Boolean Circuit Depth

Yijia Chen
Fudan University

1. Compared to Turing machines, circuits have much simpler structures.

1. Compared to Turing machines, circuits have much simpler structures. Thus, some of the best lower bounds we can prove so far are in circuit complexity.

1. Compared to Turing machines, circuits have much simpler structures. Thus, some of the best lower bounds we can prove so far are in circuit complexity. Circuits are in some sense more powerful than Turing machines, so their lower bounds are also lower bounds for Turing machines.

1. Compared to Turing machines, circuits have much simpler structures. Thus, some of the best lower bounds we can prove so far are in circuit complexity. Circuits are in some sense more powerful than Turing machines, so their lower bounds are also lower bounds for Turing machines.

2. There is a tight connection between the circuit complexity of a function and the communication complexity of a corresponding relation, which is the main topic of this chapter.

1. Compared to Turing machines, circuits have much simpler structures. Thus, some of the best lower bounds we can prove so far are in circuit complexity. Circuits are in some sense more powerful than Turing machines, so their lower bounds are also lower bounds for Turing machines.

2. There is a tight connection between the circuit complexity of a function and the communication complexity of a corresponding relation, which is the main topic of this chapter.

3. I'm not an expert in communication complexity, so please ask questions and correct mistakes.

Plan

Plan

1. Establish the correspondence.

Plan

1. Establish the correspondence.
2. Show a few examples.

Introduction

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

1. inputs with in-degree 0, each labelled by either z_i or $\bar{z}_i$,

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

1. inputs with in-degree 0, each labelled by either z_i or $\bar{z}_i$,
2. gates with in-degree 2, each labelled by a Boolean operation, either $\vee$ or $\wedge$.

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

1. inputs with in-degree 0, each labelled by either z_i or $\bar{z}_i$,
2. gates with in-degree 2, each labelled by a Boolean operation, either $\vee$ or $\wedge$.

There is a single node with out-degree 0 that is called the output node.

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

1. inputs with in-degree 0, each labelled by either z_i or $\bar{z}_i$,
2. gates with in-degree 2, each labelled by a Boolean operation, either $\vee$ or $\wedge$.

There is a single node with out-degree 0 that is called the output node.

A monotone circuit is a circuit in which all input nodes are labelled by variables (and none is labelled by a negated variable $\bar{z}_i$).

Definition

Let $z_1, \ldots, z_n$ be a set of variables. A Boolean circuit on $z_1, \ldots, z_n$ is a directed acyclic graph with two types of nodes:

1. inputs with in-degree 0, each labelled by either z_i or $\bar{z}_i$,
2. gates with in-degree 2, each labelled by a Boolean operation, either $\vee$ or $\wedge$.

There is a single node with out-degree 0 that is called the output node.

A monotone circuit is a circuit in which all input nodes are labelled by variables (and none is labelled by a negated variable $\bar{z}_i$).

A circuit in which each node has out-degree 1 (except for the output node) is called a formula. (Note that we allow many input nodes to have the same label).

Definition

The function computed by a Boolean circuit is defined inductively:

Definition

The function computed by a Boolean circuit is defined inductively:

1. the function computed by an input node is $g(z_1, \ldots, z_n) = z_i$ if the node is labelled by z_i and $g(z_1, \ldots, z_n) = \bar{z}_i$ if the node is labelled by $\bar{z}_i$.

Definition

The function computed by a Boolean circuit is defined inductively:

1. the function computed by an input node is $g(z_1, \ldots, z_n) = z_i$ if the node is labelled by z_i and $g(z_1, \ldots, z_n) = \bar{z}_i$ if the node is labelled by $\bar{z}_i$.

2. If one of the two nodes entering the gate computes the function $g_1(z_1, \ldots, z_n)$ and the other node computes the function $g_2(z_1, \ldots, z_n)$ the the gate computes the function

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n)$$

if the gate is labelled $\vee$ and it computes

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \wedge g_2(z_1, \ldots, z_n)$$

if it is labelled $\wedge$.

Definition

The function computed by a Boolean circuit is defined inductively:

1. the function computed by an input node is $g(z_1, \ldots, z_n) = z_i$ if the node is labelled by z_i and $g(z_1, \ldots, z_n) = \bar{z}_i$ if the node is labelled by $\bar{z}_i$.

2. If one of the two nodes entering the gate computes the function $g_1(z_1, \ldots, z_n)$ and the other node computes the function $g_2(z_1, \ldots, z_n)$ the the gate computes the function

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n)$$

   if the gate is labelled $\vee$ and it computes

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \wedge g_2(z_1, \ldots, z_n)$$

   if it is labelled $\wedge$.

For every Boolean function $f : \{0,1\}^n \to \{0,1\}$ there is a circuit or even formula computing f, but possibly of huge size.

Definition

For $x, y \in \{0,1\}^n$ we say that $x \leq y$ if $x_i \leq y_i$ for all $i \in [n]$. A function $f : \{0,1\}^n \to \{0,1\}$ is monotone if $x \leq y$ implies $f(x) \leq f(y)$.

Definition

For $x, y \in \{0,1\}^n$ we say that $x \leq y$ if $x_i \leq y_i$ for all $i \in [n]$. A function $f : \{0,1\}^n \to \{0,1\}$ is monotone if $x \leq y$ implies $f(x) \leq f(y)$.

Lemma

1. *The function computed by a monotone circuit is monotone.*

Definition

For $x, y \in \{0,1\}^n$ we say that $x \leq y$ if $x_i \leq y_i$ for all $i \in [n]$. A function $f : \{0,1\}^n \to \{0,1\}$ is monotone if $x \leq y$ implies $f(x) \leq f(y)$.

Lemma

1. *The function computed by a monotone circuit is monotone.*
2. *For every monotone function there is a monotone circuit computing it.*

Definition

The depth $d(C)$ of a circuit C is the length of the longest path from the output node to an input node. The size $L(F)$ of a formula F is the number of its input nodes.

Definition

The depth $d(C)$ of a circuit C is the length of the longest path from the output node to an input node. The size $L(F)$ of a formula F is the number of its input nodes.

For a function f, the depth complexity $d(f)$ is the minimum depth of a circuit computing f and the size complexity $L(f)$ is the minimum size of a formula computing f.

Definition

The depth $d(C)$ of a circuit C is the length of the longest path from the output node to an input node. The size $L(F)$ of a formula F is the number of its input nodes.

