The Complexity of Gradient Descent: CLS = PPAD ∩ PLS

John Fearnley[1] Paul Goldberg[2] Alexandros Hollender[2]

Rahul Savani[1]

[1]University of Liverpool

[2]University of Oxford

Gradient descent

$$\textbf{minimise } f(x) \quad \text{s.t.} \quad x \in [0,1]^n$$

assume f continuously differentiable, but not necessarily convex

Gradient descent

$$\textbf{minimise } f(x) \quad \text{s.t.} \quad x \in [0,1]^n$$

NP-hard even for a quadratic polynomial given explicitly

Gradient descent

minimise $f(x)$ s.t. $x \in [0,1]^n$ **NP−hard**

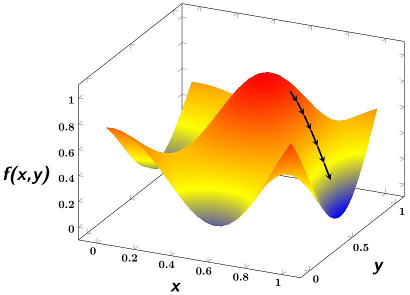
Gradient Descent: $x_{k+1} \leftarrow x_k - \eta\nabla(f(x_k))$ **(η : step size)**

Intuition: “move in the direction of steepest descent”

Gradient descent

(1) : **minimise $f(x)$** s.t. $x \in [0,1]^n$ **NP−hard**

Gradient Descent: $x_{k+1} \leftarrow x_k - \eta\nabla(f(x_k))$ **(η : step size)**



Gradient descent being applied to a function $f : [0,1]^2 \mapsto [0,1]$

Gradient descent

(1) : **minimise** $\boldsymbol{f(x)}$ s.t. $\boldsymbol{x \in [0,1]^n}$ **NP−hard**

Gradient Descent: $\boldsymbol{x_{k+1} \leftarrow x_k - \eta\nabla(f(x_k))}$ **(**η **: step size)**

Doesn't actually solve (1); can get stuck in any **stationary point**

Gradient descent

minimise $f(x)$ s.t. $x \in [0,1]^n$ **NP−hard**

Gradient Descent: $x_{k+1} \leftarrow x_k - \eta \nabla(f(x_k))$ (η **: step size)**

Doesn't actually solve (1); can get stuck in any **stationary point**

actually a Karush-Kuhn-Tucker point (due to boundaries)

Gradient descent

minimise $f(x)$ s.t. $x \in [0,1]^n$ **NP–hard**

Gradient Descent: $x_{k+1} \leftarrow x_k - \eta \nabla(f(x_k))$ **(**η **: step size)**

What is the complexity of finding a solution where gradient descent terminates?

Gradient descent

minimise $f(x)$ s.t. $x \in [0,1]^n$ **NP−hard**

Gradient Descent: $x_{k+1} \leftarrow x_k - \eta\nabla(f(x_k))$ **(**η **: step size)**

What is the complexity of finding a solution where gradient descent terminates?

Let's explore how to formalise this...

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

$[x' := x - \eta \nabla p(x))]$

GD-Local-Search: find x s.t. $f(x') \geq f(x) - \epsilon$

limited improvement

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

GD-Local-Search: find x s.t. $f(x') \geq f(x) - \epsilon$

limited improvement

GD-Fixed-Point: find x s.t. $\|x' - x\| \leq \epsilon$

x not moved by much

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

GD-Local-Search: find x s.t. $f(x') \geq f(x) - \epsilon$

limited improvement

GD-Fixed-Point: find x s.t. $\|x' - x\| \leq \epsilon$

x not moved by much

These two problems are **polynomial-time equivalent**

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

One way to solve this problem: **run Gradient Descent**!

Running time: **polynomial in $1/\epsilon$, not in input size**

Gradient descent problem

Input: C^1 function $f : [0,1]^n \mapsto \mathbb{R}$, stepsize $\eta > 0$, precision $\epsilon > 0$
(f and ∇f given as arithmetic circuits)

Goal: find a point where gradient descent terminates

Can it be solved in time polynomial in $\log(1/\epsilon)$?

