Separations in Query Complexity Based on Pointer Functions (with a slight hint of quantum complexity)

Alexander Belov
CWI

Joint work with: Andris Ambainis, Kaspars Balodis, Troy Lee, Miklos Santha, and Juris Smotrovs

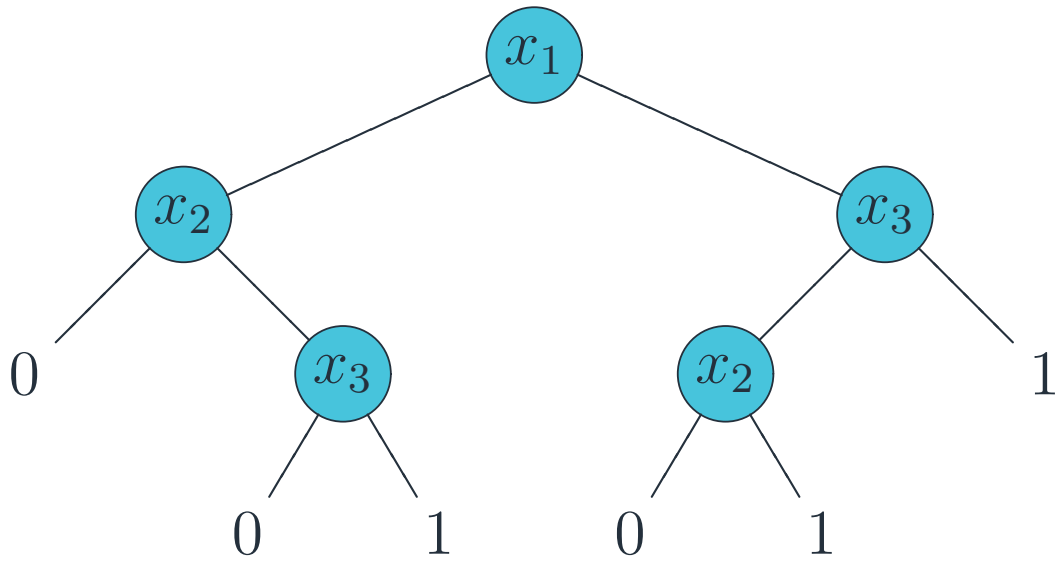
(presented at QIP'16, to appear in STOC'16)

Introduction


SEPARATIONS

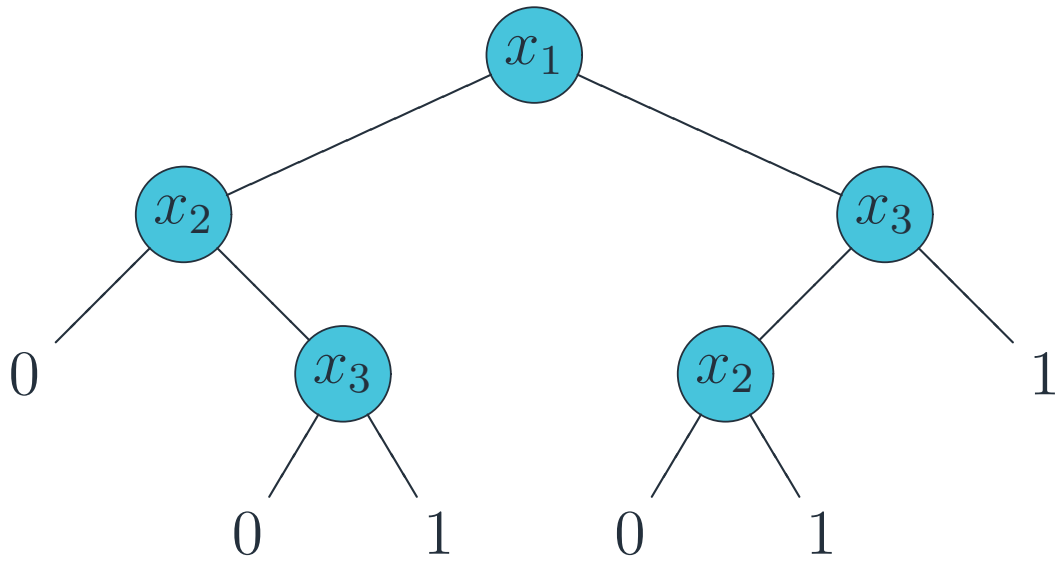
Computational Models: Deterministic

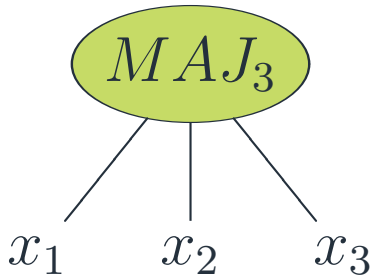
D: Deterministic (Decision Tree)



Computational Models: Deterministic

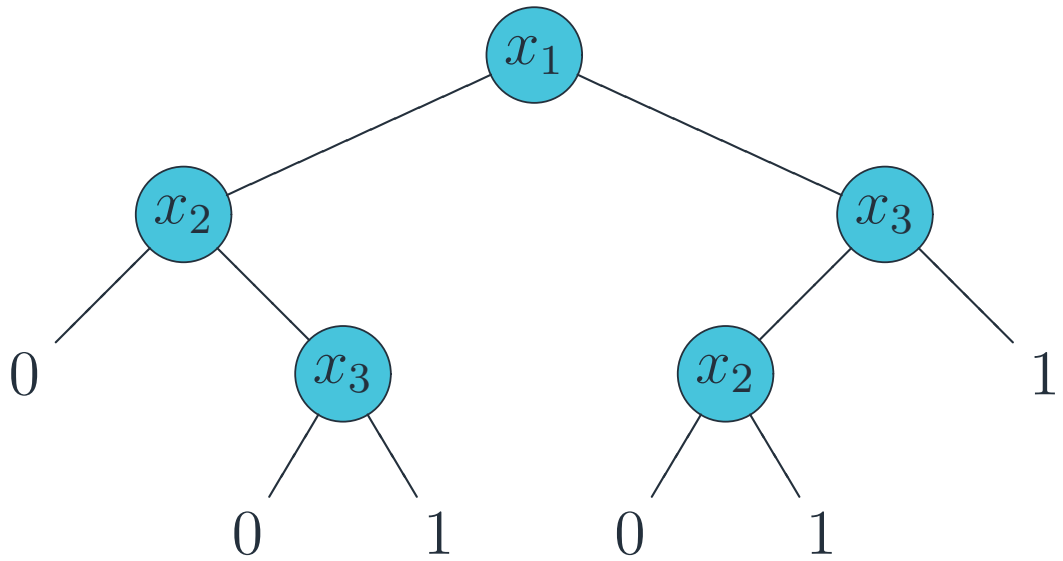
D: Deterministic (Decision Tree)





Computational Models: Deterministic

D: Deterministic (Decision Tree)



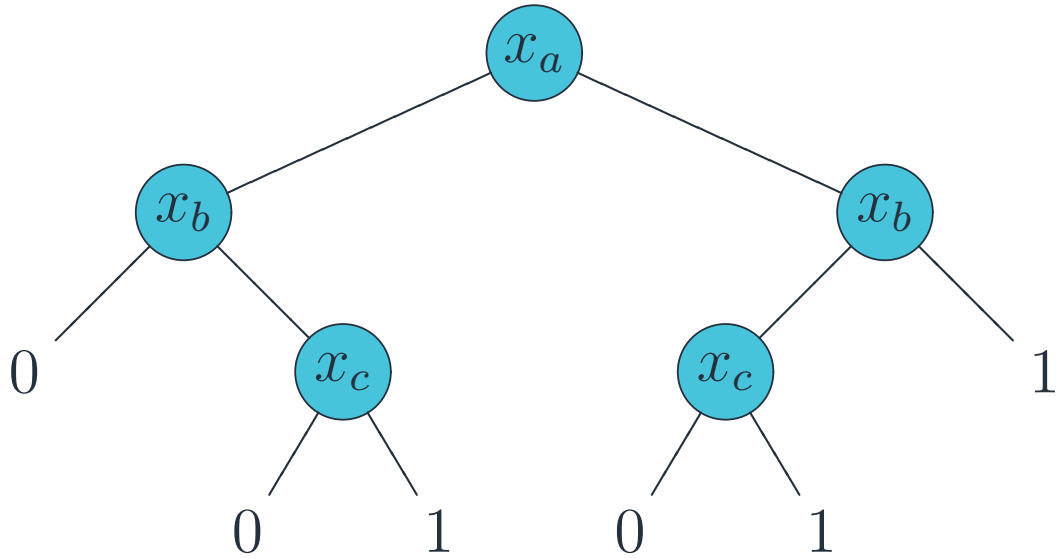
Complexity

- on input: Number of queries (length of the path) 2 or 3
- in total: Worst input (depth of the tree) 3

Computational Models: Randomised

D: Deterministic (Decision Tree)

R: Randomised (Probability distribution on decision trees)



a, b, c: uniform random permutation of $1, 2, 3$.

Complexity

- on input: Expected number of queries 2 or $\frac{8}{3}$
- in total: Worst input $\frac{8}{3}$

Computational Model: Randomised

D: Deterministic (Decision Tree)

R: Randomised (Probability distribution on decision trees)

R_0: Zero-error (Las Vegas)

- always outputs the correct output

Computational Models: Randomised

D: Deterministic (Decision Tree)

R: Randomised (Probability distribution on decision trees)

R_0: Zero-error (Las Vegas)

- always outputs the correct output

R_2: Bounded-error (Monte Carlo)

- rejects a negative input with probability $\geq \frac{2}{3}$
- accepts a positive input with probability $\geq \frac{2}{3}$

Computational Models: Randomised

D: Deterministic (Decision Tree)

R: Randomised (Probability distribution on decision trees)

- R_0: Zero-error (Las Vegas)
  - always outputs the correct output
- R_1: One-sided error
  - always rejects a negative input
  - accepts a positive input with probability $\geq \frac{1}{2}$ (or vice versa)
- R_2: Bounded-error (Monte Carlo)
  - rejects a negative input with probability $\geq \frac{2}{3}$
  - accepts a positive input with probability $\geq \frac{2}{3}$

Computational Models: Quantum

D: Deterministic (Decision Tree)

R: Randomised (Probability distribution on decision trees)

- R_0: Zero-error (Las Vegas)
  - always outputs the correct output
- R_1: One-sided error
  - always rejects a negative input
  - accepts a positive input with probability $\geq \frac{1}{2}$ (or vice versa)
- R_2: Bounded-error (Monte Carlo)
  - rejects a negative input with probability $\geq \frac{2}{3}$
  - accepts a positive input with probability $\geq \frac{2}{3}$

Q: Quantum

- Q_E: Exact
- Q_2: Bounded-error

Separations

Easy for **partial** functions

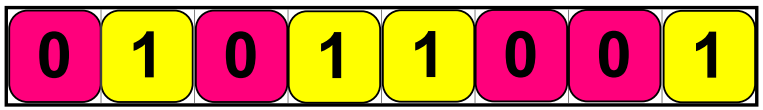
Separations

Easy for **partial** functions

Example: Deutsch-Jozsa problem (almost)

- **Reject** iff all input variables are zeroes

- **Accept** iff exactly half of the variables are ones



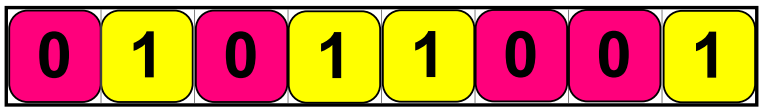
Separations

Easy for **partial** functions

Example: Deutsch-Jozsa problem (almost)

- **Reject** iff all input variables are zeroes

- **Accept** iff exactly half of the variables are ones



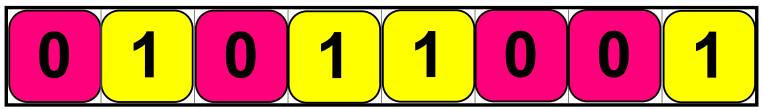
$$R_1 = 1$$

Easy for **partial** functions

Example: Deutsch-Jozsa problem (almost)

- **Reject** iff all input variables are zeroes

- **Accept** iff exactly half of the variables are ones



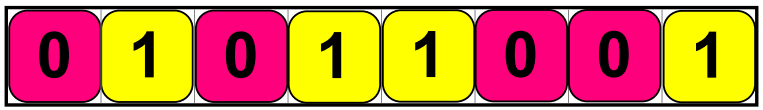
$$R_1 = 1, \quad Q_E = 1,$$

Easy for **partial** functions

Example: Deutsch-Jozsa problem (almost)

- **Reject** iff all input variables are zeroes

- **Accept** iff exactly half of the variables are ones



$$R_1 = 1, \quad Q_E = 1, \qquad R_0 = n/2 + 1$$

Easy for **partial** functions

Example: Deutsch-Jozsa problem (almost)

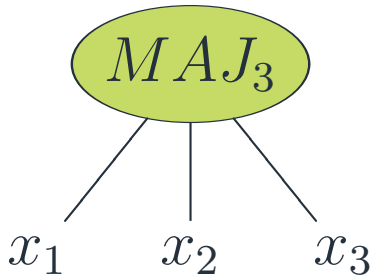
- **Reject** iff all input variables are zeroes

- **Acc**ept iff exactly half of the variables are ones

Total Functions — ???

