Decision Trees

Lecture 23

To left or to right

Decision Trees

Decision Trees

- A different complexity measure

Decision Trees

- A different complexity measure
  - Number of bits of input read

Decision Trees

- A different complexity measure
  - Number of bits of input read
    - For simpler problems

Decision Trees

- A different complexity measure
  - Number of bits of input read
    - For simpler problems
  - Interested in lower-bounds

Decision Trees

- A different complexity measure
  - Number of bits of input read
    - For simpler problems
  - Interested in lower-bounds
    - So even allow unbounded computational power

Decision Trees

- A different complexity measure
  - Number of bits of input read
    - For simpler problems
  - Interested in lower-bounds
    - So even allow unbounded computational power
    - Simpler combinatorial structure (need not understand P vs. NP etc.)

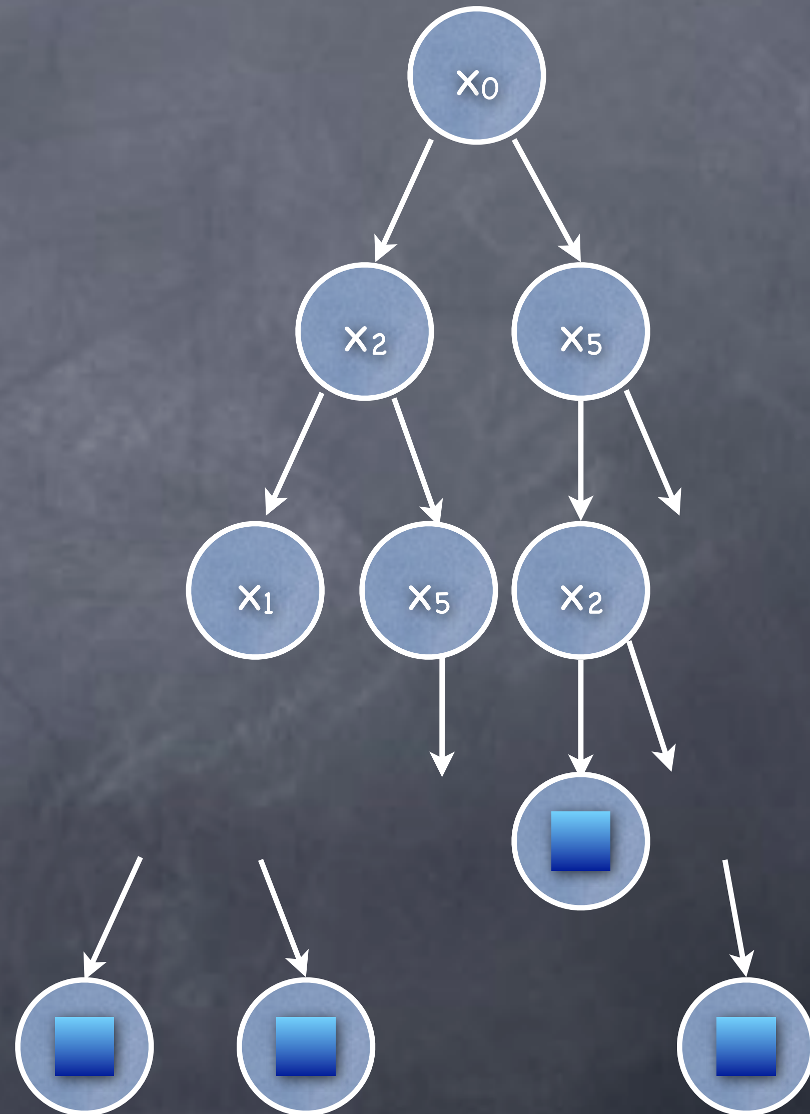
Decision Trees

Decision Trees

- Configuration graph of a computation, as it reads each bit

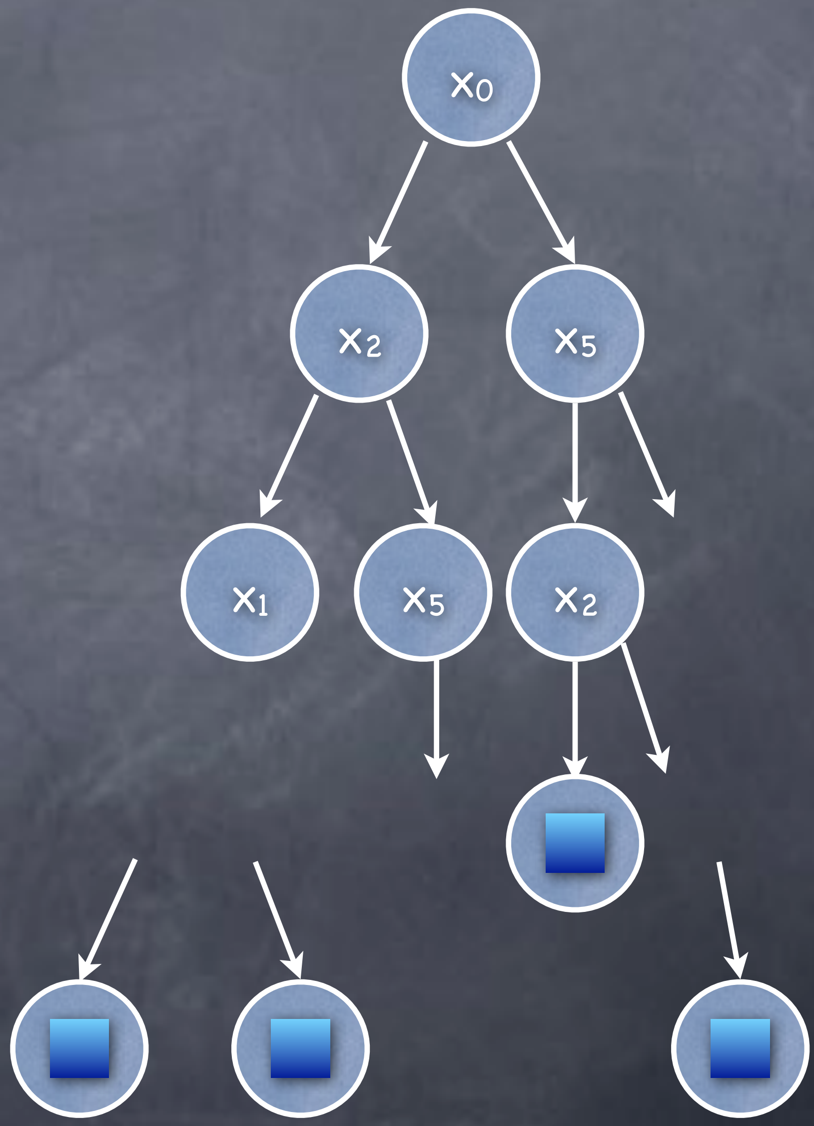
Decision Trees

- Configuration graph of a computation, as it reads each bit



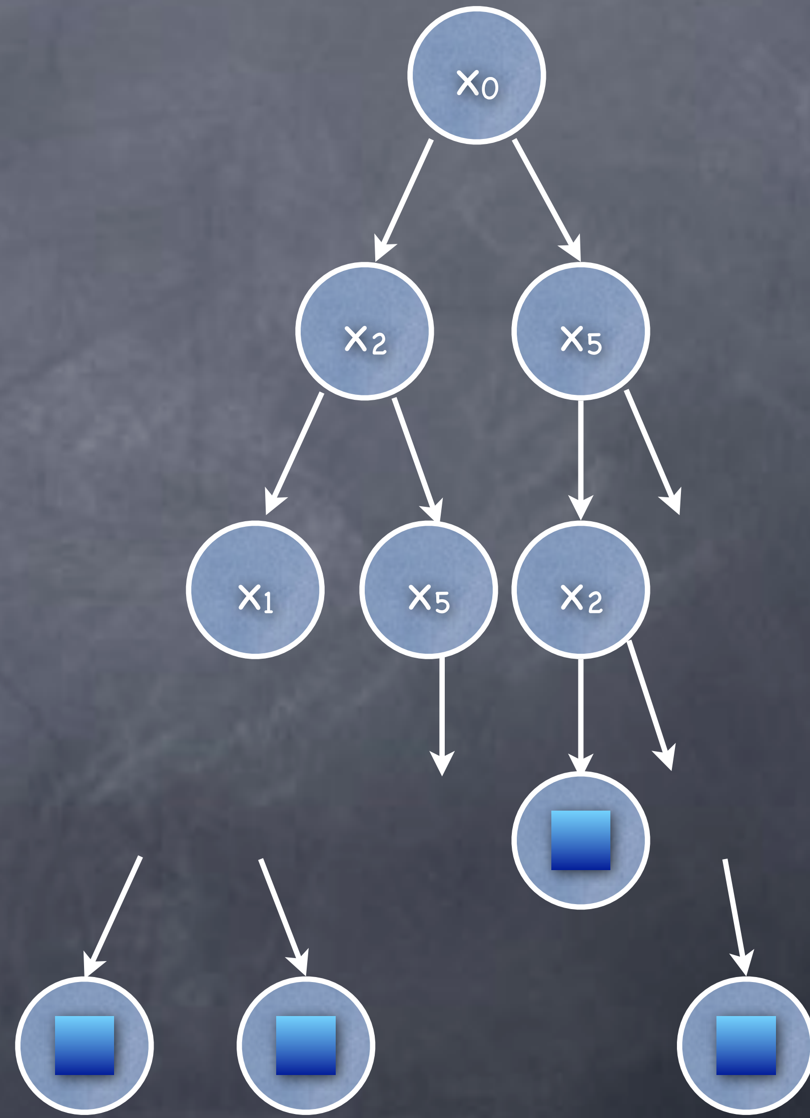
Decision Trees

- Configuration graph of a computation, as it reads each bit
  - For n-bit input, depth at most n