For a function f, the depth complexity $d(f)$ is the minimum depth of a circuit computing f and the size complexity $L(f)$ is the minimum size of a formula computing f.

The measure $d_m(C)$, $L_m(F)$, $d_m(f)$, and $L_m(f)$ are defined similarly for monotone circuits, formulas, and functions respectively.

The Connection to Communication Complexity

Definition

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ let

$$X = f^{-1}(1) \quad \text{and} \quad Y = f^{-1}(0).$$

Definition

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ let

$$X = f^{-1}(1) \quad \text{and} \quad Y = f^{-1}(0).$$

We define

$$R_f = \big\{(x,y,i) \;\big|\; x \in X,\ y \in Y, \text{ and } i \in \{1,\dots,n\} \text{ with } x_i \neq y_i\big\}.$$

Definition

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ let

$$X = f^{-1}(1) \quad \text{and} \quad Y = f^{-1}(0).$$

We define

$$R_f = \big\{(x,y,i) \;\big|\; x \in X,\ y \in Y, \text{ and } i \in \{1,\dots,n\} \text{ with } x_i \neq y_i\big\}.$$

For monotone f we also define

$$M_f = \big\{(x,y,i) \;\big|\; x \in X,\ y \in Y, \text{ and } i \in \{1,\dots,n\} \text{ with } x_i = 1 \text{ and } y_i = 0\big\}.$$

Lemma

For every circuit C for f there is a corresponding protocol $\mathcal{P}$ for R_f in which at most $d(C)$ bits are exchanged.

Proof (1)

Proof (1)

Alice and Bob traverse the nodes of the circuit C, starting from the output node and continuing towards the input nodes, while maintaining the following invariant on the function g computed by the current node

$$g(x) = 1 \quad \text{and} \quad g(y) = 0.$$

Proof (1)

Alice and Bob traverse the nodes of the circuit C, starting from the output node and continuing towards the input nodes, while maintaining the following invariant on the function g computed by the current node

$$g(x) = 1 \quad \text{and} \quad g(y) = 0.$$

The invariant if trivially true for the output node.

Proof (2)

Proof (2)

Suppose that the current node is a $\vee$-gate, and let g_1 and g_2 be the functions corresponding to the nodes entering the current node. Then

$$g(z_1, \dots, z_n) = g_1(z_1, \dots, z_n) \vee g_2(z_1, \dots, z_n).$$

Proof (2)

Suppose that the current node is a $\vee$-gate, and let g_1 and g_2 be the functions corresponding to the nodes entering the current node. Then

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n).$$

By the invariant $g(y) = 0$ and $g(x) = 1$ we have

$$\begin{aligned} & g_1(y) = g_2(y) = 0, \\ \text{and} \quad & \big(\text{either } g_1(x) = 1 \text{ or } g_2(x) = 1\big). \end{aligned}$$

Proof (2)

Suppose that the current node is a $\vee$-gate, and let g_1 and g_2 be the functions corresponding to the nodes entering the current node. Then

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n).$$

By the invariant $g(y) = 0$ and $g(x) = 1$ we have

$$\begin{aligned} & g_1(y) = g_2(y) = 0, \\ \text{and} \qquad & \big(\text{either } g_1(x) = 1 \text{ or } g_2(x) = 1\big). \end{aligned}$$

Then Alice who knows x sends a single bit indicating for which $i \in \{1, 2\}$ we have $g_i(x) = 1$. So Alice and Bob can move to the same g_i and maintain the invariant.

Proof (2)

Suppose that the current node is a $\vee$-gate, and let g_1 and g_2 be the functions corresponding to the nodes entering the current node. Then

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n).$$

By the invariant $g(y) = 0$ and $g(x) = 1$ we have

$$g_1(y) = g_2(y) = 0,$$
$$\text{and} \qquad \big(\text{either } g_1(x) = 1 \text{ or } g_2(x) = 1\big).$$

Then Alice who knows x sends a single bit indicating for which $i \in \{1, 2\}$ we have $g_i(x) = 1$. So Alice and Bob can move to the same g_i and maintain the invariant.

The case for a $\wedge$-gate is symmetric.

Proof (3)

Proof (3)

Finally, when the players reach an input node, labelled by either z_i or $\bar{z}_i$. Then they both know that i is an appropriate output, i.e., $(x, y, i) \in R_f$. □

Lemma

For every protocol $\mathcal{P}$ for R_f there is a corresponding circuit C for f such that $d(C)$ is at most the communication complexity of $\mathcal{P}$.

Proof (1)

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

1. Each internal node in which Alice sends a bit is labelled by $\vee$.

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

1. Each internal node in which Alice sends a bit is labelled by $\vee$.

2. Each internal node in which Bob sends a bit is labelled by $\wedge$.

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

1. Each internal node in which Alice sends a bit is labelled by $\vee$.
2. Each internal node in which Bob sends a bit is labelled by $\wedge$.
3. Each leaf of the tree is a monochromatic rectangle $A \times B$ with whom an output i is associated.

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

1. Each internal node in which Alice sends a bit is labelled by $\vee$.
2. Each internal node in which Bob sends a bit is labelled by $\wedge$.
3. Each leaf of the tree is a monochromatic rectangle $A \times B$ with whom an output i is associated. We claim
   3.1 either $x_i = 1$ for all $x \in A$ and $y_i = 0$ for all $y \in B$ in which case this leaf is labelled by z_i;

Proof (1)

We convert the protocol tree for $\mathcal{P}$ to a circuit as follows.

1. Each internal node in which Alice sends a bit is labelled by $\vee$.
2. Each internal node in which Bob sends a bit is labelled by $\wedge$.
3. Each leaf of the tree is a monochromatic rectangle $A \times B$ with whom an output i is associated. We claim
   3.1 either $x_i = 1$ for all $x \in A$ and $y_i = 0$ for all $y \in B$ in which case this leaf is labelled by z_i;
   3.2 or $x_i = 0$ for all $x \in A$ and $y_i = 1$ for all $y \in B$ in which case this leaf is labelled by $\bar{z}_i$.

Proof of the claim

Proof of the claim

Take any $x \in A$ and let $\sigma = x_i$.

Proof of the claim

Take any $x \in A$ and let $\sigma = x_i$. Because for all $y \in B$ the value i is a legal output on (x, y), we conclude $y_i = \bar{\sigma}$ for all $y \in B$.