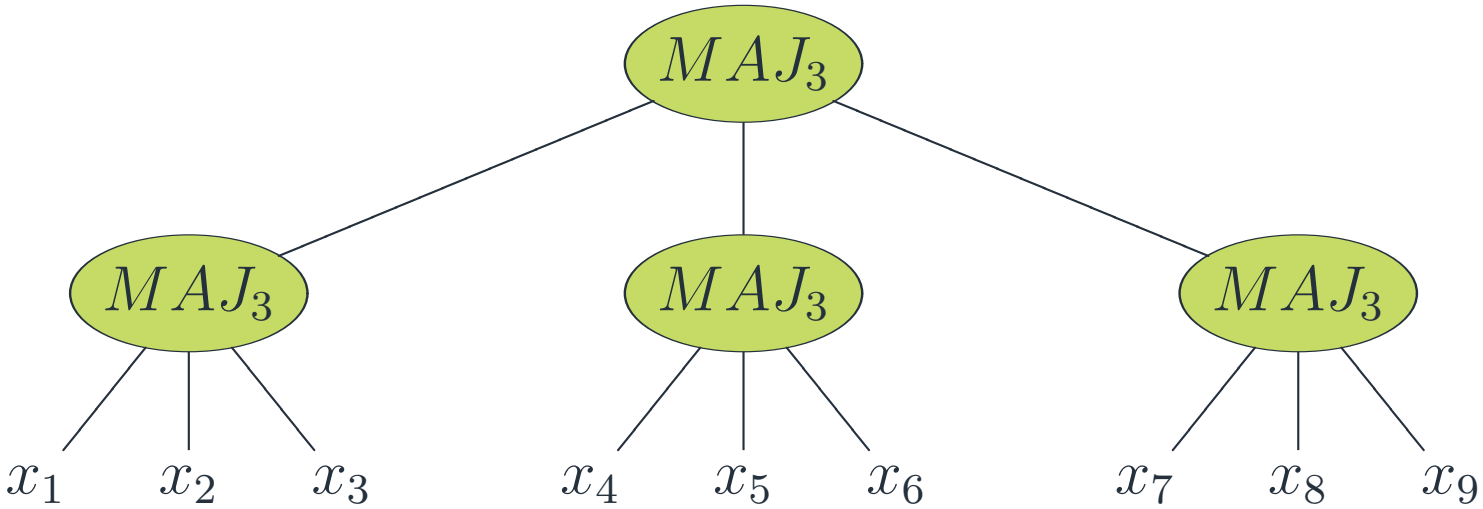
Overview of Results

We have just seen $D(MAJ_3) = 3$ and $R_0(MAJ_3) = 8/3$.



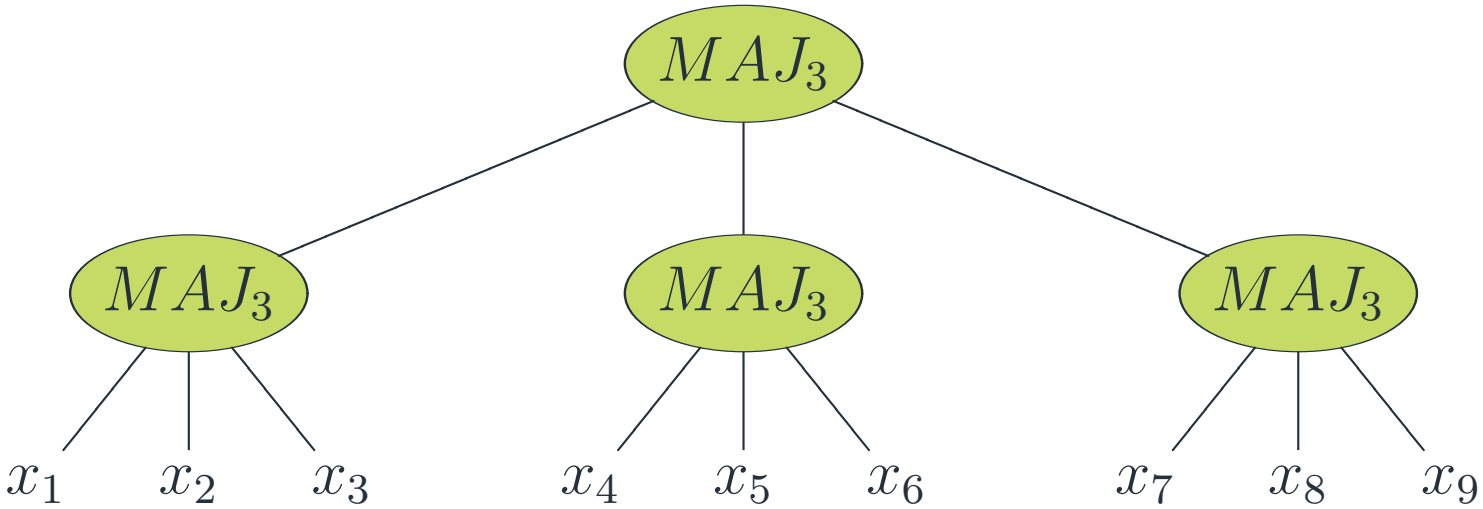
We have just seen $D(MAJ_3) = 3$ and $R_0(MAJ_3) = 8/3$.

Iterate it:



We have just seen $D(MAJ_3) = 3$ and $R_0(MAJ_3) = 8/3$.

Iterate it:



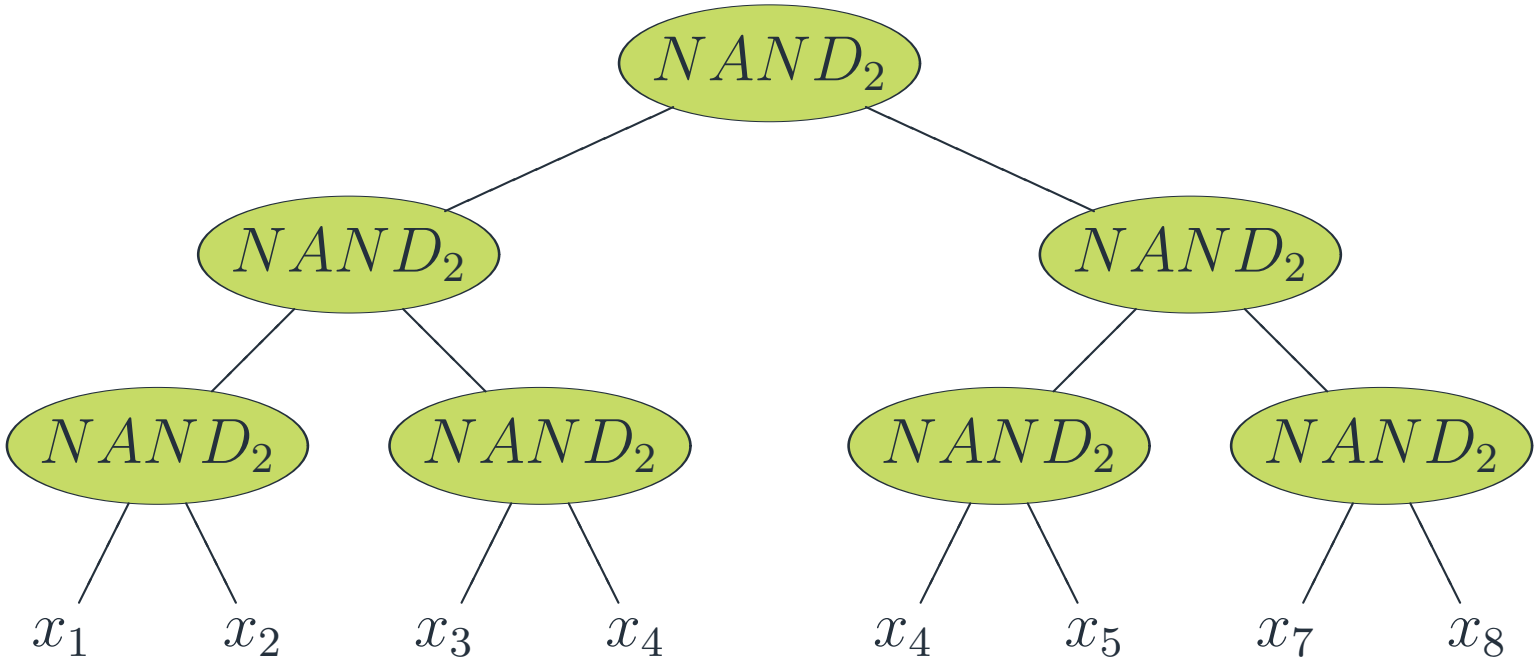
We get

$$D(MAJ_3^d) = 3^d \quad \text{and} \quad R_0(MAJ_3^d) \leq (8/3)^d.$$

(Actually, it is less...)