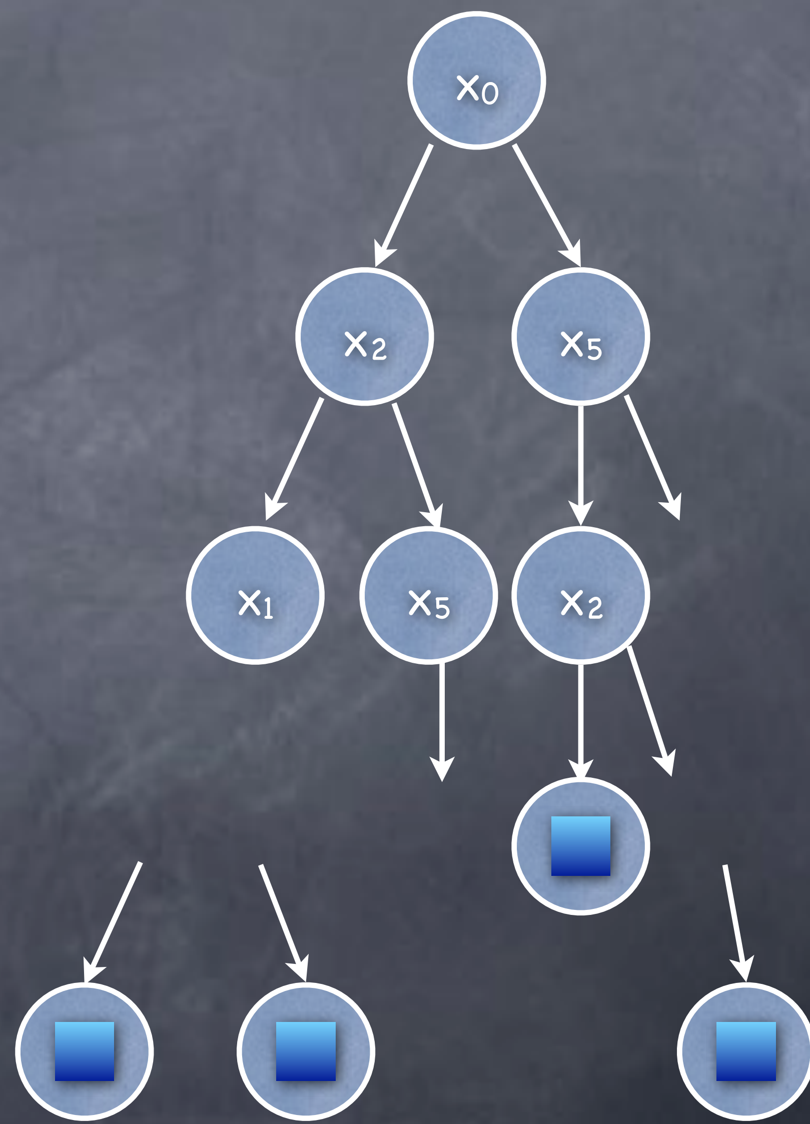
Decision Trees

- Configuration graph of a computation, as it reads each bit
  - For n-bit input, depth at most n
  - Some paths may be shorter



Decision Trees

- Configuration graph of a computation, as it reads each bit
  - For n-bit input, depth at most n
  - Some paths may be shorter
- DTree(L) = $\min_{\text{alg } A} \max_{\text{input } x} T_{A,x}$ where $T_{A,x}$ is the number of bits of x read by A



Examples

Examples

- Simpler problems

Examples

- Simpler problems
  - OR(x)=1 if at least one bit of x is 1

Examples

- Simpler problems
  - OR(x)=1 if at least one bit of x is 1
  - PARITY(x)=1 if odd number of bits of x are 1

Examples

- Simpler problems
  - OR(x)=1 if at least one bit of x is 1
  - PARITY(x)=1 if odd number of bits of x are 1
  - $SAT_C(x)$ if x is a satisfying assignment for circuit (or circuit family) C

Examples

- Simpler problems
  - OR(x)=1 if at least one bit of x is 1
  - PARITY(x)=1 if odd number of bits of x are 1
  - $SAT_C(x)$ if x is a satisfying assignment for circuit (or circuit family) C
  - CONNECTED(G) = 1 if G is the adjacency matrix of a connected graph

Examples

- Simpler problems
  - OR(x)=1 if at least one bit of x is 1
  - PARITY(x)=1 if odd number of bits of x are 1
  - $SAT_C(x)$ if x is a satisfying assignment for circuit (or circuit family) C
  - CONNECTED(G) = 1 if G is the adjacency matrix of a connected graph
- We are interested in showing DTree lower-bounds for these problems

Adversary Argument

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs
  - e.g.: DTree(OR) = n (i.e., any correct decision tree will need to read all bits in the worst case)

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs
  - e.g.: DTree(OR) = n (i.e., any correct decision tree will need to read all bits in the worst case)
    - Given any decision tree: Start with all inputs

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs
  - e.g.: DTree(OR) = n (i.e., any correct decision tree will need to read all bits in the worst case)
    - Given any decision tree: Start with all inputs
    - At first node restrict to inputs which answer 0, and consider the tree's behavior on such inputs

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs
  - e.g.: DTree(OR) = n (i.e., any correct decision tree will need to read all bits in the worst case)
    - Given any decision tree: Start with all inputs
    - At first node restrict to inputs which answer 0, and consider the tree's behavior on such inputs
    - On second node, further restrict to inputs which answer 0

Adversary Argument

- Identifying one input which will cause a shallow decision tree to go wrong: Given a decision tree find inputs which lead it to the same leaf but must have different outputs
  - e.g.: DTree(OR) = n (i.e., any correct decision tree will need to read all bits in the worst case)
    - Given any decision tree: Start with all inputs
    - At first node restrict to inputs which answer 0, and consider the tree's behavior on such inputs
    - On second node, further restrict to inputs which answer 0
    - Before n nodes, set of inputs contain 0^n and another input, no matter what bits where queried at the nodes

Graph Connectivity

Graph Connectivity

- DTree(CONNECTED) = $n(n-1)/2$ (i.e., all possible edges)

Graph Connectivity

- DTree(CONNECTED) = n(n-1)/2 (i.e., all possible edges)
  - If possible, answer "No," but maintain the invariant that edges answered "Yes" plus unqueried edges form a connected graph.