Proof of the claim

Take any $x \in A$ and let $\sigma = x_i$. Because for all $y \in B$ the value i is a legal output on (x, y), we conclude $y_i = \bar{\sigma}$ for all $y \in B$. This in turn implies that $x_i = \sigma$ for all $x \in A$.

Proof (3)

Proof (3)

The depth of the circuit equals the depth of the protocol tree, i.e., the communication complexity of $\mathcal{P}$.

Proof (3)

The depth of the circuit equals the depth of the protocol tree, i.e., the communication complexity of $\mathcal{P}$.

We prove that the circuit computes f by showing

Proof (3)

The depth of the circuit equals the depth of the protocol tree, i.e., the communication complexity of $\mathcal{P}$.

We prove that the circuit computes f by showing

for every node of the circuit, the function g corresponding to that node satisfies $g(z) = 1$ for all $z \in A$ and $g(z) = 0$ for all $z \in B$, where $A \times B$ are the inputs that reach the corresponding node of the protocol.

Proof (3)

The depth of the circuit equals the depth of the protocol tree, i.e., the communication complexity of $\mathcal{P}$.

We prove that the circuit computes f by showing

for every node of the circuit, the function g corresponding to that node satisfies $g(z) = 1$ for all $z \in A$ and $g(z) = 0$ for all $z \in B$, where $A \times B$ are the inputs that reach the corresponding node of the protocol.

The claim is proved by induction starting from the input nodes towards the output node.

Proof (3)

The depth of the circuit equals the depth of the protocol tree, i.e., the communication complexity of $\mathcal{P}$.

We prove that the circuit computes f by showing

for every node of the circuit, the function g corresponding to that node satisfies $g(z) = 1$ for all $z \in A$ and $g(z) = 0$ for all $z \in B$, where $A \times B$ are the inputs that reach the corresponding node of the protocol.

The claim is proved by induction starting from the input nodes towards the output node.

It is true in the input nodes by our construction and the claim.

Proof (4)

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree.

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node.

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node. Her bit partitions A into A_1 and A_2.

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node. Her bit partitions A into A_1 and A_2.

By the induction hypothesis,

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node. Her bit partitions A into A_1 and A_2.

By the induction hypothesis,

1. $g_1(x) = 1$ for all $x \in A_1$ and $g_1(y) = 0$ for all $y \in B$;

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node. Her bit partitions A into A_1 and A_2.

By the induction hypothesis,

1. $g_1(x) = 1$ for all $x \in A_1$ and $g_1(y) = 0$ for all $y \in B$;
2. $g_2(x) = 1$ for all $x \in A_2$ and $g_2(y) = 0$ for all $y \in B$;.

Proof (4)

Now consider an internal node computing a function g such that the claim was already proved for its two children (computing the functions g_1 and g_2).

Let $A \times B$ be the inputs reaching this node in the protocol tree. Assume, without loss of generality, that Alice sends a bit in this node. Her bit partitions A into A_1 and A_2.

By the induction hypothesis,

1. $g_1(x) = 1$ for all $x \in A_1$ and $g_1(y) = 0$ for all $y \in B$;
2. $g_2(x) = 1$ for all $x \in A_2$ and $g_2(y) = 0$ for all $y \in B$;.

So our construction

$$g(z_1, \ldots, z_n) = g_1(z_1, \ldots, z_n) \vee g_2(z_1, \ldots, z_n)$$

satisfies $g(y) = 0$ for all $y \in B$ and $g(x) = 1$ for all $x \in A = A_1 \cup A_2$. $\square$

Theorem

$$d(f) = D(R_f) \quad and \quad L(f) = C^P(R_f).$$

Theorem

$$d(f) = D(R_f) \quad \textit{and} \quad L(f) = C^P(R_f).$$

Theorem

$$d_m(f) = D(CM_f) \quad \textit{and} \quad L_m(f) = C^P(M_f).$$

Matching and ST-Connectivity

Matching

Matching

A matching in a graph $G = (V, E)$ is a set of edges such that no pair of them has a common vertex.

Matching

A matching in a graph $G = (V, E)$ is a set of edges such that no pair of them has a common vertex.

Given a graph G on n vertices, represented by $n' = \binom{n}{2}$ Boolean variables (each indicating whether a certain edge (i, j) appears in the graph or not).

$$\textsc{Match}(G) = \begin{cases} 1, & \text{if there is a matching of size } \geq n/3 \text{ in } G, \\ 0, & \text{otherwise.} \end{cases}$$

Matching

A matching in a graph $G = (V, E)$ is a set of edges such that no pair of them has a common vertex.

Given a graph G on n vertices, represented by $n' = \binom{n}{2}$ Boolean variables (each indicating whether a certain edge (i, j) appears in the graph or not).

$$\text{MATCH}(G) = \begin{cases} 1, & \text{if there is a matching of size } \geq n/3 \text{ in } G, \\ 0, & \text{otherwise.} \end{cases}$$

MATCH is monotone.

Matching

A matching in a graph $G = (V, E)$ is a set of edges such that no pair of them has a common vertex.

Given a graph G on n vertices, represented by $n' = \binom{n}{2}$ Boolean variables (each indicating whether a certain edge (i, j) appears in the graph or not).

$$\text{MATCH}(G) = \begin{cases} 1, & \text{if there is a matching of size} \geq n/3 \text{ in } G, \\ 0, & \text{otherwise.} \end{cases}$$

MATCH is monotone.

With our loss of generality we assume $n = 3m$ for some $m \in \mathbb{N}$.

M_{MATCH}

M_{MATCH}

The relation M_{MATCH} is defined by:

M_{MATCH}

The relation M_{MATCH} is defined by:

1. X is the set of all graphs of $n = 3m$ vertices with a matching of size m.

M_{MATCH}

The relation M_{MATCH} is defined by:

1. X is the set of all graphs of $n = 3m$ vertices with a matching of size m.
2. Y is the set of all graphs of $n = 3m$ vertices without such a matching.

M_{MATCH}

The relation M_{MATCH} is defined by:

1. X is the set of all graphs of $n = 3m$ vertices with a matching of size m.
2. Y is the set of all graphs of $n = 3m$ vertices without such a matching.
3. Alice is given $x \in X$ and Bob $y \in Y$, and they have to find an edge that is in x but not in y,

M_{MATCH}

The relation M_{MATCH} is defined by:

1. X is the set of all graphs of $n = 3m$ vertices with a matching of size m.
2. Y is the set of all graphs of $n = 3m$ vertices without such a matching.
3. Alice is given $x \in X$ and Bob $y \in Y$, and they have to find an edge that is in x but not in y, or equivalently an index i such that $x_i = 1$ and $y_i = 0$.