Previous Record-Holder

Iterated NAND: record-holder for R_0, R_1, R_2 versus D



We have [Snir'85, Saks & Wigderson'86]:

$$R_0 = R_1 = R_2 = O(n^{0.7537\ldots}), \qquad D = n$$

State of the Art

We have [Snir'85, Saks & Wigderson'86]:

$$R_0 = R_1 = R_2 = O(n^{0.7537\ldots}), \qquad D = n$$

It is known [Nisan'89]

$$D = O(R_1^2)$$

Our Main Results

It is known [Nisan'89]

$$D = O(R_1^2)$$

We get functions with:

$$D = \widetilde{\Theta}(R_0^2)$$

$$R_0 = \widetilde{\Theta}(R_1^2)$$

Our Main Results

It is known [Nisan'89]

$$D = O(R_1^2)$$

We get functions with:

$$D = \widetilde{\Theta}(R_0^2)$$

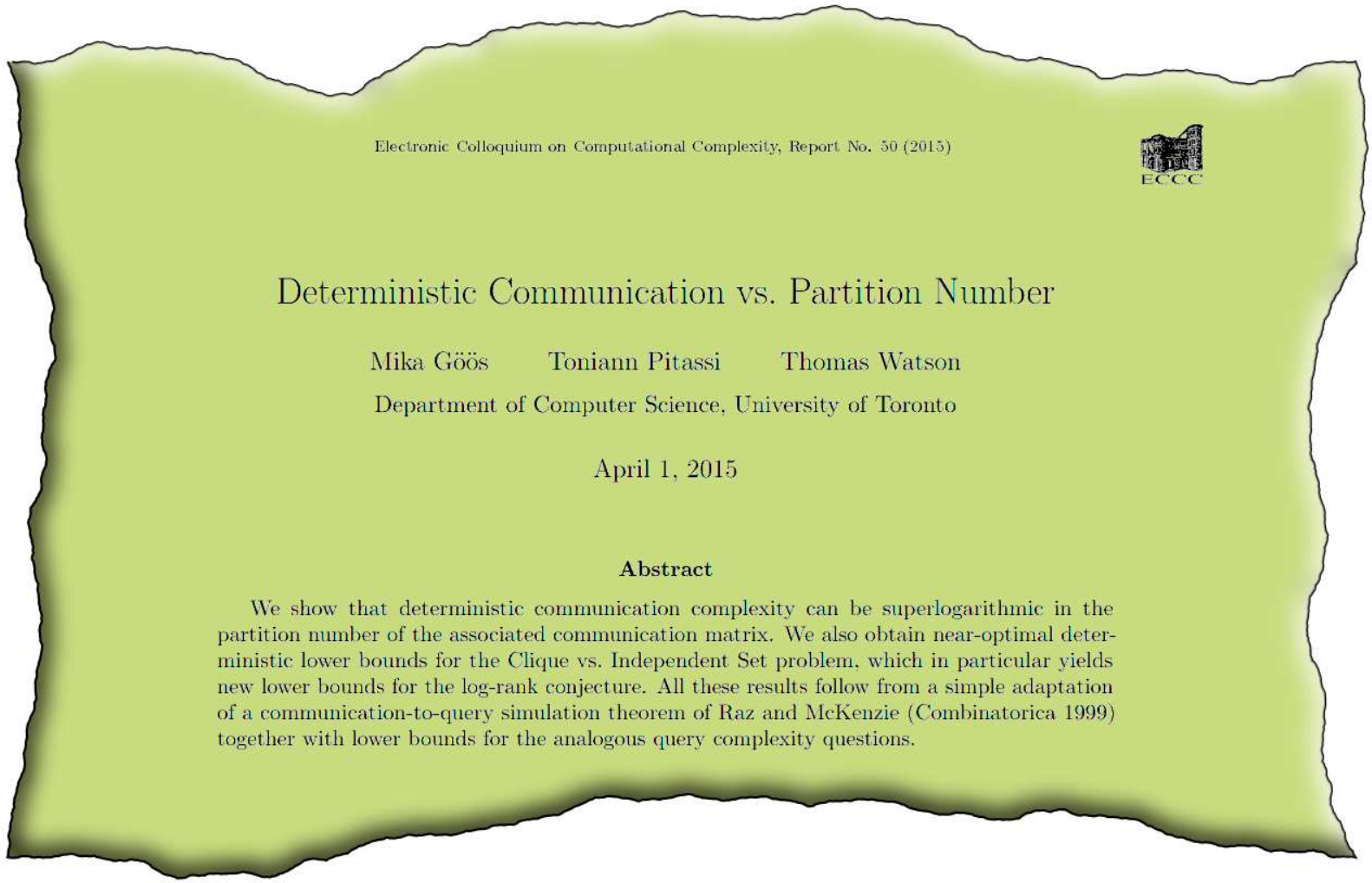
$$R_0 = \widetilde{\Theta}(R_1^2)$$

The last one also saturates [Kulkarni & Tal'13, Midrijānis'05]

$$R_0 = \widetilde{O}(R_2^2)$$

Göös-Pitassi-Watson

Paper



Electronic Colloquium on Computational Complexity, Report No. 50 (2015)

ECCC

Deterministic Communication vs. Partition Number

Mika Göös Toniann Pitassi Thomas Watson

Department of Computer Science, University of Toronto

April 1, 2015

Abstract

We show that deterministic communication complexity can be superlogarithmic in the partition number of the associated communication matrix. We also obtain near-optimal deterministic lower bounds for the Clique vs. Independent Set problem, which in particular yields new lower bounds for the log-rank conjecture. All these results follow from a simple adaptation of a communication-to-query simulation theorem of Raz and McKenzie (Combinatorica 1999) together with lower bounds for the analogous query complexity questions.

Goal

- Clique vs. Independent Set in communication complexity

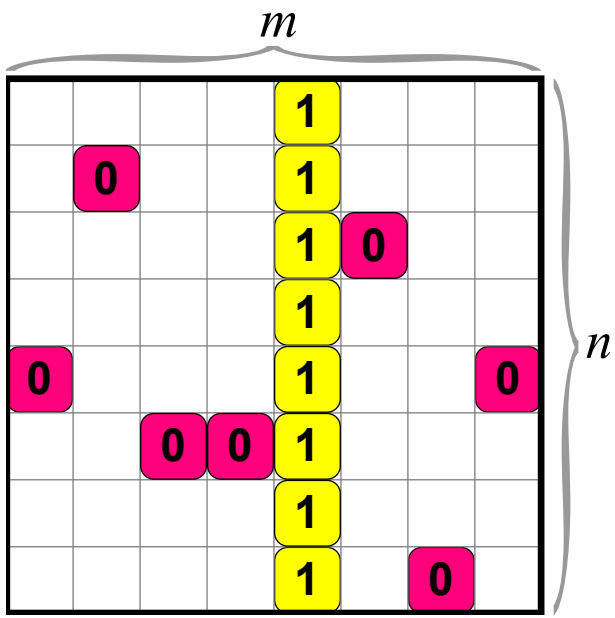
- Reduce to a problem in query complexity: Find a function that
  - has large deterministic complexity
  - has small unambiguous 1-certificates

    There exists a number of 1-certificates such that each positive input satisfies exactly one of them.

D versus 1-certificates

Function on nm Boolean variables

- **Accept** iff there exists a unique all-1 column

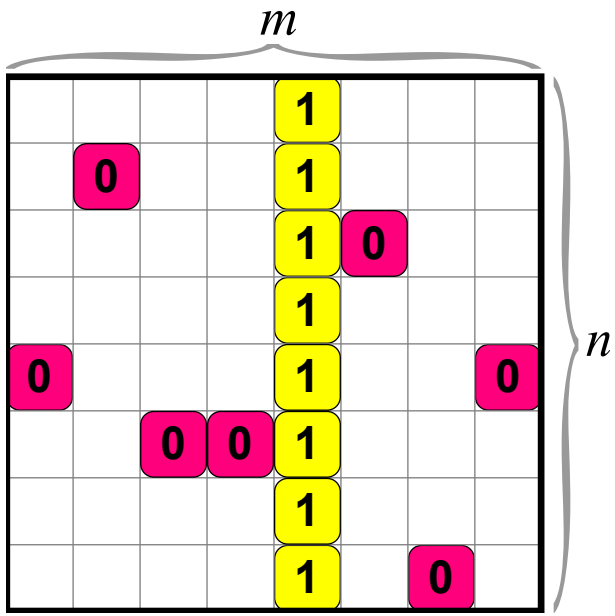


- $D = nm$
- short 1-certificates ($n + m - 1$), **BUT not** unambiguous.

D versus 1-certificates

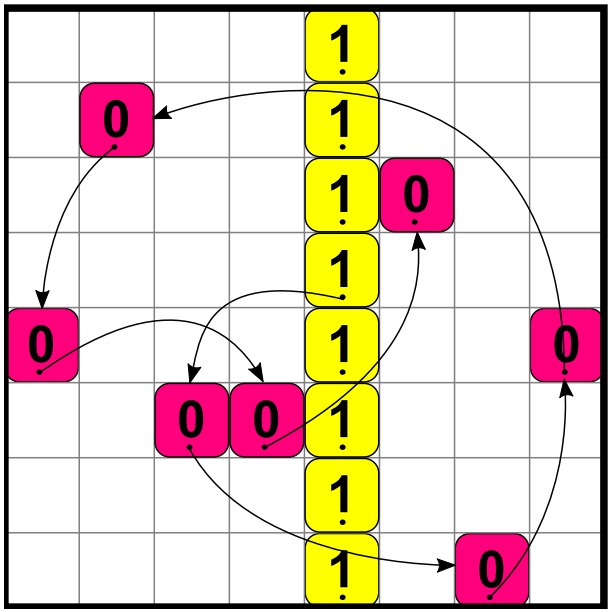
Function on nm Boolean variables

- **Accept** iff there exists a unique all-1 column



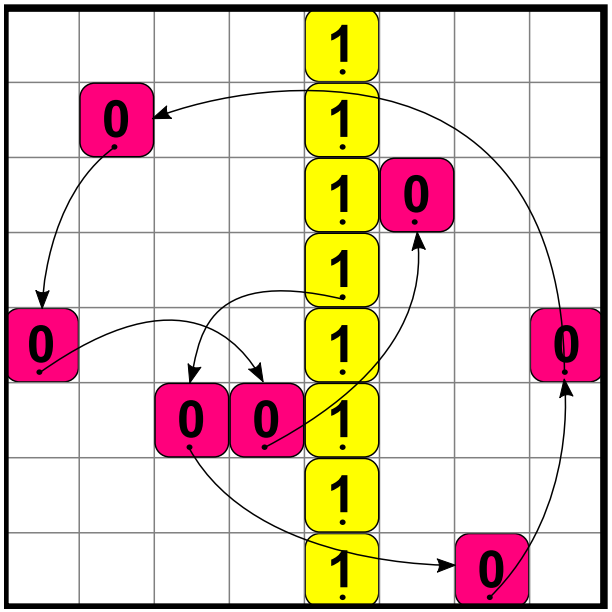
- $D = nm$
- short 1-certificates ($n + m - 1$), **BUT not** unambiguous.
  Should specify which zero to take in each column!

Pointers



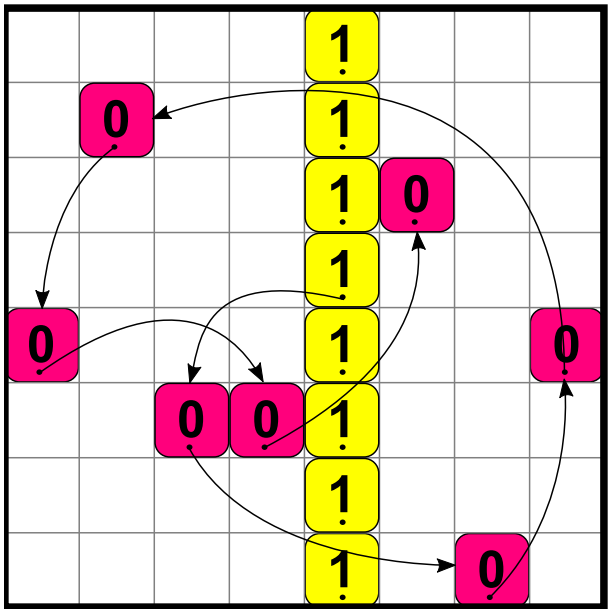
- **Alphabet:** $\{0,1\} \times ([n] \times [m] \cup \{\perp\})$
  Not Boolean, but we can encode using $O(\log(n+m))$ bits.
- **Accept** iff
  - There is a (unique) all-1 column b;
  - in b, there is a unique element r with non-zero pointer;
  - following the pointers from r, we traverse through exactly one zero in each column but b.