Graph Connectivity

- DTree(CONNECTED) = $n(n-1)/2$ (i.e., all possible edges)
  - If possible, answer "No," but maintain the invariant that edges answered "Yes" plus unqueried edges form a connected graph.
  - Yes edges by themselves connect the entire graph only if set of unqueried edges is empty

Graph Connectivity

- DTree(CONNECTED) = n(n-1)/2 (i.e., all possible edges)
  - If possible, answer "No," but maintain the invariant that edges answered "Yes" plus unqueried edges form a connected graph.
  - Yes edges by themselves connect the entire graph only if set of unqueried edges is empty
    - Otherwise some Yes edge was unforced: consider the cycle formed by an unqueried edge and the connected Yes graph

Graph Connectivity

- DTree(CONNECTED) = n(n-1)/2 (i.e., all possible edges)
  - If possible, answer "No," but maintain the invariant that edges answered "Yes" plus unqueried edges form a connected graph.
  - Yes edges by themselves connect the entire graph only if set of unqueried edges is empty
    - Otherwise some Yes edge was unforced: consider the cycle formed by an unqueried edge and the connected Yes graph
  - Until then, graph can be connected or disconnected: by setting all unqueried edges to Yes or all to No

Elusive Languages

Elusive Languages

- Languages which require the decision tree to read all the bits in the worst case

Elusive Languages

- Languages which require the decision tree to read all the bits in the worst case
  - e.g.: OR, PARITY, CONNECTED

Elusive Languages

- Languages which require the decision tree to read all the bits in the worst case
  - e.g.: OR, PARITY, CONNECTED
    - Argued using adversary strategies

Elusive Languages

- Languages which require the decision tree to read all the bits in the worst case
  - e.g.: OR, PARITY, CONNECTED
    - Argued using adversary strategies
  - Maj(x) = 1 iff #1s in x > #0s (assume |x| odd)

Elusive Languages

- Languages which require the decision tree to read all the bits in the worst case
  - e.g.: OR, PARITY, CONNECTED
    - Argued using adversary strategies
  - $\mathrm{Maj}(x) = 1$ iff #1s in x > #0s (assume $|x|$ odd)
    - Adversary strategy: alternately answer 0 and 1

Monotonic Tree Circuits

Monotonic Tree Circuits

- Tree of AND gates and OR gates (monotonic)

Monotonic Tree Circuits

- Tree of AND gates and OR gates (monotonic)
- Each variable (leaf) used only once

Monotonic Tree Circuits

- Tree of AND gates and OR gates (monotonic)
- Each variable (leaf) used only once
- Is elusive

Monotonic Tree Circuits

- Tree of AND gates and OR gates (monotonic)
- Each variable (leaf) used only once
- Is elusive
  - Answer so that each gate kept undetermined until all its leaf-descendants are queried

Monotonic Tree Circuits

- Tree of AND gates and OR gates (monotonic)
- Each variable (leaf) used only once
- Is elusive
  - Answer so that each gate kept undetermined until all its leaf-descendants are queried
    - Exercise

Certificate Complexity

Certificate Complexity

- 1-certificate

Certificate Complexity

- 1-certificate
  - For x s.t. $L(x)=1$, a subset of the bits of x which proves that $L(x)=1$: c s.t. $x|c \Rightarrow x \in L$ (i.e., no x' s.t. $L(x')=0$ and has the same values at those positions)

Certificate Complexity

- 1-certificate
  - For x s.t. $L(x)=1$, a subset of the bits of x which proves that $L(x)=1$: c s.t. $x|c \Rightarrow x \in L$ (i.e., no x' s.t. $L(x')=0$ and has the same values at those positions)
- 0-certificate: similarly for $x \notin L$, c s.t. $x|c \Rightarrow x \notin L$

Certificate Complexity

- 1-certificate
  - For x s.t. L(x)=1, a subset of the bits of x which proves that L(x)=1 : c s.t. $x|c \Rightarrow x \in L$ (i.e., no x′ s.t. L(x′)=0 and has the same values at those positions)
- 0-certificate: similarly for $x \notin L$, c s.t. $x|c \Rightarrow x \notin L$
- Can be much lower than DTree(L) because for different x's different sets of bits can be used

Certificate Complexity

- 1-certificate
  - For x s.t. $L(x)=1$, a subset of the bits of x which proves that $L(x)=1$: c s.t. $x|c \Rightarrow x \in L$ (i.e., no x' s.t. $L(x')=0$ and has the same values at those positions)
- 0-certificate: similarly for $x \notin L$, c s.t. $x|c \Rightarrow x \notin L$
- Can be much lower than DTree(L) because for different x's different sets of bits can be used
  - Produced by someone who has seen all bits of x

Certificate Complexity

- 1-certificate
  - For x s.t. L(x)=1, a subset of the bits of x which proves that L(x)=1 : c s.t. $x|c \Rightarrow x \in L$ (i.e., no x' s.t. L(x')=0 and has the same values at those positions)
- 0-certificate: similarly for $x \notin L$, c s.t. $x|c \Rightarrow x \notin L$
- Can be much lower than DTree(L) because for different x's different sets of bits can be used
  - Produced by someone who has seen all bits of x
- 1-Cert(L): $\max_{x \in L} \min_{c:\ x|c \Rightarrow x \in L} |c|$ (e.g. 1-Cert(OR) = 1)

