Computational Complexity of Games

Kyle Burke

October 23, 2017

The Big Question

- Does Left have a winning move going first?

The Big Question

- Does Left have a winning move going first?
- i.e. $G \in \mathcal{N} \cup \mathcal{L}$?

The Big Question

- Does Left have a winning move going first?
- i.e. $G \in \mathcal{N} \cup \mathcal{L}$?
- CS: Fastest algorithm to answer this for entire ruleset?

Computational Counting

- Count time as algorithmic steps.

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation
  - $10n^2 + 37.4n + 124$

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation
  - $10n^2 + 37.4n + 124 \rightarrow 10n^2$

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation
  - $10n^2 + 37.4n + 124 \to 10n^2 \to n^2$

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation
  - $10n^2 + 37.4n + 124 \rightarrow 10n^2 \rightarrow n^2$
  - $O(101n^2) = O(n^2)$

Computational Counting

- Count time as algorithmic steps.
- Cheat: use Big-O notation
  - $10n^2 + 37.4n + 124 \rightarrow 10n^2 \rightarrow n^2$
  - $O(101n^2) = O(n^2)$
  - $O(n^3 + n^2 + n + 50) = O(n^3)$

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ $\oslash$

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ $\oslash$
- $O(4^n)$ $\oslash$

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ ⊘
- $O(4^n)$ ⊘
- $O(n!)$ ⊘

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ $\oslash$
- $O(4^n)$ $\oslash$
- $O(n!)$ $\oslash$

- Top 3: P (Not $\mathcal{P}$)

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ $\oslash$
- $O(4^n)$ $\oslash$
- $O(n!)$ $\oslash$

- Top 3: P (Not $\mathcal{P}$)
- Bottom 3: $\mathrm{EXPTIME}$ ($\mathrm{P} \subsetneq \mathrm{EXPTIME}$)

What is Tractible ("Easy")?

Runs in steps polynomial in input size (parameters).

- $O(n^2 m)$ ✓
- $O(n^2 m^6 p^{1.5})$ ✓
- $O(n^{88})$ ✓
- $O(2^n)$ $\oslash$
- $O(4^n)$ $\oslash$
- $O(n!)$ $\oslash$

- Top 3: P (Not $\mathcal{P}$)
- Bottom 3: $\mathrm{EXPTIME}$ ($\mathrm{P} \subsetneq \mathrm{EXPTIME}$)

Note: No fixed-size rulesets!

Some Games are Easy!

- Brussels Sprouts

Some Games are Easy!

- BRUSSELS SPROUTS
  - n crosses, k edges

Some Games are Easy!

- Brussels Sprouts
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even

Some Games are Easy!

- Brussels Sprouts
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even
  - $O(1)$ steps

Some Games are Easy!

- Brussels Sprouts
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even
  - $O(1)$ steps
- Nim

Some Games are Easy!

- Brussels Sprouts
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even
  - $O(1)$ steps
- Nim
  - n piles, each up to m sticks.

Some Games are Easy!

- BRUSSELS SPROUTS
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even
  - $O(1)$ steps
- NIM
  - n piles, each up to m sticks.
  - $\mathcal{P} \Leftrightarrow$ Nim-sum is zero.

Some Games are Easy!

- Brussels Sprouts
  - n crosses, k edges
  - $\mathcal{P} \Leftrightarrow n$ is even
  - $O(1)$ steps
- Nim
  - n piles, each up to m sticks.
  - $\mathcal{P} \Leftrightarrow$ Nim-sum is zero.
  - $O(n \log(m))$ steps.

Some Initial Positions Are Easy!

- CHOMP

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start

Some Initial Positions Are Easy!

- Chomp
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- Cram
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start
  - $O(4nm) = O(nm)$ otherwise

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start
  - $O(4nm) = O(nm)$ otherwise
- SPROUTS (maybe)

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start
  - $O(4nm) = O(nm)$ otherwise
- SPROUTS (maybe)
  - Sprouts Conjecture: $n \bmod 6 \leq 2 \iff \mathcal{P}$

Some Initial Positions Are Easy!

- CHOMP
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- CRAM
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start
  - $O(4nm) = O(nm)$ otherwise
- SPROUTS (maybe)
  - Sprouts Conjecture: $n \bmod 6 \leq 2 \iff \mathcal{P}$
  - $O(\log(n))$ if I know it's a start

Some Initial Positions Are Easy!

- Chomp
  - $G \in \mathcal{N}$ for any $n \times m$ rectangle bigger than 1×1
  - $O(1)$ if I know it's start
  - $O(nm)$ otherwise
- Cram
  - $G \in \mathcal{P}$ for any $2n \times 2m$ rectangle.
  - $O(1)$ if I know it's start
  - $O(4nm) = O(nm)$ otherwise
- Sprouts (maybe)
  - Sprouts Conjecture: $n \bmod 6 \leq 2 \iff \mathcal{P}$
  - $O(\log(n))$ if I know it's a start
  - (otherwise unknown)

General Algorithm (A) for Short Games

For any G, $A(G)$:

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree
- All the leaves are in $\mathcal{P}$.

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree
- All the leaves are in $\mathcal{P}$.
- Use CGT rules to outcome classes all the way back up the tree until you evaluate G.

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree
- All the leaves are in $\mathcal{P}$.
- Use CGT rules to outcome classes all the way back up the tree until you evaluate G.
- Return whether $G \in \mathcal{N} \cup \mathcal{L}$.

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree
- All the leaves are in $\mathcal{P}$.
- Use CGT rules to outcome classes all the way back up the tree until you evaluate G.
- Return whether $G \in \mathcal{N} \cup \mathcal{L}$.

Worst case: have to evaluate all nodes of the game tree.

General Algorithm (A) for Short Games

For any G, $A(G)$:

- Draw the Game Tree
- All the leaves are in $\mathcal{P}$.
- Use CGT rules to outcome classes all the way back up the tree until you evaluate G.
- Return whether $G \in \mathcal{N} \cup \mathcal{L}$.

Worst case: have to evaluate all nodes of the game tree. ... so A uses exponential time.

Completeness

Problems where the *best* algorithm both:

Completeness

Problems where the *best* algorithm both:

- Runs in at most exponential time ("inclusion": in $\mathrm{EXPTIME}$) and

Completeness

Problems where the *best* algorithm both:

- Runs in at most exponential time ("inclusion": in $\mathrm{EXPTIME}$) and
- Requires exponential time in the worst cases ("hardness": $\mathrm{EXPTIME}$-hard)

Completeness

Problems where the *best* algorithm both:

- Runs in at most exponential time ("inclusion": in $\mathrm{EXPTIME}$) and
- Requires exponential time in the worst cases ("hardness": $\mathrm{EXPTIME}$-hard)

... are $\mathrm{EXPTIME}$-complete.

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

---

[1] Fraenkel, Lichtenstein - http://www.sciencedirect.com/science/article/pii/0097316581900169

[2] Hearn, Demaine - http://erikdemaine.org/papers/GPC/

[3] Robson, "The Complexity of Go", IFIP Congress 1983

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

- (Generalized) CHESS[1]

[1] Fraenkel, Lichtenstein - http://www.sciencedirect.com/science/article/pii/0097316581900169

[2] Hearn, Demaine - http://erikdemaine.org/papers/GPC/

[3] Robson, "The Complexity of Go", IFIP Congress 1983

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

- (Generalized) CHESS[1]
- UNBOUNDED CONSTRAINT LOGIC[2]

[1] Fraenkel, Lichtenstein - `http://www.sciencedirect.com/science/article/pii/0097316581900169`

[2] Hearn, Demaine - `http://erikdemaine.org/papers/GPC/`

[3] Robson, "The Complexity of Go", IFIP Congress 1983

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

- (Generalized) CHESS[1]
- UNBOUNDED CONSTRAINT LOGIC[2]
- GO (without Superko)[3]

[1] Fraenkel, Lichtenstein - http://www.sciencedirect.com/science/article/pii/0097316581900169
[2] Hearn, Demaine - http://erikdemaine.org/papers/GPC/
[3] Robson, "The Complexity of Go", IFIP Congress 1983

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

- (Generalized) CHESS[1]
- UNBOUNDED CONSTRAINT LOGIC[2]
- GO (without Superko)[3]

Notice: All loopy games!

[1] Fraenkel, Lichtenstein - http://www.sciencedirect.com/science/article/pii/0097316581900169

[2] Hearn, Demaine - http://erikdemaine.org/papers/GPC/

[3] Robson, "The Complexity of Go", IFIP Congress 1983

EXPTIME-complete Rulesets?

