6-1-2010

Block Sensitivity versus Sensitivity

Karn Seth
Dartmouth College

Block Sensitivity versus Sensitivity

Karn Seth
Advisor : Amit Chakrabarti

Senior Honours Thesis
Submitted to the faculty in partial fulfillment of the
requirements for a Major in Computer Science

Dartmouth Computer Science Technical Report TR2010-673

June 1, 2010

Abstract

Sensitivity and block sensitivity are useful and well-studied measures of computational complexity, but in spite of their similarities, the largest possible gap between them is still unknown. Rubinstein showed that this gap must be at least quadratic, and Kenyon and Kutin showed that it is at worst exponential, but many strongly suspect that the gap is indeed quadratic, or at worst polynomial.

Our work shows that for a large class of functions, which includes Rubinstein's function, the quadratic gap between sensitivity and block sensitivity is the best we can possibly do.

Contents

1 Introduction **2**
1.1 Boolean Functions 2
1.2 Decision Trees 2

2 Sensitivity and Block Sensitivity **5**

3 Previous Results **7**
3.1 Rubinstein's Construction 7
3.2 Kenyon and Kutin's results 8
3.3 Other results 8

4 An Alternative Proof of Kenyon and Kutin's result **10**

5 Rubinstein's Function is the Best in its Class **14**

6 Concluding remarks **17**
6.1 Discussion of results 17
6.2 Open Problems 17

Chapter 1

Introduction

The field of computational complexity is concerned with understanding how difficult functions are to compute. Difficulty is generally defined in two ways: how much computational power is needed to compute a function, and how much of the input needs to be looked at in order to determine the functions value. We will be primarily concerned with the latter sort of difficulty. There are a variety of models used by researchers in this field, and in this chapter we will introduce some of them.

1.1 Boolean Functions

Boolean functions are functions of the form $f : \{0,1\}^n \to \{0,1\}$. They serve as a simple, yet powerful class of functions that allow us to investigate the broader field of functional complexity in a simplified environment. We will be focusing only on Boolean functions throughout our work, and for convenience and later use, we will define some useful notation relating to them:

For an input $x \in \{0,1\}^n$, we define x_i to be the ith bit of the input.

We define the *weight* of x, written as $|x|$, to be the number of i such that x_i is 1.

For any subset $S \subseteq [n]$, we write $\mathbb{1}_S$ to denote the characteristic vector of S, defined as the vector (string) x in $\{0,1\}^n$ for which $x_i = 1 \iff i \in S$. We will sometimes abuse notation slightly, writing $f(S)$ instead of $f(\mathbb{1}_S)$.

We define x^S to be $x \oplus \mathbb{1}_S$, which is simply the input x with each bit in the subset S flipped. For sets consisting of a single bit, we will shorten $x^{\{i\}}$ to x^i.

Some simple examples of Boolean functions are as follows:

AND: $f(x) = 1 \iff x_i = 1$ for all i

OR: $f(x) = 1 \iff x_i = 1$ for at least one i

DICTATOR: $f(x) = 1 \iff x_i = 1$ for a fixed particular i, the dictator.

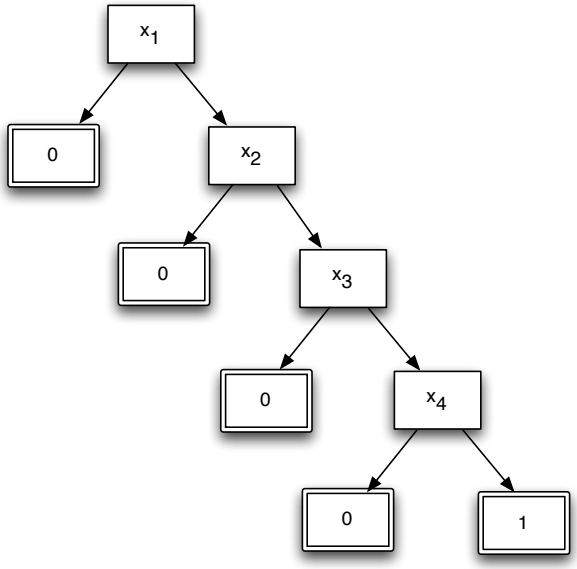
THRESHOLD: $f(x) = 1 \iff |x| \geq t$, where t is the threshold.

With the exception of DICTATOR, all the above functions are *symmetric*, meaning that the value of $f(x)$ depends only on $|x|$.

1.2 Decision Trees

Decision trees are an important and useful model of boolean functional complexity. A decision tree is an algorithm to compute a Boolean function $f : \{0,1\}^n \to \{0,1\}$ by making single-bit queries to the input. Each

Figure 1.1: A decision tree for the AND function on 4 bits



query asks for the value of bit x_i, with the answer to the query being the value of this bit. The algorithm is adaptive, that is, the bit requested in the kth query may depend on the answers of the previous $k-1$ queries.

The algorithm can be pictured as descending a binary tree in which the nodes are marked with indices in $[n]$ and the leaves are marked with either 1 or 0. We descend the tree as follows: when we reach a node marked i, we query bit x_i, taking the left branch if the query returns 0 and the right branch if it returns 1. When we reach a leaf, we take the value of the leaf as the output of the function.