Certificate Complexity

- 1-certificate
  - For x s.t. $L(x)=1$, a subset of the bits of x which proves that $L(x)=1$: c s.t. $x|c \Rightarrow x \in L$ (i.e., no x' s.t. $L(x')=0$ and has the same values at those positions)
- 0-certificate: similarly for $x \notin L$, c s.t. $x|c \Rightarrow x \notin L$
- Can be much lower than DTree(L) because for different x's different sets of bits can be used
  - Produced by someone who has seen all bits of x
- 1-Cert(L): $\max_{x \in L} \min_{c:\ x|c \Rightarrow x \in L} |c|$ (e.g. 1-Cert(OR) = 1)
- 0-Cert(L): $\max_{x \notin L} \min_{c:\ x|c \Rightarrow x \notin L} |c|$ (e.g. 0-Cert(OR) = n)

$$\mathrm{DTree}(L) \leq \mathrm{0Cert}(L) \times \mathrm{1Cert}(L)$$

DTree(L) $\leq$ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)
  - While both pools non-empty

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)
  - While both pools non-empty
    - Pick a 0-certificate, and query all (remaining) bits in it

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)
  - While both pools non-empty
    - Pick a 0-certificate, and query all (remaining) bits in it
    - If a good 0-certificate, terminate with 0. Else, remove all 0 and 1 certificates inconsistent with the bits revealed

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)
  - While both pools non-empty
    - Pick a 0-certificate, and query all (remaining) bits in it
    - If a good 0-certificate, terminate with 0. Else, remove all 0 and 1 certificates inconsistent with the bits revealed
  - One pool must be non-empty. Output the corresponding answer

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- A Decision tree algorithm
  - Start with a pool of all 0-certificates and all 1-certificates (for various x)
  - While both pools non-empty
    - Pick a 0-certificate, and query all (remaining) bits in it
    - If a good 0-certificate, terminate with 0. Else, remove all 0 and 1 certificates inconsistent with the bits revealed
  - One pool must be non-empty. Output the corresponding answer
  - Clearly correct. Number of bits read?

$$\mathrm{DTree}(L) \leq \mathrm{0Cert}(L) \times \mathrm{1Cert}(L)$$

$DTree(L) \leq 0Cert(L) \times 1Cert(L)$

- An undetermined 0-certificate has at least one unrevealed conflicting bit with each undetermined 1-certificate

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- An undetermined 0-certificate has at least one unrevealed conflicting bit with each undetermined 1-certificate
  - Otherwise it is possible to have an x consistent with both those certificates!

$DTree(L) \leq 0Cert(L) \times 1Cert(L)$

- An undetermined 0-certificate has at least one unrevealed **conflicting bit** with each undetermined 1-certificate
  - Otherwise it is possible to have an x consistent with both those certificates!
- Picking such a 0-certificate and querying reduces number of unrevealed bits of each remaining 1-certificate by at least 1

$DTree(L) \leq 0Cert(L) \times 1Cert(L)$

- An undetermined 0-certificate has at least one unrevealed conflicting bit with each undetermined 1-certificate
    - Otherwise it is possible to have an x consistent with both those certificates!
- Picking such a 0-certificate and querying reduces number of unrevealed bits of each remaining 1-certificate by at least 1
    - Initially at most 1Cert(L) bits in each 1-certificate

DTree(L) $\leq$ 0Cert(L) x 1Cert(L)

- An undetermined 0-certificate has at least one unrevealed conflicting bit with each undetermined 1-certificate
  - Otherwise it is possible to have an x consistent with both those certificates!
- Picking such a 0-certificate and querying reduces number of unrevealed bits of each remaining 1-certificate by at least 1
  - Initially at most 1Cert(L) bits in each 1-certificate
    - So at most 1Cert(L) iterations

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- An undetermined 0-certificate has at least one unrevealed conflicting bit with each undetermined 1-certificate
  - Otherwise it is possible to have an x consistent with both those certificates!
- Picking such a 0-certificate and querying reduces number of unrevealed bits of each remaining 1-certificate by at least 1
  - Initially at most 1Cert(L) bits in each 1-certificate
    - So at most 1Cert(L) iterations
  - In each iteration at most 0Cert(L) bits queried

$$\text{DTree}(L) \leq \text{0Cert}(L) \times \text{1Cert}(L)$$

DTree(L) $\leq$ 0Cert(L) x 1Cert(L)

- Example: AND-OR trees

DTree(L) $\leq$ 0Cert(L) x 1Cert(L)

- Example: AND-OR trees
  - 0-certificate: enough variables so that can evaluate just one input wire for AND gates, and all input wires for OR gates

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- Example: AND-OR trees
  - 0-certificate: enough variables so that can evaluate just one input wire for AND gates, and all input wires for OR gates
  - 1-certificate: enough variables so that can evaluate just one input wire for OR gates, and all input wires for AND gates

DTree(L) ≤ 0Cert(L) x 1Cert(L)

- Example: AND-OR trees
  - 0-certificate: enough variables so that can evaluate just one input wire for AND gates, and all input wires for OR gates
  - 1-certificate: enough variables so that can evaluate just one input wire for OR gates, and all input wires for AND gates
  - If regular AND-OR tree, 0Cert(L) x 1Cert(L) = number of leaves = DTree(L)

Studying DTree(L)

Studying DTree(L)

- Various techniques

Studying DTree(L)

