Lower Bounds on the Communication Complexity of Shifting

Marco D. Adelfio

Advisor: Amit Chakrabarti

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

June 2, 2005

Contents

Chapter 1

Introduction

The study of complexity focuses on classifying problems by their difficulty. Why are some problems inherently *harder* to solve than others? In complexity theory, there are standard ways of classifying problems by the time or space required by an algorithm that solves a specific problem, such as sorting a list of numbers or multiplying two matrices. Other measures of complexity involve different models and measures in order to classify the difficulty of problems.

1.1 Communication Complexity

One such measure of a problem is communication complexity, in which an input is distributed among different parties who wish to compute a specific function of those inputs. Since each party is missing a part of the input, they must communicate in order to compute the function. The most basic model of communication complexity was introduced by Yao in 1979 (see [2] for a thorough introduction). There are two parties (traditionally called Alice and Bob), each with a specific part of the problem's input. Both Alice and Bob are assumed to have unlimited computation power, and the goal is to minimize the amount of communication between them, as a function of the input size, to solve the problem.

Some examples of interesting functions in this model are the following, where Alice starts with input x and Bob starts with input y:

- The equality function EQ(x, y), where $x, y \in \{0,1\}^n$, and EQ$(x, y) = 1$ iff $x = y$.
- The disjointness function DISJ(x, y), where $x, y \subseteq \{1, \ldots, n\}$, and DISJ$(x, y) = 1$ iff $x \cap y = \emptyset$.
- The inner product function IP(x, y), where $x, y \in \{0,1\}^n$, and IP$(x, y) = \Sigma_{i=1}^{n} x_i y_i \mod 2$.

To evaluate each of these functions, some communication must take place between Alice and Bob. Their communication follows a fixed *protocol*, a sequence of steps taken by both parties that depends on the input each has and any earlier communication. At the end of the protocol, at least one of the parties must have evaluated the function $f(x, y)$.

The simplest way to solve a problem in this model is to send all the inputs to a single party who calculates the result. This protocol requires n bits of communication for an input of size n bits and this protocol can be applied to any function. So this is the trivial upper bound for all problems in this model. For some functions, such as EQ, this bound cannot be improved without altering the model—if there are not n bits communicated, then not all n bits of the inputs can be tested for equality and the function cannot be

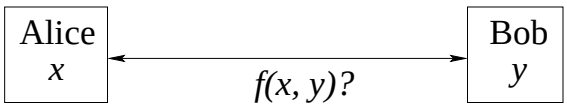

Figure 1.1: The basic model for analyzing communication complexity.

computed (in fact, both DISJ and IP share this $\Theta(n)$ bound on communication in this model). However, there are communication models where the complexity of EQ is less than $O(n)$. Allowing randomized computation by the parties allows them to compute a different value for a function over multiple trials. If we bind the acceptable error rate close enough to zero, we can create a protocol that gives us a "good enough" approximation of the function with much less communication. In the case of EQ the randomized bound becomes $O(1)$.

Another variation on the standard communication model is to add a third player—a *referee*—and allow only a single round of communication from Alice and Bob to the referee. In the *simultaneous* model for computing a function f, the referee does not have initial access to any of the input and computes the value of f only from the information he receives from Alice and Bob. This is also known as the "Number on the Forehead" model because in the extensions of this model, each player sees all but one of the inputs. That is, for k inputs $x_0, \ldots, x_{k-1}$, player i sees inputs $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{k-1}$. The *almost simultaneous* model is similar, but the referee *is* given access to some of the inputs (i.e. the referee is one of the players who can see all but 1 input). If one input is signifi cantly larger than the others, the communication complexity of the simultaneous and almost simultaneous models will be asymptotically equal, since everyone could send the small inputs to the referee without affecting the order of the complexity. In the three-player version of this model, a function $f : X \times Y \times Z \to \{0, 1\}$ with three inputs $x \in X, y \in Y, z \in Z$ is evaluated. Alice knows x and z, Bob knows y and z, and the referee knows x and y, and communication proceeds as before.

1.2 Circuit Complexity

One fundamental measure of complexity is the complexity of a circuit that computes a certain function, measured in terms of the size and depth of the circuit. A circuit is defi ned to be a directed acyclic graph, where the source nodes are associated with input bits, and all other nodes are associated with Boolean gates. The gates are divided into levels, which connect the set of input nodes to an output node, or multiple output nodes. The gates in the fi rst level can only have incoming edges from the input nodes. All subsequent levels can have incoming edges from the input nodes or from gates that came in earlier levels. We call the number of levels in a circuit its *depth*, and the number of edges between gates in a circuit its *size*. For our purposes, we limit the gate-types to AND, OR, and NOT gates, and limit the number of inputs for each gate to two.

The motivation for using circuits to measure complexity is their relative simplicity when compared to computers, Turing machines, or other models of computation. Information flows in one direction in a circuit, from the input to the output through a fi xed number of intermediate levels. But circuits in this simple model, even when restricted to the three gate types above with limited inputs, are still capable of computing all Boolean functions. With that in mind, functions are not interesting in this model because they *can* be computed by a circuit, it is the scale of such a circuit that concerns us.