(f convex: yes, e.g., via the Ellipsoid method)

Total search problems

A search problem is **total** if a solution is **guaranteed to exist**

Examples:

- **NASH:**
  Find a mixed Nash equilibrium of a game
- **PURE-CONGESTION:**
  Find a pure Nash equilibrium of a congestion game

Total search problems

A search problem is **total** if a solution is **guaranteed to exist**

Examples:

- **NASH:**
  Find a mixed Nash equilibrium of a game
- **PURE-CONGESTION:**
  Find a pure Nash equilibrium of a congestion game
- **FACTORING:**
  Find a prime factor of a number ≥ 2
- **BROUWER:**
  Find a fixed point of a continuous function $f : [0,1]^3 \mapsto [0,1]^3$

Total search problems

A search problem is **total** if a solution is **guaranteed to exist**

Examples:

- **NASH:**
  Find a mixed Nash equilibrium of a game
- **PURE-CONGESTION:**
  Find a pure Nash equilibrium of a congestion game
- **FACTORING:**
  Find a prime factor of a number ≥ 2
- **BROUWER:**
  Find a fixed point of a continuous function $f : [0,1]^3 \mapsto [0,1]^3$
- **GRADIENT-DESCENT**

NP Total Search Problems (TFNP)

NASH, **PURE-CONGESTION**, **FACTORING**, **BROUWER**, **GRADIENT-DESCENT**, . . .

In addition to being total, these problems have more in common:

They are **NP** function problems with **easy-to-verify solutions**

NP Total Search Problems (TFNP)

NASH, **PURE-CONGESTION**, **FACTORING**, **BROUWER**, **GRADIENT-DESCENT**, . . .

In addition to being total, these problems have more in common:

They are **NP** function problems with **easy-to-verify solutions**

Can a **TFNP** problem be **NP**-hard?

NP Total Search Problems (TFNP)

NASH, **PURE-CONGESTION**, **FACTORING**, **BROUWER**, **GRADIENT-DESCENT**, . . .

In addition to being total, these problems have more in common:

They are **NP** function problems with **easy-to-verify solutions**

Can a **TFNP** problem be **NP**-hard? Not unless **NP = co-NP ...**

[Megiddo-Papadimitriou, 1991]

NP Total Search Problems (TFNP)

NASH, **PURE-CONGESTION**, **FACTORING**, **BROUWER**, **GRADIENT-DESCENT**, . . .

In addition to being total, these problems have more in common:

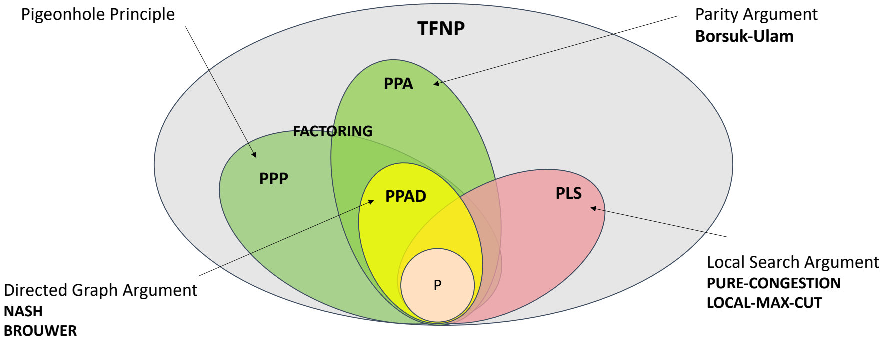
They are **NP** function problems with **easy-to-verify solutions**

Can a **TFNP** problem be **NP**-hard? Not unless **NP = co-NP ...**

[Megiddo-Papadimitriou, 1991]