Pointers



- short unambiguous 1-certificates ($n + m - 1$)

Pointers



- short unambiguous 1-certificates ($n + m - 1$)
- Still have $D = nm$ (Adversary argument, next slide)

Adversary Method

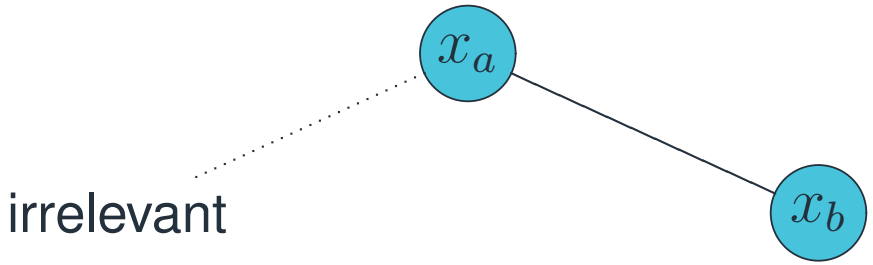
Adversary finds a bad input for each deterministic decision tree, by playing along with the decision tree.

Adversary Method

Adversary finds a bad input for each deterministic decision tree, by playing along with the decision tree.

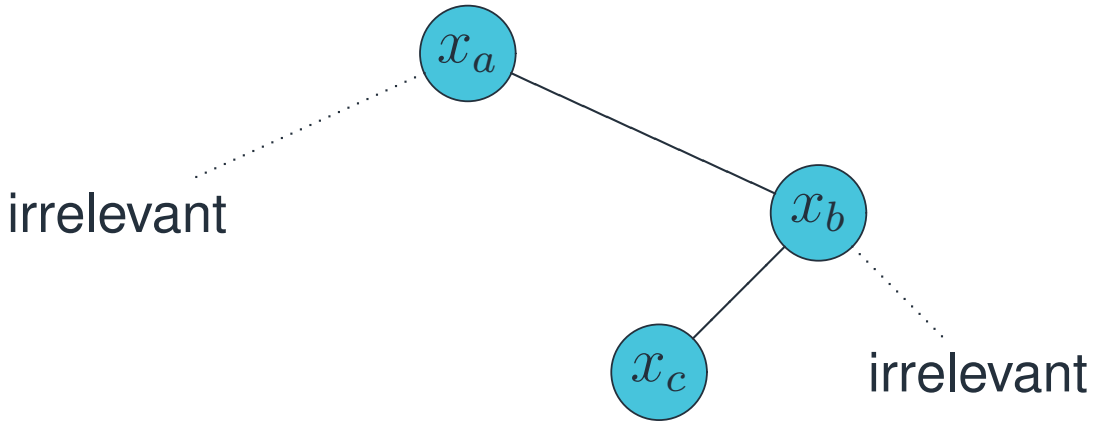
Adversary Method

Adversary finds a bad input for each deterministic decision tree, by playing along with the decision tree.



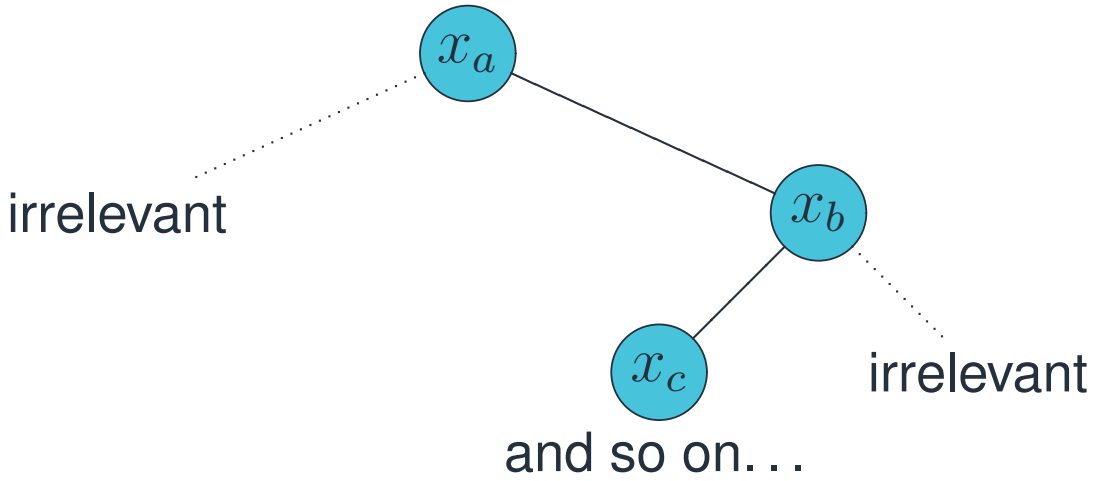
Adversary Method

Adversary finds a bad input for each deterministic decision tree, by playing along with the decision tree.



Adversary Method

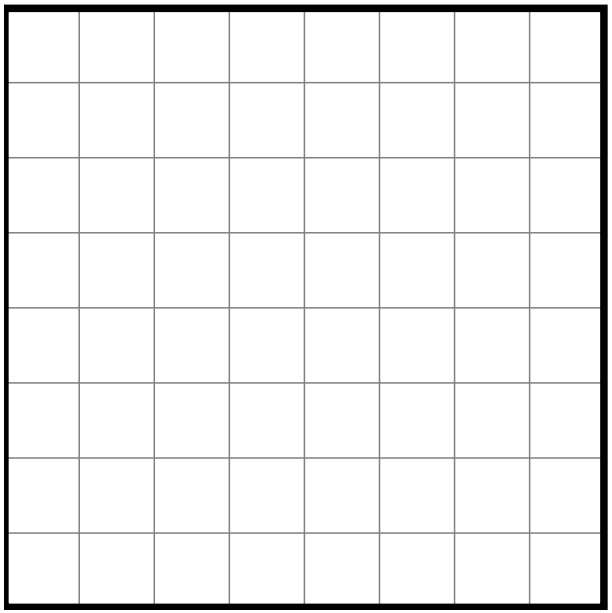
Adversary finds a bad input for each deterministic decision tree, by playing along with the decision tree.



For each queried variable, the adversary provides the value, so that the value of the function is unknown as long as possible.

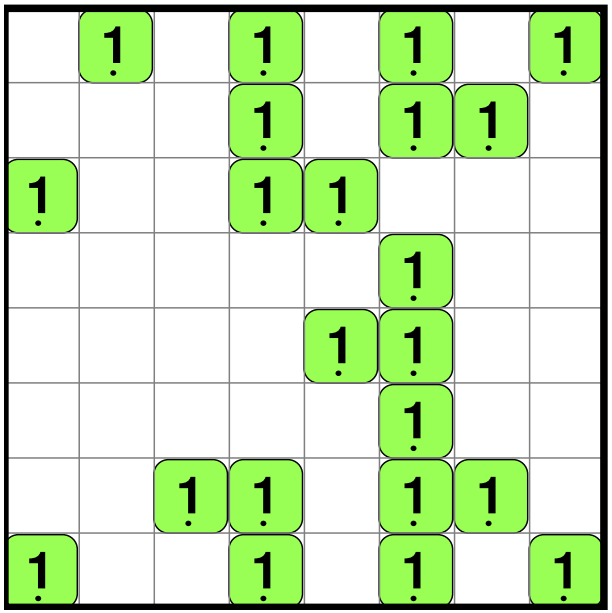
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



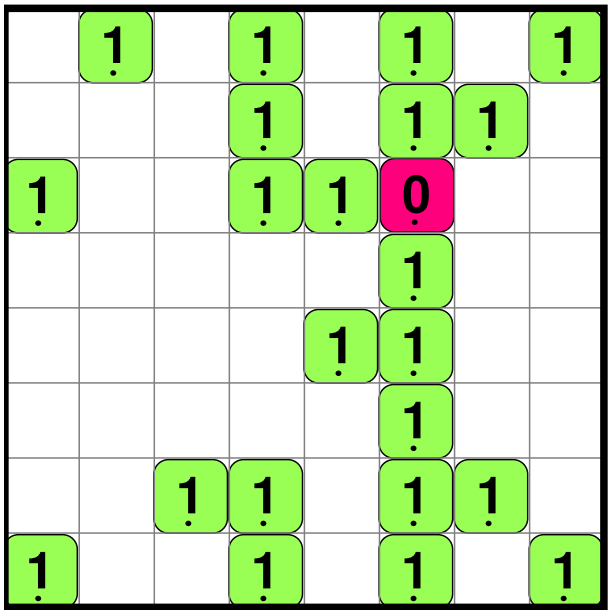
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



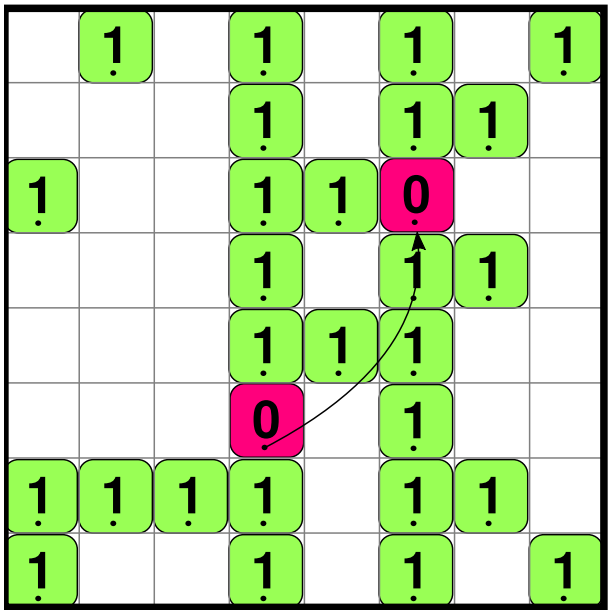
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



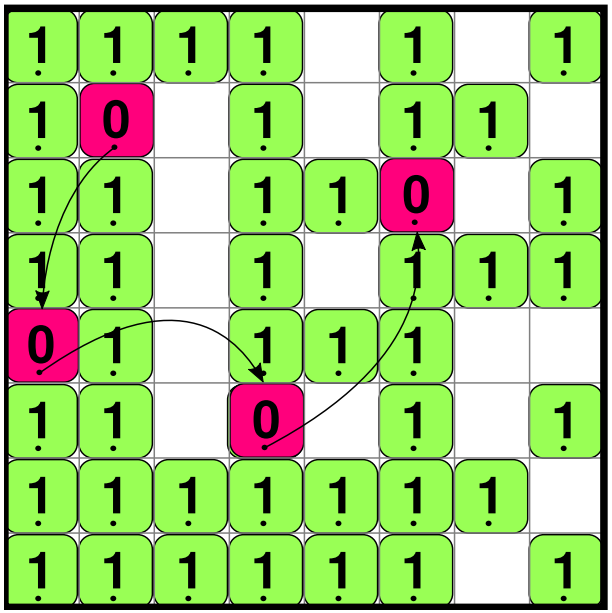
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



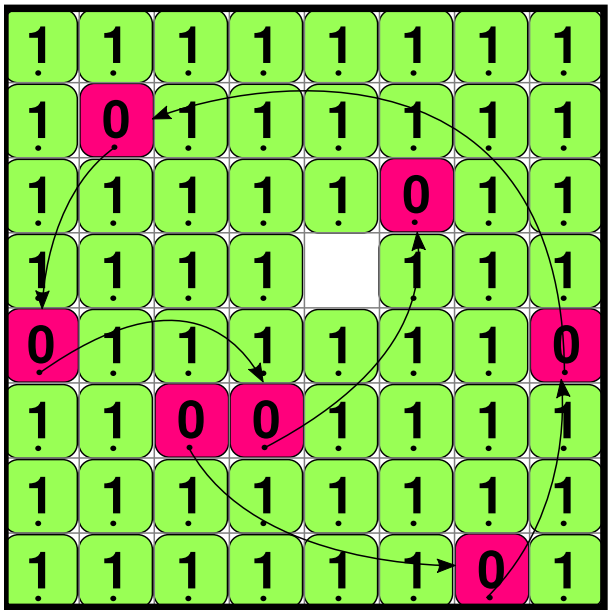
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



