The Complexity of Computing the Solution Obtained by a Specific Algorithm

Paul W. Goldberg

Department of Computer Science
University of Oxford, U. K.

LSE
Oct 2013

Starting-point

It is **PSPACE**-complete to find any Nash equilibria of a game, that are computed by the Lemke-Howson algorithm. [1]

[1](Papadimitriou, G, and Savani '11); "no short cuts"
not every solution is a L-H solution; result also applies to some related game-solving algorithms

Starting-point

It is **PSPACE**-complete to find any Nash equilibria of a game, that are computed by the Lemke-Howson algorithm. [1]

So, we have taken a specific (exponential-time) algorithm $\mathcal{A}$ for a specific problem (NASH), and found out the complexity of computing $\mathcal{A}$'s solutions...

[1] (Papadimitriou, G, and Savani '11); "no short cuts"
not every solution is a L-H solution; result also applies to some related game-solving algorithms

Starting-point

It is **PSPACE**-complete to find any Nash equilibria of a game, that are computed by the Lemke-Howson algorithm. [1]

So, we have taken a specific (exponential-time) algorithm $\mathcal{A}$ for a specific problem (NASH), and found out the complexity of computing $\mathcal{A}$'s solutions... NASH is a funny choice of problem, it's believed to be hard but not known to be as hard as **NP**...

[1](Papadimitriou, G, and Savani '11); "no short cuts"
not every solution is a L-H solution; result also applies to some related game-solving algorithms

Starting-point

It is **PSPACE**-complete to find any Nash equilibria of a game, that are computed by the Lemke-Howson algorithm. [1]

So, we have taken a specific (exponential-time) algorithm $\mathcal{A}$ for a specific problem (NASH), and found out the complexity of computing $\mathcal{A}$'s solutions... NASH is a funny choice of problem, it's believed to be hard but not known to be as hard as **NP**... should we care about generalizations of the above result? (to other problems/exp-time algorithms)

[1](Papadimitriou, G, and Savani '11); "no short cuts"
not every solution is a L-H solution; result also applies to some related game-solving algorithms

A more general class of questions

Given problem X and (exp-time) algorithm $\mathcal{A}$ for X, what is the complexity of computing $\mathcal{A}$'s solutions?

A more general class of questions

Given problem X and (exp-time) algorithm $\mathcal{A}$ for X, what is the complexity of computing $\mathcal{A}$'s solutions?

Example: X=SAT, $\mathcal{A}$=lexicographic search

A more general class of questions

Given problem X and (exp-time) algorithm $\mathcal{A}$ for X, what is the complexity of computing $\mathcal{A}$'s solutions?

Example: X=SAT, $\mathcal{A}$=lexicographic search

LEXMINSAT (find the lexicographically min satisfying assignment) is complete for **OptP** (Krentel '88).

Definition

An **OptP** function f_M has associated poly-time non-det TM M; M outputs a binary number at each branch of computation; $f_M(x)$ is largest number for all accepting branches.

"easier" than **PSPACE**

Conjecture (attempt to generalize Slide 1)

Given any **PPAD**-complete problem X, and "path-following" algorithm $\mathcal{A}$ for X, it's **PSPACE**-complete to compute $\mathcal{A}$'s output on instances of X.

- **PPAD**?
- "path-following"?

PPAD

parity argument on a directed graph (Papadimitriou '91):

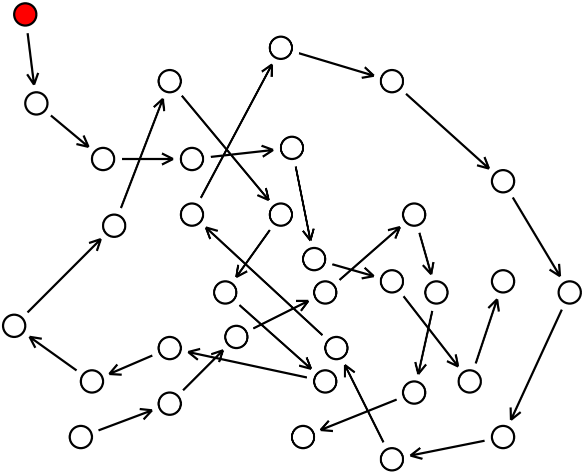
END OF LINE

Given directed graph G of indegree/outdegree at most 1, and a "source" vertex of indegree 0, find another vertex of degree 1. G has vertices $\{0,1\}^n$ and edges represented by boolean circuits S, P.