It is believed that **TFNP** does **not** have complete problems

TFNP Landscape

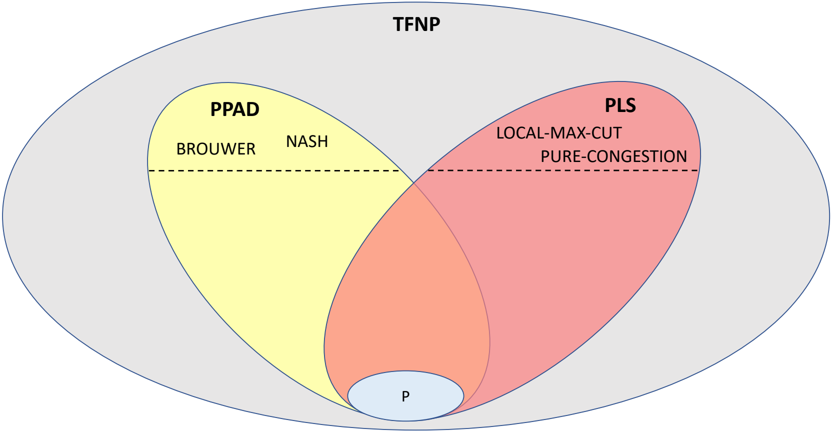


TFNP subclasses

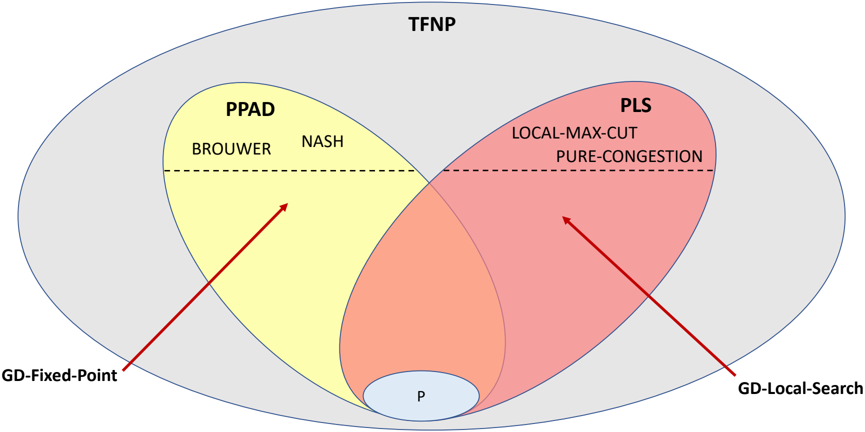
Why believe that **PPAD** $\neq$ **P**, **PLS** $\neq$ **P**, etc. ?

- many seemingly hard problems lie in **PPAD**, **PLS**, ...
- oracle separations (in particular **PPAD** $\neq$ **PLS**)
- hard under cryptographic assumptions

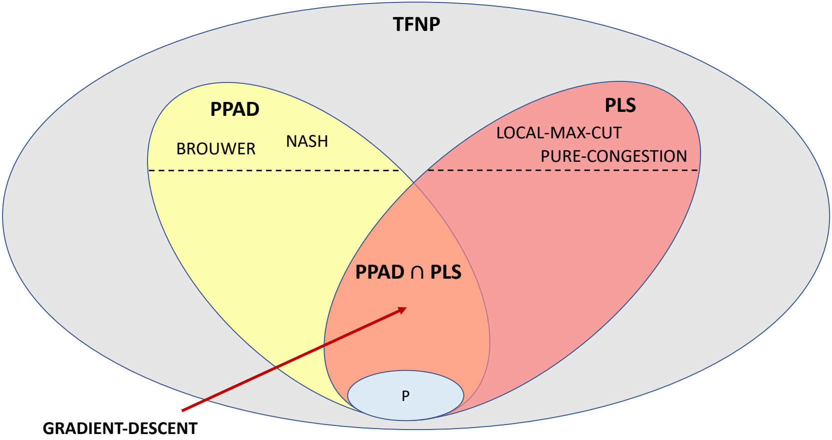
PPAD ∩ PLS



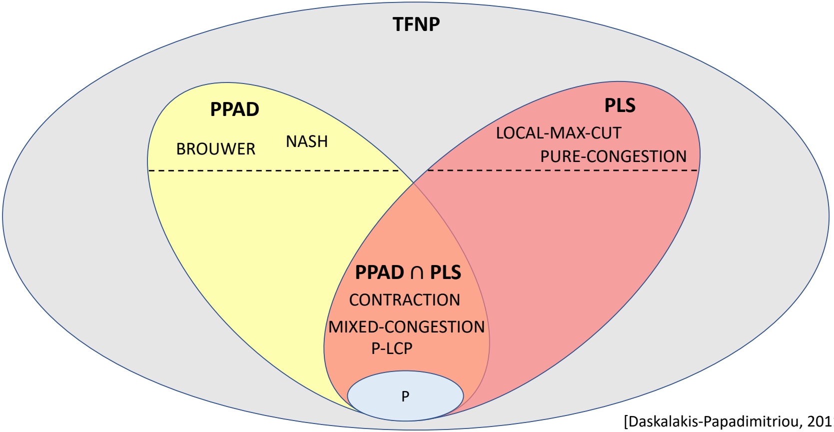
PPAD ∩ PLS


TFNP
PPAD
BROUWER
NASH
PLS
LOCAL-MAX-CUT
PURE-CONGESTION
P
GD-Fixed-Point
GD-Local-Search

PPAD ∩ PLS


TFNP
PPAD
BROUWER
NASH
PLS
LOCAL-MAX-CUT
PURE-CONGESTION
PPAD ∩ PLS
P
GRADIENT-DESCENT

PPAD ∩ PLS



[Daskalakis-Papadimitriou, 2011]

Unlikely containments

Consider a problem $\boldsymbol{A}$ in PPAD $\cap$ PLS

Since $\boldsymbol{A}$ is in both classes:

- If $\boldsymbol{A}$ is PPAD-hard then PPAD $\subseteq$ PLS
- If $\boldsymbol{A}$ is PLS-hard then PLS $\subseteq$ PPAD

Unlikely containments

Consider a problem $\boldsymbol{A}$ in PPAD $\cap$ PLS

Since $\boldsymbol{A}$ is in both classes:

- If $\boldsymbol{A}$ is PPAD-hard then PPAD $\subseteq$ PLS
- If $\boldsymbol{A}$ is PLS-hard then PLS $\subseteq$ PPAD

We do not believe that either containments holds, so
we do not believe *A* is PPAD-hard or PLS-hard

PPAD ∩ PLS seems unnatural...

