Avoiding distractions in parity games

Tom van Dijk – `t.vandijk@utwente.nl`

ISoLA 2024

OVERVIEW

Outline of the talk

- Parity games
- Distractions
- Tangle learning
- Recursive tangle learning (**this work**)
- Perspectives

WHY PARITY GAMES?

Formal methods

- The model checking problem of modal μ-calculus...
  ...is equivalent to the problem of solving a parity game.
- The synthesis problem of ω-regular specifications (LTL, etc)...
  ...translates to the problem of solving a parity game.

WHY PARITY GAMES?

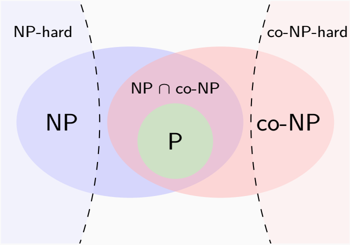
Formal methods

- The model checking problem of modal μ-calculus...
  ...is equivalent to the problem of solving a parity game.
- The synthesis problem of ω-regular specifications (LTL, etc)...
  ...translates to the problem of solving a parity game.

- Good news! We can solve large practical parity games very fast!
  (TvD, TACAS 2018)

WHY PARITY GAMES?

Famous open problem: P vs NP



- P: answer computed in polynomial time
- NP: proof checked in polynomial time
- co-NP: refutation checked in polynomial time
- NP-complete: can simulate every NP problem

Parity games

- Are in NP ∩ co-NP; even in UP ∩ co-UP
- Since 2017: quasi-polynomial solutions:
  strictly above polynomial time, and below exponential time
- Goal: a polynomial-time algorithm or a superpolynomial lower bound

WHY TANGLE LEARNING?

Main algorithm families

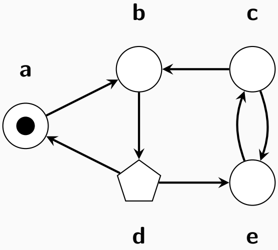
- Strategy improvement (policy iteration) little progress in years
- Value iteration underlying universal tree
- Decomposition-based underlying universal tree for some variants
- Universal trees have quasi-polynomial size! lower bound

Tangle learning (our approach)

- Based on decomposition with "attractors" (controlled predecessor)
- Not as rigid as Zielonka's recursive algorithm
- Explicitly targets won subgames (tangles)

PARITY GAMES

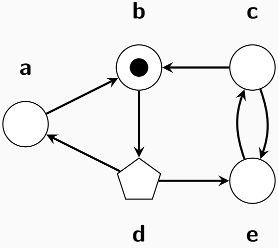
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a**

PARITY GAMES

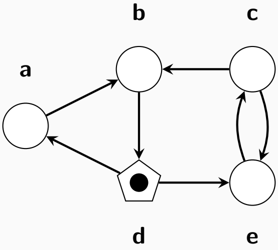
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b**

PARITY GAMES

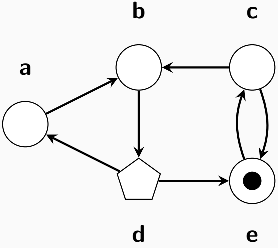
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d**

PARITY GAMES

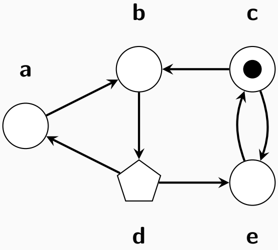
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e**

PARITY GAMES

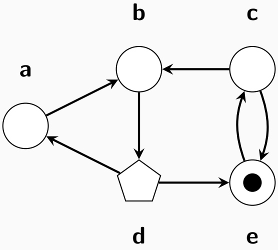
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c**

PARITY GAMES

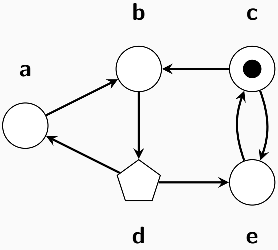
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e**