END OF LINE characterizes **PPAD**; poly-time reductions between NASH and END OF LINE establish **PPAD**-completeness of NASH[2].

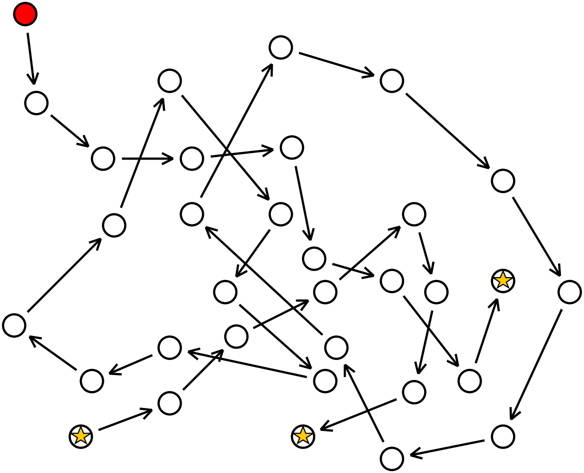
[2] Daskalakis, G, and Papadimitriou '05,'06; Chen, Deng, and Teng '06

END OF LINE graph



You are given a node with degree 1 (colored red here)

END OF LINE graph



The highlighted nodes are **PPAD**-complete to find.

How hard is **PPAD**?

- "between **P** and **NP**"

How hard is **PPAD**?

- "between **P** and **NP**"
- NOT **NP**-complete unless **NP**=co-**NP** (Megiddo'86) since it's an **NP** total search problem (like FACTORING)

How hard is **PPAD**?

- “between **P** and **NP**”
- NOT **NP**-complete unless **NP**=co-**NP** (Megiddo’86) since it’s an **NP** total search problem (like FACTORING)
  (could there be some other way to prove **PPAD** is as hard as **NP**?)

How hard is **PPAD**?

- "between **P** and **NP**"
- NOT **NP**-complete unless **NP**=co-**NP** (Megiddo'86) since it's an **NP** total search problem (like FACTORING)
  (could there be some other way to prove **PPAD** is as hard as **NP**?)
- anyway, it's assumed not solvable in poly-time, based on effort to find a poly-time algorithm, and usage of general boolean circuits in problem instances

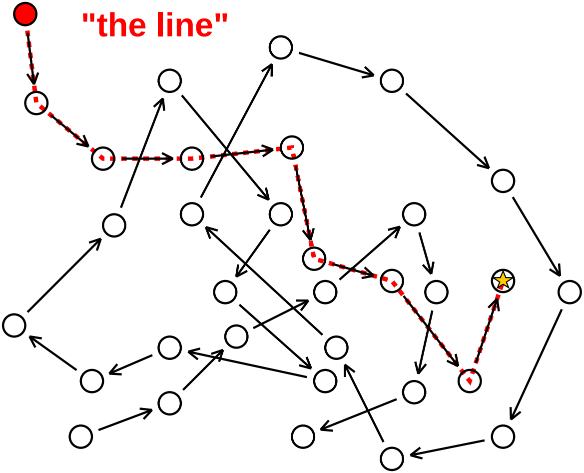
Two "natural" exponential-time algorithms

- lexicographic search
- follow the line

Search for lexicographically-least solution is **OptP**-complete. The search for line-following solution is **PSPACE**-complete!

OTHER END OF THIS LINE (OEOTL) denotes the **PSPACE**-complete search problem.

END OF LINE graph



The node attached to the red node is **PSPACE**-complete to find!

The **PSPACE**-hardness of OEOTL

- The circuits S and P that comprise an instance of END-OF-LINE are like a space-bounded time-reversible TM. (nodes of big graph $\leftrightarrow$ configurations)
- It's **PSPACE**-complete to find the config of a space-n TM after 2^n transitions
- TMs can be made time-reversible[3] (by remembering some of the previous configs, during a computation)

[3] Bennett '73,'89; Crescenzi and Papadimitriou '95 (NTMs: depth-bounded tree-like circuit for NTM $\rightarrow$ (S, P)-graph G; TRUE gates are reachable in G)

- The circuits S and P that comprise an instance of END-OF-LINE are like a space-bounded time-reversible TM. (nodes of big graph $\leftrightarrow$ configurations)
- It's **PSPACE**-complete to find the config of a space-n TM after 2^n transitions
- TMs can be made time-reversible[3] (by remembering some of the previous configs, during a computation)

Slide 1: Lemke-Howson serves as a proxy for generic polynomial-space bounded computation.