For a Boolean function f, we define its deterministic decision tree complexity $D(f)$ to be the height of the smallest decision tree required to compute f exactly. In other words, $D(f)$ is the minimum number of queries that an optimal deterministic algorithm for f needs to make, in order to be able to correctly calculate the value of the function on all inputs in $\{0,1\}^n$. A low decision tree complexity means that we need need to examine only a small portion of the input bits in order to correctly determine the functions value on that input.

The decision tree complexity of some simple functions is easy to determine. Consider DICTATOR, which has value 1 if and only if the ith bit of the input is 1. This function can be computed by a decision tree that has a single node. That node queries bit i and has a 1-leaf as its right child, a 0-leaf as its left child. Since this tree has depth 1, $D(\text{DICTATOR}) = 1$.

Now consider the AND function on n bits, where $f(x) = 1$ if and only if all n bits of the input are 1. We will see that any decision tree for this function must have depth n. Let T be any decision tree for AND. Starting at the root of T and taking the right-hand-side branch at each node (as we would if the input consisted of all 1s), let $S \subseteq [n]$ be the set of indices encountered at the query nodes along the way. Suppose $i \in [n]$ is not in S, and let $x = \mathbb{1}_{[n]}$, $y = \mathbb{1}_{[n]-\{i\}}$. Then, since x and y agree at every bit except the ith, T will drive y to the same leaf as x, namely the rightmost one, thus returning the same output for both of them. However, $f(x) = 1$ and $f(y) = 0$, thus one of these answers must be incorrect. Hence our assumption that there is an $i \notin S$ must be false, and $S = [n]$, implying that the right-most path must have at least n nodes. Hence $D(\text{AND}) = n$.

In order to better understand decision trees and decision tree complexity, a variety of related measures have been defined and studied. These include *certificate complexity*, *sensitivity*, *block sensitivity*, the *degree of the representing polynomial*, and the *degree of an approximating polynomial*. In addition, there are variants of decision trees, namely *randomized* and *quantum* decision trees. We are interested in two measures in particular : *sensitivity* and *block sensitivity*. A full introduction to these measures can be found in [2].

Chapter 2

Sensitivity and Block Sensitivity

Definition 2.1. We define the sensitivity $s_x(f)$ of f on input x to be the number of $i \in [n]$ such that $f(x) \neq f(x^i)$. The overall sensitivity s(f) of f is the maximum of $s_x(f)$ over all x.

In other words, $s_x(f)$ is the number of bits of x which, if flipped, would change the value of f(x). The sensitivity of a function can vary over different inputs, sometimes quite dramatically.Taking *OR* for example, we note that the input $x = \overrightarrow{0}$ has sensitivity n, because $f(x) = 0$, while $f(x^i) = 1$ for all i. However, the input x^i has a sensitivity of just 1, because flipping any bit x_j such that $j \neq i$ doesn't alter the function's value.

Definition 2.2. The block sensitivity $bs_x(f)$ of f on x is the maximum number t such that there is collection $\mathcal{B} = \{B_1, \ldots, B_t\}$ of disjoint subsets of $[n]$ with $f(x) \neq f(x^{B_i})$ for all $i \in [t]$. $\mathcal{B}$ is said to be a *sensitive block decomposition* of f at x. The overall block sensitivity bs(f) of f is the maximum of $bs_x(f)$ over all x.

Note that sensitivity is just a special case of block sensitivity, where the block sizes are restricted to 1. Thus sensitivity is a lower bound for block sensitivity.

There can be a significant gap between sensitivity and block sensitivity on particular inputs. Take f to be the THRESHOLD function, for example, with the threshold set to 2. Then for the input $x = \overrightarrow{0}$, we can create the sensitive block decomposition $\mathcal{B} = \{B_1, \ldots, B_{\lfloor n/2 \rfloor}\}$, where each $B_i = \{2i - 1, 2i\}$. Thus $bs_x(f) = \lfloor n/2 \rfloor$. However, $s_x(f) = 0$ because there is no single-bit flip that can change the weight of the input from 0 to 2. However, if we consider instead the input $y = x^i$ for any i, then the sensitivity becomes $n - 1$, because $f(y) \neq f(y^j)$ for all $j \neq i$. The block sensitivity also jumps to $n - 1$, since sensitivity is a lower bound for block sensitivity. So when we consider s(f) and bs(f), we have s(f) = bs(f) = $n - 1$.

We do not know what is the largest possible gap between s(f) and bs(f), but Rubinstein gives a function with $bs(f) = \Omega(s(f)^2)$.

We will sometimes differentiate between $s^0(f)$ and $s^1(f)$, which are the sensitivities of the function restricted to inputs x for which $f(x) = 0$ and $f(x) = 1$ respectively. Similarly, we can also differentiate between $bs^0(f)$ and $bs^1(f)$.

An important fact motivating these measures is that bs(f) provides a lower bound on $D(f)$. This can be shown as follows: consider the input x with $bs_x(f) = \text{bs}(f)$, and let $\mathcal{B} = \{B_1, \ldots, B_t\}$ be its sensitive block decomposition. Let T be a decision tree for f. Then we will show that the path that x takes as it descends T must query at least 1 bit from each block B_i. Suppose that for some i, no bit in B_i is queried by T along the path taken by x. Then T will drive both x and x^{B_i} to the same leaf. But because x is block sensitive to B_i, we know that $f(x) \neq f(x^{B_i})$. Thus T returns an incorrect answer for either x or x^{B_i}. Hence T must query at least one bit for every block in $\mathcal{B}$, implying that it has depth at least $|\mathcal{B}| = \text{bs}(f)$. Combining this with our

earlier observation, we have that

$$\mathrm{s}(f) \leq \mathrm{bs}(f) \leq D(f)$$

Nisan [5] showed that $D(f)$ can also be upper bounded by $\mathrm{s}(f) \cdot \mathrm{bs}(f)^2$. This results in the following chain of inequalities:

$$\mathrm{s}(f) \leq \mathrm{bs}(f) \leq D(f) \leq s(f) \cdot bs(f)^2 \leq \mathrm{bs}(f)^3$$

Chapter 3

Previous Results

Over the last few decades, there has been plenty of work relating to sensitivity and block sensitivity. One of the earliest results is Rubinstein's construction of a function f demonstrating a quadratic gap between s(f) and bs(f)[6]. This is the largest known gap between these two measures.