- Various techniques
  - Arithmetization: write the boolean function for L as a multi-linear polynomial of n boolean variables. Then degree is a lower-bound on DTree(L)

Studying DTree(L)

- Various techniques
  - Arithmetization: write the boolean function for L as a multi-linear polynomial of n boolean variables. Then degree is a lower-bound on DTree(L)
  - Topological criterion for monotone functions: construct a simplicial complex corresponding to the monotone boolean function. If the simplicial complex "not collapsible" then DTree(L)=n

Studying DTree(L)

- Various techniques
  - Arithmetization: write the boolean function for L as a multi-linear polynomial of n boolean variables. Then degree is a lower-bound on DTree(L)
  - Topological criterion for monotone functions: construct a simplicial complex corresponding to the monotone boolean function. If the simplicial complex "not collapsible" then DTree(L)=n
  - "Sensitivity" is a lower-bound on DTree(L)

Studying DTree(L)

- Various techniques
  - Arithmetization: write the boolean function for L as a multi-linear polynomial of n boolean variables. Then degree is a lower-bound on DTree(L)
  - Topological criterion for monotone functions: construct a simplicial complex corresponding to the monotone boolean function. If the simplicial complex "not collapsible" then DTree(L)=n
  - "Sensitivity" is a lower-bound on DTree(L)
- Will explore some in exercises

Randomized Decision Trees

Randomized Decision Trees

- Recall two views of randomized computation

Randomized Decision Trees

- Recall two views of randomized computation
  - Randomly decide (based on fresh coin flips, and queries and answers so far) what variable to query

Randomized Decision Trees

- Recall two views of randomized computation
  - Randomly decide (based on fresh coin flips, and queries and answers so far) what variable to query
  - Flip all coins up front and then run a deterministic computation

Randomized Decision Trees

- Recall two views of randomized computation
  - Randomly decide (based on fresh coin flips, and queries and answers so far) what variable to query
  - Flip all coins up front and then run a deterministic computation
    - i.e., randomly choose a (deterministic) decision tree

Randomized Decision Trees

Randomized Decision Trees

- Complexity measure

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs
  - Note: No error allowed (Las Vegas)

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs
  - Note: No error allowed (Las Vegas)
- Random decision tree chosen independent of the (adversarial) input. i.e., input chosen "before" the random choice

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs
  - Note: No error allowed (Las Vegas)
- Random decision tree chosen independent of the (adversarial) input. i.e., input chosen "before" the random choice
  - Gets more power over the "adversary"

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs
  - Note: No error allowed (Las Vegas)
- Random decision tree chosen independent of the (adversarial) input. i.e., input chosen "before" the random choice
  - Gets more power over the "adversary"
    - Adversary can't find a single pair of inputs that force many reads for all random choices

Randomized Decision Trees

- Complexity measure
  - Expected number of bits read, max over all inputs
  - Note: No error allowed (Las Vegas)
- Random decision tree chosen independent of the (adversarial) input. i.e., input chosen "before" the random choice
  - Gets more power over the "adversary"
    - Adversary can't find a single pair of inputs that force many reads for all random choices
- Question: How to prove lower-bounds against randomization?

Yao's Min-Max

Yao's Min-Max

- Interested in expected cost (running time)

Yao's Min-Max

- Interested in expected cost (running time)

	(Deterministic) Algorithms			
Input s				
		$T_{A,x}$		

Yao's Min-Max

- Interested in expected cost (running time)

a randomized algorithm

	0.125	0.25	0.5	0.125
	(Deterministic) Algorithms			
Inputs		$T_{A,x}$		

Yao's Min-Max

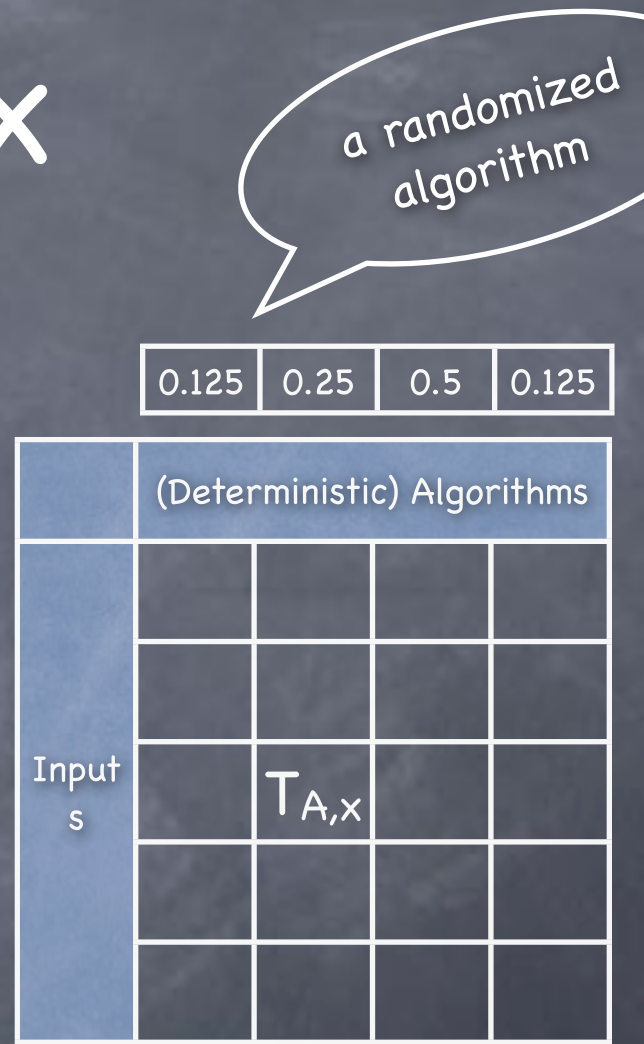
- Interested in expected cost (running time)
- Standard setting: Pick your randomized algorithm R; input x given adversarially