Suppose problem A is **PPAD**-complete

Suppose problem B is **PLS**-complete

The following problem is **PPAD ∩ PLS**-complete:

EITHER(A,B)

Input: an instance I_A of A, an instance I_B of B

Output: a solution of I_A, or a solution of I_B

PPAD ∩ PLS seems unnatural...

BROUWER (PPAD-complete):
Input: continuous function $f : [0, 1]^3 \mapsto [0, 1]^3$, precision $\epsilon > 0$
Output: approximate fixpoint x:

$$||f(x) - x|| \leq \epsilon$$

PPAD ∩ PLS seems unnatural...

BROUWER (PPAD-complete):
Input: continuous function $f : [0,1]^3 \mapsto [0,1]^3$, precision $\epsilon > 0$
Output: approximate fixpoint x:

$$\|f(x) - x\| \leq \epsilon$$

LOCAL-OPT (PLS-complete):
Input: continuous function $p : [0,1]^3 \mapsto [0,1]$, (non-continuous) function $g : [0,1]^3 \mapsto [0,1]^3$, precision $\epsilon > 0$
Output: local minimum x of p w.r.t. g:

$$p(g(x)) \geq p(x) - \epsilon$$

PPAD ∩ PLS seems unnatural...

BROUWER (PPAD-complete):
Input: continuous function $\boldsymbol{f} : [0,1]^3 \mapsto [0,1]^3$, precision $\epsilon > 0$
Output: approximate fixpoint $\boldsymbol{x}$:

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}\| \leq \epsilon$$

LOCAL-OPT (PLS-complete):
Input: continuous function $\boldsymbol{p} : [0,1]^3 \mapsto [0,1]$, (non-continuous) function $\boldsymbol{g} : [0,1]^3 \mapsto [0,1]^3$, precision $\epsilon > 0$
Output: local minimum $\boldsymbol{x}$ of $\boldsymbol{p}$ w.r.t. $\boldsymbol{g}$:

$$\boldsymbol{p}(\boldsymbol{g}(\boldsymbol{x})) \geq \boldsymbol{p}(\boldsymbol{x}) - \epsilon$$

EITHER(BROUWER,LOCAL-OPT) is **PPAD** ∩ **PLS**-complete

Continuous Local Search (CLS)

Daskalakis & Papadimitriou [SODA 2011] defined a new class via:

CONTINUOUS-LOCAL-OPT

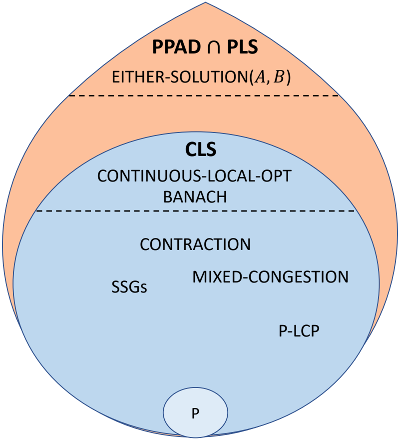
Input:
continuous $p : [0,1]^3 \mapsto [0,1]$ and
continuous $f : [0,1]^3 \mapsto [0,1]^3$, precision $\epsilon > 0$

Output: local minimum x of p w.r.t. f:

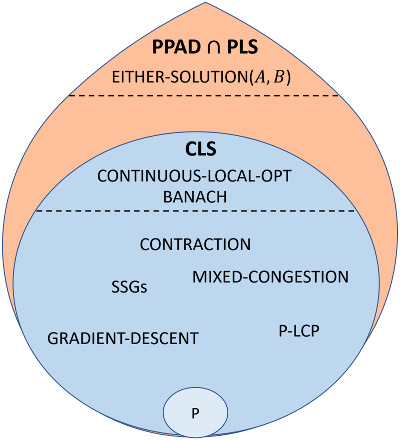
$$p(f(x)) \geq p(x) - \epsilon$$

CLS is the class of all problems that are polynomial-time reducible to **CONTINUOUS-LOCAL-OPT**