[3] Bennett '73,'89; Crescenzi and Papadimitriou '95 (NTMs: depth-bounded tree-like circuit for NTM $\rightarrow$ (S, P)-graph G; TRUE gates are reachable in G)

Path-following algorithms

definition

A *path-following algorithm* for a **PPAD**-complete problem X uses a reduction to convert X to END OF LINE, follows the line, and uses the same reduction to convert that end-of-line to a solution of X.

Lemke-Howson is path-following, so the result of slide 1 is a special case of the path-following algorithms conjecture.

Path-following algorithms

definition

A *path-following algorithm* for a **PPAD**-complete problem X uses a reduction to convert X to END OF LINE, follows the line, and uses the same reduction to convert that end-of-line to a solution of X.

Lemke-Howson is path-following, so the result of slide 1 is a special case of the path-following algorithms conjecture.

Challenge instances for the path-following algorithms conjecture

- X =NASH, $\mathcal{A}$ =Scarf's algorithm
- X =2D-DISCRETE BROUWER, $\mathcal{A}$ = "the natural algorithm"

"Paradox"

PPAD is no harder than **NP** (maybe easier); Lemke-Howson is efficient in practice; *but* it's "harder" to compute the output of Lemke-Howson than the "obviously inefficient" lexicographic search

PPAD easier than **NP**?

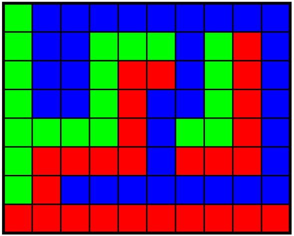
General intuition for the hardness of **PPAD** is that unrestricted boolean circuits are hard to work with...
But note **PPAD** instances have polynomial "query complexity": consider a computationally unbounded algorithm that wants a solution given the circuits S and P and is able to query their input/output behaviour...

2D-DISCRETE BROUWER

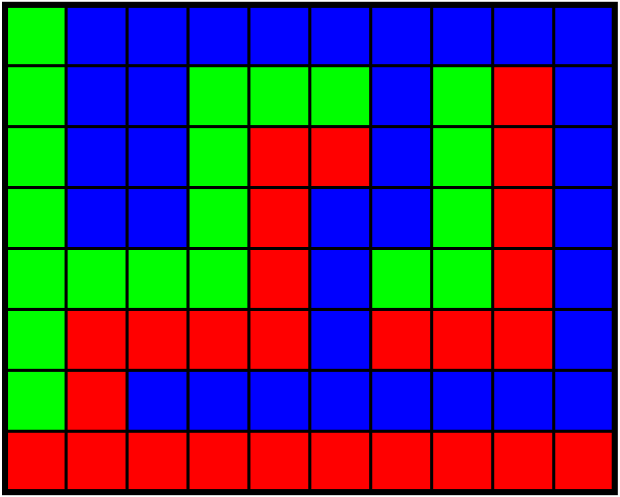
Search for a *panchromatic point* of a *discrete Brouwer function* — in 2D, a function $f : N \times N' \longrightarrow \{0, 1, 2\}$ where

- the bottom row has color 1 (e.g. red)
- the left-hard side has color 2 (e.g. green)
- the top and RHS have color 0 (e.g. blue)
- internal points colored by a poly-size boolean circuit C