a randomized algorithm

	0.125	0.25	0.5	0.125
	(Deterministic) Algorithms			
Inputs		$T_{A,x}$		

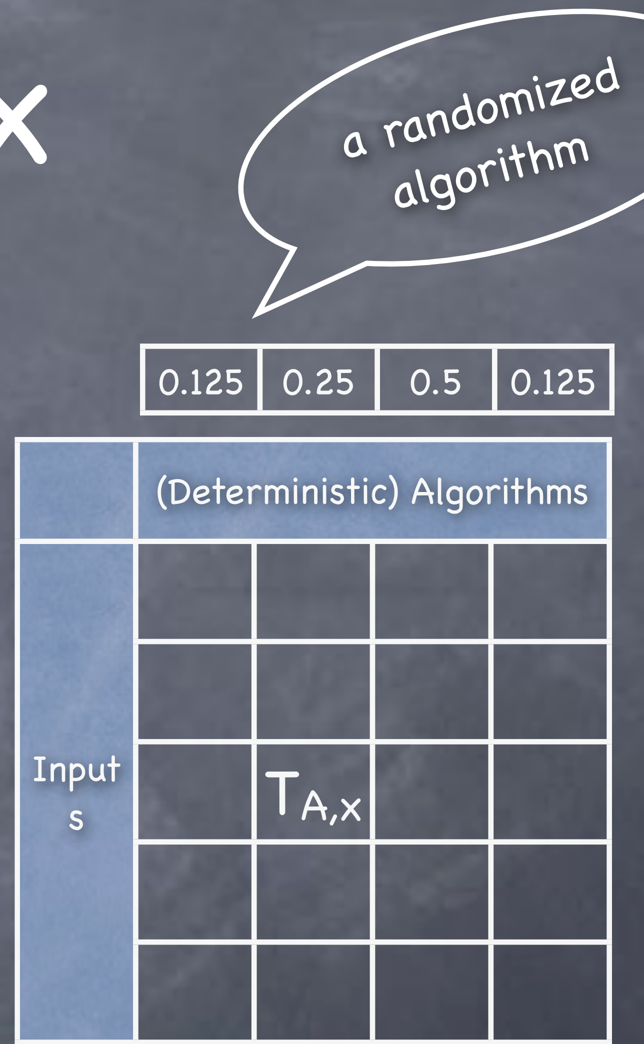
Yao's Min-Max

- Interested in expected cost (running time)
- Standard setting: Pick your randomized algorithm R; input x given adversarially
    - (Or may allow random input: not useful to the adversary)



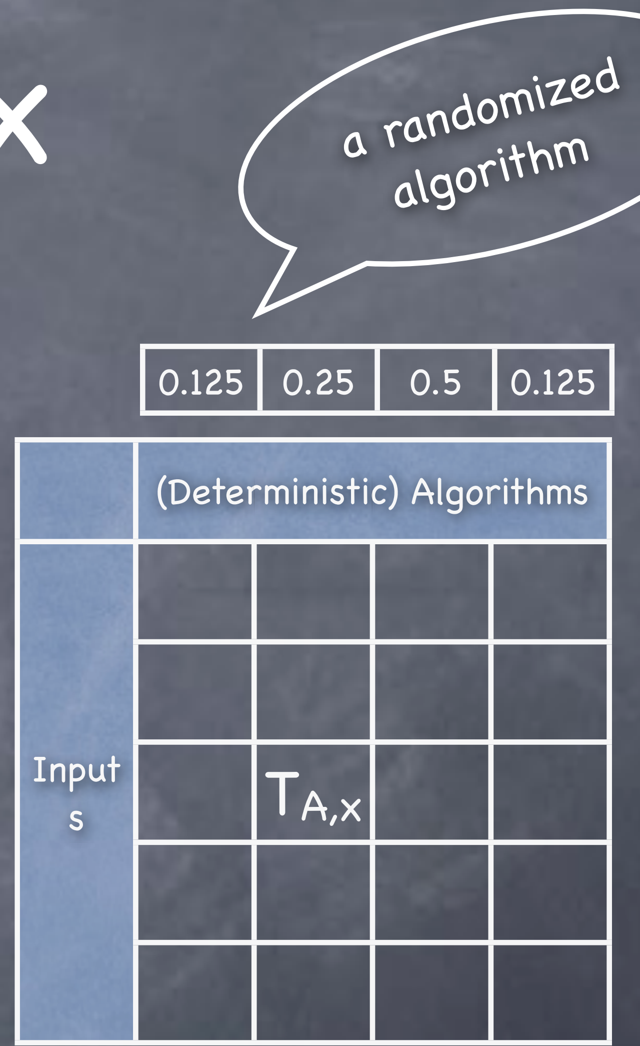
Yao's Min-Max

- Interested in expected cost (running time)
- Standard setting: Pick your randomized algorithm R; input x given adversarially
  - (Or may allow random input: not useful to the adversary)
- Another setting: Given adversarial input distribution X; pick your deterministic algorithm A