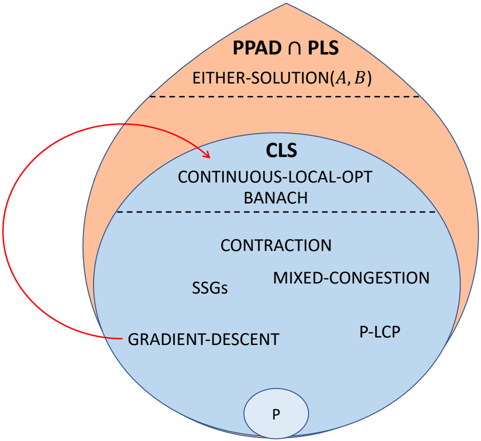
PPAD ∩ PLS and CLS



PPAD ∩ PLS and CLS

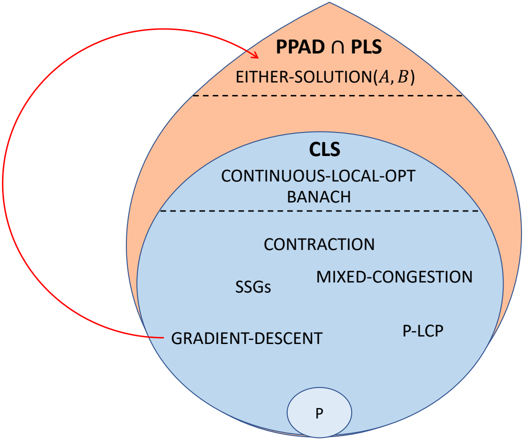


Collapse

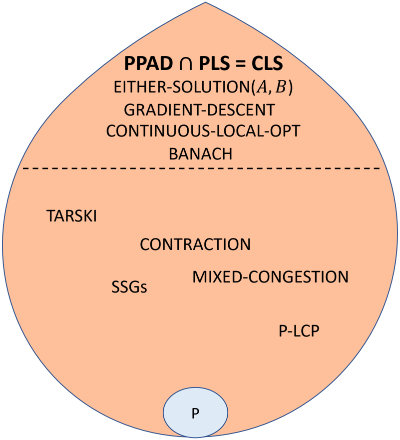


PPAD ∩ PLS
EITHER-SOLUTION(A, B)
CLS
CONTINUOUS-LOCAL-OPT
BANACH
CONTRACTION
SSGs
MIXED-CONGESTION
GRADIENT-DESCENT
P-LCP
P

Collapse


PPAD ∩ PLS
EITHER-SOLUTION(A, B)
CLS
CONTINUOUS-LOCAL-OPT
BANACH
CONTRACTION
SSGs
MIXED-CONGESTION
GRADIENT-DESCENT
P-LCP
P

Collapse



PPAD ∩ PLS = CLS
EITHER-SOLUTION(A, B)
GRADIENT-DESCENT
CONTINUOUS-LOCAL-OPT
BANACH
TARSKI
CONTRACTION
SSGs
MIXED-CONGESTION
P-LCP
P

Main Result

GRADIENT-DESCENT is **PPAD** $\cap$ **PLS** – hard

Main Result

Reduction from **EITHER(A, B)** to **2D-GRADIENT-DESCENT**

where

A is the **PPAD**-complete problem **End-of-Line**
B is the **PLS**-complete problem **ITER**

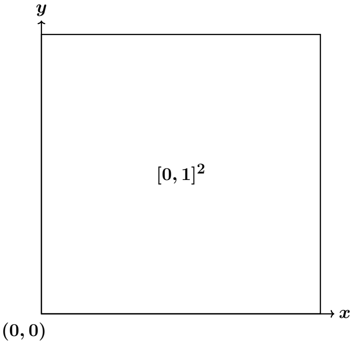
Proof Sketch

Reduction from **EITHER(A, B)** to **2D-GRADIENT-DESCENT**
where
A is the **PPAD**-complete problem **End-of-Line**
B is the **PLS**-complete problem **ITER**

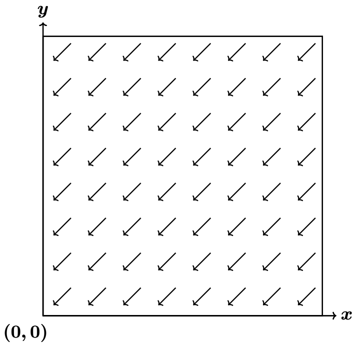
Constructing a 2D-GRADIENT-DESCENT instance *f*

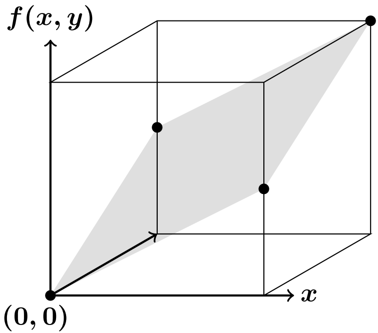
- Domain is the **square** $[0, 1]^2$
- Overlay grid and **assign values for *f* and ∇f at grid points**
- Use **bicubic interpolation** to produce smooth function
- All **stationary points** are either **End-Of-Line** or **ITER** solutions