Yes, there are rulesets that require exponential time to solve!

- (Generalized) CHESS[1]
- UNBOUNDED CONSTRAINT LOGIC[2]
- GO (without Superko)[3]

Notice: All loopy games!
How do we know there's no faster algorithm?

---

[1] Fraenkel, Lichtenstein - http://www.sciencedirect.com/science/article/pii/0097316581900169
[2] Hearn, Demaine - http://erikdemaine.org/papers/GPC/
[3] Robson, "The Complexity of Go", IFIP Congress 1983

Game Reductions!

Let's say I want to prove a loopy game Banane is EXPTIME-hard.

Game Reductions!

Let's say I want to prove a loopy game $\textsc{Banane}$ is $\mathrm{EXPTIME}$-hard.

- I need to find a function
  f : $\textsc{Chess}$ positions $\rightarrow$ $\textsc{Banane}$ positions

Game Reductions!

Let's say I want to prove a loopy game BANANE is EXPTIME-hard.

- I need to find a function
  f : CHESS positions $\rightarrow$ BANANE positions
- f must be efficiently computable

Game Reductions!

Let's say I want to prove a loopy game $\textsc{Banane}$ is $\mathrm{EXPTIME}$-hard.

- I need to find a function
  f : $\textsc{Chess}$ positions $\rightarrow$ $\textsc{Banane}$ positions
- f must be efficiently computable
- $x \in \mathcal{L} \cup \mathcal{N} \Leftrightarrow f(x) \in \mathcal{L} \cup \mathcal{N}$

Game Reductions!

Let's say I want to prove a loopy game $\textsc{Banane}$ is $\mathrm{EXPTIME}$-hard.

- I need to find a function $f : \textsc{Chess}$ positions $\rightarrow \textsc{Banane}$ positions
- f must be efficiently computable
- $x \in \mathcal{L} \cup \mathcal{N} \Leftrightarrow f(x) \in \mathcal{L} \cup \mathcal{N}$
- Now f is a *reduction*.

Game Reductions!

Let's say I want to prove a loopy game $\textsc{Banane}$ is $\mathrm{EXPTIME}$-hard.

- I need to find a function f : $\textsc{Chess}$ positions $\rightarrow$ $\textsc{Banane}$ positions
- f must be efficiently computable
- $x \in \mathcal{L} \cup \mathcal{N} \Leftrightarrow f(x) \in \mathcal{L} \cup \mathcal{N}$
- Now f is a *reduction*. ... and $\textsc{Banane}$ is $\mathrm{EXPTIME}$-hard!

Game Reductions!

Let's say I want to prove a loopy game BANANE is EXPTIME-hard.

- I need to find a function f : CHESS positions $\rightarrow$ BANANE positions
- f must be efficiently computable
- $x \in \mathcal{L} \cup \mathcal{N} \Leftrightarrow f(x) \in \mathcal{L} \cup \mathcal{N}$
- Now f is a *reduction*. ... and BANANE is EXPTIME-hard!

How does this show hardness?

Hardness Follows a Reduction

Reduction f : $\textsc{Chess} \to \textsc{Banane}$

Hardness Follows a Reduction

Reduction f : $\textsc{Chess} \to \textsc{Banane}$

- $\textsc{Chess}$ EXPTIME-hard $\to$ $\textsc{Banane}$ EXPTIME-hard

Hardness Follows a Reduction

Reduction $f : \textsc{Chess} \rightarrow \textsc{Banane}$

- $\textsc{Chess}$ EXPTIME-hard $\rightarrow$ $\textsc{Banane}$ EXPTIME-hard
- Proof-by-contradiction

Hardness Follows a Reduction

Reduction f : CHESS $\rightarrow$ BANANE

- CHESS EXPTIME-hard $\rightarrow$ BANANE EXPTIME-hard
- Proof-by-contradiction
  - Assume BANANE solvable in faster-than-exponential time by some algorithm A

Hardness Follows a Reduction

Reduction $f : \textsc{Chess} \to \textsc{Banane}$

- $\textsc{Chess}$ EXPTIME-hard $\to$ $\textsc{Banane}$ EXPTIME-hard
- Proof-by-contradiction
  - Assume $\textsc{Banane}$ solvable in faster-than-exponential time by some algorithm A
  - New $\textsc{Chess}$-solving algorithm, $B(x)$: return $A(f(x))$

Hardness Follows a Reduction

Reduction f : CHESS $\to$ BANANE

- CHESS EXPTIME-hard $\to$ BANANE EXPTIME-hard
- Proof-by-contradiction
  - Assume BANANE solvable in faster-than-exponential time by some algorithm A
  - New CHESS-solving algorithm, $B(x)$: return $A(f(x))$
  - B solves CHESS!

Hardness Follows a Reduction

Reduction f : Chess $\rightarrow$ Banane

- Chess EXPTIME-hard $\rightarrow$ Banane EXPTIME-hard
- Proof-by-contradiction
  - Assume Banane solvable in faster-than-exponential time by some algorithm A
  - New Chess-solving algorithm, $B(x)$: return $A(f(x))$
  - B solves Chess!
  - B solves Chess in faster-than-exponential time!

Hardness Follows a Reduction

Reduction $f : \textsc{Chess} \to \textsc{Banane}$

- $\textsc{Chess}$ $\mathrm{EXPTIME}$-hard $\to$ $\textsc{Banane}$ $\mathrm{EXPTIME}$-hard
- Proof-by-contradiction
  - Assume $\textsc{Banane}$ solvable in faster-than-exponential time by some algorithm A
  - New $\textsc{Chess}$-solving algorithm, $B(x)$: return $A(f(x))$
  - B solves $\textsc{Chess}$!
  - B solves $\textsc{Chess}$ in faster-than-exponential time!
  - Now $\textsc{Chess}$ is not $\mathrm{EXPTIME}$-hard $\rightarrow\leftarrow$

What about Short Games?

- Assume longest game has polynomial turns.

What about Short Games?

- Assume longest game has polynomial turns.
- … and polynomial options.

What about Short Games?

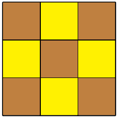
- Assume longest game has polynomial turns.
- ... and polynomial options.
- E.g. Domineering.

What about Short Games?

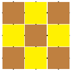
- Assume longest game has polynomial turns.
- ... and polynomial options.
- E.g. Domineering.
- Let's do 3x3 case:

What about Short Games?

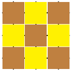
- Assume longest game has polynomial turns.
- ... and polynomial options.
- E.g. Domineering.
- Let's do 3x3 case:



What about Short Games?

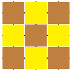


What about Short Games?

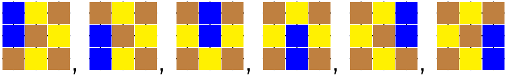


$\downarrow$ Left's potentially-winning options

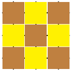
What about Short Games?



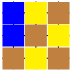
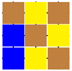
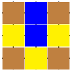
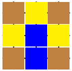
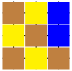
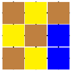
↓ Left's potentially-winning options



What about Short Games?

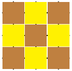


$\downarrow$ Left's potentially-winning options

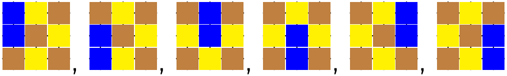
, , , , , 

$\downarrow$ Right's potentially-winning options

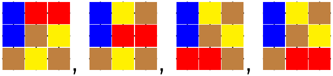
What about Short Games?



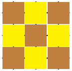
↓ Left's potentially-winning options



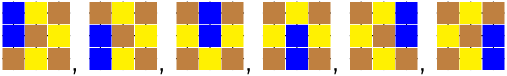
↓ Right's potentially-winning options



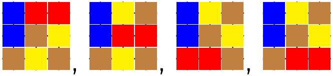
What about Short Games?



↓ Left's potentially-winning options

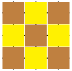


↓ Right's potentially-winning options

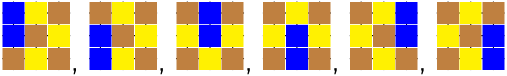


↓ Left's potentially-winning options

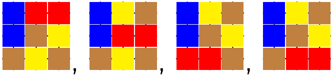
What about Short Games?