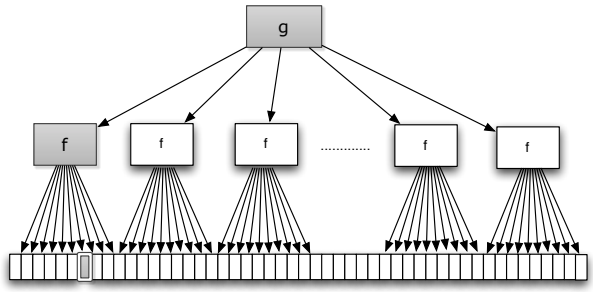
3.1 Rubinstein's Construction

First we define the function f on $2\sqrt{n}$ bits, with $f(x) = 1 \iff x_{2i} = 1$ and x_{2i-1} are both 1 for some $i \in [\sqrt{n}]$, and all other bits are 0. Then we define another function g which takes as its input the values of $\sqrt{n}$ copies of f, and applies the *OR* function. The result is a tiered function F on $2n$ bits that has bs(f) = n on the input $\vec{0}$, with blocks $B_i = \{x_{2i}, x_{2i-1}\}$ for all $i \in [n]$.

To compute s(f), we consider first $s^0(F)$ and then $s^1(F)$. For $s^0(F)$, we look at inputs where the function value is 0, but can be made 1 by turning on a single bit. This means that none of the outputs of the fs can be 1, but all of them could potentially be made 1 with a single bit change. This means they must have either a single bit on (in which case we can turn on its neighbour), or have 2 neighbouring bits on and some third bit on (in which case we can turn off the third bit). In either case, we can have at most a single sensitive bit for each copy of f, and so $s^0(F) = \sqrt{n}$.

For $s^1(F)$, we look at inputs where the value of F is 1, meaning that at least 1 copy of f must have value 1. But if 2 or more fs have value 1, then they can't all be made 0 in a single bit flip. This means that if an input to F with value 1 can be made 0 in a single bit flip, then it must have *exactly* one f with output 1. Then to change F to 0, we can simply flip any of the $2\sqrt{n}$ inputs to the active f, and this will ruin the pattern

Figure 3.1: Rubenstein's function in tree form.



going into that f, making its output a 0. Thus $s^1(f) = 2\sqrt{n}$.

For this function, $\text{bs}(F) \geq \text{s}(F)^2$, meaning that the gap between sensitivity and block sensitivity is quadratic.

Note that this function can be visualized as a 2-level tree, where the top level is the function g, the nodes in the middle level are the copies of f, and the leaves are the 2-bit blocks. It is a natural question whether tweaking some part of this construction can improve the quadratic gap. There are three obvious changes that can be made: changing the functions f and g, increasing the number of levels in the tree, or changing the size of the blocks. We will show later on that changing any or all of these factors does not improve this gap.

3.2 Kenyon and Kutin's results

Kenyon and Kutin[4] showed that for a function with high block-sensitivity, where the blocks are each of size c or smaller for some constant c, $\text{s}(f) = \Omega(\text{bs}(f)^{1/c})$. A corollary of this fact is that when $\text{bs}(f) = \Omega(n)$, $\text{s}(f)$ must be polynomially related to $\text{bs}(f)$. We will present an alternate version of their proof in a later section.

The fact that the quality of the bound deteriorates as we increase the size of the blocks suggests that we may be able to create a function that uses block sizes larger than 2 to create a superquadratic gap. In fact, our alternative proof of Kenyon and Kutin's result suggests a way to create such a function. However, we will prove later that all such efforts will have holes, and will not help us increase the quadratic gap.

3.3 Other results

The results discussed in this section, while not directly relevant to our results, are still interesting because they show the approach others have taken to studying sensitivity and block sensitivity.

The first of these results is a simple one showing that all non-constant symmetric functions have $\text{s}(f) \geq n/2$. Recall that a symmetric function is one that depends only on the value of $|x|$. Since the function is nonconstant, that means for some two different inputs x and y, $f(x) = 0$ and $f(y) = 1$. To show the lower bound, we argue as follows:

Suppose, without loss of generality, that $|x| < |y|$. Then we can increase the weight of x by flipping single 0-bits to 1, until eventually we encounter an input x' with weight t and $f(x') = 0$, such flipping a single bit in x' from 0 to 1 to get y' with weight $t + 1$ gives $f(y') = 1$. If $|x'| < \lfloor n/2 \rfloor$, then there are at least $\lceil n/2 \rceil$ places to flip a zero to a one, changing the function value from 0 to 1. Alternatively, if $|x'| \geq \lfloor n/2 \rfloor$, then y' has at least $\lceil n/2 \rceil$ places to flip a one to a zero, changing the function value from 1 to 0. One of the two cases holds, so $\text{s}(f) \geq \lceil n/2 \rceil$.