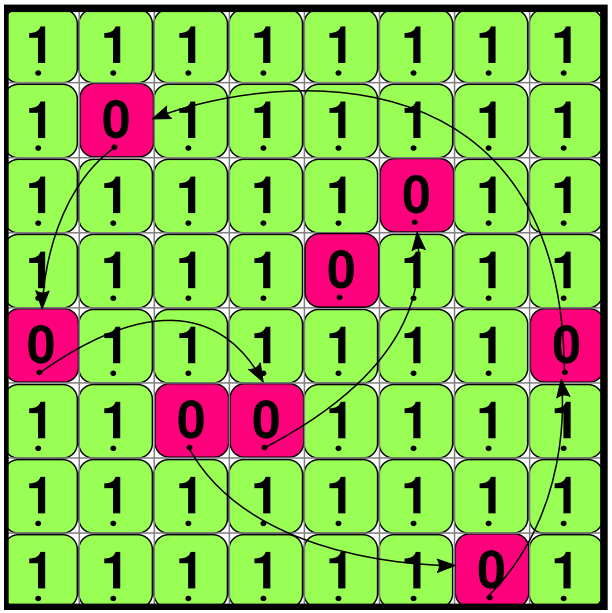
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



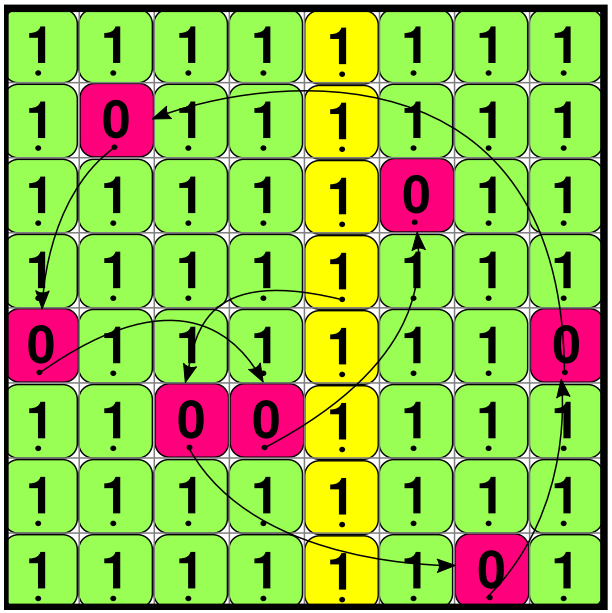
Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



Deterministic Lower Bound

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



Features of Pointers

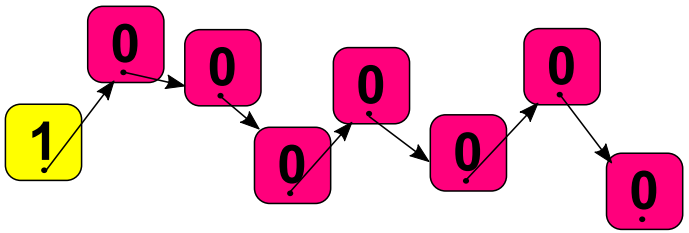
Highly elusive
(flexible)

Still traversable
(if know where to start).

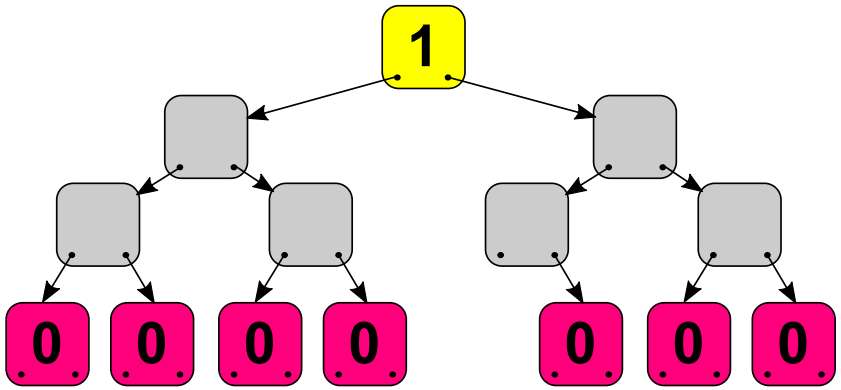
Our Modifications

Binary Tree

Instead of a list

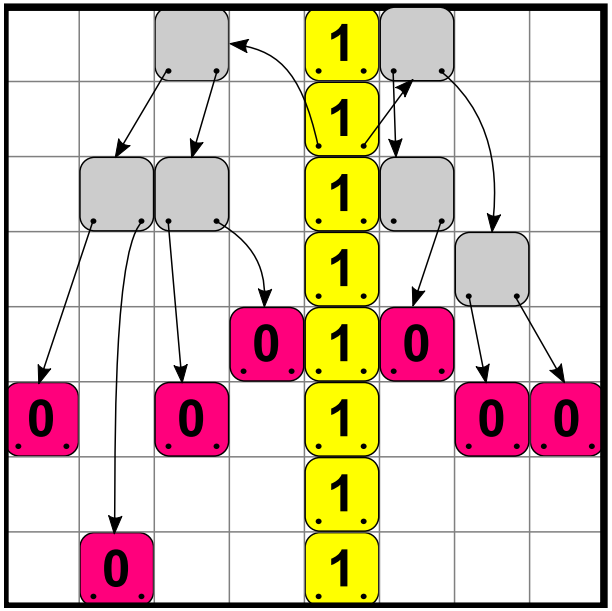


we use a balanced binary tree



- More elusive
- Random access

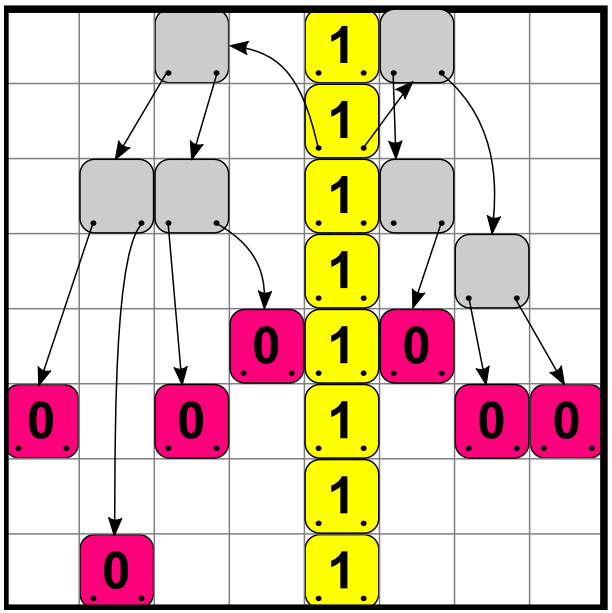
Definition (base)



Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.

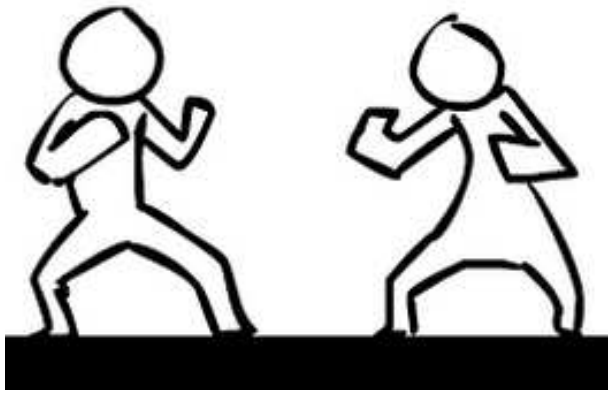
Definition (base)



Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.
- Some additional information is contained in the leaves (to be defined).

R_1 versus R_0



State of the Art

- NO separation was known even between R_2 and R_0.

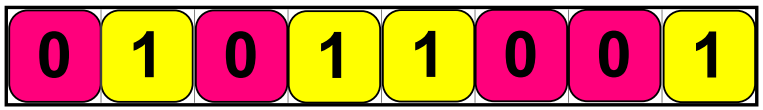
(Iterated functions are not of much help here.)

Reminder 1: Partial Separation

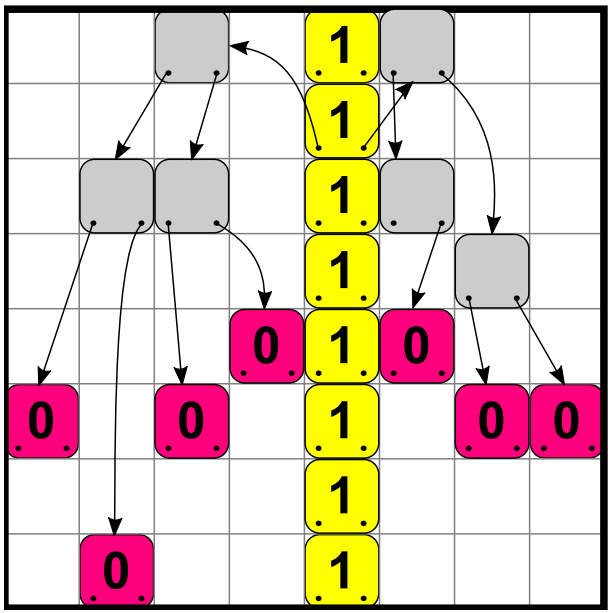
Recall the separation for a partial function

- **Reject** iff all input variables are zeroes

- **Accept** iff exactly half of the variables are ones



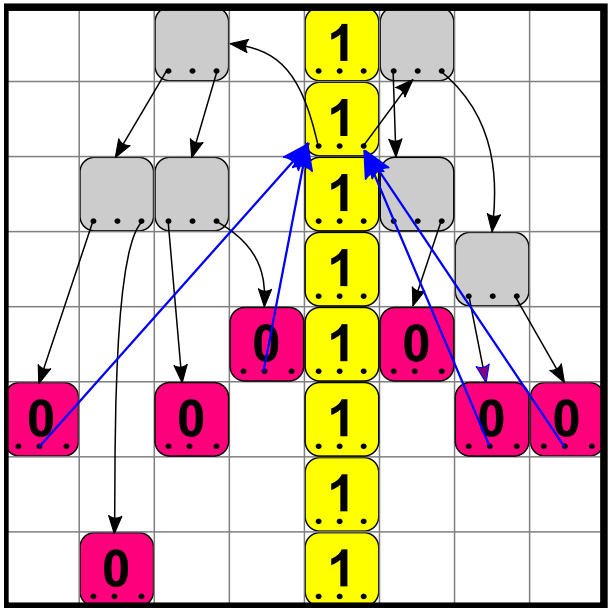
Reminder 2: Definition (base)



Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.
- Some additional information is contained in the leaves (to be defined).

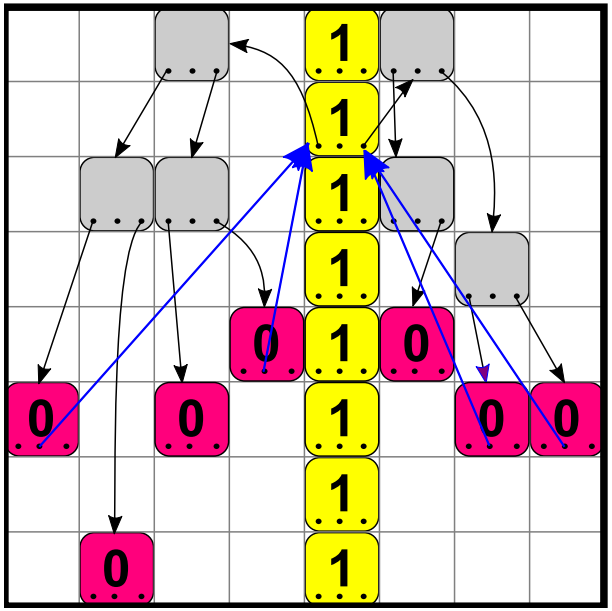
Definition



Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.
- exactly $m/2$ of the leaves back point to the root r.