It is a well-known fact of circuit complexity that there must exist functions whose computation requires large circuits (i.e. circuits with large size or depth). Simply counting the number of functions of n input bits, which is 2^{2^n}, and comparing this to the number of possible polynomial-sized circuits shows that the

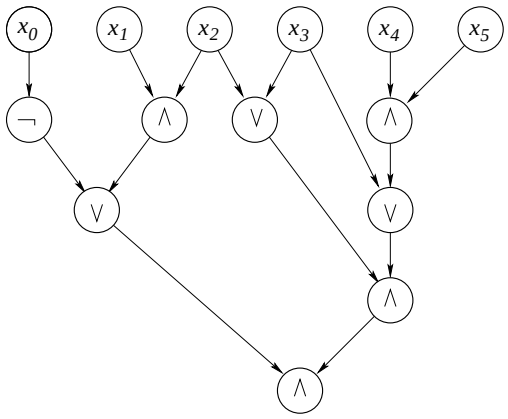

Figure 1.2: An example of a circuit to compute the truth value of the Boolean expression: $(\neg x_0 \vee (x_1 \wedge x_2)) \wedge ((x_2 \vee x_3) \wedge (x_3 \vee (x_4 \wedge x_5))))$.

ratio of functions computable by polynomial-size circuits to total functions of an n-bit input goes to zero as n gets large. However, despite this evidence that such *hard* functions exist (in fact, that almost all Boolean functions are hard), no explicit function has been found that has strong lower bounds on the circuit size, or even less strict bounds on the depth and size. It is a major open problem to find a function that has no circuit with a simultaneous $O(n)$ bound on size and $O(\log(n))$ bound on depth.

1.3 Communication Complexity and Size-Depth Bounds

An interesting quality of these two measures of complexity is that the complexity of a function in one can often be related to the complexity of a function in the other. In fact, despite their dissimilar structure, both communication complexity and circuit complexity focus on the paths information must take from input to output, with the major difference being that the actual computation is abstracted out of the communication model. The approach of translating problems from circuits to communication, and the other way around, has been applied in various ways, for different models within the realm of communication complexity and circuit complexity. Also, the function being analyzed in one model may not be the same as the function it relates to in the other model.

The best-known connection between these two types of complexity equates the circuit depth of a function to the communication complexity of a relation that derives from that function. This property is very useful, since most functions are easier to classify in one model than the other. The communication model of this relation is known as the Karchmer-Wigderson game, and creates a particularly nice isomorphism between Boolean circuits and communication protocols because the depth of the circuit exactly equals the number of bits sent in the protocol.

For our purposes, the important connection between communication complexity and circuit complexity derives from a graph-theoretic fact originally proved by L. G. Valiant in 1977 (see [8]). The application of this theorem to circuits would allow lower bounds for certain communication complexity problems to prove the existence of a *hard* function (i.e. one for which there is no circuit of size $O(n)$ and depth $O(\log n)$ simultaneously). More specifically, bounds in the almost-simultaneous three-player communication model (with certain constraints) translate into circuit size and depth bounds, and vice-versa. We make these relationships explicit in the next chapter.

Chapter 2

Complexity of Shifting

We begin exploring the connections between different models of complexity with the following precise formulation of how bounds in communication complexity and circuit size and depth can be related.

Theorem 2.1 *Let C be a circuit with depth $O(\log n)$ and size $O(n)$. Let $f(i,j,x)$ be the corresponding three argument function, so that $f(i,j,x) = y_i$, the ith output bit of the circuit on input (j,x) (where i, j, and x are proportionally sized as before: $0 \leq i < n$, $0 \leq j < n$, $x \in \{0,1\}^n$). Then f can be computed using a restricted, almost simultaneous protocol in which:*

- *Alice (holding x, j) sends $O(n/\log\log n)$ bits;*
- *Bob (holding x, i) sends $O(n^\epsilon)$ bits, for a fixed constant ϵ;*
- *Referee (holding i, j) computes $f(i,j,x)$.*

The importance of this connection will become clear as we introduce an intermediate model and formally relate bounds in all three.

2.1 The Sum-Index Problem

Let's examine a communication problem in the *almost simultaneous* model. For this problem, Alice has an n-bit Boolean array x and a positive integer $i < n$; Bob has the same array x and a different positive integer $j < n$; and the referee has the pair of indices i and j. There is no direct communication between Alice and Bob, they can only send a single message each to the referee, who wants to compute the function $f(i,j,x) = x_{i+j \mod n}$. (For simplicity, we will omit the *mod n* from now on, and assume all indices are computed this way.)

At first glance, it seems doubtful that sending fewer than n bits will be enough to evaluate the function. In the case that Alice does not send any bits, all of the referee's information about the array must come from Bob. Since any of the n elements of the array could be the desired one, Bob has no choice but to send n bits on to the referee (since the values of x must map one-to-one onto the messages Bob sends). The difficulty in finding an efficient protocol is dealing with the shared information of the three parties.