The above notion of symmetry is quite strong, and there are a variety of weaker notions of symmetry that have also been studied for their sensitivity. It appears in general that functions with a high degree of symmetry must have a high sensitivity. Some of the notions of symmetry studied are as follows:

Definition 3.1. A boolean function $f : \binom{V}{2} \to \{0, 1\}$ is a *graph property* if each bit is in the input is associated with an edge of a graph on V vertices, with $x_i = 1 \iff$ the edge associated with x_i is present in the graph. Additionally, $f(x) = 1$ if and only if the underlying graph has a particular property that is invariant under graph isomorphism.

Definition 3.2. A boolean function $f : \{0, 1\}^n \to \{0, 1\}$ is *cyclically invariant* if for all $x = x_1x_2x_3 \ldots x_n$, $f(x_1, x_2, \ldots, x_n) = f(x_2, x_3, \ldots, x_n, x_1)$

Definition 3.3. A *group of permutations* on $[n]$ is a set of permutations on n bits that is closed under composition and taking inverses. A group of permutations G is said to be *transitive* if for every pair of indices $i, j \in [n]$, there exists a permutation $\pi \in G$ such that $\pi(i) = j$.

Definition 3.4. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be *weakly symmetric* if it is invariant under the action of a transitive group of permutations G, that is, for every input x, $f(\pi(x)) = f(x)$ for every permutation $\pi \in G$.

Turán[8] showed that all graph properties have $\mathrm{s}(f) = \Omega(\sqrt{n})$. He conjectured that a similar bound holds in the case of all weakly symmetric functions. Kenyon and Kutin[4] made a similar conjecture, that $s^0(f) \cdot s^1(f) = \Omega(n)$ for all weakly symmetric functions.

Chakraborty[1] gave counterexamples to both these conjectures. He gave a cyclically invariant function with $\mathrm{s}(f) = \Theta(n^{1/3})$. (Note that all cyclically invariant functions are also weakly symmetric.) He also gave an example of a weakly symmetric function with $s^0(f) \cdot s^1(f) = \Theta(\sqrt{n})$. In order to do so, he defined two new classes of functions, minterm-cyclic functions (which are cyclically intransitive) and minterm-transitive functions (which are weakly symmetric).

He also showed that for these classes of functions, $\mathrm{s}(f) = \Omega(n^{1/3})$, and that the gap between $\mathrm{bs}(f)$ and $\mathrm{s}(f)$ is at most quadratic. Sun[7] showed that all weakly symmetric functions have $\mathrm{bs}(f) = \Omega(n^{1/3})$, while in the special case of minterm-transitive functions, Drucker[3] gave a lower bound of $\Omega(n^{3/7})$ for block sensitivity, also making an improvement to a function of Sun's which made this lower bound almost tight.

Chapter 4

An Alternative Proof of Kenyon and Kutin's result

First, we reprove Kenyon and Kutin's result showing that if $\mathrm{bs}(f) = Kn$, then $\mathrm{s}(f) = \Omega(n^{K/2})$ [4]. Our strategy will be to take the highly block-sensitive input, and turn on a single bit in a large number of different blocks, in such a way that the function remains block-sensitive to the remaining bits in those blocks. The resulting input will now have a high block sensitivity, but with blocks of size one less than the original input. We repeat this process, continually reducing the size of the blocks, until we an input with a large number of blocks of size 1, meaning that this input will have high single-bit sensitivity.

The lemma that follows shows that we can perform the first step, that is, turn on a large number of bits in different blocks, such that the resulting input is still block-sensitive to the remaining bits in those blocks.

Lemma 4.1. *Suppose that the Boolean function $f : \{0,1\}^n \to \{0,1\}$ has sensitivity s(f), and has a sensitive block decomposition $\{B_1, \ldots, B_t\}$, at a certain input a, consisting of t blocks, each of size c or less, $c \geq 2$. Then there exists an input a' such that f has a sensitive block decomposition at a' consisting of $\frac{t}{8s(f)}$ blocks, each of size $c-1$ or less.*

Proof. We will begin by replacing f with the function $x \mapsto f(x \oplus a) \oplus f(a)$.This has the effect of shifting the block-sensitive input from a to $\vec{0}$, and also ensures that $f(\vec{0}) = 0$. The remaining structure of the function is unchanged. Further, for convenience, we renumber the inputs to f, moving all the blocks of size c to the front, so that we may assume that $B_1, \ldots B_r$ consist of blocks of size c, while $B_{r+1}, \ldots, B_t$ consist of blocks of size $c-1$ or less, and further, that the blocks of size c each have $B_i = \{i, i+r, \ldots, i+r(c-1)\}$ for each $i \in [r]$. This means putting all the first bits of each block together at the front, followed by all the second bits, and so forth. (This is simply for notational convenience).