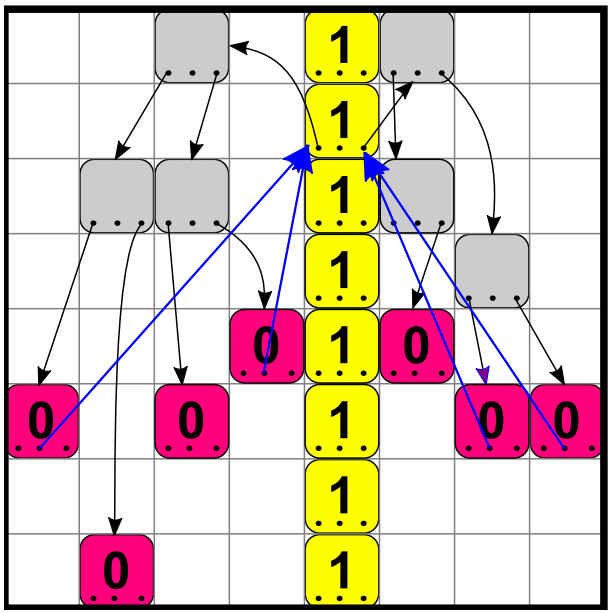
Totalisation



A column is good if it contains a leaf back pointing to the root of a legitimate tree.

- A positive input contains exactly $m/2$ good columns.
- A negative input contains no good columns.

Totalisation



A column is good if it contains a leaf back pointing to the root of a legitimate tree.

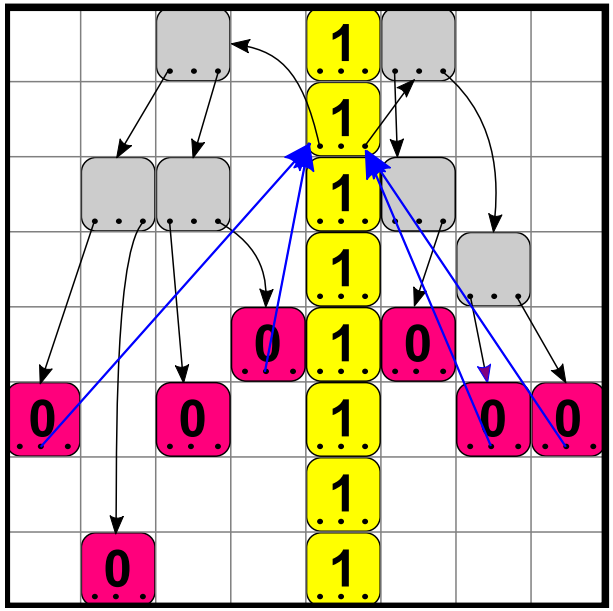
- A positive input contains exactly $m/2$ good columns.
- A negative input contains no good columns.

A total function looks like a partial function!

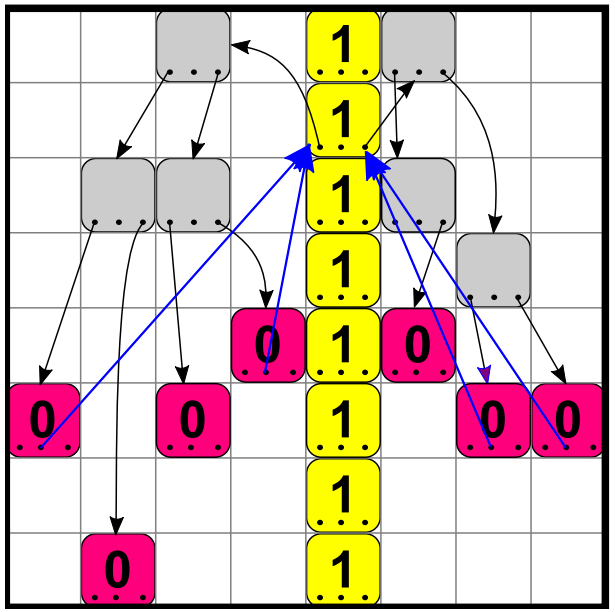
Check Column: Informal

Deterministic subroutine

Given a column $c \in [m]$, accept iff it is good.



Check Column: Informal

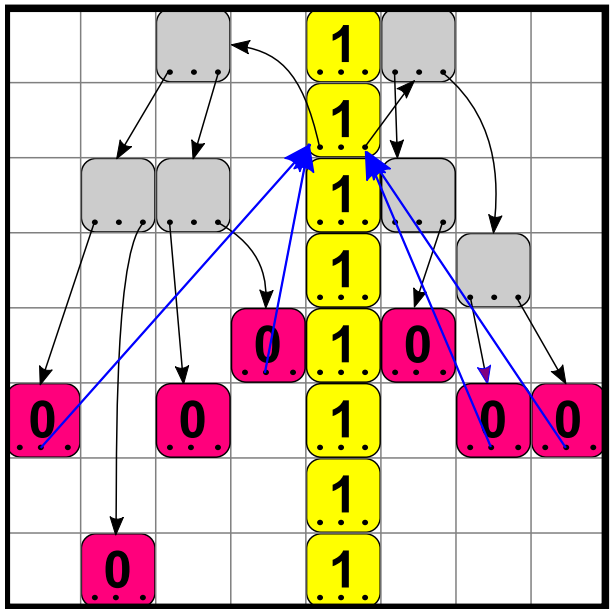


Deterministic subroutine

Given a column $c \in [m]$, accept iff it is good.

Go through column c, find the back pointer to r, and check the tree.

Check Column: Informal



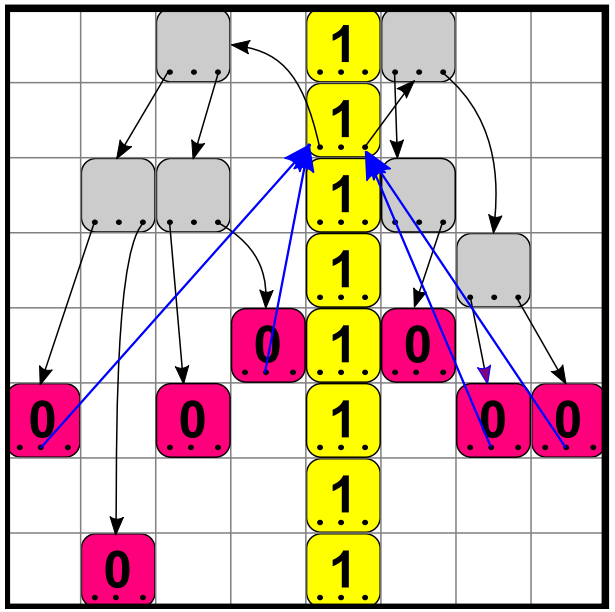
Deterministic subroutine

Given a column $c \in [m]$, accept iff it is good.

Go through column c, find the back pointer to r, and check the tree.

Wait, column c may contain many bogus pointers — ???

Check Column: Informal



Deterministic subroutine

Given a column $c \in [m]$, accept iff it is good.

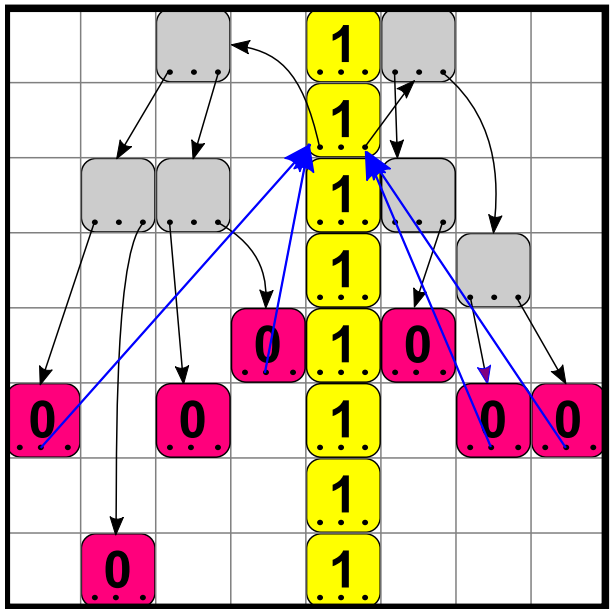
Go through column c, find the back pointer to r, and check the tree.

Wait, column c may contain many bogus pointers — ???

On each step, either

- eliminate a column: it is not the all-1 column; or
- eliminate an element in column c: it is not a leaf of the tree.

Check Column: Formal

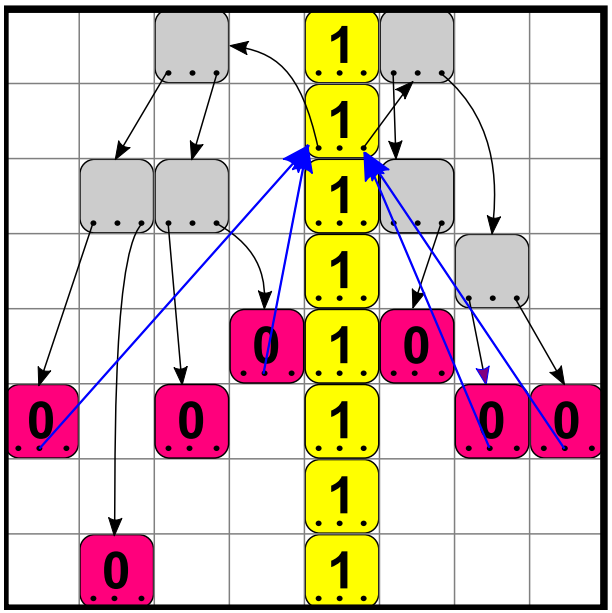


Deterministic subroutine

Given a column $c \in [m]$, accept iff it is good.

- **While** there is ≥ 2 non-eliminated columns:
  - Let a be a non-eliminated element in c. If none, **reject**.
  - Let r be the back pointer of a, and b be the column of r.
  - Let j be a non-eliminated column $\neq b$.
  - **If** the path $T(j)$ from r ends in a zero in column j, eliminate column j.
    Otherwise, eliminate element a.
- Verify the only non-eliminated column.

R_1 Upper Bound

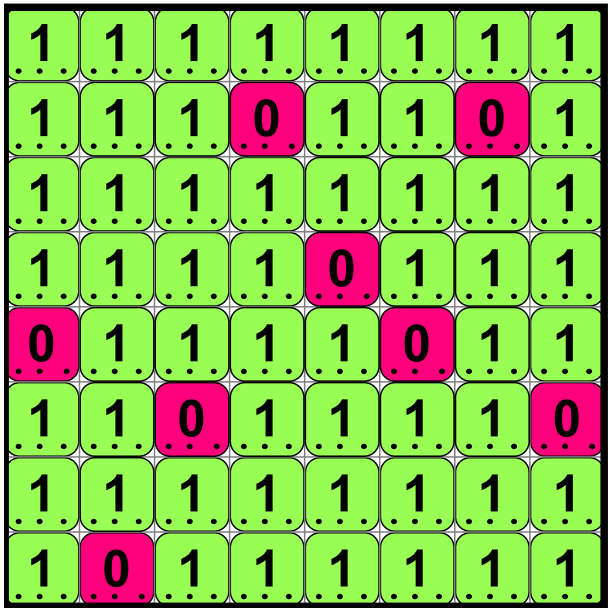


- On each iteration of the loop, either an element or a column gets eliminated. At most $n + m$ iterations.
  Complexity: $\widetilde{O}(n + m)$.