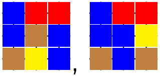
↓ Left's potentially-winning options



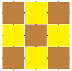
↓ Right's potentially-winning options



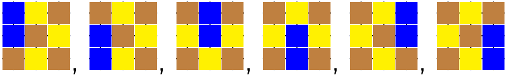
↓ Left's potentially-winning options



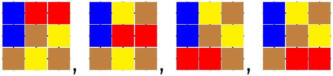
What about Short Games?



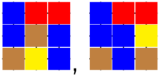
↓ Left's potentially-winning options



↓ Right's potentially-winning options

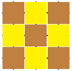


↓ Left's potentially-winning options

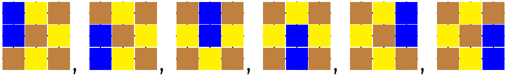


↓ Right's potentially-winning options

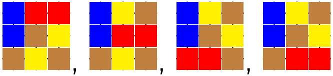
What about Short Games?



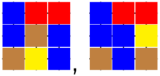
$\downarrow$ Left's potentially-winning options



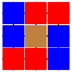
$\downarrow$ Right's potentially-winning options



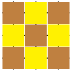
$\downarrow$ Left's potentially-winning options



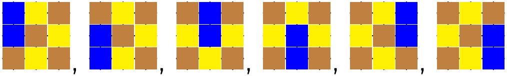
$\downarrow$ Right's potentially-winning options

 $\in \mathcal{R} \cup \mathcal{P}$ (Right wins)

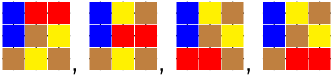
What about Short Games?



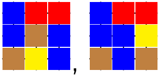
↓ Left's potentially-winning options



↓ Right's potentially-winning options



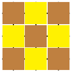
↓ Left's potentially-winning options



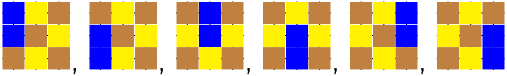
↓ Right's potentially-winning options ✓

So... previous move not a winner for Left. Back up and try the next one.

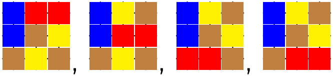
What about Short Games?



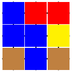
↓ Left's potentially-winning options



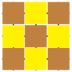
↓ Right's potentially-winning options



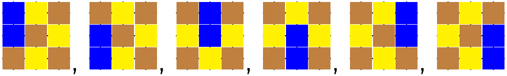
↓ Left's potentially-winning options



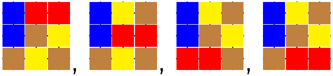
What about Short Games?



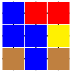
$\downarrow$ Left's potentially-winning options



$\downarrow$ Right's potentially-winning options

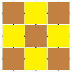


$\downarrow$ Left's potentially-winning options

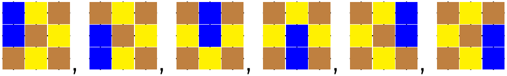


No Right move!

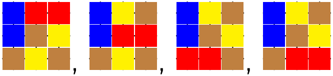
What about Short Games?



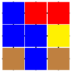
↓ Left's potentially-winning options



↓ Right's potentially-winning options

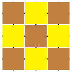


↓ Left's potentially-winning options

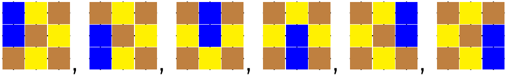


No Right move! (Left wins)

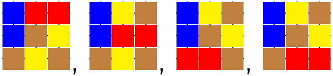
What about Short Games?



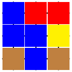
↓ Left's potentially-winning options



↓ Right's potentially-winning options

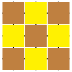


↓ Left's potentially-winning options

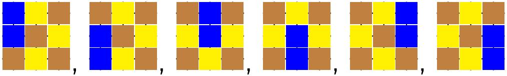


No Right move! (Left wins) Go back and change Right's last move...

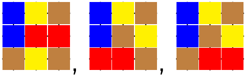
What about Short Games?



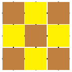
↓ Left's potentially-winning options



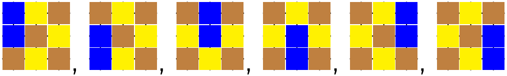
↓ Right's potentially-winning options



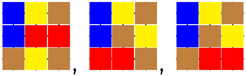
What about Short Games?



$\downarrow$ Left's potentially-winning options

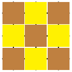


$\downarrow$ Right's potentially-winning options

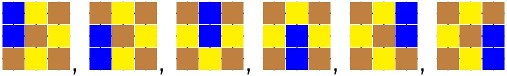


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$

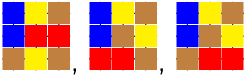
What about Short Games?



$\downarrow$ Left's potentially-winning options

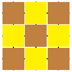


$\downarrow$ Right's potentially-winning options

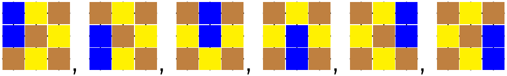


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need?

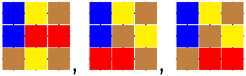
What about Short Games?



↓ Left's potentially-winning options

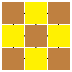


↓ Right's potentially-winning options

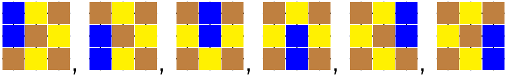


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need? (polynomial)

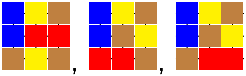
What about Short Games?



$\downarrow$ Left's potentially-winning options

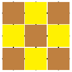


$\downarrow$ Right's potentially-winning options

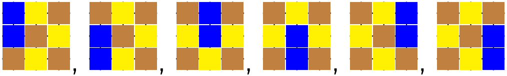


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need? (polynomial)
- How many boards in a row?

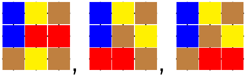
What about Short Games?



$\downarrow$ Left's potentially-winning options

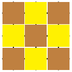


$\downarrow$ Right's potentially-winning options

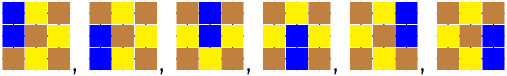


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need? (polynomial)
- How many boards in a row? (polynomial)

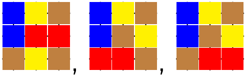
What about Short Games?



$\downarrow$ Left's potentially-winning options

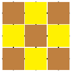


$\downarrow$ Right's potentially-winning options

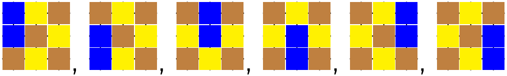


- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need? (polynomial)
- How many boards in a row? (polynomial)
- How much "workspace" do I need?

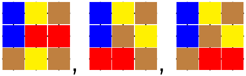
What about Short Games?



↓ Left's potentially-winning options



↓ Right's potentially-winning options



- Continue to decide $G \in \mathcal{L} \cup \mathcal{N}$
- How many rows of boards do I need? (polynomial)
- How many boards in a row? (polynomial)
- How much "workspace" do I need? (polynomial)

PSPACE

PSPACE: All problems solvable with a polynomial amount of space.

PSPACE

PSPACE: All problems solvable with a polynomial amount of space.

- $\mathrm{P} \subsetneq \mathrm{EXPTIME}$

PSPACE

PSPACE: All problems solvable with a polynomial amount of space.

- $\mathrm{P} \subsetneq \mathrm{EXPTIME}$
- $\mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

PSPACE

PSPACE: All problems solvable with a polynomial amount of space.

- $\mathrm{P} \subsetneq \mathrm{EXPTIME}$
- $\mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$
- $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

PSPACE-hard Problems

PSPACE-hard: Problems at least as hard as the hardest problem(s) in PSPACE

PSPACE-hard Problems

PSPACE-hard: Problems at least as hard as the hardest problem(s) in PSPACE

- Everything $\mathrm{EXPTIME}$-hard.

PSPACE-hard Problems

PSPACE-hard: Problems at least as hard as the hardest problem(s) in PSPACE

- Everything EXPTIME-hard.
- Games: Amazons, Geography, Hex, Konane, Node Kayles, Snort, Toads and Frogs, etc.

PSPACE-hard Problems

PSPACE-hard: Problems at least as hard as the hardest problem(s) in PSPACE

- Everything EXPTIME-hard.
- Games: Amazons, Geography, Hex, Konane, Node Kayles, Snort, Toads and Frogs, etc.
- Non-Games: Deadlock detection, periodic scheduling, etc.