Then, using set notation, we have

$$f(\varnothing) \;=\; 0 \;\text{ and }\; f(B_i) \;=\; 1 \;\forall\, i \in [t]\,. \tag{4.1}$$

For each $i \in [r]$, we define $B_i{}'$ to be all the bits in B_i except the very first.

$$B_i{}' \;:=\; \{i+r, \ldots, i+r(c-1)\}$$

Further, for each $A \subseteq [r]$, we define J_A as follows:

$$J_A \;:=\; \{j \in [r] \setminus A : f(A \cup B_j) = 0\}$$

We claim that there exists a "large" subset $A \subseteq [r]$ consisting of "first bits" of the blocks of size c, such that satisfies the following four properties.

$$f(A) = 0; \tag{4.2}$$
$$f(A \cup B_i) = 1, \text{ whenever } i \in \{r+1, \ldots, t\}; \tag{4.3}$$
$$f(A \cup B_i') = 1, \text{ whenever } i \in A; \tag{4.4}$$
$$|J_A| \leq 2 \cdot |A| \cdot s(f). \tag{4.5}$$

(4.2) ensures that the function value is still zero after all the bits in A have been turned on. (4.3) ensures that the function is still block-sensitive to all the blocks of size $c-1$ or fewer. (4.4) makes sure that the input $\mathbb{1}_A$ is block-sensitive to the sub-block B_i' for every block B_i which has its first bit included in A. Note that each block mentioned in (4.3) and (4.4) has size $c-1$ or less. Hence, if we can find an A large enough so that $|A| + (t-r) \geq \frac{t}{8 \cdot s(f)}$, then choosing $a' := \mathbb{1}_A$ gives us the input satisfying the lemma.

To motivate (4.5), we note that in order for A to be able to grow large, there should be a large number of bits that can potentially be added to A. Observe, however, that any bit in J_A is an invalid addition to A (because, for every $j \in J_A$, $A \cup \{j\}$ violates (4.4) when $i = j$). Hence, the purpose of the condition in (4.5) is to bound the size of J_A, ensuring that there are a large number of potential additions to A.

We now prove that there exists an A satisfying (4.2) - (4.5) such that $|A| + (t-r) \geq \frac{t}{8 \cdot s(f)}$

Observe that each of the above properties is satisfied for $A = \varnothing$, following from (4.1).

Suppose A is maximal satisfying the above properties, and that $|A| + (t-r) < \frac{t}{8 \cdot c \cdot s(f)}$. We will show that there is a bit $j \in [r] \setminus A$ that can be added to A while still maintaining the above conditions.

Let $C \subseteq [r] \setminus A$ denote the bits in $[r] \setminus A$ which, if added to A, would violate (4.2), (4.3) or (4.4) (for $i \neq c$). Note, however, that if $c \in C$ violates (4.2), then $f(A) = 0$ but $f(A \cup c) = 1$), hence f must be sensitive to c on input $\mathbb{1}_A$. But there can be at most s(f)cs that $\mathbb{1}_A$ can be sensitive to. Similarly, if $c \in C$ violates (4.3), then f must be sensitive to c on input $\mathbb{1}_{A \cup B_i}$ for some $i \in \{r+1, \ldots, t\}$, and if $c \in C$ violates (4.4), then f must be sensitive to c on input $\mathbb{1}_{A \cup B_i}$ for some $i \in A$. In total, there are $|A| + (t-r) + 1 \leq \frac{t}{8 \cdot c \cdot s(f)}$ such inputs, and each of them can be sensitive to at most s(f) bits, it follows that

$$|C| \leq \frac{t}{8}$$

As mentioned earlier, J_A denotes the bits $j \in [r] \setminus A$ that violate (4.4) for $i = j$. Define D as follows:

$$D := [r] \setminus (A \cup C \cup J_A)$$

Then, adding any bit from D to A would preserve (4.2), (4.3) and (4.4). Also

$$\begin{aligned}
|D| &= r - (|A| + |C| + |J_A|) \\
&= t - (|A| + (t - r) + |C| + |J_A|) \\
&\geq t - \Big(\frac{t}{8 \cdot s(f)} + \frac{t}{8} + 2 \cdot \frac{t}{8 \cdot s(f)} \cdot s(f)\Big) \\
&\geq t - (\frac{t}{8} + \frac{t}{8} + \frac{t}{4}) \\
&\geq\ t - \frac{t}{2} \\
&= \frac{t}{2}
\end{aligned}$$

We need to show that there is a way to choose $j \in D$ so that $A \cup j$ satisfies (4.5). It suffices to find a j such that it adds at most $2 \cdot s(f)$ bits to J_A.

What would cause a bit $i \in [r] \setminus (A \cup J_A)$ to be added to J_A? We know that this i is not already a member of J_A, hence $f(A \cup B_i) = 1$. However, when j is added to A, i will be added to J_A if and only if $f(A \cup \{j\} \cup B_i) = 0$. This means that the input $\mathbb{1}_{A \cup B_i}$ is sensitive to the bit j. Thus, for each bit $i \in [r] \setminus (A \cup J_A)$, we define:

$$S_A(i) \ :=\ \{j \in D : f(A \cup B_i \cup j) = 0\}$$

That is, $S_A(i)$ consists of all bits j that the input $\mathbb{1}_{A \cup B_i}$ is sensitive to, and thus also all js such that if j were added to A, then i would be added to J_A. If we want as few bits as possible to be added to J_A, we should choose a $j \in D$ such that it is in as few $S_A(i)$s as possible. Noting that there are t $S_A(i)$s and $t/2$ js in D, and that each $S_A(i)$ can contain at most s(f) js, we can apply the pigeonhole principle to get a j that is in at most $2 \cdot s(f)$ $S_A(i)$s.