M'

M'

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M' be the restriction of M_{MATCH} to $X' \times Y'$.

M'

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M' be the restriction of M_{MATCH} to $X' \times Y'$. Clearly

$$D(M') \le D(M_{\text{MATCH}}).$$

M'

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M' be the restriction of M_{MATCH} to $X' \times Y'$. Clearly

$$D(M') \leq D(M_{\text{MATCH}}).$$

1. X' is the set of graphs on n vertices that are matchings of size m.

M'

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M' be the restriction of M_{MATCH} to $X' \times Y'$. Clearly

$$D(M') \leq D(M_{\text{MATCH}}).$$

1. X' is the set of graphs on n vertices that are matchings of size m.
2. Y' is the set of graphs in which the vertices are partitions into two sets S of size $m-1$ and T of size $2m+1$, and the edges are all the pairs in which at least one vertex is in S.

The pair-disjointness relation M

The pair-disjointness relation M

1. $n = 3m$.

The pair-disjointness relation M

1. $n = 3m$.
2. X consists of all ordered sets P of m pairs of $\{1, \ldots, n\}$, where the $2m$ elements in P are pairwise distinct.

The pair-disjointness relation M

1. $n = 3m$.
2. X consists of all ordered sets P of m pairs of $\{1, \ldots, n\}$, where the $2m$ elements in P are pairwise distinct.
3. Y consists of all sets S of $m - 1$ elements of $\{1, \ldots, n\}$.

The pair-disjointness relation M

1. $n = 3m$.
2. X consists of all ordered sets P of m pairs of $\{1, \ldots, n\}$, where the $2m$ elements in P are pairwise distinct.
3. Y consists of all sets S of $m - 1$ elements of $\{1, \ldots, n\}$.

Then we let

$$M = \{(P, S, i) \mid P \in X,\ S \in Y, \text{ and the } i\text{-th pair in } P \text{ contains no element of } S\}.$$

The pair-disjointness relation M

1. $n = 3m$.
2. X consists of all ordered sets P of m pairs of $\{1, \ldots, n\}$, where the $2m$ elements in P are pairwise distinct.
3. Y consists of all sets S of $m - 1$ elements of $\{1, \ldots, n\}$.

Then we let

$$M = \Big\{(P, S, i) \;\Big|\; P \in X,\ S \in Y, \text{ and the } i\text{-th pair in } P \text{ contains no element of } S\Big\}.$$

Theorem
$D(M) = \Omega(m)$.

From M' to the pair-disjointness M

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.

2. Bob, given a set S of $m - 1$ elements in $\{1, \ldots, 3m\}$, transforms it into a graph $y \in Y'$ corresponding to this set S.

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.
2. Bob, given a set S of $m - 1$ elements in $\{1, \ldots, 3m\}$, transforms it into a graph $y \in Y'$ corresponding to this set S.
3. Hence the protocol for M' will output a pair of P that contains no elements of S.

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.

2. Bob, given a set S of $m - 1$ elements in $\{1, \ldots, 3m\}$, transforms it into a graph $y \in Y'$ corresponding to this set S.

3. Hence the protocol for M' will output a pair of P that contains no elements of S.

4. Finally, Alice sends the index of this pair in the list P using $\log m$ bits.

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.

2. Bob, given a set S of $m - 1$ elements in $\{1, \ldots, 3m\}$, transforms it into a graph $y \in Y'$ corresponding to this set S.

3. Hence the protocol for M' will output a pair of P that contains no elements of S.

4. Finally, Alice sends the index of this pair in the list P using $\log m$ bits.

Hence we get $D(M) \leq D(M') + \log m$, and recall $D(M) = \Omega(m)$.

From M' to the pair-disjointness M

1. Alice, given a list P of m mutually disjoint pairs of elements in $\{1, \ldots, 3m\}$, transforms it into a matching of size m in a graph with $n = 3m$ vertices, hence obtains a graph $x \in X'$.
2. Bob, given a set S of $m - 1$ elements in $\{1, \ldots, 3m\}$, transforms it into a graph $y \in Y'$ corresponding to this set S.
3. Hence the protocol for M' will output a pair of P that contains no elements of S.
4. Finally, Alice sends the index of this pair in the list P using $\log m$ bits.

Hence we get $D(M) \leq D(M') + \log m$, and recall $D(M) = \Omega(m)$. Altogether

$$d_m(\mathrm{MATCH}) = D(M_{\mathrm{MATCH}}) \geq D(M') \geq D(M) - \log m = \Omega(m) = \Omega(n).$$

STCON

The s-t-connectivity function STCON is defined as follows:

The s-t-connectivity function STCON is defined as follows: Given a directed graph G on n nodes,

$$\textsc{stcon}(G) = \begin{cases} 1, & \text{if there is a path in } G \text{ from vertex 1 to vertex } n \\ 0, & \text{otherwise.} \end{cases}$$

M_{STCON}

1. X is the set of all directed graphs G on n vertices with a directed path from vertex 1 to vertex n.

M_{STCON}

1. X is the set of all directed graphs G on n vertices with a directed path from vertex 1 to vertex n.
2. Y is the set of all directed graphs G on n vertices with no directed paths from vertex 1 to vertex n.

1. X is the set of all directed graphs G on n vertices with a directed path from vertex 1 to vertex n.
2. Y is the set of all directed graphs G on n vertices with no directed paths from vertex 1 to vertex n.
3. The task of Alice and Bob is given $x \in X$ and $y \in Y$ to find an edge that appears in x but not in y.

Restricting M_{STCON}

Restricting M_{STCON}

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M be the restriction of M_{STCON} to $X' \times Y'$.