PSPACE-hard Problems

PSPACE-hard: Problems at least as hard as the hardest problem(s) in PSPACE

- Everything EXPTIME-hard.
- Games: Amazons, Geography, Hex, Konane, Node Kayles, Snort, Toads and Frogs, etc.
- Non-Games: Deadlock detection, periodic scheduling, etc.

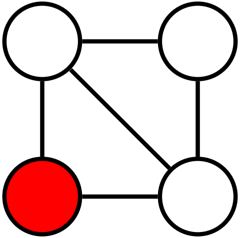
PSPACE-complete: Both PSPACE-hard and in PSPACE.

Example: Impartial Col is PSPACE-hard

Impartial Col: 2-coloring placement game

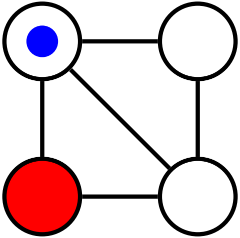
Example: Impartial Col is PSPACE-hard

Impartial Col: 2-coloring placement game



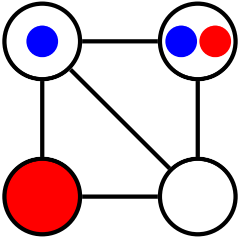
Example: Impartial Col is PSPACE-hard

Impartial Col: 2-coloring placement game



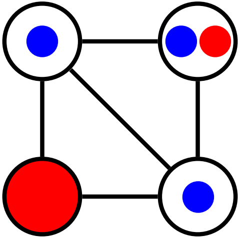
Example: Impartial Col is PSPACE-hard

Impartial Col: 2-coloring placement game



Example: Impartial Col is PSPACE-hard

Impartial Col: 2-coloring placement game



Example: Impartial Col is PSPACE-hard

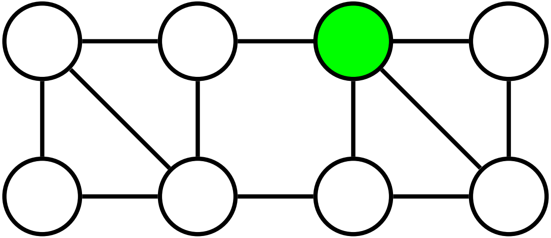
We'll reduce from Node Kayles (known to be PSPACE-hard.)

Example: Impartial Col is PSPACE-hard

We'll reduce from Node Kayles (known to be PSPACE-hard.)
Node Kayles: (Impartial) Each turn, place a token on an empty vertex not adjacent to another token.

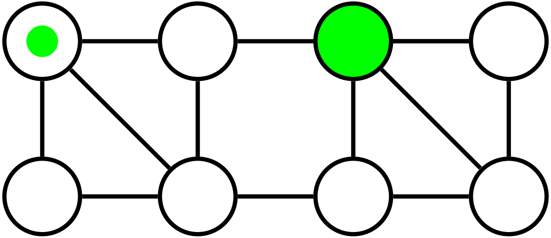
Example: Impartial Col is PSPACE-hard

We'll reduce from Node Kayles (known to be PSPACE-hard.)
Node Kayles: (Impartial) Each turn, place a token on an empty vertex not adjacent to another token.



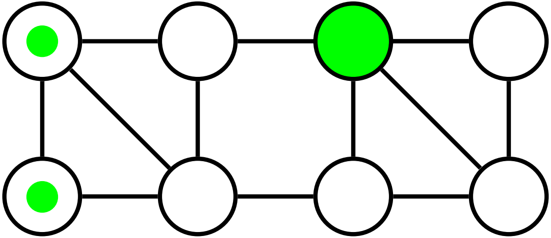
Example: Impartial Col is PSPACE-hard

We'll reduce from Node Kayles (known to be PSPACE-hard.)
Node Kayles: (Impartial) Each turn, place a token on an empty vertex not adjacent to another token.



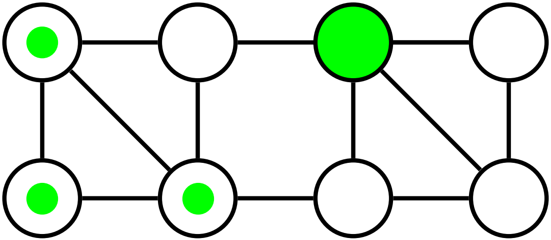
Example: Impartial Col is PSPACE-hard

We'll reduce from Node Kayles (known to be PSPACE-hard.)
Node Kayles: (Impartial) Each turn, place a token on an empty vertex not adjacent to another token.



Example: Impartial Col is PSPACE-hard

We'll reduce from Node Kayles (known to be PSPACE-hard.)
Node Kayles: (Impartial) Each turn, place a token on an empty vertex not adjacent to another token.

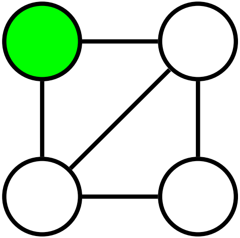


Example: Impartial Col is PSPACE-hard

Proof: reduction from Node Kayles to Impartial Col.

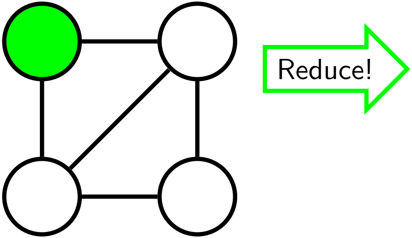
Example: Impartial Col is PSPACE-hard

Proof: reduction from Node Kayles to Impartial Col.



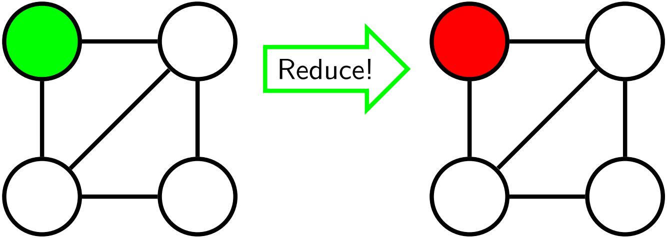
Example: Impartial Col is PSPACE-hard

Proof: reduction from Node Kayles to Impartial Col.



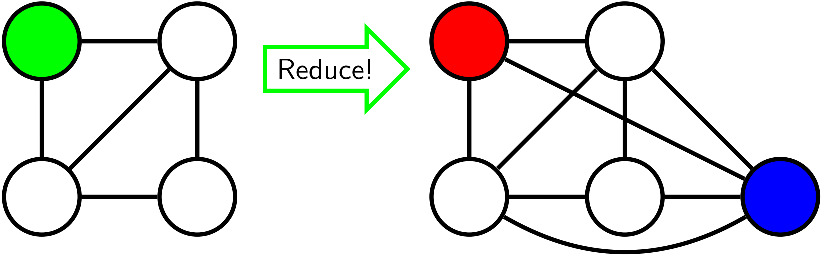
Example: Impartial Col is PSPACE-hard

Proof: reduction from Node Kayles to Impartial Col.



Example: IMPARTIAL COL is PSPACE-hard

Proof: reduction from NODE KAYLES to IMPARTIAL COL.

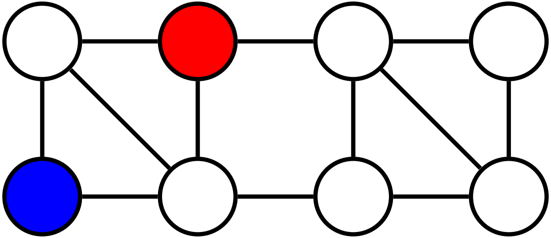


Example: Snort is PSPACE-hard

Snort: Can't play adjacent to opponent.

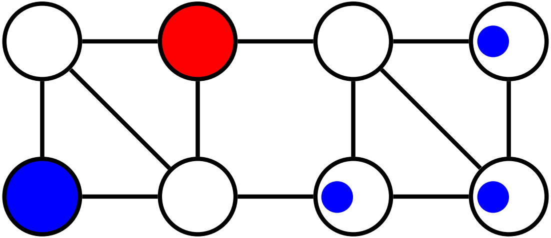
Example: Snort is PSPACE-hard

Snort: Can't play adjacent to opponent.



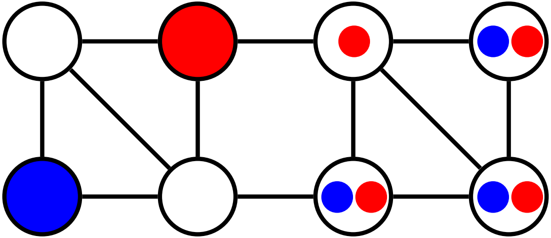
Example: Snort is PSPACE-hard

Snort: Can't play adjacent to opponent.