PARITY GAMES

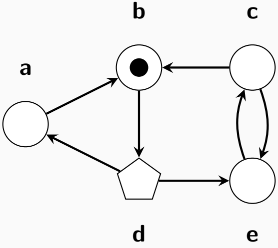
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬠
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c**

PARITY GAMES

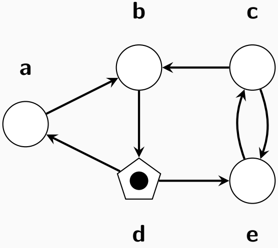
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b**

PARITY GAMES

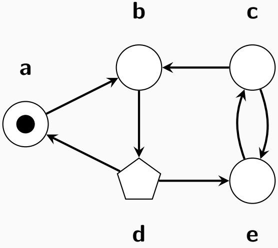
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d**

PARITY GAMES

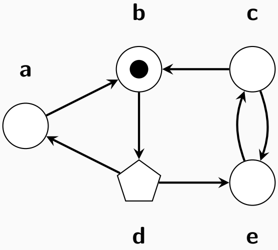
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬠
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a**

PARITY GAMES

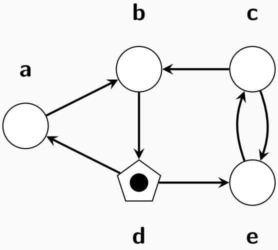
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b d a b**

PARITY GAMES

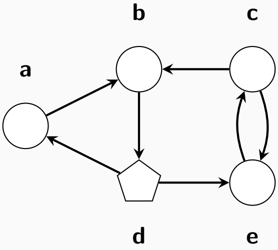
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬠
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



The play π: **a b d e c e c b (d a b)**$^{\omega}$

PARITY GAMES

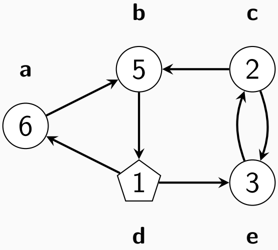
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



We need a winning condition...

PARITY GAMES

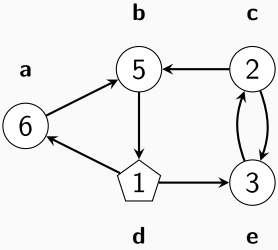
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Player Even wins if the highest priority seen infinitely often is even

PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor

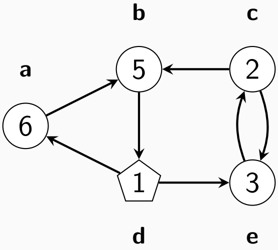


The play π: 6 5 1 3 2 3 2 5 $(1\ 6\ 5)^\omega$

Who wins this play?

PARITY GAMES

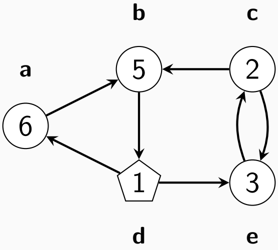
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



How do we determine who wins a vertex?

PARITY GAMES

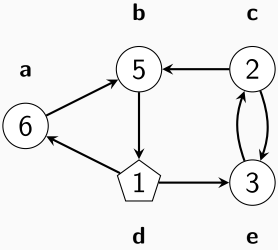
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬠
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins vertices V if they have a positional strategy $\sigma\colon V \to V$ such that every play in V consistent with σ is won by that player.

PARITY GAMES

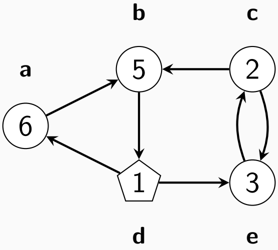
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



A player wins a (subgame) $\mathcal{G}$ if they have a positional strategy σ such that all cycles in the induced graph $\mathcal{G}[\sigma]$ are won by that player

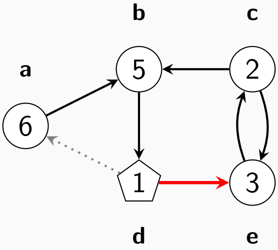
PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Which vertices are won by which player?