Choosing this j means that $A \cup j$ satisfies all of (4.2) - (4.5). This contradicts our assumption that A is maximal. Thus we can have A large enough so that $|A| + (t - r) \geq \frac{t}{8 \cdot s(f)}$, which gives us the lemma.

□

We can now apply this repeatedly, continuously turning on single bits in blocks, until those blocks have only one bit remaining in them. The function will then be sensitive to those single bits. This process is captured in the following theorem.

Theorem 4.2. *Suppose that the Boolean function $f : \{0,1\}^n \to \{0,1\}$ has sensitivity s(f), and has a sensitive block decomposition $\{B_1, \ldots, B_t\}$, at a certain input a, consisting of t blocks, each of size c or less, $c \geq 1$. Then* s$(f) \geq \frac{t^{\frac{1}{c}}}{8^{c-1}}$.

Proof. We will prove the theorem by induction on c.

When $c = 1$ each sensitive block consists of a single bit, and so sensitivity = block sensitivity = t.

Now suppose $c > 1$ and that s$(f) < \frac{t^{\frac{1}{c}}}{8^{c-1}}$. Using the lemma, we can construct an input that has a sensitive block decomposition into $\geq t^{\frac{c-1}{c}} \cdot 8^{c-2}$ blocks of size c-1 or less. Using the inductive hypothesis on this input,

$$\text{s}(f) \geq \frac{t^{\frac{1}{c}} \cdot 8^{\frac{c-2}{c-1}}}{8^{c-1}} > \frac{t^{\frac{1}{c}}}{8^{c-1}},$$

contradicting our assumption that s$(f) < \frac{t^{\frac{1}{c}}}{8^{c-1}}$.

Hence the theorem holds for all $c \geq 1$, and is tight at $c = 1$. □

From the theorem, it follows that $s(f) = \Omega(bs(f)^{1/c})$ when the blocks are of size c or less for some constant c. Also, sensitivity and block sensitivity must be polynomially related when $bs(f) \geq n/c$ for a constant c, because the input with block sensitivity Kn must have atleast $n/2c$ blocks of size $2c$ or less.

Notice that the input we created over the induction has a tiered form, where we first turn on lots of single bits in large blocks, then turn on more single bits in a subset of those partially filled blocks, and so on, throwing away some blocks at each level. An immediate question is whether we can generalize Rubinstein's function to get an example of a function that makes this theorem "tight", by causing us to throw away the maximum number of blocks at each step. (Note that Rubinstein's function makes this theorem tight for $c = 2$). However, as our next result shows, all natural extensions of Rubinstein's function to blocks of size greater than 2 cannot improve the gap.

Chapter 5

Rubinstein's Function is the Best in its Class

Here we will show that a large class of generalizations of Rubinstein's functions fail to create a super-quadratic gap. We will change Rubinstein's function in the following ways:

- by increasing the number of levels
- by allowing a more general class of functions at each level

To create a function F corresponding to a tree of functions of height h, where h is some constant, at each level i we will use intermediate functions of the form $f_i : \Sigma_{i+1}^{k_i} -> \Sigma_i$. Each f_i will take the output of k_i copies of f_{i+1} as input. The root function f_1, has its output alphabet Σ_1 restricted to $\{0, 1\}$, and the leaf functionf_h has input alphabet $\{0, 1\}$ as well. The remaining Σ_is can be chosen arbitrarily, as long as they are of at most constant size. This restriction ensures that $F : \{0, 1\}^n \to \{0, 1\}$, where $n = k_1 \cdot k_2 \cdot \cdots \cdot k_h$

We add the additional restriction that the intermediate functions f_i must also be *symmetric*, that is, their output is determined solely by the *number of occurrences* of each variable in the input, without depending on their position. We will call these functions *SSAF*s, for *symmetric small-alphabet functions*. We will call the function F an *HSF*, for *hierarchically symmetric function*. We will show that every *HSF*, consisting of a tree with *SSAF*s at each level, must have $s^0(f) \cdot s^1(f) = \Omega(n)$. It follows from this that $\mathrm{s}(f) = \Omega(\sqrt{n})$, meaning that *HSF*s can't have more than a quadratic gap between $\mathrm{s}(f)$ and $bs(f)$. Our proof strategy will be to create an input such that every level is highly sensitive, either in the $0 \to 1$ direction or the $1 \to 0$ direction.

Note that some of the k_is can be constant, and can simulate blocks if f_i is the weight function. Then it is clear that Rubinstein's function is an *HSF*, where $f_1 = OR$, f_2 is the function that is $1 \iff$ exactly 1 input bit is 2 and the rest are all zero, and f_3 is the weight function for 2 bits.

Also, observe that cyclically invariant functions and *HSF*s are are distinct classes of functions. There are some functions that are both cyclically invariant and *HSF*s, such as the symmetric functions. But there are *HSF*s that are not cyclically invariant (for example Rubinstein's function), and also other functions that are cyclically invariant but cannot be decomposed into *HSF*s. It is the case, however, that all *HSF*s are weakly symmetric.

We will now define some notions that are useful for our proof.

Definition 5.1. Two variables A_i and $B_i \in \Sigma_i$ are called *adjacent* if there is an input $x \in {\Sigma_{i+1}}^{k_i}$ such that $f_i(x) = A_i$, and there is a single variable change in x that produces x' such that $f_i(x') = B_i$

Definition 5.2. For two adjacent variables A_i and $B_i \in \Sigma_i$, $x \in \Sigma_{i+1}^{k_i}$ is a *highly sensitive forward interface* (*HSI*) from A_i to B_i if $f_i(x) = A_i$, and there are $\geq \frac{k_i}{2 \cdot |\Sigma_{i+1}|}$ places where a single variable change from A_{i+1} to

B_{i+1} will create x' with $f_i(x') = B_i$. (Here, A_{i+1} are B_{i+1} are elements of Σ_{i+1}). We call x' a *highly sensitive backward interface* (*HSBI*) from B_i to A_i.