Sticking into Deutsch-Jozsa, get R_1 and Q_E upper bound of

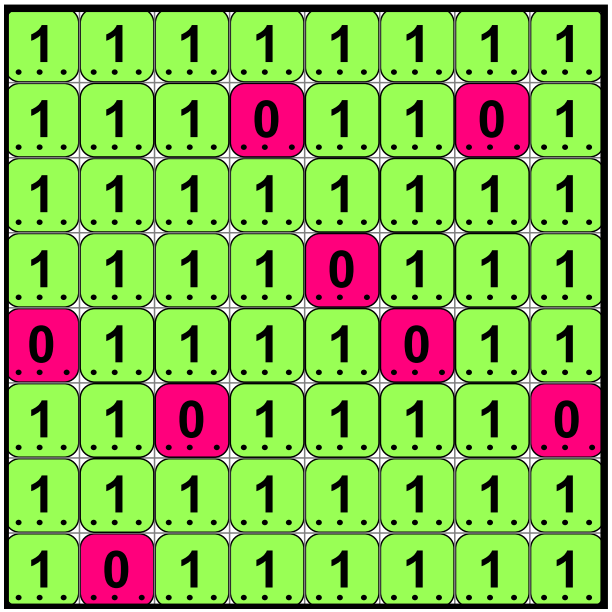
$$\widetilde{O}(n + m).$$

R_0 Lower Bound



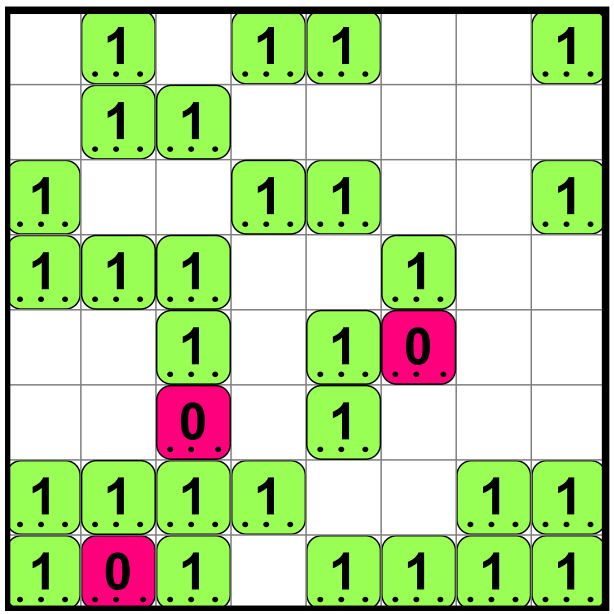
(Negative) input with exactly one zero in each column.

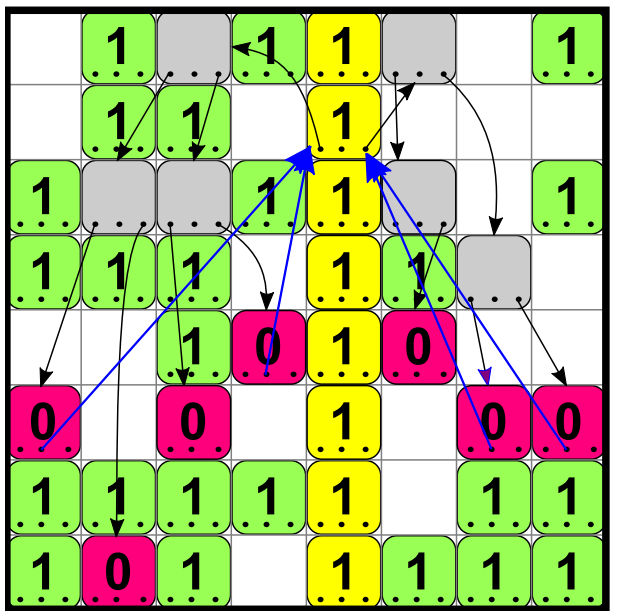
R_0 Lower Bound



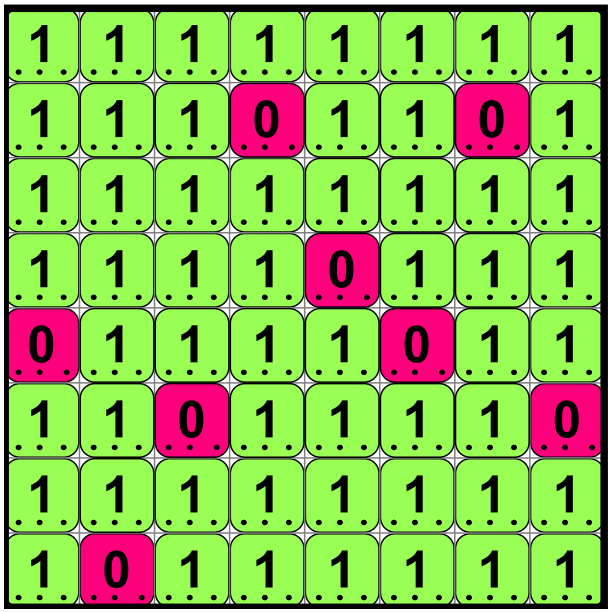
(Negative) input with exactly one zero in each column.

- An R_0 algorithm can reject only if it has found $m/2$ zeroes.





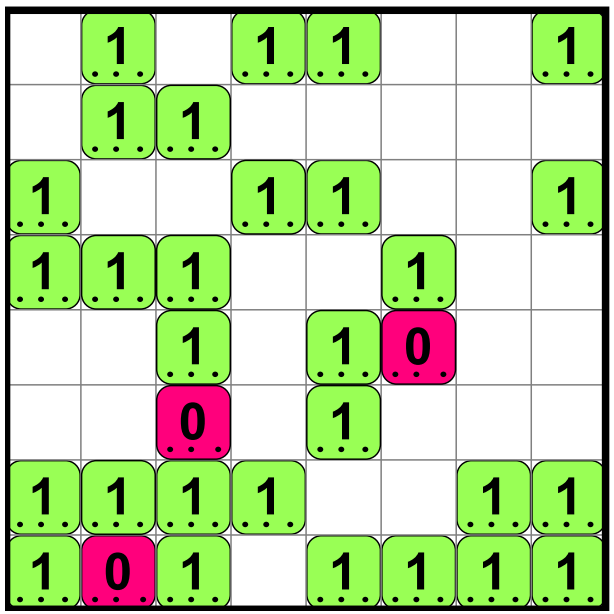
R_0 Lower Bound

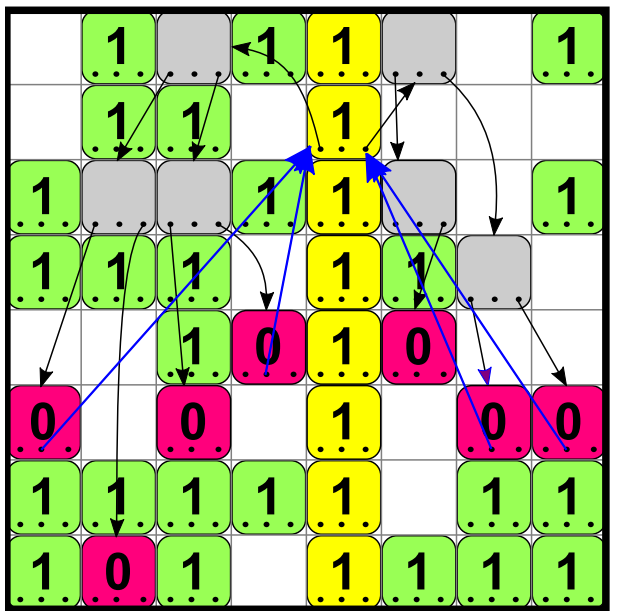


(Negative) input with exactly one zero in each column.

- An R_0 algorithm can reject only if it has found $m/2$ zeroes.

Requires $\Omega(nm)$ queries.





Summary

- Upper bound for R_1 and Q_E is $\widetilde{O}(n+m)$.
- Lower bound for a R_0 algorithm is $\Omega(nm)$.

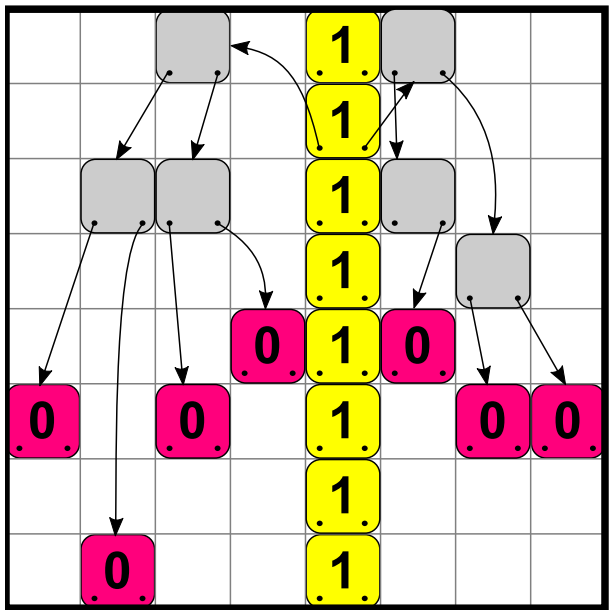
Taking $n = m$, we get a quadratic separation between R_1 and R_0, as well as between Q_E and R_0

NB. The previous separation was [Ambainis'12]:

$$Q_E = O(R_0^{0.8675\ldots})$$

R_0 versus D

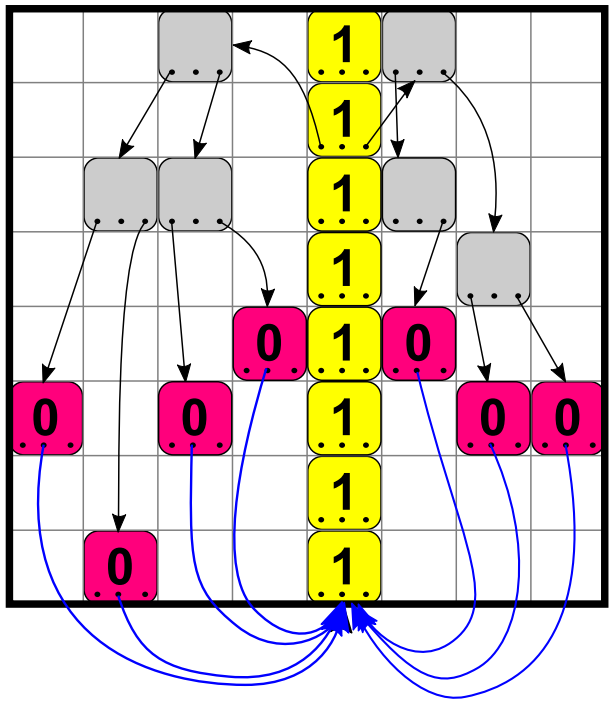
Reminder: Definition (base)



Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.
- Some additional information is contained in the leaves (to be defined).

Definition

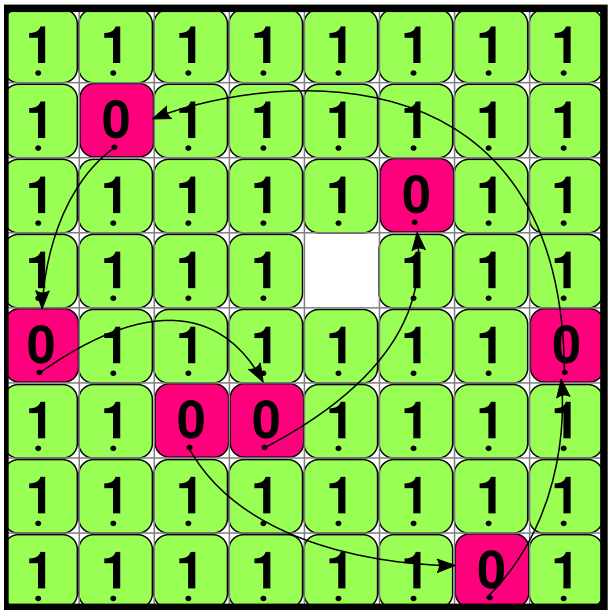


Accept iff

- There is a (unique) all-1 column b;
- in b, there is a unique element r with non-zero pointers;
- for each $j \neq b$, following a path $T(j)$ from r gives a zero in the jth column.
- all the leaves back point to the all-1 column b.