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor



Player Odd wins with strategy $\{\, \mathbf{d} \to \mathbf{e} \,\}$
Only two (Odd) cycles remain

PARITY GAMES

Solving a parity game

- Compute the winning regions $W_{\bullet}$ and $W_{\pentagon}$
- Compute the winning positional strategies $\sigma_{\bullet}$ and $\sigma_{\pentagon}$

HOW DO THEY WORK?

How do parity game solving algorithms work?

- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

HOW DO THEY WORK?

How do parity game solving algorithms work?

- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

- Sounds simple?

HOW DO THEY WORK?

How do parity game solving algorithms work?

- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

- Sounds simple?

- Follow edges to (good) high even vertices
- Avoid edges to (bad) high odd vertices

HOW DO THEY WORK?

How do parity game solving algorithms work?

- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

- Sounds simple?

- Follow edges to (good) high even vertices
- Avoid edges to (bad) high odd vertices
- ... and (somehow) generalize from edges to paths

HOW DO THEY WORK?

How do parity game solving algorithms work?

- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

- Sounds simple?

- Follow edges to (good) high even vertices
- Avoid edges to (bad) high odd vertices
- ... and (somehow) generalize from edges to paths
- ... and (somehow) generalize from paths to (sets of) cycles

HOW DO THEY WORK?

How do parity game solving algorithms work?

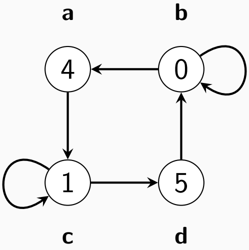
- Systematically investigate possible paths (cycles)
- Keep good paths (even cycles)
- Reject bad paths (odd cycles)

- Sounds simple?

- Follow edges to (good) high even vertices
- Avoid edges to (bad) high odd vertices
- ... and (somehow) generalize from edges to paths
- ... and (somehow) generalize from paths to (sets of) cycles

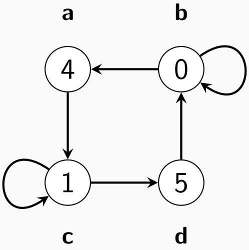
- BUT...

DISTRACTIONS



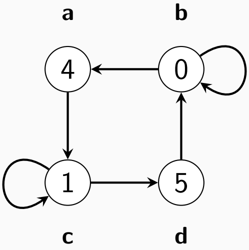
- Find the winning strategy for player Even.

DISTRACTIONS



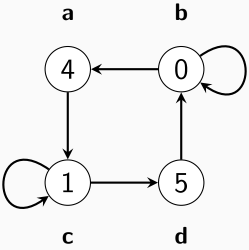
- Find the winning strategy for player Even.
- Pick a random strategy inside the winning region?

DISTRACTIONS



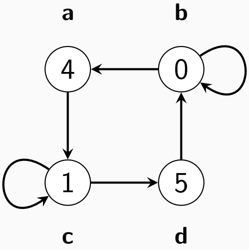
- Find the winning strategy for player Even.
- Pick a random strategy inside the winning region?
  If you play from **c** to **c** you lose.

DISTRACTIONS



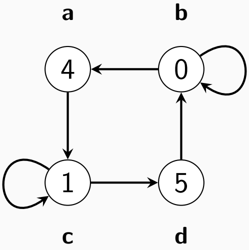
- Find the winning strategy for player Even.
- Pick a random strategy inside the winning region?
  If you play from **c** to **c** you lose.
- Play to nice high Even vertices?

DISTRACTIONS



- Find the winning strategy for player Even.
- Pick a random strategy inside the winning region?
  If you play from **c** to **c** you lose.
- Play to nice high Even vertices?
  If you play from **b** to **a** you lose.