Background “landscape”

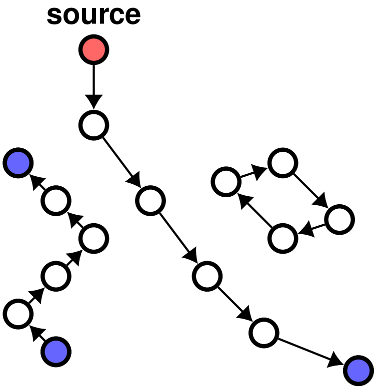


Background “landscape”





PPAD-complete problem: End-Of-Line

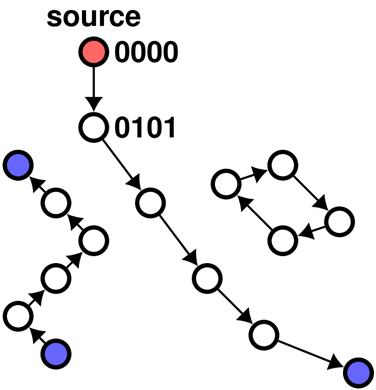


Given a graph of
indegree/outdegree at most 1

and a **source**
(indegree 0, outdegree 1)

find another vertex of degree 1

PPAD-complete problem: End-Of-Line



Catch:

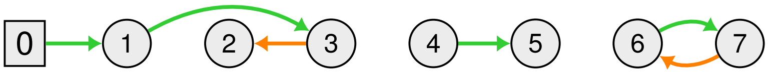
graph is **exponentially large**

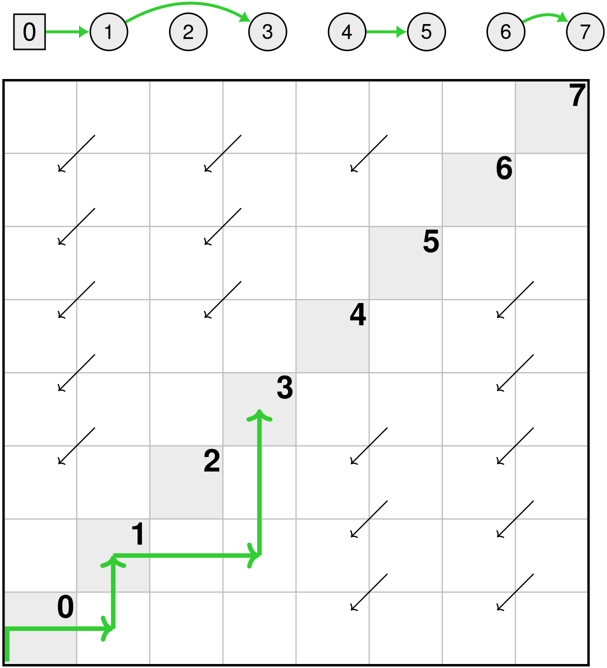
defined by boolean circuits $\boldsymbol{S}$, $\boldsymbol{P}$ that map a vertex $\{\mathbf{0},\mathbf{1}\}^n$ to its successor and predecessor

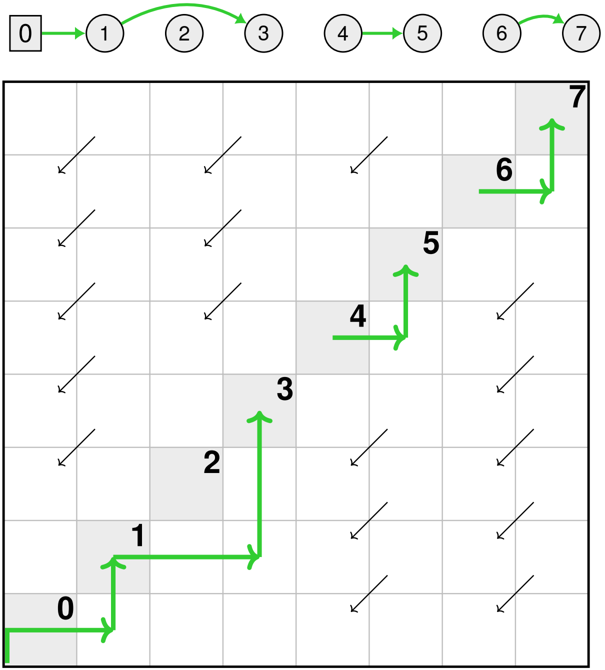
$$\boldsymbol{S}(\mathbf{0000}) = \mathbf{0101}$$

