set-bit-enumerator.

```
Algorithm ComputeRegularEnumeration
input:  x
X_p:=set of p-prefixes given by E(x,1) (= list of values for #_1(f(x)))
FOR i := p - 1 TO 0 DO
   Expand:  X_i:=:=⋃_{z∈X_{i+1}} {z0, z1}
   Prune :  IF |X_i| > q THEN
            Compute coefficients b_i according to lemma 6.3 such that
               b × y ≠ b × y' for any i-set-bit-vectors y, y'
               representing different i-prefixes from X_i.
            Compute E' := E(x, 1^{b_0}, ..., 1^{b_{p-i}})
            X_i:=set of i-vectors from X_i which correspond to some
               value from E'
```

Finally we have to show that E indeed provides us with a list of at most q values.
As the expanding step doubles the number of elements of X, we have at most $2q$ i-prefixes as inputs for E. Lemma 6.3 assures that the b_i are bounded by $(2q+1)^2 \leq 5q^2$. The length of $h(x, 1^{b_0}, \ldots, 1^{b_{p-i}})$ can thus be bounded by $5q^2p^2$ (p coefficients of length at most $5q^2$ times the length of $f(x)$). Therefore, if $p > 5$, the number of values of E' is at most $(5q^2p^2)^{\frac{1}{2}-\epsilon} \leq (p^{2t+3})^{\frac{1}{2}-\epsilon} \leq p^{t+2-2t\epsilon-3\epsilon} \leq p^t = q$ by the choice of t. □

7 Open problems

We would firstly be interested in knowing if the bounds of Theorem 6.1 are optimal or can be further improved. This would be possible for instance if lemma 6.3 could be strengthened. A lemma with lower bounds for the b_i would even help if the inner products could not be guaranteed to be pairwise different but to have enough different values (e.g. $k/2$).

We would like to know if any interesting function classes have approximations like polynomial-time set-bit-enumerators. Counting class variants of #P or outside EXP could be possible candidates. On the other hand it would be interesting to classify the exact complexity of set-bit-enumeration for certain function classes.

As enumerators and set-bit-enumerators can be thought of as computing restrictions of the range of a function, relationships to similar notions for languages are also another direction to look in. For example Amir, Beigel and Gasarch's in their research on bounded query classes, Amir, Beigel and Gasarch define $F_k^A(x_1, \ldots, x_k) = (A(x_1), \ldots, A(x_k))$, where $A(x)$ is the characteristic function of A: 1 if x is in A, 0 otherwise. They then draw conclusions from the hypothesis F_k^A is computable with less than k queries to A. Their results also implied the main theorem of [CH91].
Other generalizations could involve a further weakening of the information available from a set-bit-enumerator.

Acknowledgements

We would like to thank Clemens Lautemann for helping us improve the readibility and correctness of this paper and Johannes Köbler for helpful comments.

References

[ABG90] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proc. Fifth Annual IEEE Structure in Complexity Theory Conference*, pages 232–243, 1990.

[BDG88] J. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, 1988.

[CH89] J. Cai and L. Hemachandra. Enumerative counting is hard. *Information and Computation*, 82:34–44, 1989.

[CH91] J. Cai and L. Hemachandra. A note on enumerative counting. *Information Processing Letters*, 38:215–219, 1991.

[Kre88] M. Krentel. The complexity of optimization problems. *J. Computer Systems Sci.*, 36:490–509, 1988.

[Ram92] J. Ramachandran. Set bit enumeration is hard. In *Proceedings, Seventh Annual Structure in Complexity Theory*, pages 58–70, 1992. IEEE Computer Society Press.