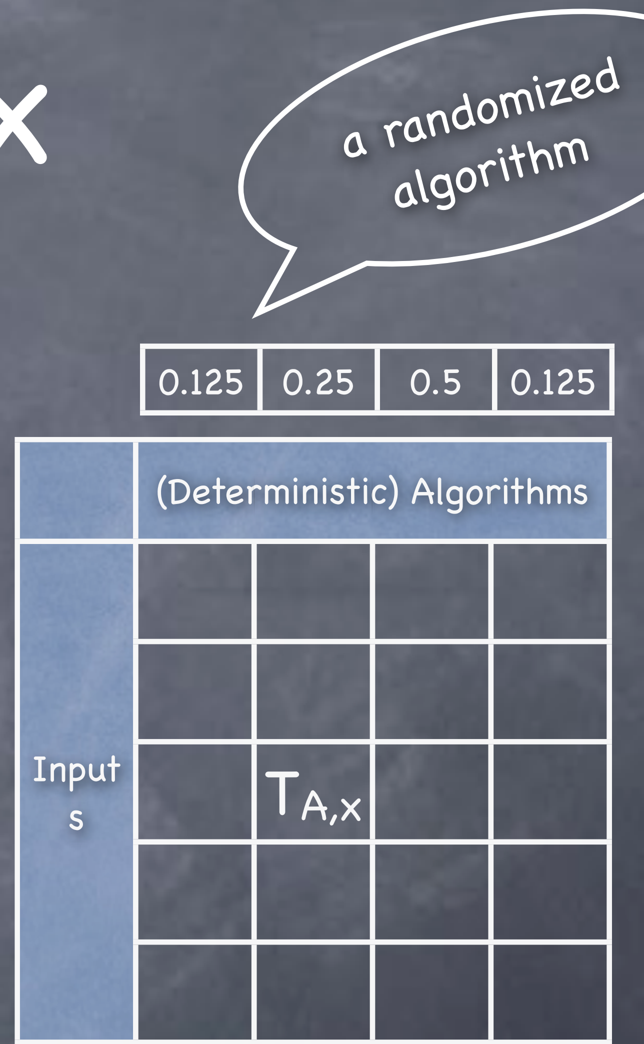
Yao's Min-Max

- Interested in expected cost (running time)
- Standard setting: Pick your randomized algorithm R; input x given adversarially
  - (Or may allow random input: not useful to the adversary)
- Another setting: Given adversarial input distribution X; pick your deterministic algorithm A
  - (Allowing randomized algorithm no better)



Yao's Min-Max

- Interested in expected cost (running time)
- Standard setting: Pick your randomized algorithm R; input x given adversarially
  - (Or may allow random input: not useful to the adversary)
- Another setting: Given adversarial input distribution X; pick your deterministic algorithm A
  - (Allowing randomized algorithm no better)
- Both have the same expected cost!! (not obvious: follows from LP duality)



Yao's Min-Max

Yao's Min-Max

- $\min_{\text{rand-alg } R} \max_{\text{input } x} E_{A \leftarrow R}[T_{A,x}] = \max_{\text{inp-distr } X} \min_{\text{alg } A} E_{x \leftarrow X}[T_{A,x}]$

Yao's Min-Max

- $\min_{\text{rand-alg } R} \max_{\text{input } x} E_{A \leftarrow R}[T_{A,x}] = \max_{\text{inp-distr } X} \min_{\text{alg } A} E_{x \leftarrow X}[T_{A,x}]$
- Simpler, but useful direction: for <u>any</u> randomized alg R and <u>any</u> input-distribution X, $\max_{\text{input } x} E_{A \leftarrow R}[T_{A,x}] \geq \min_{\text{alg } A} E_{x \leftarrow X}[T_{A,x}]$

Yao's Min-Max

- $\min_{\text{rand-alg } R} \max_{\text{input } x} E_{A\leftarrow R}[T_{A,x}] = \max_{\text{inp-distr } X} \min_{\text{alg } A} E_{x\leftarrow X}[T_{A,x}]$
- Simpler, but useful direction: for <u>any</u> randomized alg R and <u>any</u> input-distribution X, $\max_{\text{input } x} E_{A\leftarrow R}[T_{A,x}] \geq \min_{\text{alg } A} E_{x\leftarrow X}[T_{A,x}]$
  - If every algorithm A performs badly on an <u>**input-distribution**</u> X, then a randomized combination of those algorithms also perform badly on X. If R does badly on X, on some x in its support it does at least as badly (x depends on R)

Yao's Min-Max

- $\min_{\text{rand-alg } R} \max_{\text{input } x} E_{A \leftarrow R}[T_{A,x}] = \max_{\text{inp-distr } X} \min_{\text{alg } A} E_{x \leftarrow X}[T_{A,x}]$
- Simpler, but useful direction: for <u>any</u> randomized alg R and <u>any</u> input-distribution X, $\max_{\text{input } x} E_{A \leftarrow R}[T_{A,x}] \geq \min_{\text{alg } A} E_{x \leftarrow X}[T_{A,x}]$
  - If every algorithm A performs badly on an **<u>input-distribution</u>** X, then a randomized combination of those algorithms also perform badly on X. If R does badly on X, on some x in its support it does at least as badly (x depends on R)
  - Useful: Can show lower-bound for randomized algorithms via lower-bound on distributional complexity for deterministic algorithms