If Alice sends just a single bit, it is not at all obvious that this can reduce Bob's message significantly (by any more than one bit). The sum $i + j$ could take any value from 0 to n, so Alice and Bob must each send messages which take into account all n of these possibilities. If Alice and Bob together send fewer

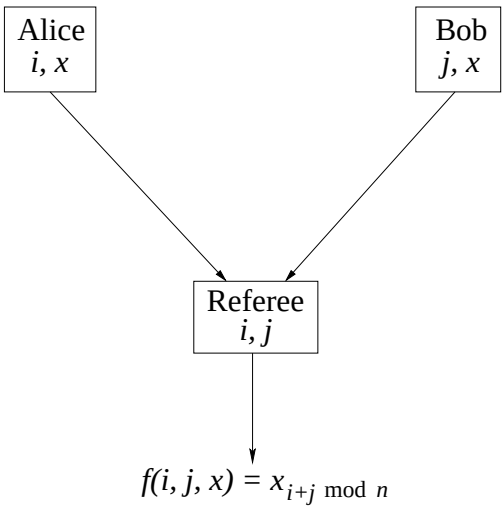

Figure 2.1: The three player communication model, computing the Sum-Index function.

than n bits, it seems unlikely that x_{i+j} will always be among them. Since they are sending fewer than n bits, there are cases when they send the exact same messages, even for different inputs. One way to approach the problem is to explore how to reconcile these overlapping messages.

It turns out that there is a protocol in which Alice sends a single bit, and Bob sends $n/2$ bits. It was introduced by Pavel Pudlak, Vojtech Rodl, and Jiri Sgall in 1997, and follows the approach outlined above to determine the wanted bit of the original array [6]. The messages sent in the protocol are as follows:

- **Alice** sends the bit x_{-i}
- **Bob** sends the bits $x_0 \oplus x_j, x_1 \oplus x_{j-1}, x_2 \oplus x_{j-2}, \ldots$. There will be $n/2$ distinct pairs whose indices sum to j, so Bob sends $n/2$ bits.
- **The Referee** computes $x_{-i} \oplus (x_{-i} \oplus x_{i+j}) = x_{i+j}$.

The crux of the protocol is the pairing scheme—the pairing has to depend solely on j but must send the parity of x_{i+j} with another bit for every possibly value of i. (Note: to achieve the $\lfloor n/2 \rfloor$ bit message for Bob, he omits sending the bit $x_k \oplus x_{j-k}$ when $k = j - k$, since this parity is always 0.) Seeing that one bit from Alice can halve the necessary bits from Bob provokes an interesting question: Can we reduce Bob's message size further?

2.2 An Initial Lower Bound

The best-known lower bound for the overall communication complexity of the Sum-Index function is $\Omega(\sqrt{n})$ bits, much lower than the upper bound shown above of $\lfloor n/2 \rfloor + 1$ bits. This is proved generally for k players in [6], by comparing the number of unique messages generated by the players to the number of input strings. For two players, take any protocol P, for the Sum-Index function. We restrict the inputs of Alice and Bob so that Alice's index ranges over $0, \sqrt{n}, 2\sqrt{n}, \ldots, n - \sqrt{n}$, and Bob's index ranges over $0, 1, \ldots, \sqrt{n} - 1$. Each index can only take $\sqrt{n}$ values, but their sum can still be any number in the original range $0, 1, \ldots, n$. Now, for every combination of indices i and j in this restricted range, we can write down the messages that Alice

and Bob would send. If we express the communication complexity of the Sum-Index problem by CC(SI), the total size of all these messages is at most nCC(SI). Since there are n combinations of indices, and each player has only $\sqrt{n}$ possible indices, each player repeats his or her message for $\sqrt{n}$ combinations. So only $\frac{n}{\sqrt{n}}$CC(SI) $= \sqrt{n}$CC(SI) bits of these messages are unique and not sent for earlier combinations of indices. Thus the entire input x can be reconstructed from these $\sqrt{n}$CC(SI) bits. So there is a one-to-one mapping from values of x to the $\sqrt{n}$CC(SI)-bit list of messages, for any protocol P for the Sum-Index function. Therefore CC(SI) $\geq \Omega(\sqrt{n})$

We would like to prove a stronger lower bound, so we introduce the following model to simplify the analysis of our problem.

2.3 Valiant's Graph Model

One approach to resolve questions of both circuit complexity and communication complexity uses a model orginally proposed by L. G. Valiant [9]. Valiant examines the properties of circuits in terms of graphs that mirror their structure. By proving a graph-theoretic fact about the number of paths in a graph, he shows that lower-bounds in the following model imply lower bounds for a general circuit.

Definition 2.2 *A graph is an ordered pair $G = (V, E)$ where V is a set of* vertices *or* nodes*, and E is a set of edges between vertices. In a directed graph, the edges are* ordered *pairs, while an undirected graph has edges made up of unordered pairs of vertices.*

Consider a graph G which consists of vertices $X \cup Y$ where $X = \{x_0, x_1, \ldots, x_{n-1}\}$ corresponds to n inputs and $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ corresponds to n outputs. The edges of G are defined implicitly by the function $\tau : Y \to 2^X$ where $\tau(y_i) \subseteq X$ is the set of input vertices which are adjacent to y_i in the graph. Now we define two sets of Boolean functions: m Boolean functions $r_0(x), \ldots, r_{m-1}(x)$, which we will call the graph's *common bits*; and n further Boolean functions $f_0(\ldots), \ldots, f_{n-1}(\ldots)$. There are $m + |\tau(y_i)|$ Boolean arguments for each f_i, namely the values of all the functions $r_0(x), \ldots r_{m-1}(x)$ and the values of the inputs in the set $\tau(y_i)$. In other words, we have m functions of the inputs, and each output y_i can be an arbitrary function of these *m common bits* and of the inputs $\tau(y_i)$ to which it is connected directly in G.