Definition 5.3. A function f_i is said to have a *perfect subtree* from A_i to B_i with input $x \in \Sigma_{i+1}^{k_i}$, if x is either an HSFI or an HSBI from A_i to B_i, and if the associated variables that are to be flipped in x are A_{i+1} and B_{i+1}, and if $i \neq h$, then f_{i+1} also has a perfect subtree from A_{i+1} to B_{i+1}.

As we see, the recursive definition of perfect subtrees means that they have either *HSFI*s or *HSBI*s at each level.

Notice that the perfect subtree property is reflexive, that is, if f_i has a perfect subtree from A_i to B_i with input x, then f_i also has a perfect subtree from B_i to A_i for some input x'. To see this, simply flip the single variable in x that can change the output of f_i from A_i to B_i, and note that if x was an *HSFI*, then the resulting x' is an *HSBI* and vice versa.

Definition 5.4. For every function f_i, starting with f_h, we define 0_i to be the variable in Σ_i that is output by f_i on the input $\vec{0}$.

$$0_h = f_h(\vec{0})$$

$$0_i = f_i({0_{i+1}}^{k_i}) \text{ for } i \neq h$$

Definition 5.5. A *0-sequence* for a variable $A_i \in \Sigma_i$ is a sequence of variables $B_i^1, B_i^2, ..., B_i^t$ with each $B_i^j \in \Sigma_i$, $B_i^1 = A_i$ and $B_i^t = 0_i$, where each pair B_i^j, B_i^{j+1} is *adjacent*.

Definition 5.6. A *perfect 0-sequence* for a variable $A_i \in \Sigma_i$ is a $0-$*sequence* for A_i where each pair B_i^j, B_i^{j+1} in the sequence has a perfect subtree.

Note that if two variables A_i and B_i both have perfect 0-sequences, then we can create a perfect sequence of adjacent variables from A_i to B_i by concatenating the perfect 0-sequence for A_i with the reverse of the perfect 0-sequence for B_i.

Lemma 5.7. *Let F be a HSF of height h with intermediate functions of the form $f_i : \Sigma_{i+1}^{k_i} \to \Sigma_{i+1}$. Then, for all $i \in [h]$, each non-trivial variable $A_i \in \Sigma_i$ has a perfect 0-sequence. (A non-trivial variable A_i is one that has at least 1 input x such that $f_i(x) = A_i$.)*

Proof. We prove the claim by induction on the level of the tree, starting from the leaves.

At level h, notice that having a perfect subtree is the same as having either a *HSFI* or a *HSBI*.

0_h has a trivial perfect 0-sequence. Choose any other nontrivial variable A_h, and let x be any input to f_h yielding A_h. Then we can turn off individual bits in x until we get to $\vec{0}$. We let $B_h^1, \ldots, B_h^t$ be all the values the function f_h takes as we do this, in order. We claim that every pair of adjacent variables in this sequence has either an *HSFI* or an *HSBI*. It has an *HSFI* if there were more than $k_h/2$ ones in the input at the time of the flip, because there are $\frac{k_h}{2}$ choices of 1 to change to 0. Similarly, if there were fewer than $\frac{k_h}{2}$ ones in the input, we have an *HSBI*. Note that this argument is nearly identical to the one used to show that symmetric functions have sensitivity $\geq n/2$.

Thus, we can construct a perfect 0-sequence for every variable at level h.

At level i, assume that for the function one level lower, f_{i+1}, every variable $A_{i+1} \in \Sigma i + 1$ has a perfect 0-sequence. Again, 0_i has a perfect 0-sequence. Pick any other nontrivial variable in Σ_i, say A_i, and let x be an input for it. Choose the variable with the largest number occurrences in x, call this A_{i+1}. Then there are at least $\frac{k_i}{|\Sigma_{i+1}|}$ variables with value A_{i+1}. Suppose $B_{i+1}^1, \ldots, B_{i+1}^t$ is the 0-sequence for A_{i+1}. Then change every occurrence of A_{i+1} to B_{i+1}^2, one variable at a time. Then change every occurrence of B_{i+1}^2 to B_{i+1}^3, and so on, following the perfect 0-sequence for A_{i+1}, until each of these variables has been turned into 0_{i+1}. Then

there are at least $\frac{k_i}{|\Sigma_{i+1}|}$ such variables. Turn each occurrence of 0_{i+1} into the variable with the second-highest frequency in the input, following the reverse of the perfect 0-sequence for that variable. Now there are at least $\frac{k_i}{|\Sigma_{i+1}|}$ variables of this type. Repeat this process, moving to 0_{i+1} and back to the variable with the next highest number of occurrences each time, until the entire input consists of a single variable. Finally, change every input variable to 0_{i+1}. Now the output of f_i is 0_i. Let the perfect 0-sequence for A_i be the values that f_i takes as we perform this process, in order.