Reminder 2: Adversary Argument

- **While** there are non-queried elements in a column:
  - Return 1.
- **When** the last element in a column is queried:
  - Return 0, linking it to the last returned 0.



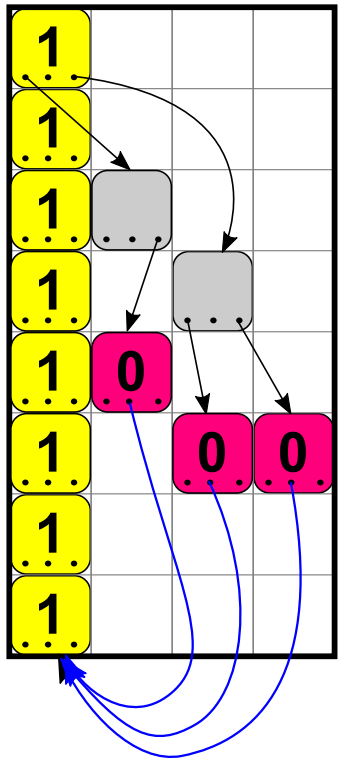
Deterministic Lower Bound

Adversary Method.

Let $n = 2m$.

If the kth element is queried in a column:

- If $k \leq m$, return 1.
- Otherwise, return 0 with back pointer to column $k - m$.



At the end, the column contains m 1 and m 0 with back pointers to all columns $1, 2, \ldots, m$.

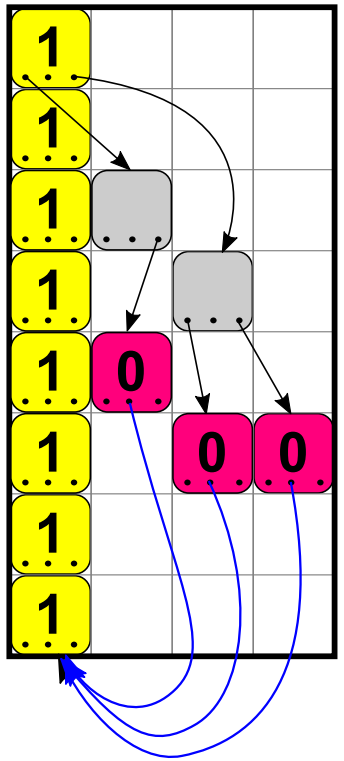
Deterministic Lower Bound

Adversary Method.

Let $n = 2m$.

If the kth element is queried in a column:

- If $k \leq m$, return 1.
- Otherwise, return 0 with back pointer to column $k - m$.



At the end, the column contains m 1 and m 0 with back pointers to all columns $1, 2, \ldots, m$.

- The algorithm does not know the value of the function until it has queried $> m$ elements in each of m columns.

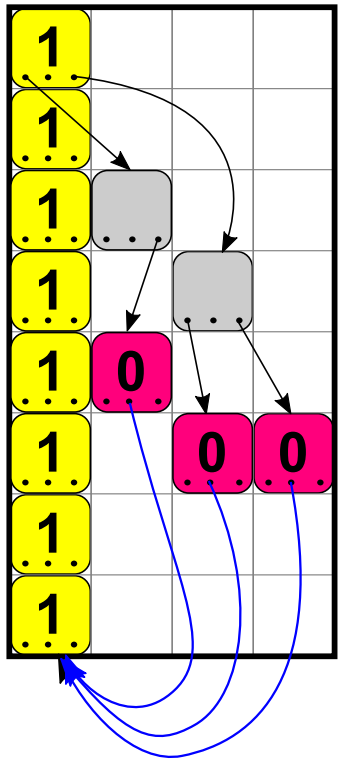
Deterministic Lower Bound

Adversary Method.

Let $n = 2m$.

If the kth element is queried in a column:

- If $k \leq m$, return 1.
- Otherwise, return 0 with back pointer to column $k - m$.

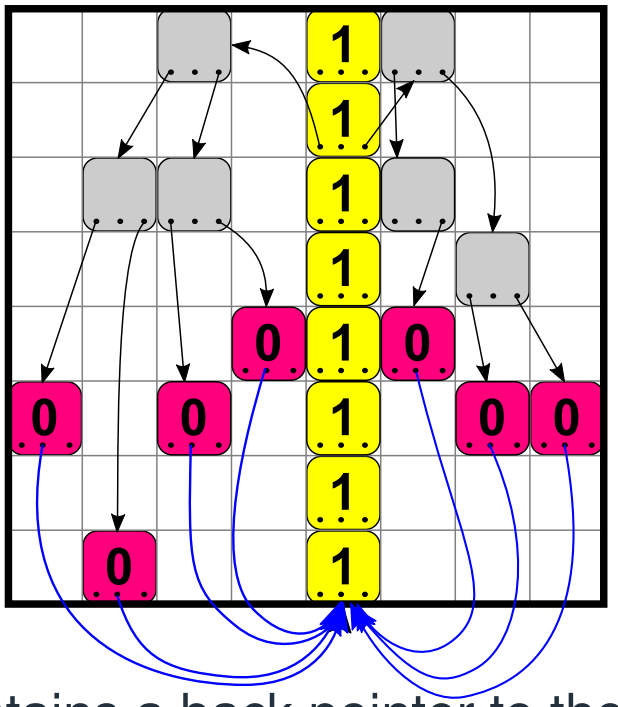


At the end, the column contains m 1 and m 0 with back pointers to all columns $1, 2, \ldots, m$.

- The algorithm does not know the value of the function until it has queried $> m$ elements in each of m columns.

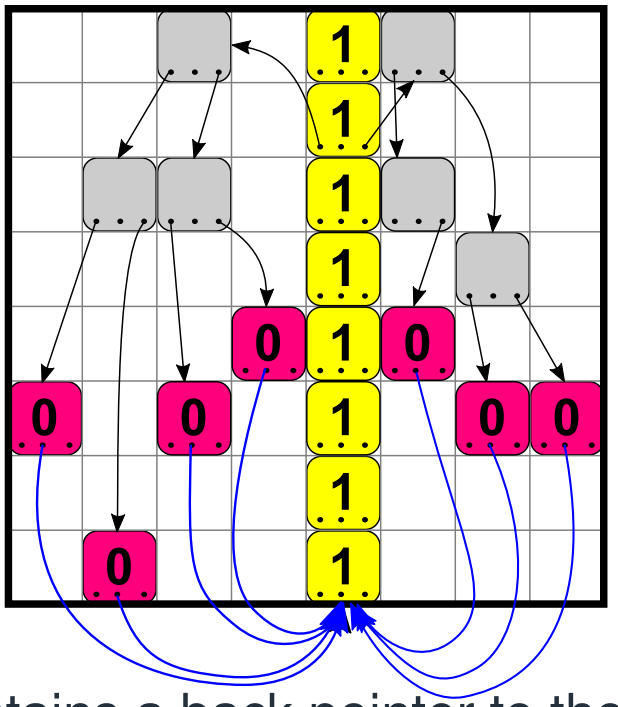
Lower bound: $\Omega(m^2)$.

R_0 Upper Bound: Informal



- Each column contains a back pointer to the all-1 column.
  BUT which one is the right one—?

R_0 Upper Bound: Informal



- Each column contains a back pointer to the all-1 column.
  BUT which one is the right one—?

We try each back pointer by quering few elements in the column, and proceed to a one where no zeroes were found.

- Even if this is not the all-1 column,
  we can arrange that it contains fewer zeroes whp.

R_0 Upper Bound: Formal

Algorithm

- Let c be the first column, and $k \leftarrow n$.
- **While** $k > 1$,
  - Let $c \leftarrow$ProcessColumn(c, k), and $k \leftarrow k/2$.

ProcessColumn(column c, integer k)

- Query all elements in column c.
- **If** there are no zeroes, **verify** column c.
- **If** there are $> k$ zeroes, query all nm variables, and output the value of the function.
- **For** each zero a:
  - Let j be the back pointer of a.
  - Query $\widetilde{O}(n/k)$ elements in column j. (Probability $< \frac{1}{(nm)^2}$ that no zero found if there are $> k/2$ of them).
  - **If** no zero was found, **return** j.
- **Reject**

Summary

Take $n = 2m$.

- Lower bound for a D algorithm is $\Omega(m^2)$.
- Upper bound for a R_0 algorithm is $\widetilde{O}(n + m)$.

We get a quadratic separation between R_0 and D.

Summary

Take $n = 2m$.

- Lower bound for a D algorithm is $\Omega(m^2)$.
- Upper bound for a R_0 algorithm is $\widetilde{O}(n + m)$.

We get a quadratic separation between R_0 and D.

- Also, upper bound for a Q_2 algorithm is $\widetilde{O}(\sqrt{n + m})$.

We get a quartic separation between Q_2 and D.

NB. Previous separation was quadratic: Grover's search.

Conclusion

Results

$$R_1 = \widetilde{O}(R_0^{1/2})$$
$$Q_E = \widetilde{O}(R_0^{1/2})$$
$$R_0 = \widetilde{O}(D^{1/2})$$
$$Q_2 = \widetilde{O}(D^{1/4})$$

$$Q_2 = \widetilde{O}(R_0^{1/3})$$
$$Q_E = \widetilde{O}(R_2^{2/3})$$
$$\widetilde{\deg} = \widetilde{O}(R_2^{1/4})$$

Open Problems

We have resolved $R_2 \leftrightarrow R_0$ and $R_1 \leftrightarrow D$.

Can we resolve $R_2 \leftrightarrow D$ too?
Known: $R_2 = \Omega(D^{1/3})$ and $R_2 = \widetilde{O}(D^{1/2})$.

- Can we overcome the "certificate complexity barrier"?
  Obtain a function with $R_2 = o(C)$?

- The same about $Q_2 \leftrightarrow D$
  Known: $Q_2 = \Omega(D^{1/6})$ and $Q_2 = \widetilde{O}(D^{1/4})$.
- and $Q_E \leftrightarrow D$?
  Known: $Q_E = \Omega(D^{1/3})$ and $Q_E = \widetilde{O}(D^{1/2})$.

Cheat Sheets

Aaronson, Ben-David, and Kothari came up with the Cheat-Sheet technique.

Cheat Sheets

Aaronson, Ben-David, and Kothari came up with the Cheat-Sheet technique.

- also uses pointers
- is incomparable to our results
- prove a number of interesting results, e.g., a total Boolean function f with

$$R_2(f) = \widetilde{\Omega}(Q_2(f)^{2.5}).$$

Aaronson, Ben-David, and Kothari came up with the Cheat-Sheet technique.

- also uses pointers
- is incomparable to our results
- prove a number of interesting results, e.g., a total Boolean function f with

$$R_2(f) = \widetilde{\Omega}(Q_2(f)^{2.5}).$$

- Actually, $R_2(f) = \widetilde{\Omega}(Q_2(f)^3)$, if there exists a partial function g on n variables with

$$Q_2(g) = O(\log n) \qquad \text{and} \qquad R_2(g) = \widetilde{\Omega}(n).$$

Any questions?