Definition 2.3 *We say that G* realizes *or* computes *function $F : \{0,1\}^n \to \{0,1\}^n$ with m common bits if there exist $r_0, \ldots, r_{m-1}$, and $f_0, \ldots, f_{n-1}$ such that for all $i, 0 \leq i \leq n-1$:*

$$F_i(x) = f_i(r_0(x), \ldots, r_{m-1}(x), \tau(y_i)) \text{ (where } F_i \text{ denotes bit } i \text{ of } F(x))$$

By the degree of G*, we mean* $\max_{y \in Y} |\tau(y)|$*. By the* common bits *we mean the values of the functions* $r_0, \ldots, r_{m-1}$.

This is a slight abuse of notation, since the final argument $\tau(y_i)$ is actually a set of input nodes. The intended meaning is that f_i is a function of all the common bits in addition to exactly those inputs that it is connected to in G. So not all input bits are directly available to each output function f_i, all other significant input values must be communicated via the common bits. Thus a graph with n common bits can realize any function, since the common bits can simply mirror the input bits (e.g. $r_i = x_i$), leaving each output function with direct access to all of the input. Similarly, a graph with degree n and 0 common bits computes any function, since each output node is connected to each input. Thus, the important properties of graphs in this model are the number of common bits and the degree of the graph. We are interested in minimizing both these quantities in order to classify the complexity of functions in this model.

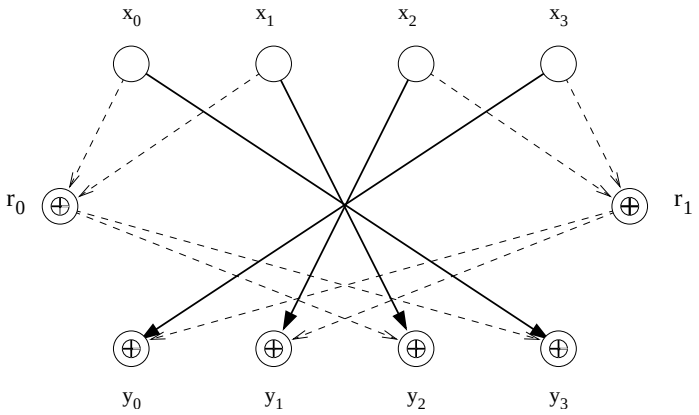

Figure 2.2: Computation of a shift, where $y_i = x_{(i+2) \mod 4}$.

Figure 2.2 illustrates a small graph that computes a *shift* function. The common bits r_1, r_2, and the dotted edges connected to them are not technically part of the graph, but are shown here to illustrate the passage of information from the input nodes to the output nodes. The significance of the shift function will become clear in the next sections, but for now we define it as a function that shifts the input bits, in this case by 2 bits so that $y_i = x_{(i+2) \mod 4}$. The common bits and output nodes are assigned the *XOR* function ($\oplus$), which is evaluated on the values of adjacent input nodes and selected common bits. In a manner similar to the protocol of Section 2.1, the output values are correctly assigned the values of the corresponding inputs.

As we will see, this graph model is especially useful for analyzing functions that permute the input. Computing a *single* permutation would be extremely simple: connect each input to the output that is to take its value. This graph, of degree 1, realizes the given permutation without the need for *any* common bits. The non-trivial power of a graph is to compute *multiple* functions. Unlike a circuit, graphs in this model can compute multiple functions, since the only fixed component of the model is the graph itself, not the functions at the nodes.

2.4 Connection to Circuit Size and Depth

In particular, the following reduction gives a specific bound to aim for in the graph model, and was the primary motivation for its original development.

Theorem 2.4 *([6], [9]) For every $\epsilon > 0, c$, and d there exists K such that if a function F can be computed by a circuit of size cn and depth $d \log n$, then it can be computed by a graph G of degree at most n^ϵ with $Kn/\log\log n$ common bits.*

The proof proceeds from a graph-theoretic property proved earlier by Valiant. Namely that for any $\epsilon > 0$ and circuit C with size $O(n)$ and depth $O(\log n)$, there is a set S of $O(n/\log\log n)$ edges such that every path of length $\epsilon \log n$ in C contains an edge from S. To convert a circuit of this form into a graph G, first assign values of the edges in S to the common bits of G. Since each output bit is a function of the values at these $o(n/\log\log n)$ edges and any other input bits reachable by a path avoiding S (and hence shorter than $\epsilon \log n$), we connect each output in G to the corresponding inputs. The maximum number of these inputs (i.e. the degree of G) is then $2^{\epsilon \log n} = n^\epsilon$. Thus, if we have a circuit of this form that computes a specific function, the function can also be computed by a graph with these properties.

The reverse of Theorem 2.4 is what we are most interested in. If there is a function such that no graph G with degree n^ϵ and $o(n)$ common bits can compute it, the only circuits that can compute the function are simultaneously large and deep (that is, size $= \Omega(n)$, depth $= \Omega(\log n)$). If we can find functions which are easy to analyze using this model, we should be able to find strong lower bounds on that function's complexity.