Assume N and N' are exponentially large
C maps coordinates to colors

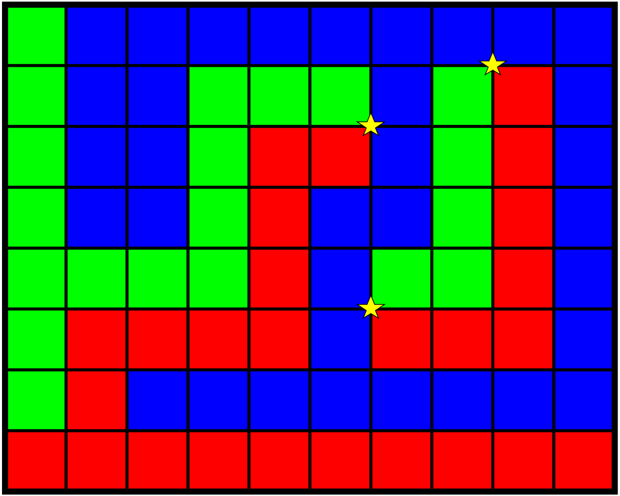


2D-DISCRETE BROUWER example



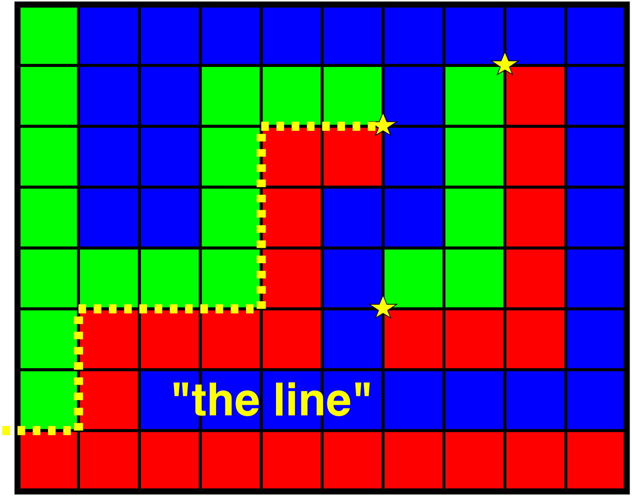
Search for trichromatic point

2D-DISCRETE BROUWER example



Search for trichromatic point... they are **PPAD**-complete to find (Chen and Deng ('06, '09))

The “natural” path-following algorithm



Follow the line! How hard is it to find this solution?