$$\boldsymbol{P}(\mathbf{0101}) = \mathbf{0000}$$

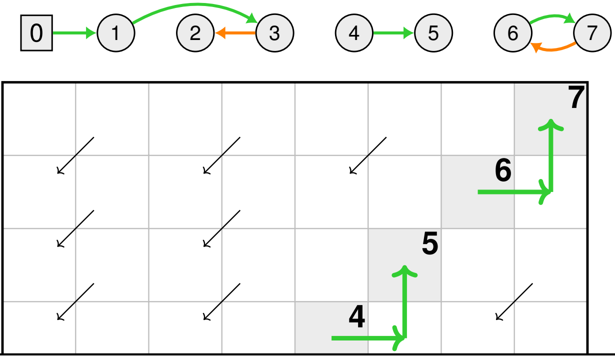
PPAD-complete problem: End-Of-Line



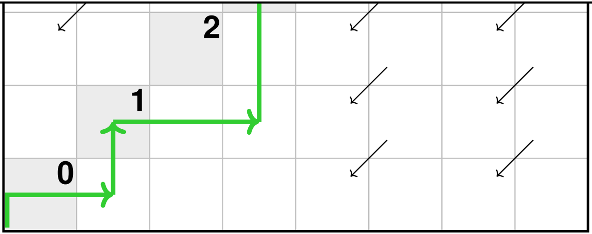

0
1
2
3
4
5
6
7

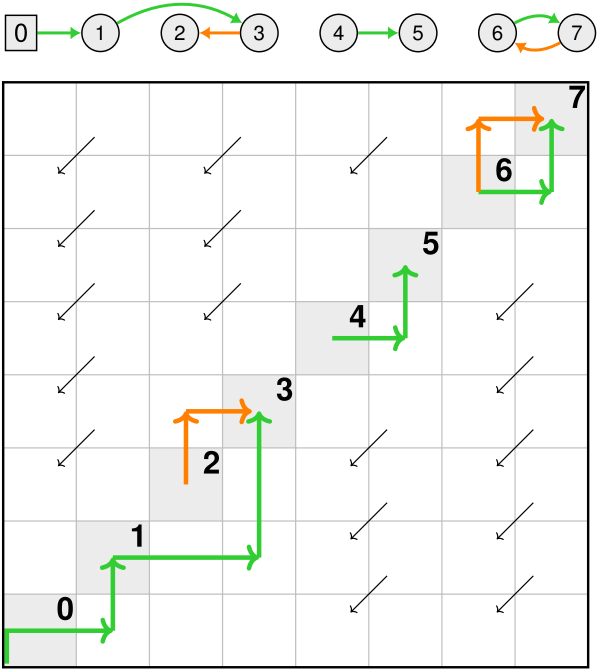


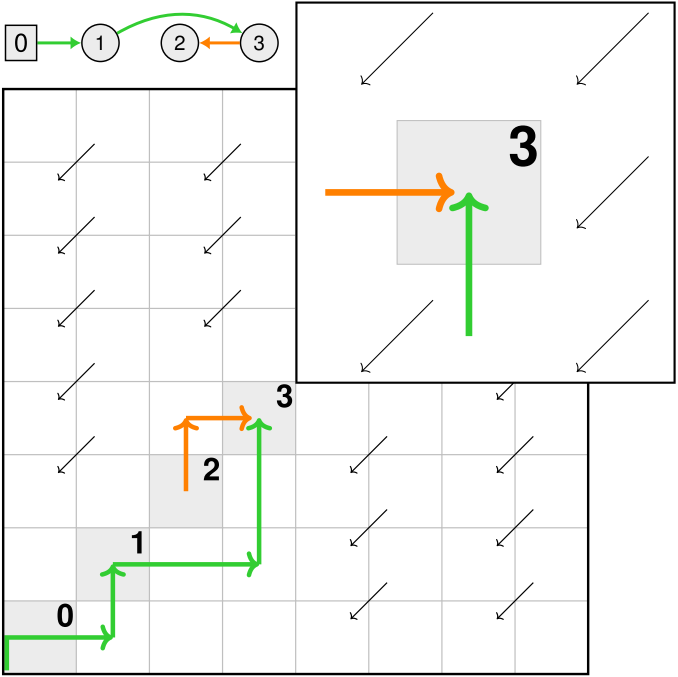
0
1
2
3
4
5
6
7



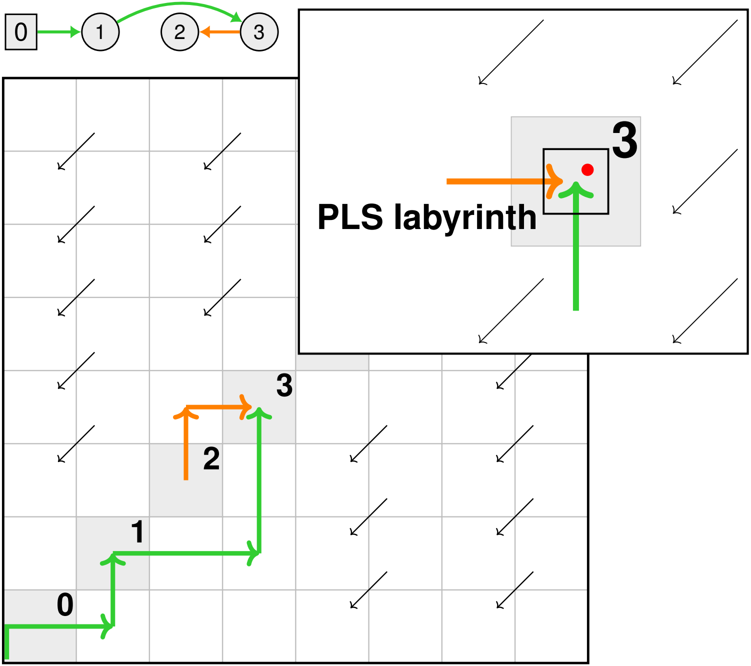
Locally-computable **green** paths: **Hubáček and Yogev SODA’17**
(used to show conditional hardness of CLS)



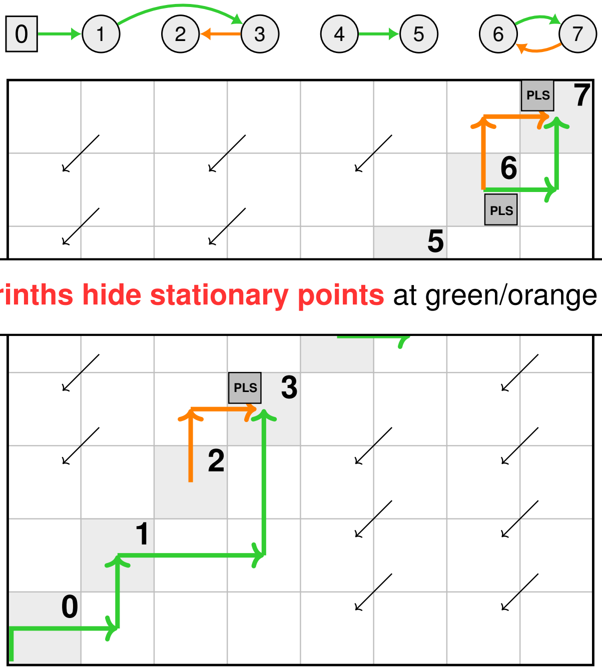

0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7



0
1
2
3
0
1
2
3
3

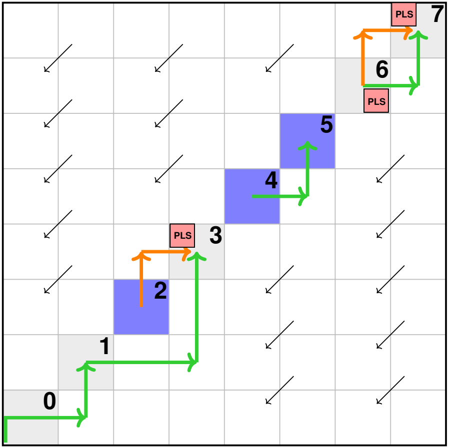


0
1
2
3
PLS labyrinth
3
0
1
2
3



PLS labyrinths hide stationary points at green/orange meetings

All stationary points are:
solutions of **End-of-Line** instance; or
solutions of **PLS-complete** labyrinth



We have shown: **2D-GRADIENT-DESCENT** is **PPAD ∩ PLS** – hard

Take home message: PPAD ∩ PLS

Before:

- **PPAD** and **PLS** both successful classes
- **PPAD ∩ PLS** not believed to have interesting complete problems
- **CLS** introduced as “natural” (presumed distinct) counterpart

Now:

- **PPAD ∩ PLS** is a **natural class with complete problems**
- Captures complexity of problems solved by **gradient descent**
- **PPAD ∩ PLS = CLS**
- **Many important problems are now candidates for hardness**

Open Problems

The following are candidates for **PPAD** $\cap$ **PLS**-completeness:

- **POLYNOMIAL-KKT**
- **MIXED-CONGESTION**
- **CONTRACTION**
- **TARSKI**
- **COLORFUL-CARATHEODORY**

Open Problems

The following are candidates for **PPAD ∩ PLS**-completeness:

- ~~**POLYNOMIAL-KKT**~~
- ~~**MIXED-CONGESTION**~~ [Babichenko, Rubinstein STOC'21]
- **POLYNOMIAL-KKT for degree < 5**
- **MIXED-NETWORK-CONGESTION**
- **CONTRACTION**
- **TARSKI**
- **COLORFUL-CARATHEODORY**

Thank you!