Example: SNORT is PSPACE-hard

SNORT: Can't play adjacent to opponent.



Example: Snort is PSPACE-hard

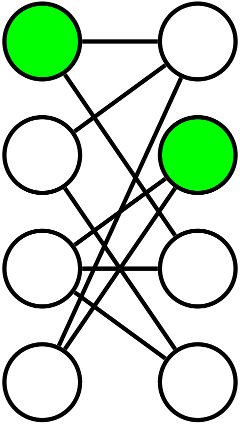
Reduce from Bigraph Node-Kayles (known to be hard).

Example: Snort is PSPACE-hard

Reduce from Bigraph Node-Kayles (known to be hard).
Bigraph Node-Kayles: Kayles on Bipartite Graph where each player gets one side.

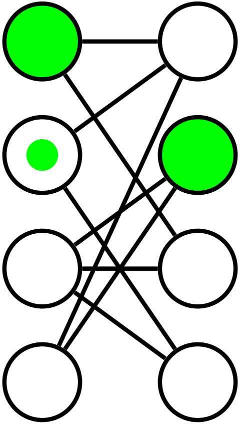
Example: Snort is PSPACE-hard

Reduce from Bigraph Node-Kayles (known to be hard).
Bigraph Node-Kayles: Kayles on Bipartite Graph where each player gets one side.



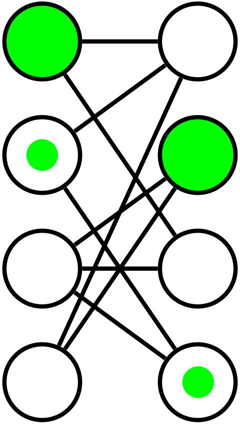
Example: Snort is PSPACE-hard

Reduce from Bigraph Node-Kayles (known to be hard).
Bigraph Node-Kayles: Kayles on Bipartite Graph where each player gets one side.



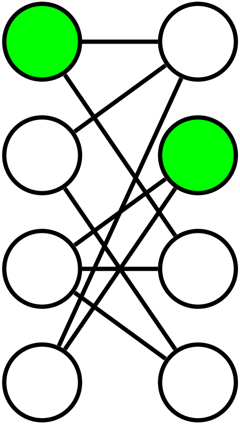
Example: Snort is PSPACE-hard

Reduce from Bigraph Node-Kayles (known to be hard).
Bigraph Node-Kayles: Kayles on Bipartite Graph where each player gets one side.



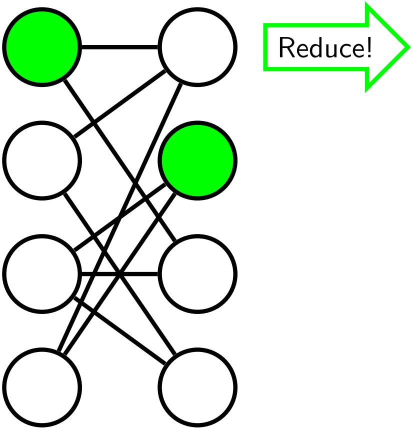
Example: Snort is PSPACE-hard

Reduce Bigraph Node Kayles to Snort



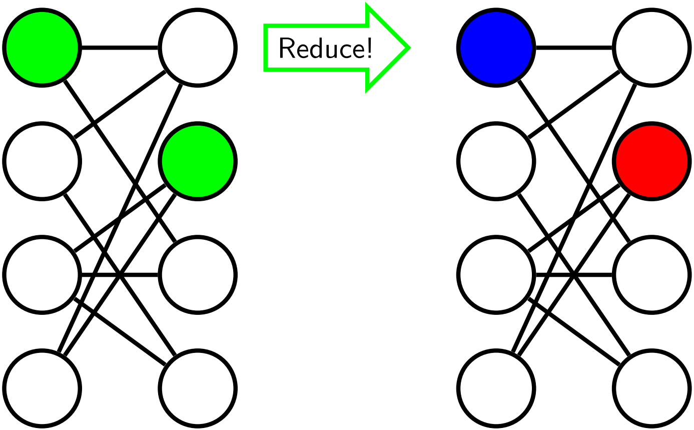
Example: Snort is PSPACE-hard

Reduce Bigraph Node Kayles to Snort



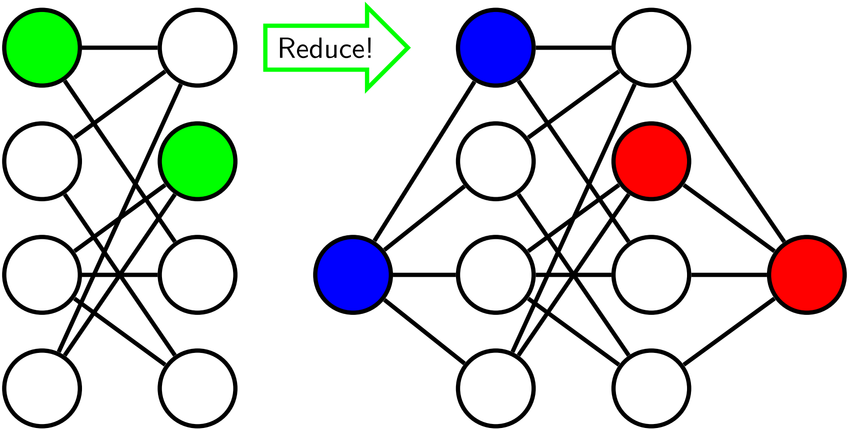
Example: Snort is PSPACE-hard

Reduce Bigraph Node Kayles to Snort



Example: SNORT is PSPACE-hard

Reduce BIGRAPH NODE KAYLES to SNORT

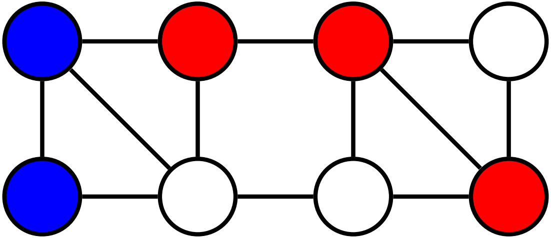


Example: Graph NoGo is Hard

Graph NoGo: Go, without capture moves.

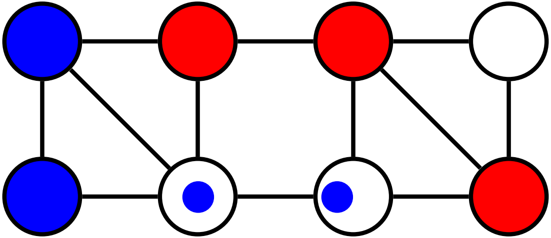
Example: Graph NoGo is Hard

Graph NoGo: Go, without capture moves.



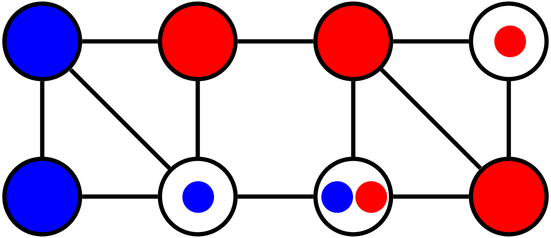
Example: Graph NoGo is Hard

Graph NoGo: Go, without capture moves.



Example: Graph NoGo is Hard

Graph NoGo: Go, without capture moves.



Example: Graph NoGo is Hard

Reduce: Col $\rightarrow$ NoGo

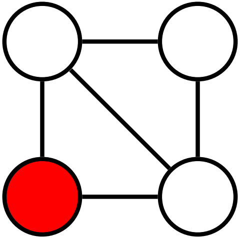
Example: Graph NoGo is Hard

Reduce: Col $\rightarrow$ NoGo
Col: Can't play adjacent to yourself.

Example: Graph NoGo is Hard

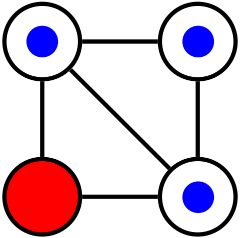
Reduce: Col $\rightarrow$ NoGo

Col: Can't play adjacent to yourself.



Example: GRAPH NOGO is Hard

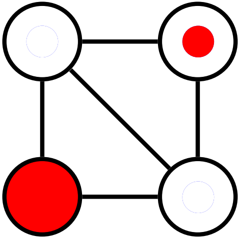
Reduce: COL $\rightarrow$ NOGO
COL: Can't play adjacent to yourself.