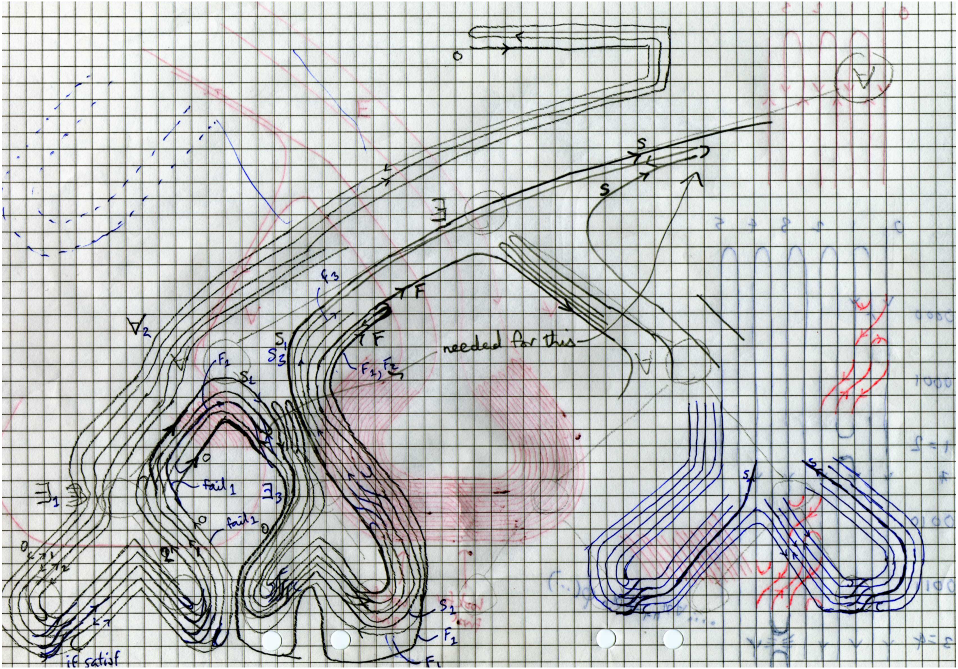
2D DISCRETE BROUWER

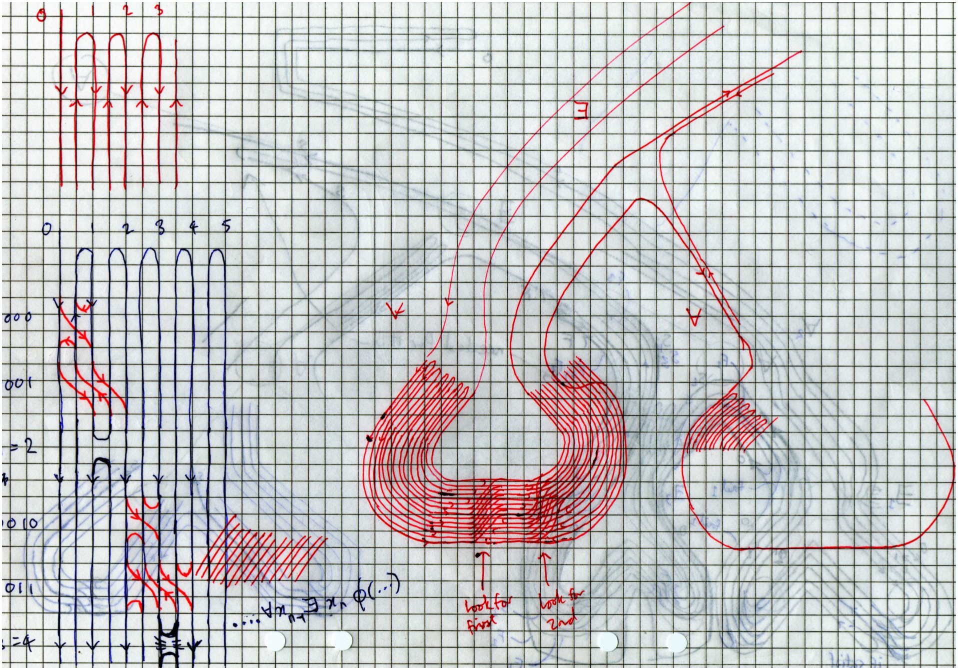
END OF LINE $\leq_p$ 2D-BROUWER (Chen & Deng '06, '09)

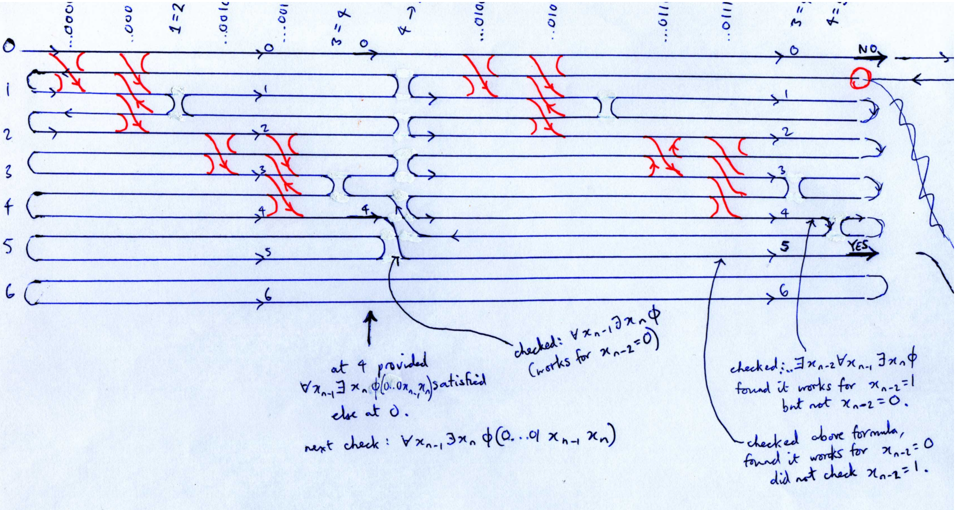
theorem

2D-DISCRETE BROUWER is **PSPACE**-complete, if you want the "natural line-following" solution.

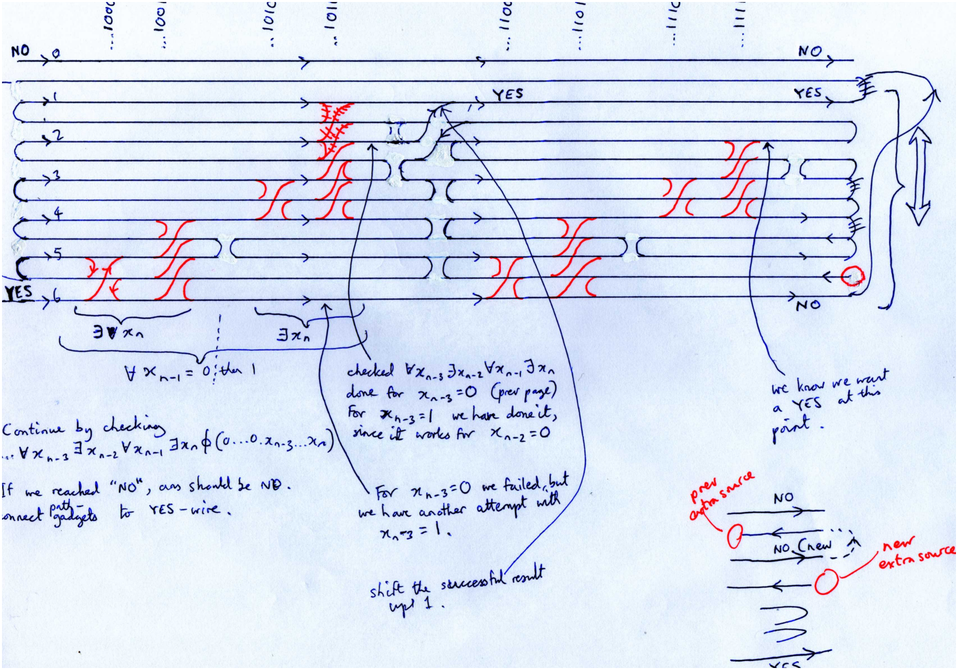
The 3D version is easier; 2D needed more work...