Restricting M_{STCON}

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M be the restriction of M_{STCON} to $X' \times Y'$. Thus

$$D(M) \leq D(M_{\mathrm{STCON}}).$$

Restricting M_{STCON}

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M be the restriction of M_{STCON} to $X' \times Y'$. Thus

$$D(M) \leq D(M_{\mathrm{STCON}}).$$

The domains X' and Y' are obtained by restricting our attention to layered graphs that consist of $\ell + 2$ layers $0, 1, \ldots, \ell, \ell + 1$ each of them with w vertices with

$$\ell + 2 = w = \sqrt{n}.$$

Restricting M_{STCON}

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M be the restriction of M_{STCON} to $X' \times Y'$. Thus

$$D(M) \leq D(M_{\text{STCON}}).$$

The domains X' and Y' are obtained by restricting our attention to layered graphs that consist of $\ell + 2$ layers $0, 1, \ldots, \ell, \ell + 1$ each of them with w vertices with

$$\ell + 2 = w = \sqrt{n}.$$

1. Every edge connects a vertex in some layer i and a vertex in the adjacent layer $i + 1$.

Restricting M_{STCON}

We will choose some $X' \subset X$ and $Y' \subset Y$ and let M be the restriction of M_{STCON} to $X' \times Y'$. Thus

$$D(M) \leq D(M_{\text{STCON}}).$$

The domains X' and Y' are obtained by restricting our attention to layered graphs that consist of $\ell + 2$ layers $0, 1, \ldots, \ell, \ell + 1$ each of them with w vertices with

$$\ell + 2 = w = \sqrt{n}.$$

1. Every edge connects a vertex in some layer i and a vertex in the adjacent layer $i + 1$.
2. Vertex 1 belongs to layer 0 and vertex n belongs to layer $\ell + 1$.

Reducing FORK to M

Reducing FORK to M

1. Alice considers her string $a \in \{1, \ldots, w\}^\ell$ as a directed path from vertex 1 to vertex n (this will be her graph x) by choosing from each layer i its a_i-th vertex and connecting them.

Reducing FORK to M

1. Alice considers her string $a \in \{1, \ldots, w\}^\ell$ as a directed path from vertex 1 to vertex n (this will be her graph x) by choosing from each layer i its a_i-th vertex and connecting them.
2. Bob considers his string $b \in \{1, \ldots, w\}^\ell$ as a path p from vertex 1 to another vertex in the last layer (say $n-1$), and construct a graph $y \in Y'$ that contains this path.

Reducing FORK to M

1. Alice considers her string $a \in \{1, \ldots, w\}^\ell$ as a directed path from vertex 1 to vertex n (this will be her graph x) by choosing from each layer i its a_i-th vertex and connecting them.
2. Bob considers his string $b \in \{1, \ldots, w\}^\ell$ as a path p from vertex 1 to another vertex in the last layer (say $n-1$), and construct a graph $y \in Y'$ that contains this path. In addition, edges connecting each vertex not in the path to all the vertices in the next layer.

Reducing FORK to M

1. Alice considers her string $a \in \{1, \ldots, w\}^{\ell}$ as a directed path from vertex 1 to vertex n (this will be her graph x) by choosing from each layer i its a_i-th vertex and connecting them.
2. Bob considers his string $b \in \{1, \ldots, w\}^{\ell}$ as a path p from vertex 1 to another vertex in the last layer (say $n-1$), and construct a graph $y \in Y'$ that contains this path. In addition, edges connecting each vertex not in the path to all the vertices in the next layer.

   Observe that the path corresponding to b does not reach vertex n,

Reducing FORK to M

1. Alice considers her string $a \in \{1, \dots, w\}^\ell$ as a directed path from vertex 1 to vertex n (this will be her graph x) by choosing from each layer i its a_i-th vertex and connecting them.
2. Bob considers his string $b \in \{1, \dots, w\}^\ell$ as a path p from vertex 1 to another vertex in the last layer (say $n - 1$), and construct a graph $y \in Y'$ that contains this path. In addition, edges connecting each vertex not in the path to all the vertices in the next layer.

   Observe that the path corresponding to b does not reach vertex n, and vertex 1 is not connected to vertex n in y.

Reducing FORK to M (cont'd)

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y.

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y. In addition u belongs to layer i and v to layer $i + 1$.

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y. In addition u belongs to layer i and v to layer $i + 1$.
4. (u, v) belongs to the path a.

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y. In addition u belongs to layer i and v to layer $i + 1$.
4. (u, v) belongs to the path a. On the other hand, u belongs to b but v does not.

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y. In addition u belongs to layer i and v to layer $i + 1$.
4. (u, v) belongs to the path a. On the other hand, u belongs to b but v does not. Thus i is a legal output for FORK.

Reducing FORK to M (cont'd)

3. Alice and Bob use the protocol for M on x and y to get an output edge (u, v) that appears in x but not in y. In addition u belongs to layer i and v to layer $i + 1$.

4. (u, v) belongs to the path a. On the other hand, u belongs to b but v does not. Thus i is a legal output for FORK.

We conclude

$$d_m(\text{STCON}) = D(M_{\text{STCON}}) \geq D(M) \geq D(\text{FORK}) = \Omega(\log \ell \cdot \log w) = \Omega(\log^2 n).$$

Set Cover

Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ whose deterministic communication complexity $D(g)$ is significantly larger than its nondeterministic communication complexity $N(g)$.

Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ whose deterministic communication complexity $D(g)$ is significantly larger than its nondeterministic communication complexity $N(g)$.

Let $R_1, \ldots, R_t$ be a cover (possibly with intersections) of the matrix M_g corresponding to g with monochromatic rectangles.

Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ whose deterministic communication complexity $D(g)$ is significantly larger than its nondeterministic communication complexity $N(g)$.

Let $R_1, \ldots, R_t$ be a cover (possibly with intersections) of the matrix M_g corresponding to g with monochromatic rectangles. Thus

$$N(g) \leq t.$$

Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ whose deterministic communication complexity $D(g)$ is significantly larger than its nondeterministic communication complexity $N(g)$.

Let $R_1, \ldots, R_t$ be a cover (possibly with intersections) of the matrix M_g corresponding to g with monochromatic rectangles. Thus

$$N(g) \leq t.$$

We define

$$M = \{(x, y, i) \mid x, y \in \{0,1\}^n \text{ and } (x, y) \in R_i\}.$$

M is a total relation,

Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ whose deterministic communication complexity $D(g)$ is significantly larger than its nondeterministic communication complexity $N(g)$.

Let $R_1, \ldots, R_t$ be a cover (possibly with intersections) of the matrix M_g corresponding to g with monochromatic rectangles. Thus

$$N(g) \leq t.$$

We define

$$M = \{(x, y, i) \mid x, y \in \{0,1\}^n \text{ and } (x, y) \in R_i\}.$$

M is a total relation, and

$$D(g) \leq D(M).$$

We construct a function $f : \{0,1\}^t \to \{0,1\}$ such that $D(M_f) \geq D(M)$.