Example: Graph NoGo is Hard

Reduce: Col $\rightarrow$ NoGo

Col: Can't play adjacent to yourself.



Example: Graph NoGo is Hard

Reduce from Col

Example: Graph NoGo is Hard

Reduce from Col

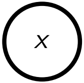
Separate "gadgets" to replace vertices and edges.

Example: Graph NoGo is Hard

Reduce from Col
Separate "gadgets" to replace vertices and edges.
Here's the gadget for each vertex:

Example: GRAPH NOGO is Hard

Reduce from COL
Separate "gadgets" to replace vertices and edges.
Here's the gadget for each vertex:



Example: GRAPH NOGO is Hard

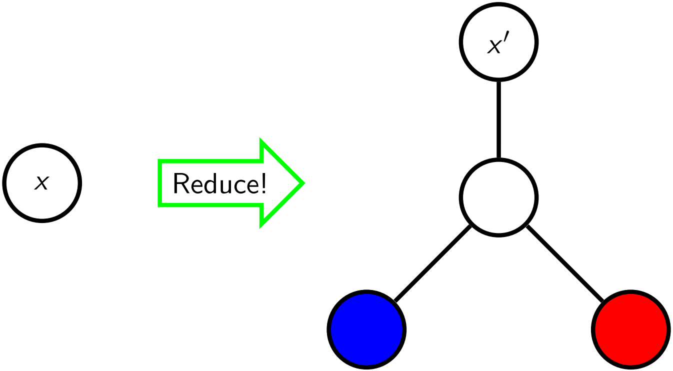
Reduce from COL

Separate "gadgets" to replace vertices and edges.

Here's the gadget for each vertex:

Example: GRAPH NOGO is Hard

Reduce from COL
Separate "gadgets" to replace vertices and edges.
Here's the gadget for each vertex:

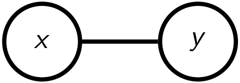


Example: GRAPH NOGO is Hard

Here's the reduction for each COL edge:

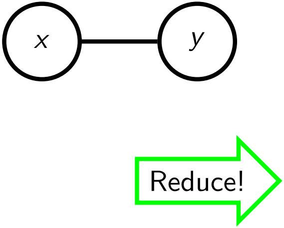
Example: Graph NoGo is Hard

Here's the reduction for each Col edge:



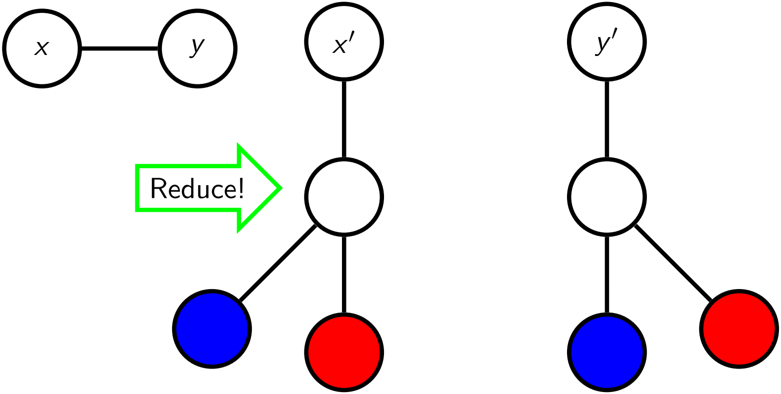
Example: GRAPH NOGO is Hard

Here's the reduction for each COL edge:



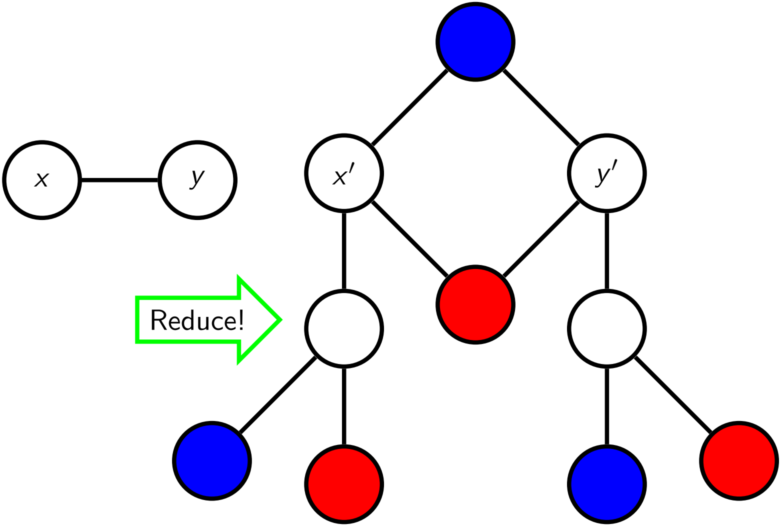
Example: Graph NoGo is Hard

Here's the reduction for each Col edge:



Example: Graph NoGo is Hard

Here's the reduction for each Col edge:



Locality is Tougher

For some games, we have to play adjacent to other moves.

Locality is Tougher

For some games, we have to play adjacent to other moves.

- Geography

Locality is Tougher

For some games, we have to play adjacent to other moves.

- Geography
- Slimetrail

Locality is Tougher

For some games, we have to play adjacent to other moves.

- Geography
- Slimetrail
- Constraint Logic

Locality is Tougher

For some games, we have to play adjacent to other moves.

- GEOGRAPHY
- SLIMETRAIL
- CONSTRAINT LOGIC

Some of the first PSPACE-hard games were these, proven from QSAT.

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \lor \overline{x_1} \lor x_2) \land \cdots \land (\overline{x_{27}} \lor \overline{x_1} \lor x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true;

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true; Right otherwise

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true; Right otherwise
- Phrase winnability with quantifiers!

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true; Right otherwise
- Phrase winnability with quantifiers!
- $G \in \mathcal{L} \cup \mathcal{N} \iff$

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true; Right otherwise
- Phrase winnability with quantifiers!
- $G \in \mathcal{L} \cup \mathcal{N} \iff$
  $\exists x_0 : \forall x_1 : \exists x_2 : \forall x_3 : \ldots : \forall x_{27} :$

QSAT (is a game)

- QSAT: Quantified Boolean Satisfiability
- 3CNF: $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$
- Play: create an assignment of variables.
  - Left assigns to x_0
  - Right assigns to x_1
  - Left: x_2, etc
- Left wins if formula is true; Right otherwise
- Phrase winnability with quantifiers!
- $G \in \mathcal{L} \cup \mathcal{N} \iff$
  $\exists x_0 : \forall x_1 : \exists x_2 : \forall x_3 : \ldots : \forall x_{27} :$
  $(x_0 \vee \overline{x_1} \vee x_2) \wedge \cdots \wedge (\overline{x_{27}} \vee \overline{x_1} \vee x_{12})$

Sample Reduction: Geography

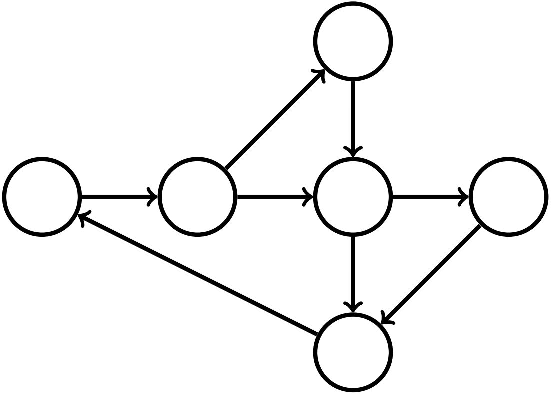
- Reduce QSAT to Geography

Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.

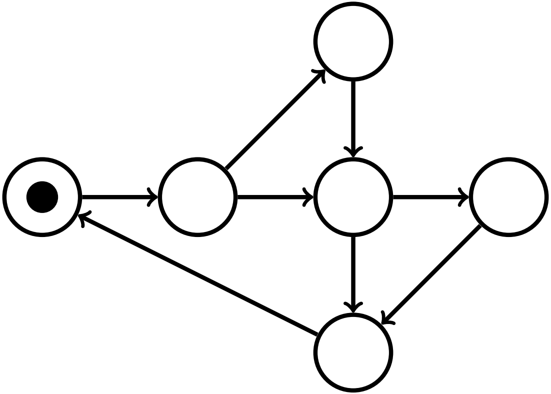
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.