To see that this is indeed a perfect 0-sequence, note that for every change between B_i^j and B_i^{j+1} in this sequence, where we let the variable changed from be A_{i+1}, the variable being changed to be B_{i+1}, and the input being changed from be x, we have that x must be either a *HSFI* or a *HSBI* from B_i^j to B_i^{j+1}. This is because, by our ordering, there were at least $\frac{k_i}{|\Sigma_{i+1}|}$ copies of A_{i+1} before we changed a single A_{i+1} to a B_{i+1}, and so at x, at least one of A_{i+1} and B_{i+1} appears $\geq \frac{k_i}{2\cdot|\Sigma_{i+1}|}$ times. If it is A_{i+1}, then x must be a *HSFI*, and if it is B_{i+1}, then x is a *HSBI*. Further, because A_{i+1} and B_{i+1} are neighbours in a perfect 0-sequence, we know that A_{i+1} and B_{i+1} must have a perfect subtree. It follows that B_i^j and B_i^{j+1} also have a perfect subtree, with input x. Since this holds for all pairs B_i^j and B_i^{j+1}, the sequence is in fact a perfect 0-sequence. This completes the inductive step.

Hence, all nontrivial variables in a *HSF* have a perfect 0-sequence.

□

Theorem 5.8. *Let F be a nonconstant HSF of height h with intermediate functions of the form $f_i : \Sigma_{i+1}^{k_i} \to \Sigma_{i+1}$. Then $s^0(F) \cdot s^1(F) = \Omega(k_1 \cdot k_2 \cdot \cdots \cdot k_h)$.*

Proof. It follows easily from the lemma that there is a perfect subtree from 0 to 1 with some input x. Then if x is a *HSFI*, there are $\frac{k_1}{2\cdot|\Sigma_2|}$ places to change some A_2 to B_2 to flip the function value from 0 to 1. If x is a *HSBI* there may be at most 1 place to make such a change. Now look at each such A_2 where we can make such a change to B_2. We know that there must be a perfect subtree from A_2 to B_2, with an input that is either a *HSFI* or a *HSBI*. As for level 1, if it is a *HSFI*, there are at least $\frac{k_2}{2\cdot|\Sigma_3|}$ places to make a change in it to flip the output of f_2 from A_2 to B_2, whereas if it is a *HSBI* there may be at most 1 place to make such a change. Continuing in this fashion all the way to the leaves, we see that if $S \subseteq [h]$ is the subset of levels where the flip in variables is in the *HSFI* direction, then we have $\prod_{i\in S} \frac{k_i}{2\cdot|\Sigma_{i+1}|}$ bits that can be flipped at the lowest level to flip the function from 0 to 1, that is, $s^0(f) \geq \prod_{i\in S} \frac{k_i}{2\cdot|\Sigma_{i+1}|}$ sensitive bits.

Now consider the input we get with a single sensitive bit flipped, so that F is 1, and can be changed to 0 in a single bit flip. Since the perfect subtree property is transitive, there is still a perfect subtree from 0 to 1. We can make the same argument as we did in the first case in this case as well, except observe that everywhere we could change A_is to B_is in the first case, we must now change B_is to A_is. This means that all the levels that had *HSFI*s in the first case are *HSBI*s in this case, and vice versa. If we go all the way down the tree as we did in the first case, we get that $s^1(f) \geq \prod_{i\notin S} \frac{k_i}{2\cdot|\Sigma_{i+1}|}$.

Taking both equations together, we have that $s^0(F) \cdot s^1(F) = \Omega(k_1 \cdot k_2 \cdot \cdots \cdot k_h) = \Omega(n)$.

□

Chapter 6

Concluding remarks

6.1 Discussion of results

Our results relate well to previous work in the field. Kenyon and Kutin conjectured that for "nice" functions, $s^0(f) \cdot s^1(f) = \Omega(n)$. They were vague as to what was meant by "nice", but our results show that this exact bound holds for a large class of generalizations of Rubinstein's function.

It remains open whether a superquadratic gap between sensitivity and block sensitivity is possible, but our work suggests that any function that creates such a gap must be significantly different from Rubinstein's function, and, arguably, more complex than it.

6.2 Open Problems

There are several improvements to our theorem that appear possible. One is to remove the restriction that the intermediate alphabets Σ_i of *HSF*s be of at most constant size. We suspect that such an improvement may be possible.

The most significant open problem is the same one we introduced at the very start, namely what the largest possible gap between sensitivity and block sensitivity. We have provided a partial answer, but we feel the question is still quite open.

Bibliography

[1] S. Chakraborty. On the Sensitivity of Cyclically-Invariant Boolean Functions. ECCC Report No. TR05-020.

[2] H. Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey, *Theoretical Computer Science*, Volume 288, Number 1, 9 October 2002 , pp. 21-43(23)

[3] Andrew Drucker. Block Sensitivity of Minterm-Transitive Functions

[4] Claire Kenyon and Samuel Kutin. Sensitivity, block sensitivity, and l-block sensitivity of Boolean functions. *Information and Computation*, vol 189 (2004), no. 1, 43-53.

[5] Noam Nisan. CREW PRAMs and decision trees. *SIAM J. Comput*. 20 (1991), no. 6, 999-1070.

[6] David Rubinstein. Sensitivity vs. block sensitivity of Boolean functions. *Combinatorica* 15 (1995), no. 2, 297-299.

[7] Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theor. Comput. Sci.* , 384(1):8791, 2007.

[8] György Turán. The critical complexity of graph properties. *Inf. Process*. Lett., 18(3):151153, 1984.