We construct a function $f : \{0,1\}^t \to \{0,1\}$ such that $D(M_f) \geq D(M)$.

$$f(z_1, \ldots, z_t) = \begin{cases} 1, & \text{if there exists a row } x \text{ of } M_g \text{ such that} \\ & \qquad \text{for all } i \text{ we have } (x \in R_i \implies z_i = 1) \\ 0, & \text{otherwise.} \end{cases}$$

We construct a function $f : \{0,1\}^t \to \{0,1\}$ such that $D(M_f) \geq D(M)$.

$$f(z_1, \ldots, z_t) = \begin{cases} 1, & \text{if there exists a row } x \text{ of } M_g \text{ such that} \\ & \qquad \text{for all } i \text{ we have } (x \in R_i \implies z_i = 1) \\ 0, & \text{otherwise.} \end{cases}$$

f is monotone.

Reduction from M to M_f

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise.

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise. So $f(x') = 1$.

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise. So $f(x') = 1$.

2. Bob, given $y \in \{0,1\}^n$, constructs $y' \in \{0,1\}^t$ by assigning $y'_i = 0$ if the column y belongs to R_i and 1 otherwise.

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise. So $f(x') = 1$.

2. Bob, given $y \in \{0,1\}^n$, constructs $y' \in \{0,1\}^t$ by assigning $y'_i = 0$ if the column y belongs to R_i and 1 otherwise. So $f(y') = 0$.

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise. So $f(x') = 1$.

2. Bob, given $y \in \{0,1\}^n$, constructs $y' \in \{0,1\}^t$ by assigning $y'_i = 0$ if the column y belongs to R_i and 1 otherwise. So $f(y') = 0$.

3. Alice and Bob use the protocol for the relation M_f on (x', y') to get an index i with $x'_i = 1$ and $y'_i = 0$. Thus, both x and y intersect R_i, i.e., $(x, y, i) \in M$.

Reduction from M to M_f

1. Alice, given $x \in \{0,1\}^n$, constructs $x' \in \{0,1\}^t$ by assigning $x'_i = 1$ if the the row x belongs to R_i and 0 otherwise. So $f(x') = 1$.
2. Bob, given $y \in \{0,1\}^n$, constructs $y' \in \{0,1\}^t$ by assigning $y'_i = 0$ if the column y belongs to R_i and 1 otherwise. So $f(y') = 0$.
3. Alice and Bob use the protocol for the relation M_f on (x', y') to get an index i with $x'_i = 1$ and $y'_i = 0$. Thus, both x and y intersect R_i, i.e., $(x, y, i) \in M$.

Assume $D(g) = N^2(g)$, then the function f has $t = 2^{N(g)}$ variables and

$$d_m(f) = D(M_f) \geq D(g) = \log^2 t.$$

Similarly $L(f) = \Omega(t^{\log t})$.

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \implies (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \implies (z_t = 1)\big].$$

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \Longrightarrow (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \Longrightarrow (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP,

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \Longrightarrow (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \Longrightarrow (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP, and can be rewritten to a 3-CNF formula

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big),$$

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \Longrightarrow (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \Longrightarrow (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP, and can be rewritten to a 3-CNF formula

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big),$$

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \Longrightarrow (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \Longrightarrow (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP, and can be rewritten to a 3-CNF formula

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big),$$

where

1. $x_{n+1}, \ldots, x_p$ are auxiliary variables,

Note we can write

$$f(z_1, \ldots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \Longrightarrow (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \Longrightarrow (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP, and can be rewritten to a 3-CNF formula

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big),$$

where

1. $x_{n+1}, \ldots, x_p$ are auxiliary variables,
2. each φ_i is a disjunction of 3 literals on the variables $x_1, \ldots, x_p$,

Note we can write

$$f(z_1, \dots, z_t) \equiv \exists x \in \{0,1\}^n : \\ \big[(x \in R_1) \implies (z_1 = 1)\big] \wedge \cdots \wedge \big[(x \in R_t) \implies (z_t = 1)\big].$$

If deciding "$x \in R_i$" can be done in time polynomial in t, then f is a function in NP, and can be rewritten to a 3-CNF formula

$$f(z_1, \dots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big),$$

where

1. $x_{n+1}, \dots, x_p$ are auxiliary variables,
2. each φ_i is a disjunction of 3 literals on the variables $x_1, \dots, x_p$,
3. and both p and s are polynomially bounded in t.

The set-cover problem

The set-cover problem

SET-COVER

Input:	A collection of m sets over a universe of ℓ elements and a number d.
Problem:	Is there a subcollection of d sets that covers the whole universe?

Reduction to the set-cover problem (1)

Reduction to the set-cover problem (1)

Recall

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big).$$

Reduction to the set-cover problem (1)

Recall

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big).$$

1. The universe is of size $s + p$, one element for each φ_i, and one element for each $x_i \vee \bar{x}_i$.

Reduction to the set-cover problem (1)

Recall

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s \big).$$

1. The universe is of size $s + p$, one element for each φ_i, and one element for each $x_i \vee \bar{x}_i$.
2. For every x_i there are two sets $A_{x_i=1}$ and $A_{x_i=0}$. $A_{x_i=1}$ contains all terms in which x_i appears, and $A_{x_i=0}$ contains all terms in which $\bar{x}_i$ appears.

Reduction to the set-cover problem (1)

Recall

$$f(z_1, \ldots, z_t) \equiv \exists x_1 \cdots x_p \big(\varphi_1 \wedge \cdots \wedge \varphi_s\big).$$

1. The universe is of size $s + p$, one element for each φ_i, and one element for each $x_i \vee \bar{x}_i$.
2. For every x_i there are two sets $A_{x_i=1}$ and $A_{x_i=0}$. $A_{x_i=1}$ contains all terms in which x_i appears, and $A_{x_i=0}$ contains all terms in which $\bar{x}_i$ appears.
3. Finally, set $d = p$.

Reduction to the set-cover problem (2)

Reduction to the set-cover problem (2)

If f is 1, then there exists an assignment for $x_1, \ldots, x_p$ that satisfies all the terms. Then the corresponding p sets from a cover.

Reduction to the set-cover problem (2)

If f is 1, then there exists an assignment for $x_1, \ldots, x_p$ that satisfies all the terms. Then the corresponding p sets from a cover.

If there is cover, then for every i at least one of $A_{x_1=1}$ and $A_{x_i=0}$ is in the cover in order to cover the term $x_i \vee \bar{x}_i$.