2.5 Connection to Communication Complexity

Valiant's graph model can also be expressed in terms of a problem in communication complexity, a reduction established in [6].

Theorem 2.5 *A function $F(p, x)$ can be computed by a graph of maximal degree d with r common bits if and only if there exists a restricted protocol for computing $bit_F(i, p, x)$ in which Alice sends r bits and Bob sends d bits.*

For a function of two variables, $F(p, x)$, to be computed by a graph, we mean that a single graph can be used to compute $F(p, x)$ for any given value of p. The functions that calculate the common bits and the output values may change, but the graph itself must be the same in each case. The function $bit_F(i, p, x)$ computes bit i of $F(p, x)$.

By *restricted protocol*, we mean a protocol in which one of the parties is restricted to sending a determined subset of the bits in his available inputs. Although this restriction limits slightly the power of the communication model, it enhances the connection between the graph model and circuits if you follow Valiant's line of reasoning.

The proof of Theorem 2.5 is purely constructive. If we have a graph G that computes $F(p, x)$, we generate a restricted protocol for the three-party model that computes a function $bit_F(i, p, x)$ as follows: Alice sends the value of all inputs x_k connected to output y_i in G, Bob sends the values of all the common bits for the given input (p, x), and the referee computes the output using the function associated with y_i (that is, $f_i(r_0(x), \ldots, r_{m-1}(x), \tau(y_i))$), for which he has all the input values.

On the other hand, we can construct a graph that computes a function $F(p, x)$ from a restricted protocol that computes it as well. Without loss of generality, assume Alice is the restricted party, sending only bits that appear in the input x. There is an edge (x_k, y_i) for all x_k such that Alice sends x_k on input i. Now assign functions to the common bits so that they take on the values transmitted by Bob for all inputs (p, x).

Both constructions introduce bijections between Alice's communication and the degree of the graph, and Bob's communication and the number of common bits, so clearly these quantities are maintained throughout the construction. We have now shown how to translate complexity of functions in the 3-player communication model to the complexity of related functions in our graph model. In a manner similar to the previous section, we can now relate bounds in one model to bounds in the other.

2.6 Sum-Index Problem, Revisited

Following Theorem 2.5, we can translate the Sum-Index function into a function of two variables that is computed by a graph. Instead of $f(i, j, x) = x_{i+j}$, we have a function that shifts the array x by j bits: shift(j, x). If we let y represent the array that the function outputs, then $y_i = x_{i+j}$.

Now that we have a function to compute, we can translate our protocol for the Sum-Index problem into a structure for a graph that computes the shift function. The corresponding graph G that computes shift(j, x)

has $\lfloor n/2 \rfloor$ common bits and degree 1. Each output bit y_i is adjacent to input bit x_{-i}, which corresponds to the bit sent by Alice. Common bit r_i computes the function $x_i \oplus x_{j-i}$. And output bit l computes the function $x_{-i} \oplus r_{-i}$. The graph in Figure 2.2 illustrates this type of graph structure in the simple 4-bit case, and could compute the other 3 shifts, also with 2 common bits. This general structure gives us an upper bound on the number of common bits required to compute all shifts in a degree 1 graph.

From the time that Valiant introduced the model of the graph and common bits until Pudlak, Rodl, and Sgall proved otherwise, several researchers, including Valiant, believed that super-linear lower bounds for $O(n)$ depth circuits could be established by proving that all n shift functions cannot be computed by a graph with an $o(n)$ limit on the degree and common bits. It turns out that this approach to finding an explicit, hard function would not work out, since in 1997, Pudlak, Rodl, and Sgall derived a surprising and complex graph structure with simultaneous bounds of $O(n \log \log n / \log n)$ on both the degree and number of common bits. Their construction of such a graph (and concurrently a protocol for the Sum-Index function) follows similar principles to the graph defined above, in that elements are grouped and the parity of each group is computed as a single common bit, while all but one of each group's values are connected to specific outputs. The groupings are generated this way using properties of arithmetic progressions, and for a graph of degree $O(n/\log n)$, it turns out that $O(n \log \log n / \log n)$ common bits are required to compute all n cyclic shifts. The bound does not imply this model will be unable to prove the existence of explicitly hard functions. Since the $O(n \log \log n / \log n)$ degree of the graph is still significantly larger than the $O(n^{\epsilon})$ lower bound we would need for such a proof, it instead suggests that we look at graphs (or protocols) in which the numbers of common bits and edges differ considerably (or the parties send messages of disparate size). It is these types of graphs and protocols that are relevant to Valiant's circuit reduction, and we examine them in the next chapter.

Now that we have introduced the graph model, we have a framework to prove a much stronger lower bound than the general bound of $\Omega(\sqrt{n})$ presented above.

Chapter 3

A Lower Bound for Shifts

After seeing that a degree 1 graph does, in fact, significantly decrease the necessary number of common bits in computing shift permutations, it is reasonable to suspect we may be able to reduce the graph complexity further. However, we cannot.