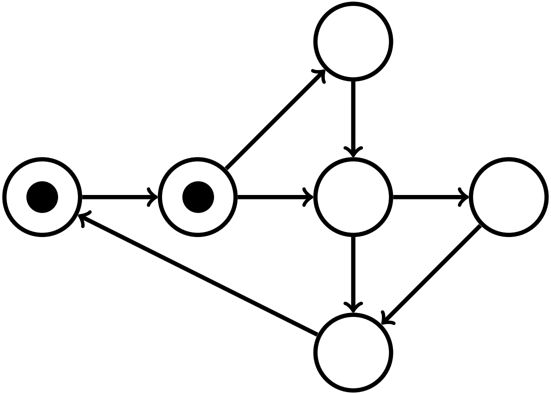
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.



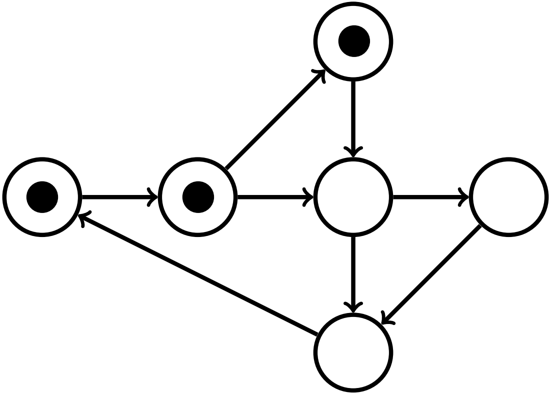
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.



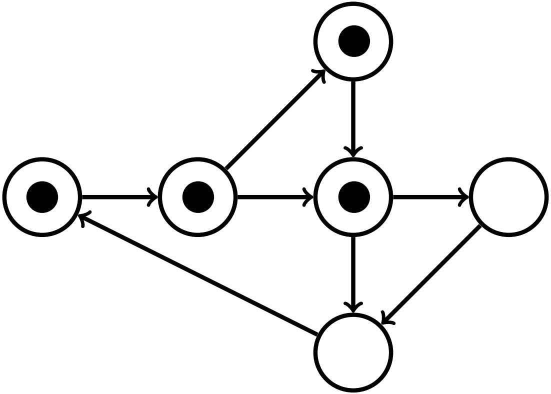
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.



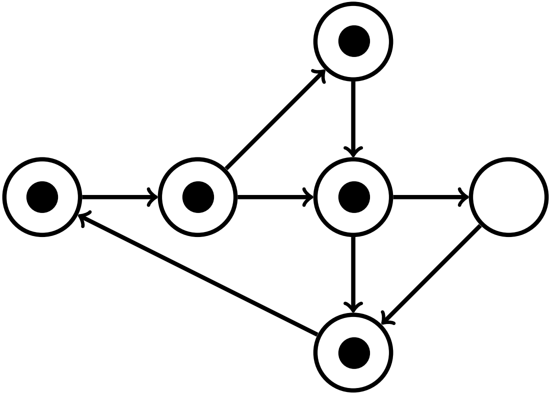
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- GEOGRAPHY: Move around on a directed graph, but you can't visit a vertex twice.



Sample Reduction: Geography

- Reduce QSAT to Geography
- Geography: Move around on a directed graph, but you can't visit a vertex twice.



Sample Reduction: GEOGRAPHY

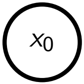
- Reduce QSAT to GEOGRAPHY

Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- Variable Gadget:

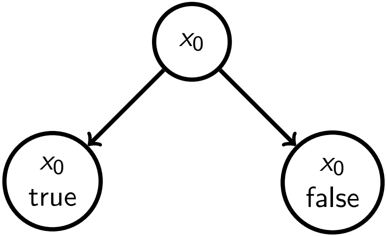
Sample Reduction: Geography

- Reduce QSAT to Geography
- Variable Gadget:



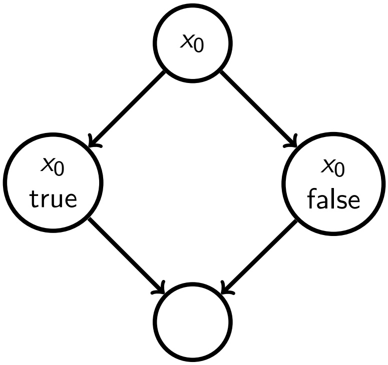
Sample Reduction: Geography

- Reduce QSAT to Geography
- Variable Gadget:



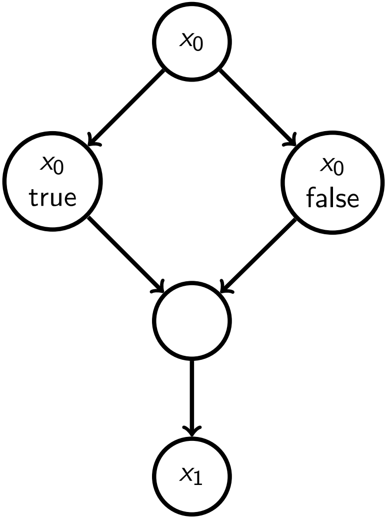
Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- Variable Gadget:



Sample Reduction: GEOGRAPHY

- Reduce QSAT to GEOGRAPHY
- Variable Gadget:



Sample Reduction: GEOGRAPHY

After all variables are chosen, Right will choose a clause,

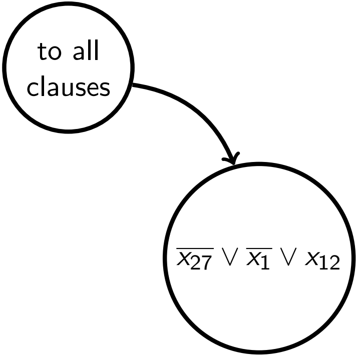
Sample Reduction: GEOGRAPHY

After all variables are chosen, Right will choose a clause,

... then Left will choose a variable in that clause.

Sample Reduction: GEOGRAPHY

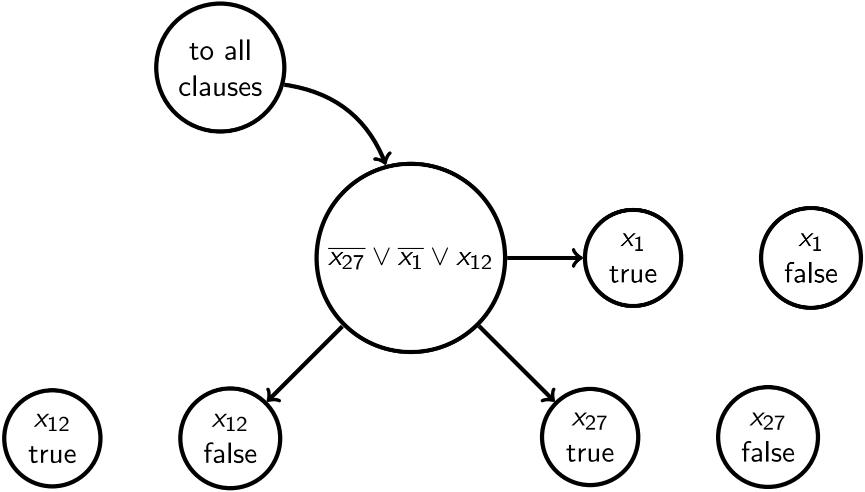
After all variables are chosen, Right will choose a clause,
... then Left will choose a variable in that clause.



Sample Reduction: Geography

After all variables are chosen, Right will choose a clause,

... then Left will choose a variable in that clause.



Issues!

At this point, I expect you have some problems:

Issues!

At this point, I expect you have some problems:

- NoGo is only played on grids! Not general graphs!

Issues!

At this point, I expect you have some problems:

- NoGo is only played on grids! Not general graphs!
- Same with Snort!

Issues!

At this point, I expect you have some problems:

- NoGo is only played on grids! Not general graphs!
- Same with Snort!
- What about starting positions? What if we never reach these positions in the range of the reduction?

Issues!

At this point, I expect you have some problems:

- NoGo is only played on grids! Not general graphs!
- Same with Snort!
- What about starting positions? What if we never reach these positions in the range of the reduction?

Let's address the starting positions problem first.

Starting Positions

- Just determining winnability, not strategy.

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. CHOMP, HEX)

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. CHOMP, HEX)
- Some are conjectured (e.g. SPROUTS, CRAM)

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. Chomp, Hex)
- Some are conjectured (e.g. Sprouts, Cram)
- Reason: parameterized by number (e.g. 13×13 Cram)

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. CHOMP, HEX)
- Some are conjectured (e.g. SPROUTS, CRAM)
- Reason: parameterized by number (e.g. 13×13 CRAM)
  - Can describe with logarithmic information.