Reduction to the set-cover problem (2)

If f is 1, then there exists an assignment for $x_1, \ldots, x_p$ that satisfies all the terms. Then the corresponding p sets from a cover.

If there is cover, then for every i at least one of $A_{x_1=1}$ and $A_{x_i=0}$ is in the cover in order to cover the term $x_i \vee \bar{x}_i$. Since the cover is of size p, exactly one of $A_{x_1=1}$ and $A_{x_i=0}$ is in the cover.

Reduction to the set-cover problem (2)

If f is 1, then there exists an assignment for $x_1, \ldots, x_p$ that satisfies all the terms. Then the corresponding p sets from a cover.

If there is cover, then for every i at least one of $A_{x_1=1}$ and $A_{x_i=0}$ is in the cover in order to cover the term $x_i \vee \bar{x}_i$. Since the cover is of size p, exactly one of $A_{x_1=1}$ and $A_{x_i=0}$ is in the cover.

Then the cover induces a satisfying assignment, since the universe contains all the terms.

Reduction to the set-cover problem (3)

Reduction to the set-cover problem (3)

The reduction can be performed in a small depth $O(\log t)$.

Reduction to the set-cover problem (3)

The reduction can be performed in a small depth $O(\log t)$. Hence

$$d_m(\textsc{set-cover}) \geq d(f) - O(\log t) = \Omega(\log^2 t).$$

Monotone Constant-Depth Circuits

Circuits of unbounded fan-in

Circuits of unbounded fan-in

Now $\wedge$- and $\vee$-gates can have unbounded number of inputs.

Circuits of unbounded fan-in

Now $\wedge$- and $\vee$-gates can have unbounded number of inputs. Among others, constant-depth circuits become meaningful.

Circuits of unbounded fan-in

Now $\wedge$- and $\vee$-gates can have unbounded number of inputs. Among others, constant-depth circuits become meaningful.

We can define similarly $d(f)$ and $L(f)$.

Circuits of unbounded fan-in

Now $\wedge$- and $\vee$-gates can have unbounded number of inputs. Among others, constant-depth circuits become meaningful.

We can define similarly $d(f)$ and $L(f)$.

It is still the case that $L(F)$, the size of a formula F, translate to the protocol partition number $C^P(f)$.

Circuits of unbounded fan-in

Now $\wedge$- and $\vee$-gates can have unbounded number of inputs. Among others, constant-depth circuits become meaningful.

We can define similarly $d(f)$ and $L(f)$.

It is still the case that $L(F)$, the size of a formula F, translate to the protocol partition number $C^P(f)$.

However, the depth $d(f)$ is equal to the round complexity of the protocol, the number of alternations between the communication from Alice to Bob and the communication from Bob to Alice.

Depth k vs. depth $k-1$ for monotone circuits

Depth k vs. depth $k-1$ for monotone circuits

We construct a formula $f : \{0,1\}^n \to \{0,1\}$ with $n = m^k$ as follows.

Depth k vs. depth $k-1$ for monotone circuits

We construct a formula $f : \{0,1\}^n \to \{0,1\}$ with $n = m^k$ as follows.

1. f consists of a complete m-ary tree of depth k.

Depth k vs. depth $k-1$ for monotone circuits

We construct a formula $f : \{0,1\}^n \to \{0,1\}$ with $n = m^k$ as follows.

1. f consists of a complete m-ary tree of depth k.
2. Each of its m^k leaves is labelled by a unique variable in $\{x_1, \dots, x_n\}$.

Depth k vs. depth $k-1$ for monotone circuits

We construct a formula $f : \{0,1\}^n \to \{0,1\}$ with $n = m^k$ as follows.

1. f consists of a complete m-ary tree of depth k.
2. Each of its m^k leaves is labelled by a unique variable in $\{x_1, \ldots, x_n\}$.
3. The gates in the odd levels (including the root) are labelled by $\wedge$, and those in the even levels are labelled by $\vee$.

Depth k vs. depth $k-1$ for monotone circuits

We construct a formula $f : \{0,1\}^n \to \{0,1\}$ with $n = m^k$ as follows.

1. f consists of a complete m-ary tree of depth k.
2. Each of its m^k leaves is labelled by a unique variable in $\{x_1, \dots, x_n\}$.
3. The gates in the odd levels (including the root) are labelled by $\wedge$, and those in the even levels are labelled by $\vee$.

We show that any depth $k-1$ formula computing f has size exponential in m.

The tree problem T_k

The tree problem T_k

Consider the complete m-ary tree of depth k.

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \dots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

An input to the tree problem is a labelling of the tree, where

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

An input to the tree problem is a labelling of the tree, where

1. Bob gets as his input the labels of all nodes in the odd levels,

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

An input to the tree problem is a labelling of the tree, where

1. Bob gets as his input the labels of all nodes in the odd levels,
2. and Alice gets her input the labels of all nodes in even level.

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

An input to the tree problem is a labelling of the tree, where

1. Bob gets as his input the labels of all nodes in the odd levels,
2. and Alice gets her input the labels of all nodes in even level.

The goal is to compute the label of the leaf reached by the path induced by the labelling.

The tree problem T_k

Consider the complete m-ary tree of depth k. A labelling of the tree assigns to each leaf a bit, and to each internal node a number in $\{1, \ldots, n\}$.

The labels of the internal nodes define a (unique) path from the root to a leaf, where the label of each internal node is viewed as a pointer to one of its children.

An input to the tree problem is a labelling of the tree, where

1. Bob gets as his input the labels of all nodes in the odd levels,
2. and Alice gets her input the labels of all nodes in even level.

The goal is to compute the label of the leaf reached by the path induced by the labelling.

It is known that the $k-1$-round communication complexity $D^{k-1}(T_k)$ of T_k is

$$D^{k-1}(T_k) = \Omega\big(m/\text{polylog}(m)\big).$$

Reduction from T_k to M_f (1)

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then

     $$S_{i+1} = \left\{ \text{the child of } v \text{ defined by the labelling given to Alice} \;\middle|\; v \in S_i \right\}$$

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then
     $$S_{i+1} = \left\{\text{the child of } v \text{ defined by the labelling given to Alice} \;\middle|\; v \in S_i\right\}$$
   - If i is odd, then
     $$S_{i+1} = \left\{\text{all the children of } v \;\middle|\; v \in S_i\right\}$$

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then
     $$S_{i+1} = \left\{\text{the child of } v \text{ defined by the labelling given to Alice} \;\middle|\; v \in S_i\right\}$$
   - If i is odd, then
     $$S_{i+1} = \left\{\text{all the children of } v \;\middle|\; v \in S_i\right\}$$
2. Bob computes a sequence of sets $Q_1, \ldots, Q_k$ inductively:

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then
     $$S_{i+1} = \left\{ \text{the child of } v \text{ defined by the labelling given to Alice} \;\middle|\; v \in S_i \right\}$$
   - If i is odd, then
     $$S_{i+1} = \left\{ \text{all the children of } v \;\middle|\; v \in S_i \right\}$$
2. Bob computes a sequence of sets $Q_1, \ldots, Q_k$ inductively:
   - Q_1 contains only the root of the tree.