Theorem 3.1 (Main Theorem) *For a graph of degree 1 to realize shift(p, x) for all n values of p, G must have at least $\lfloor n/2 \rfloor$ common bits.*

In this proof, we show that there is a tight bound on the number of common bits needed to compute shift functions in degree 1 graphs. This implies a similar tight bound on the communication complexity of the restricted Sum-Index function.

The following definitions will be helpful in proving our theorem.

Definition 3.2 *A* permutation *is a bijection that maps a set onto itself. Let S_n represent the set of all permutations of the set $\{0, \ldots, n-1\}$. We define a special class of permutations, s_p to be shift permutations, such that $s_p(i) = i + p$.*

Permutations are an important concept in numerous sections of this proof. The shift functions introduced in the previous chapter are simply permutations of the input. Of course, shift permutations are only a subset of all permutations, and graphs can compute families of permutations other than shifts. However, shifts provide simpler analysis, and unless it is found that shifts are *not* hard functions to build circuits for, shifts seem to be a better tool with which to analyze our problem.

Definition 3.3 *Given a function $F : A \to A$, a* cycle *is a subset $C \subseteq A$, where $C = \{c_1, \ldots c_m\}$, and $F(c_1) = c_2, F(c_2) = c_3, \ldots, F(c_m) = c_1$. The* length *of the cycle is the number of steps in the path of the cycle, m.*

As we shall see, cycles and their relationship with permutations allow us to convert seemingly complex and unwieldy associations between paths of information in a graph into much cleaner combinatorial problems that facilitate simpler counting arguments.

Now we begin our proof. Since we want to determine the minimum number of common bits necessary to compute a function, we must argue that there are at least a certain number of different common bit strings that will be generated by a specific graph for a specific function (in our case, a shift). For any graph with fewer than n common bits, there must be multiple input values that map to the same values of the common bits. In the communication model, this is equivalent to Bob sending fewer than n bits, which implies that

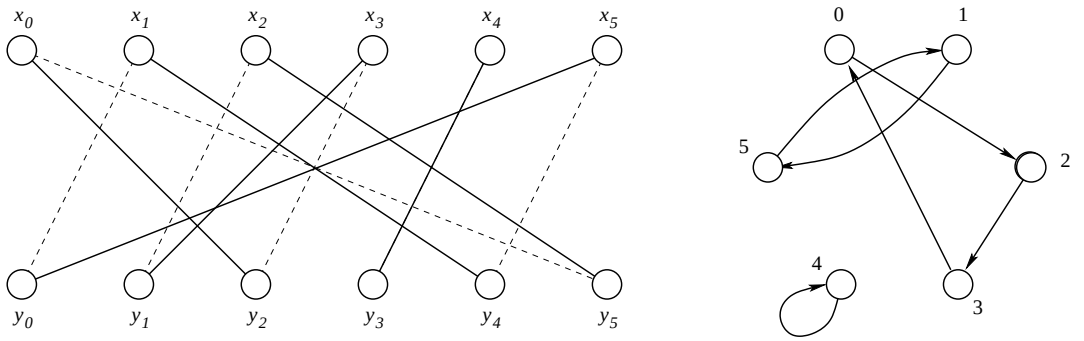

Figure 3.1: The consequence graph corresponding to a specific graph and a shift by one bit.

he sends the same message for more than one value of the input array. To further simplify the analysis of inputs that *share* an assignment of the common bits, we introduce the following construction:

Definition 3.4 *Given a graph G and a permutation $\pi \in S_n$ that the graph is to compute, define the* consequence graph *G_π to be the composition of the permutation and G's τ function. That is, G_π is a graph with vertices labeled $\{0, \ldots, n-1\}$ and there is an edge (a, b) in G_π iff $x_b \in \tau(y_{\pi(a)})$.*

Our goal is to find the maximum number of input strings that can share an assignment of the common bits. The consequence graph's utility in approaching this goal stems from one important property, expressed in Lemma 3.5. This property will allow us to give a combinatorial argument about the number of common bits required to realize a specific permutation. In essence, the consequence graph helps us significantly by directly representing the connections between inputs that share common bit assignments.

Figure 3.1 shows an arbitrary graph G, overlayed with a shift permutation. We want to determine how many common bits are required to compute these types of functions on a graph. On the right is the consequence graph determined by G and the 2-bit shift. Let's examine what happens if we want two inputs x and x' to share an assignment of the common bits, if x_0 is flipped between them ($x_0 \neq x'_0$). Since the graph is computing a one-bit shift, a change in x_0 must be reflected by a change in y_5. Since the only non-common bit input to y_5 is x_2, x_2 must also change to allow this ($x_2 \neq x'_2$). Similarly, if x_2 is to change, y_1 must reflect this change, so x_3 must change. To reflect these chains of consequence, we add edges between the corresponding nodes in the consequence graph. Lemma 3.5 formalizes this characteristic of consequence graphs.

Lemma 3.5 *Let x and x' be two different input vectors to a graph G that realizes permutation π, and further suppose x and x' result in the same values of the common bits. Then, if $x_i \neq x'_i$ there is a j such that $x_j \neq x'_j$ where $j \in \{b | (i, b)$ is an edge in $G_\pi\}$.*

The proof of this lemma is based on an attribute of our graph, namely that each output bit is a function of the common bits and a specific subset of input bits. So for an output bit to change when the common bits do not, at least one of the bits in that subset must change.