DISTRACTIONS



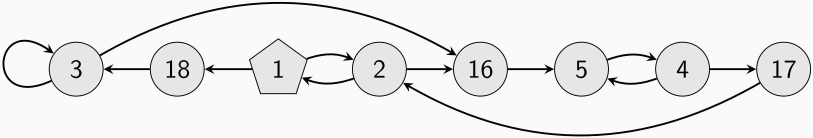
- Find the winning strategy for player Even.
- Pick a random strategy inside the winning region?
  If you play from **c** to **c** you lose.
- Play to nice high Even vertices?
  If you play from **b** to **a** you lose.
- Vertex **a** is a distraction for player Even

DISTRACTIONS

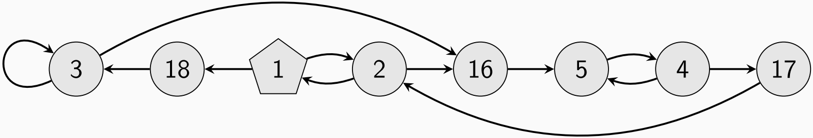
Intuition

- A distraction is a vertex that makes algorithms require more steps to solve the parity game why? because the algorithm tries paths (assumptions) that turn out to be unproductive
- A distraction is a vertex that *seems* a good target to play to but is actually bad (or less good than other targets)
- A distraction for α is a vertex with an α-priority that the opponent $\overline{\alpha}$ can win if player α always tries to visit it.

DISTRACTIONS



DISTRACTIONS



Vertex **16** is a distraction.
Play **2** $\rightarrow$ **1** instead of **2** $\rightarrow$ **16** to win!
But also play **3** $\rightarrow$ **16**!

Wrong assumption: play to **18**, **16**, **4**, **2**
To win: play to **2**, **18** $>$ rest

DISTRACTIONS

Identifying distractions

Every algorithm somehow overcomes distractions.

1. By showing that distractions lead to bad paths
   - Example: paths from **16** reach **17**, so avoid **16**
   - Decomposition-based algorithms
2. By showing that other vertices reach better paths
   - i.e. give a higher "value" to vertices along good paths (to good vertices)
   - Example: paths from **2** reach **16**, so play to **2**
   - Value iteration algorithms

TANGLE LEARNING

Normal tangle learning

- Pure decomposition based
- shows that a distraction leads to bad paths
- *does not* give higher value to vertices on good paths

Recursive tangle learning (this work)

- Further decomposes each “region” recursively
- shows that a distraction leads to bad paths
- *does* give higher value to vertices on good paths
  “I now try to play to a vertex inside a region rather than its top vertex”

ATTRACTOR COMPUTATION

Playing from **A** to **B**

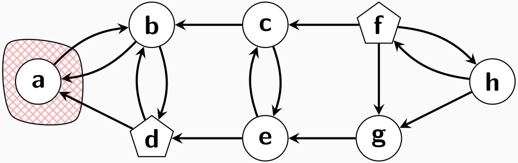
- From which vertices **A** must a play eventually reach **B**?
- What is the highest vertex that player α can reach?
- Which vertices cannot be avoided by the opponent $\overline{\alpha}$?

Attractor computation

- "Backward reachability" with an opponent
- Given target set Z and a player α,
  compute all vertices from which player α can ensure arrival in Z
- Add to Z (exhaustively):
  - all vertices of the player α that *can* play to Z
  - all vertices of opponent $\overline{\alpha}$ that *must* play to Z

Example of attractor computation

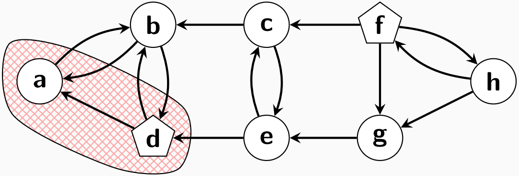
Computing the ⬟-attractor to **a**



Attractor set: $\{\mathbf{a}\}$
Can attract: **d** but not **b**

Example of attractor computation

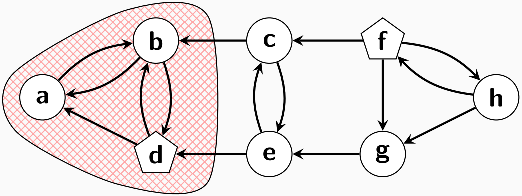
Computing the ⬠-attractor to **a**



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$
Can attract: **b** but not **e**