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \ldots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then
     $$S_{i+1} = \left\{\text{the child of } v \text{ defined by the labelling given to Alice} \,\middle|\, v \in S_i\right\}$$
   - If i is odd, then
     $$S_{i+1} = \left\{\text{all the children of } v \,\middle|\, v \in S_i\right\}$$
2. Bob computes a sequence of sets $Q_1, \ldots, Q_k$ inductively:
   - Q_1 contains only the root of the tree.
   - If i is even, then
     $$Q_{i+1} = \left\{\text{all the children of } v \,\middle|\, v \in Q_i\right\}$$

Reduction from T_k to M_f (1)

1. Alice computes a sequence of sets $S_1, \dots, S_k$ inductively:
   - S_1 contains only the root of the tree.
   - If i is even, then
     $$S_{i+1} = \left\{\text{the child of } v \text{ defined by the labelling given to Alice} \,\middle|\, v \in S_i\right\}$$
   - If i is odd, then
     $$S_{i+1} = \left\{\text{all the children of } v \,\middle|\, v \in S_i\right\}$$
2. Bob computes a sequence of sets $Q_1, \dots, Q_k$ inductively:
   - Q_1 contains only the root of the tree.
   - If i is even, then
     $$Q_{i+1} = \left\{\text{all the children of } v \,\middle|\, v \in Q_i\right\}$$
   - If i is odd, then
     $$Q_{i+1} = \left\{\text{the child of } v \text{ defined by the labelling given to Bob} \,\middle|\, v \in Q_i\right\}$$

Reduction from T_k to M_f (2)

Reduction from T_k to M_f (2)

3. Alice computes a string x of length n by putting 1 in all coordinates j for $j \in S_k$ and 0 elsewhere.

Reduction from T_k to M_f (2)

3. Alice computes a string x of length n by putting 1 in all coordinates j for $j \in S_k$ and 0 elsewhere.
4. Bob computes a string y of length n by putting 0 in all coordinates j for $j \in Q_k$ and 1 elsewhere.

Reduction from T_k to M_f (2)

3. Alice computes a string x of length n by putting 1 in all coordinates j for $j \in S_k$ and 0 elsewhere.
4. Bob computes a string y of length n by putting 0 in all coordinates j for $j \in Q_k$ and 1 elsewhere.
5. Finally, Alice and Bob use the protocol for M_f on (x, y) and output the result.

The correctness (1)

The correctness (1)

We first show

$$f(x) = 1 \quad \text{and} \quad f(y) = 0$$

The correctness (1)

We first show

$$f(x) = 1 \quad \text{and} \quad f(y) = 0$$

$f(x) = 1$ By induction on i from $k-1$ to 1, if each node in S_{i+1} computes the value 1, then so do all the nodes in S_i.

The correctness (1)

We first show

$$f(x) = 1 \quad \text{and} \quad f(y) = 0$$

$f(x) = 1$ By induction on i from $k-1$ to 1, if each node in S_{i+1} computes the value 1, then so do all the nodes in S_i.

$f(y) = 0$ By induction on i from $k-1$ to 1 if each node in Q_{i+1} computes the value 0, then so do all the nodes in Q_i.

The correctness (2)

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \ldots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \ldots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

- It is trivially true for $i = 1$, i.e., $S_1 = Q_1 = \{\mathrm{root}\}$.

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \dots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

- It is trivially true for $i = 1$, i.e., $S_1 = Q_1 = \{\mathrm{root}\}$.
- If i is odd, then we put all the children of S_i to S_{i+1}, and only those defined by the labelling to Q_{i+1}.

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \ldots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

- It is trivially true for $i = 1$, i.e., $S_1 = Q_1 = \{\mathrm{root}\}$.
- If i is odd, then we put all the children of S_i to S_{i+1}, and only those defined by the labelling to Q_{i+1}. Since $v_i \in S_i \cap Q_i$, then the next node v_{i+1} on the path is in $S_{i+1} \cap Q_{i+1}$.

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \ldots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

- It is trivially true for $i = 1$, i.e., $S_1 = Q_1 = \{\mathrm{root}\}$.
- If i is odd, then we put all the children of S_i to S_{i+1}, and only those defined by the labelling to Q_{i+1}. Since $v_i \in S_i \cap Q_i$, then the next node v_{i+1} on the path is in $S_{i+1} \cap Q_{i+1}$. Conversely, if $v \in S_{i+1} \cap Q_{i+1}$, then its father is in $S_i \cap Q_i = \{v_i\}$. Thus, $v = v_{i+1}$.

The correctness (2)

Finally, we prove that there is exactly one j with $x_j = 1$ and $y_j = 0$ by showing that for every $i \in \{1, \dots, k\}$ the set $S_i \cap Q_i$ includes a single node v_i, which is the node in level i that the path from the root reaches.

- It is trivially true for $i = 1$, i.e., $S_1 = Q_1 = \{\mathrm{root}\}$.
- If i is odd, then we put all the children of S_i to S_{i+1}, and only those defined by the labelling to Q_{i+1}. Since $v_i \in S_i \cap Q_i$, then the next node v_{i+1} on the path is in $S_{i+1} \cap Q_{i+1}$. Conversely, if $v \in S_{i+1} \cap Q_{i+1}$, then its father is in $S_i \cap Q_i = \{v_i\}$. Thus, $v = v_{i+1}$.
- The case for even i is symmetric.

The lower bound

We conclude for any constant k, the size of any depth $k-1$ formula for f is

$$C^{P,k-1}(M_f) = \Omega\left(2^{D^{k-1}(M_f)/(k-1)}\right) = \Omega\left(2^{D^{k-1}(T_f)/(k-1)}\right) = \Omega\left(2^{m/\text{polylog}(m)}\right).$$

Thank You!