Proof: Suppose x differs from x' at bit i, but both x and x' generate the same assignment to the common bits, r. Then, for G to realize the permutation π, the output bit $y_{\pi(i)}$ must change (to $y_{\pi(i)}$, let's say). Since $y_{\pi(i)} \neq y'_{\pi(i)}$, we have that $f_{\pi(i)}(r, \tau(y_{\pi(i)})) \neq f_{\pi(i)}(r, \tau(y_{\pi(i)}))$. Recall that $\tau(y_{\pi(i)}$ represents the values of the input nodes connected to $y_{\pi(i)}$, and since this is the only variable in the inequality that is not exactly

the same on both sides, the value of one of these bits must change. And these bits are exactly the bits corresponding to the neighborhood of i in G_π.

In the following lemmas, we first consider a graph G which is a perfect matching of the inputs and outputs, before proving the general statement. That is, we assume the edges of G form a bijection between the input bits and output bits. Of all degree 1 graphs, we expect the perfect matchings to be the most powerful at computation, since any graph that is not a matching has at least one input bit without any out-edges. Input bits without out-edges have no direct connection to the outputs and must transmit all their information through the common bits. So we expect non-matching graphs to require more common bits than matching graphs.

A well-known property of permutations, including the shift permutations we have explored, is that they divide a set into disjoint cycles. When a matching graph is computing a permutation of the input bits, the consequence graph will represent the composition of two permutations, and will therefore be a permutation itself. Since each node in a cycle can be reached by following a path from any other node in the cycle, a minor extension of Lemma 3.5 implies that all bits corresponding to nodes in a cycle must flip if two inputs are to share assignments of the common bits. For a given cycle, it appears that only two input substrings can share common bit assignments out of the 2^l possible input substrings (where the length of the cycle and corresponding input substring have length l). This would imply that at least $2^l/2 = 2^{l-1}$ common bit assignments are needed for each cycle, an intuition that is formalized in the following Lemma.

Lemma 3.6 *For a degree 1 graph G to realize a given permutation π, the number of common bits must be greater than or equal to the difference $n - c(G_\pi)$, where $c(G_\pi)$ is the number of connected components in the corresponding consequence graph.*

There are 2^n possible input vectors, so in order to have fewer than n common bits, some of the inputs must result in the same common bits. We need to determine which of these vectors can generate identical common bits to find a lower bound.

Proof: For a degree 1 graph G, the graph G_π has out-degree 1 at all vertices. So for an input x_i to change, the input corresponding to the vertex j in edge (i, j) must also change, by Lemma 3.5. The consequence graph for the general degree 1 case takes the form of disjoint connected components, such that paths from any node in a connected component arrive in that connected component's cycle (every path must end in a cycle, since all nodes have out-degree 1). In fact, by following the edges in G_π, each input bit in an entire cycle must flip if any one bit in a connected component does, in order to leave the common bits unchanged. Let $z_1, \ldots, z_m$ be a set of vertices in G_π, which are each in separate cycles (and therefore in separate connected components). Take x and x' to be assignments of the input bits such that the bits corresponding to vertices $z_1, \ldots, z_m$, are the same for both assignments. Since no bit in any of the connected components containing the m vectors $z_1, \ldots, z_m$ can change without a change in the common bits, $x = x'$. Thus, for any assignment of these bits, there is a one-to-one mapping from the other $n - m$ input bits to the common bits. Therefore, G must have at least $n - m$ common bits.

Lemma 3.7 *Given a permutation $\sigma \in S_n$, there exists a $p < n$, such that the composition of the two permutations $s_p \circ \sigma$ (call this permutation σ_p) has at most $\frac{n+1}{2}$ cycles. More generally, the same is true given a function $\kappa : \{1, \ldots, n\} \to \{1, \ldots, n\}$, and the composition $s_p \circ \kappa$ (call this function κ_p).*

Proof: Let c_p be the number of cycles in a permutation σ_p. Now, suppose that every shift of the original permutation does result in a large number of cycles, in this case $c_p > \frac{n+1}{2}$. Since each permutation σ_p takes

on n values, the total length of the cycles is limited to n. In fact, fewer than $\frac{n-1}{2}$ of the cycles can have a length greater than 1, leaving at least two of the cycles as *loops*, i.e. cycles of length 1. A cycle is a loop at i when $\sigma_p(i) = i$. That is, there is one loop at index i for exactly one shift, when $p = i - \pi(i)$. This would imply that there are n total loops over all shifts, but this contradicts the earlier conclusion that each of the n shifts has at least 2 loops. Therefore, there must be at least one shift such that $c_p \leq \frac{n+1}{2}$.

Similarly, the more general case is true for the same reasons. There is a loop at each vertex for exactly one shift, so n total loops. Also, as above, if there were more than $\frac{n-1}{2}$ connected components of more than 1 vertex, these would account for $n-1$ vertices, leaving only one vertex to comprise the final 2 or more cycles. Thus the lemma holds in this case as well.