Example of attractor computation

Computing the ⬠-attractor to **a**

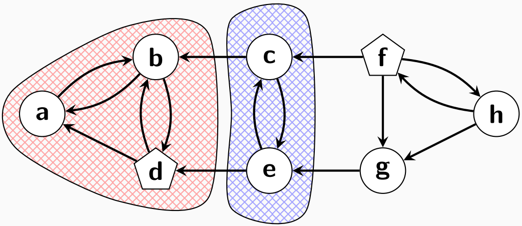


Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$
Can attract: neither **c** nor **e**

PARITY GAMES

Example of attractor computation

Computing the ⬟-attractor to **a**

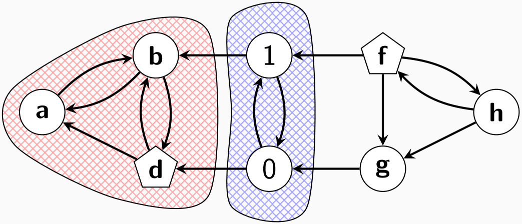


Attractor set: $\{\mathbf{a}\}$, $\{\,\mathbf{a},\mathbf{d}\}$, $\{\mathbf{a},\mathbf{b},\mathbf{d}\}$

What if player ● cannot stay in $\{\mathbf{c},\mathbf{e}\}$??

Example of attractor computation

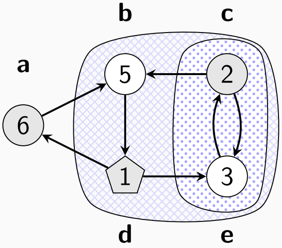
Computing the ⬠-attractor to **a**



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$
Every play inside the blue area is won by ⬠!

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.



A game with a 5-tangle and a 3-tangle

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Definition

A tangle is

- a pair $T = (U, \sigma)$ where
  - $U \subseteq V$ is a nonempty set of vertices
  - $\sigma\colon U_\alpha \to U$ is a strategy for player $\alpha := \text{pr}(U) \bmod 2$

  such that
- player α wins all cycles in the induced subgame $\mathcal{G}[U, \sigma]$
- the induced subgame $\mathcal{G}[U, \sigma]$ is strongly connected

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Properties

- The opponent must escape (or lose inside the tangle)
- The opponent can freely choose any escape (strongly connected)
- A closed tangle (no escapes) is a winning region

- Note: any “won subgame” can be decomposed into tangles

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

Tangles are fundamental

- All algorithms implicitly reason about tangles:
  - every algorithm must deal with cycles and nested cyclic structures
  - if you conclude that the opponent cannot 'hide' in some subgame
  - then this **must** be because of tangles
- Most algorithms often explore the same tangle many times!
  (This can lead to repetitive behavior and exponential blowup!)

TANGLES

Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

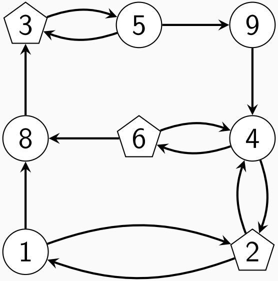
Tangle attractor

Because the opponent $\overline{\alpha}$ **must** escape the tangle, we can use tangles to attract all vertices of a tangle simultaneously.