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. Chomp, Hex)
- Some are conjectured (e.g. Sprouts, Cram)
- Reason: parameterized by number (e.g. 13×13 Cram)
  - Can describe with logarithmic information.
  - Drawing the board is exponential in that description.

Starting Positions

- Just determining winnability, not strategy.
- Some are known (e.g. CHOMP, HEX)
- Some are conjectured (e.g. SPROUTS, CRAM)
- Reason: parameterized by number (e.g. 13×13 CRAM)
  - Can describe with logarithmic information.
  - Drawing the board is exponential in that description.
- Usually, starting positions are easy.

Specific Board Geometry

"Snapping to a grid" is difficult.

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Specific Board Geometry

"Snapping to a grid" is difficult.

- Usual progression: General $\rightarrow$ more specific $\rightarrow \cdots \rightarrow$ Grid

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Specific Board Geometry

"Snapping to a grid" is difficult.

- Usual progression: General $\rightarrow$ more specific $\rightarrow \cdots \rightarrow$ Grid
- HEX: Graph[4] $\rightarrow$ Hex-Grid[5]

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Specific Board Geometry

"Snapping to a grid" is difficult.

- Usual progression: General $\rightarrow$ more specific $\rightarrow \cdots \rightarrow$ Grid
- HEX: Graph[4] $\rightarrow$ Hex-Grid[5]
- NOGO: Graph[6] $\rightarrow$ Planar Graph[7]

---

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Specific Board Geometry

"Snapping to a grid" is difficult.

- Usual progression: General $\rightarrow$ more specific $\rightarrow \cdots \rightarrow$ Grid
- Hex: Graph[4] $\rightarrow$ Hex-Grid[5]
- NoGo: Graph[6] $\rightarrow$ Planar Graph[7]
- Snort: Graph[8] $\rightarrow$ Planar Graph[9]

---

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Specific Board Geometry

"Snapping to a grid" is difficult.

- Usual progression: General $\rightarrow$ more specific $\rightarrow \cdots \rightarrow$ Grid
- HEX: Graph[4] $\rightarrow$ Hex-Grid[5]
- NOGO: Graph[6] $\rightarrow$ Planar Graph[7]
- SNORT: Graph[8] $\rightarrow$ Planar Graph[9]
- This seems so backwards! Usually the general case is strongest!

---

[4] Evans, Tarjan 1976
[5] Reisch 1981
[6] Few slides back.
[7] B., Hearn unpublished
[8] Schaefer 1978
[9] B., Hearn unpublished

Supersets

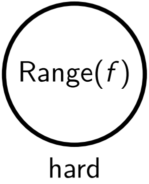
Supersets of hard sets are hard.

Supersets

Supersets of hard sets are hard. Let f be our NoGo reduction.

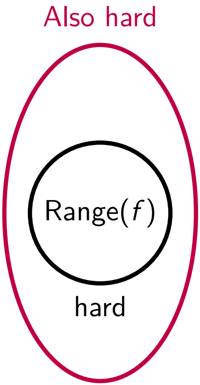
Supersets

Supersets of hard sets are hard. Let f be our NoGo reduction.



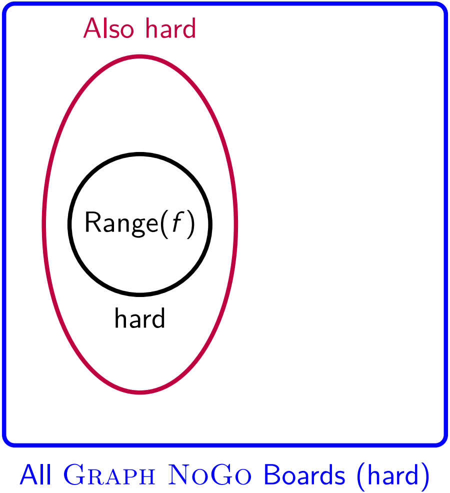
Supersets

Supersets of hard sets are hard. Let f be our NoGo reduction.



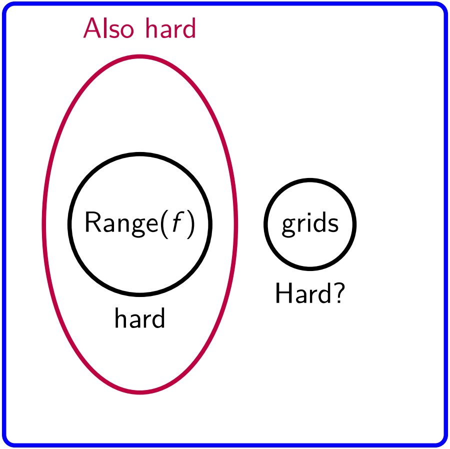
Supersets

Supersets of hard sets are hard. Let f be our NoGo reduction.



Supersets

Supersets of hard sets are hard. Let f be our NoGo reduction.



All Graph NoGo Boards (hard)

State of the Art

What is known to be hard *now*?

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https:
//link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https:
//www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓
- ATROPOS: hexagonal grid (B,Teng 2007[14]) ✓

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓
- ATROPOS: hexagonal grid (B,Teng 2007[14]) ✓
- ARC KAYLES: *no* known hardness

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓
- ATROPOS: hexagonal grid (B,Teng 2007[14]) ✓
- ARC KAYLES: *no* known hardness
- DOMINEERING: *none*

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓
- ATROPOS: hexagonal grid (B,Teng 2007[14]) ✓
- ARC KAYLES: *no* known hardness
- DOMINEERING: *none*
- CLOBBER: NP-hard on general graphs. (AGNW 2005[15])

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

State of the Art

What is known to be hard *now*?

- NODE KAYLES: general graphs (Schaeffer, 1978[10])
- SNORT: planar graphs (B, Hearn [11])
- COL: planar graphs (B, Hearn [12])
- HEX: hexagonal grid (Reisch, 1981[13]) ✓
- ATROPOS: hexagonal grid (B,Teng 2007[14]) ✓
- ARC KAYLES: *no* known hardness
- DOMINEERING: *none*
- CLOBBER: NP-hard on general graphs. (AGNW 2005[15])
- NOGO: planar graphs (B, Hearn [16])

---

[10] www.sciencedirect.com/science/article/pii/0022000078900454
[11] Not yet published.
[12] Not yet published
[13] https://link.springer.com/article/10.1007/BF00288964
[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49
[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf
[16] Not yet published.

State of the Art

What is known to be hard *now*?

- Node Kayles: general graphs (Schaeffer, 1978[10])
- Snort: planar graphs (B, Hearn [11])
- Col: planar graphs (B, Hearn [12])
- Hex: hexagonal grid (Reisch, 1981[13]) ✓
- Atropos: hexagonal grid (B,Teng 2007[14]) ✓
- Arc Kayles: *no* known hardness
- Domineering: *none*
- Clobber: NP-hard on general graphs. (AGNW 2005[15])
- NoGo: planar graphs (B, Hearn [16])
- Sprouts: *none*

[10] www.sciencedirect.com/science/article/pii/0022000078900454

[11] Not yet published.

[12] Not yet published

[13] https://link.springer.com/article/10.1007/BF00288964

[14] https://link.springer.com/chapter/10.1007%2F978-3-540-77105-0_49

[15] https://www.emis.de/journals/INTEGERS/papers/a1int2003/a1int2003.pdf

[16] Not yet published.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?
  - Not if it's in P.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?
  - Not if it's in P.
  - Hard games are better for competition.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?
  - Not if it's in P.
  - Hard games are better for competition.
- Nature of P vs. PSPACE.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?
    - Not if it's in P.
    - Hard games are better for competition.
- Nature of P vs. PSPACE.
- Limits to NP-approximation algorithms. Maybe to PSPACE as well.

Why Classify?

"But... Deep Learning can solve all of this! Why should we bother to classify all these games?"

- Deep Learning untested for many games.
- Deep Learning algorithms need huge data sets.
- Is this game even competetive?
  - Not if it's in P.
  - Hard games are better for competition.
- Nature of P vs. PSPACE.
- Limits to NP-approximation algorithms. Maybe to PSPACE as well.

Find the hardness, then use AI.

Thank you!

Thank you!

Thanks to Eric and GAG for hosting us!

Thank you!

Thanks to Eric and GAG for hosting us!
Extra thanks to Dan Burgess and Matt Ferland for proof-watching early versions of this talk.