Theorem 3.8 (Restatement of Theorem 3.1) *For a graph of degree 1 to realize all n shift permutations, G must have at least $\lfloor n/2 \rfloor$ common bits.*

Proof of Theorem 3.1: First we prove the case where the edges of G comprise a permutation. Take a graph G that realizes all cyclic shifts. For a shift by p, the consequence graph G_π will have a single edge out from each vertex: $(a, \tau(y_{a+p}))$. Let $\sigma_p(a) = \tau(y_{a+p})$. Then, σ_p is a permutation, since we are assuming the edges of G comprise a permutation, and so there is only one edge out of each vertex in X. Applying Lemma 3.7, we know that there is a p such that σ_p has no more than $\frac{n+1}{2}$ cycles. Then, applying Lemma 3.6, to realize this shift requires $n - \frac{n+1}{2} = \frac{n-1}{2} = \lfloor \frac{n}{2} \rfloor$.

Since both Lemma 3.6 and Lemma 3.7 hold for all graphs of degree 1, not only those whose edges form a permutation, the proof of the Theorem proceeds in the same way for graphs of that type. Simply replacing the permutation $\sigma_p(a)$ with $\kappa_p(a)$ gives us a function describing the edges of a general consequence graph that requires $n - c(G_\pi)$ common bits, and Lemma 3.7 proves the existence of a shift such that $c(G_\pi) \leq \frac{n+1}{2}$, and the same result.

We have now proved that the bound on common bits in the degree 1 graph is tight. In fact, it turns out that only the graphs defined in 2.6 and their offset counterparts are able to compute all shifts using only $\lfloor n/2 \rfloor$ common bits. Although this proof does not introduce new or more powerful protocols or graph structures, we have determined the precise amount of complexity that n edges provide in computing shifts. The communication complexity corollary to this theorem shows us the exact amount that a single bit from one player improves the communication complexity in the restricted Sum-Index problem.

Chapter 4

Conclusions and Open Problems

The most obvious candidate for future work is the application of the consequence graph structure to graphs with higher degree. It may seem likely that extending the structure of the consequence graph to analyze such graphs would lead quickly to more lower bounds. However, our use of the consequence graph was driven by the fact that for a given input bit, Lemma 3.5 gave us a *single* bit to which it was strongly tied. In higher degree graphs, this relationship breaks down, as there may be multiple edges out of a single node in the consequence graph. As a result, it is harder to determine the relationship between the structure of the consequence graph and the number of common bits. In a graph G of degree 1, there is at least one common bit for every two nodes not in loops in G_π, but this attribute surprisingly breaks down in graphs with higher degree as well. If a technique is developed to make use of consequence graphs of higher degree, then more lower bounds should be determined for the graph model, as well as for the corresponding shift circuits and Sum-Index communication game.

The existence of a protocol with sub-linear cost for the Sum-Index function shows us that linear bounds on the common bits will not hold for graphs with some non-constant degree. This result is cause to wonder whether shift functions are truly *hard*, in the circuit complexity sense; however, a linear bound for common bits in specific graphs is a step towards finding a similarly strong lower bound for higher degree graphs. Although several questions are implicit in the earlier chapters, two open problems seem most relevant and important to the topics we have covered:

- How restrictive is our communication model? In the general model, in which no party is required to send a determined subset of the input array, is it possible to match any of the restricted bounds?
- Are there circuits with size $O(n)$ and depth $O(\log n)$ which compute the shift function?

An interesting side note is that the Sum-Index function and the non-linear upper bound proved by Pudlak, Rodl, Sgall have been used to justify new methods in *Private Information Retrieval*. Private Information Retrieval deals with a situation in which a person wants to retrieve large sets of information from a database without allowing database administrators or other people with access to determine what queries he is performing and what data has been retrieved. In an extensive review of the subject, Chor, Goldreich, Kushilevitz, and Sudan, examine the problem of confidential retrieval of information from distributed databases (multiple servers), and cite the results of PRS as the first sign that reducing the communication costs is possible[1]. As these researchers look to improve upper bounds on the complexity of the Sum-Index function, we hope the techniques introduced in this paper are a framework for finding strong lower bounds.

Bibliography

[1] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In Proceedings of 36th FOCS, 1995.

[2] E. Kushilevitz and N. Nisan. Communication Complexity. Cambridge University Press, 1997.

[3] N. Nisan, and A. Wigderson, Rounds in Communication Complexity Revisited, SIAM J. Computing, Vol. 22, No. 1, pp. 211-219, 1993.

[4] C. Papadimitriou, Computational Complexity, Addison-Wesley (1995).

[5] P. Pudlak and V. Rodl, Modified ranks of tensors and the size of circuits, in Proc. ACM STOC, 1993, pp.523-531.

[6] P. Pudlak, V. Rodl, and J. Sgall, 1997. Boolean circuits, tensor ranks, and communication complexity. SICOMP 26, 605 633.

[7] Ran Raz: Circuit Complexity and Communication Complexity, lecture notes series from IAS summer school on Complexity Theory. Park City Mathematical Series, Volume 3, 1993.

[8] L. G. Valiant, Graph Theoretic Arguments in Low-Level Complexity, Technical Report CS 13-77, University of Edinburgh, 1977.

[9] L. G. Valiant, Why is Boolean Complexity Theory Difficult, Boolean Function Complexity ed. M.S. Paterson, Cambridge Univ. Press 1992, 84-94.