Add to Z (exhaustively):

- all vertices of the player α that *can* play to Z
- all vertices of opponent $\overline{\alpha}$ that *must* play to Z
- all vertices in an α-tangle where all escapes lead to Z

EXAMPLE PARITY GAME



Tangle learning (two-player basic variation)

- Assume that every vertex has a unique priority
  (just to simplify the presentation)
- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left

TANGLE LEARNING

Tangle learning (two-player basic variation)

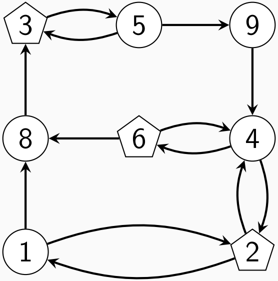
- Assume that every vertex has a unique priority
  (just to simplify the presentation)
- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left
- Every locally closed region contains a <u>new</u> tangle
  - Locally closed if the top vertex is attracted to the region
  - It is a won subgame, but it may not (yet) be strongly connected
  - Run Tarjan's SCC algorithm restricted to the attractor strategy
  - The result is the new tangle

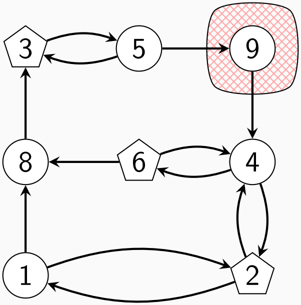
TANGLE LEARNING

Tangle learning (two-player basic variation)

- Assume that every vertex has a unique priority
  (just to simplify the presentation)
- Partition game into regions with tangle attractor
  - Attract to highest priority
  - Repeat with the remainder until nothing left
- Every locally closed region contains a <u>new</u> tangle
  - Locally closed if the top vertex is attracted to the region
  - It is a won subgame, but it may not (yet) be strongly connected
  - Run Tarjan's SCC algorithm restricted to the attractor strategy
  - The result is the new tangle
- Every closed tangle is a <u>winning region</u> (dominion)
  - After finding a dominion, maximize it with another attractor
  - Remove maximized dominions from the game

TANGLE LEARNING

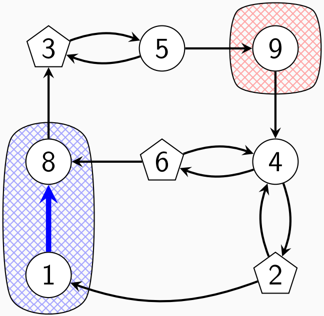




Regions:

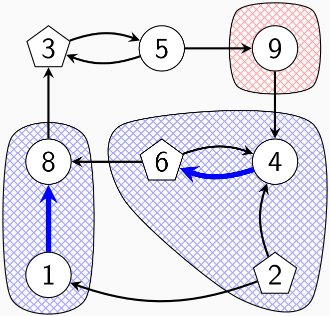
- 9 (open)

TANGLE LEARNING



Regions:

- 9 (open)
- 8 1 (open)

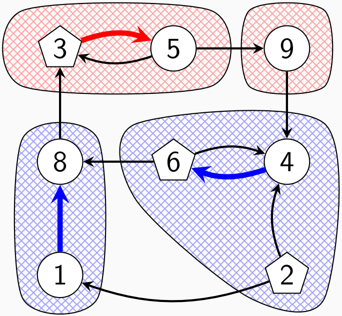


Regions:

- 9 (open)
- 8 1 (open)
- 6 4 2 (closed)

Learned tangles: $\{6, 4 \to 6\}$

TANGLE LEARNING

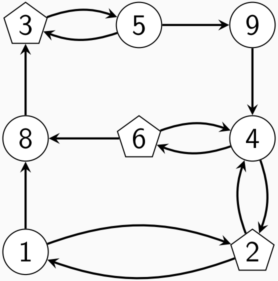


Regions:

- 9 (open)
- 8 1 (open)
- 6 4 2 (closed)
- 5 3 (closed)

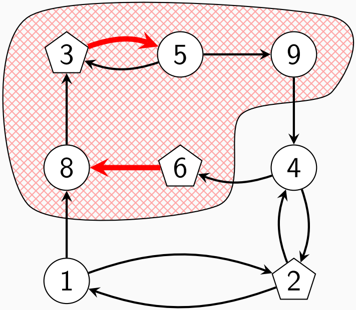
Learned tangles: $\{6, 4 \rightarrow 6\}$, $\{5, 3 \rightarrow 5\}$