needed for this
if satisf
fail 1
S
F


0 1 2 3
0 1 2 3 4 5
000
001
=2
010
011
=4
∃
∀
∀
look for first
look for 2nd
....∀x_{n-1}∃x_n φ(...)

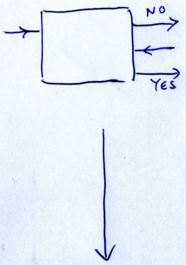


$\ldots \exists x_{n-2} \forall x_{n-1} \exists x_n \, \phi(\ldots)$

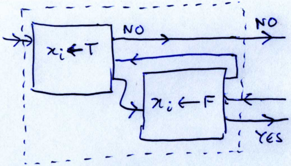


...1000
...1001
...1010
...1011
...1100
...1101
...1110
...1111
NO
YES
NO
YES
YES
NO
0
1
2
3
4
5
6
∃∀x_n
∃x_n
∀x_{n-1} = 0 then 1
checked ∀x_{n-3} ∃x_{n-2} ∀x_{n-1} ∃x_n
done for x_{n-3} = 0 (prev page)
For x_{n-3} = 1 we have done it,
since it works for x_{n-2} = 0
we know we want
a YES at this
point.
Continue by checking
... ∀x_{n-3} ∃x_{n-2} ∀x_{n-1} ∃x_n φ(0...0.x_{n-3}...x_n)
If we reached "NO", ans should be NO.
connect path-gadgets to YES-wire.
For x_{n-3} = 0 we failed, but
we have another attempt with
x_{n-3} = 1.
shift the successful result
up 1.
prev extra source
NO
NO (new
new extra source
YES

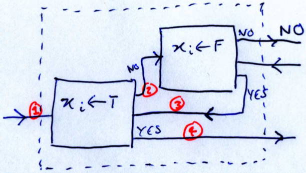
subformulas corresp. to some variables fixed



$\forall x_i$



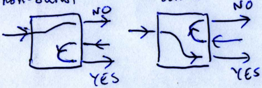
$\exists x_i$



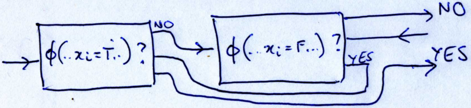
At point ③ I will exit at ④. This is because ① connects to ②, & there are no loose ends in the $\boxed{x_i \leftarrow T}$ circuit.

single-variable x_i (all others fixed)

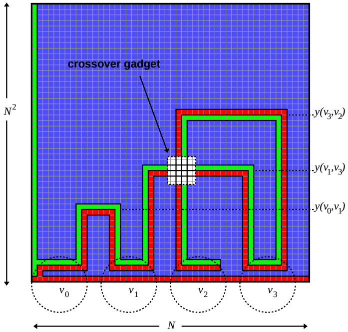
non-satisf satisf



$\exists x_i \, \phi(\ldots)$ (other vars fixed)

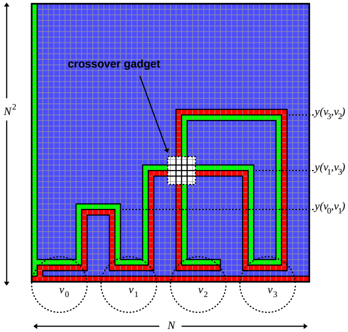


instances of 2D-DISCRETE BROUWER generated by specific reductions from END OF LINE



challenge instance (refined) for path-following conjecture

instances of 2D-DISCRETE BROUWER generated by specific reductions from END OF LINE



"theorem"

For 2D-DISCRETE BROUWER instances generated by Chen-Deng reduction, it is $\#\mathbf{P}$ hard to compute the "natural line-following algorithm" solution.

but the above is just for one specific **PPAD**-complete class of instances — we have a long way to go...