TANGLE LEARNING



Learned tangles: $\{6, 4 \to 6\}$, $\{5, 3 \to 5\}$

TANGLE LEARNING

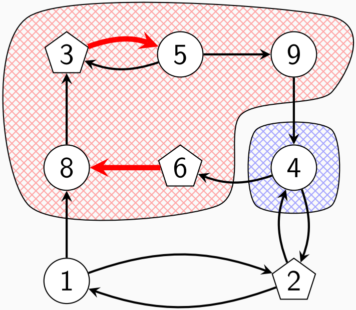


Regions:

- 9 5 3 8 6 (open)

Learned tangles: $\{6, 4 \rightarrow 6\}$, $\{5, 3 \rightarrow 5\}$

TANGLE LEARNING

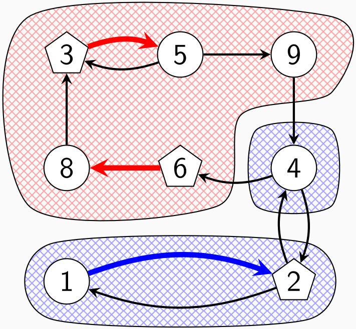


Regions:

- 9 5 3 8 6 (open)
- 4 (open)

Learned tangles: $\{6, 4 \to 6\}$, $\{5, 3 \to 5\}$

TANGLE LEARNING

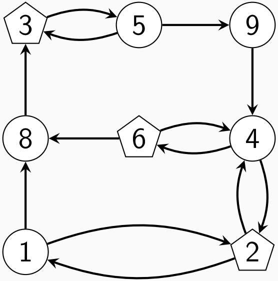


Regions:

- 9 5 3 8 6 (open)
- 4 (open)
- 2 1 (closed)

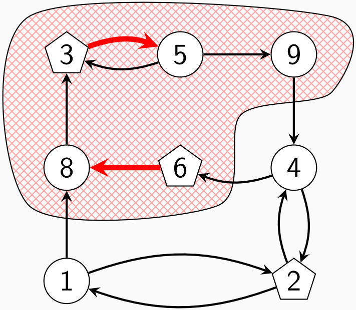
Learned tangles: $\{6, 4\to 6\}$, $\{5, 3\to 5\}$, $\{2, 1\to 2\}$

TANGLE LEARNING



Learned tangles: $\{6, 4 \to 6\}$, $\{5, 3 \to 5\}$, $\{2, 1 \to 2\}$

TANGLE LEARNING

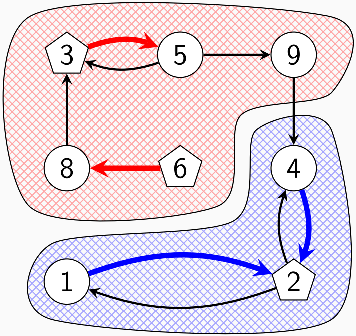


Regions:

- 9 5 3 8 6 (open)

Learned tangles: $\{6, 4{\rightarrow}6\}$, $\{5, 3{\rightarrow}5\}$, $\{2, 1{\rightarrow}2\}$

TANGLE LEARNING

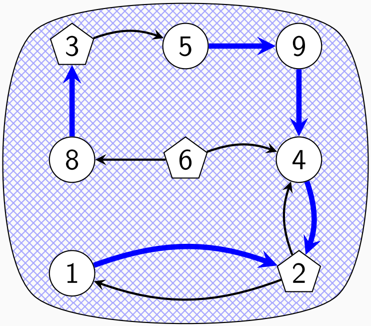


Regions:

- 9 5 3 8 6 (open)
- 4 2 1 (closed)

Learned tangles: $\{6, 4{\to}6\}$, $\{5, 3{\to}5\}$, $\{2, 1{\to}2\}$, $\underline{\{4{\to}2, 2, 1{\to}2\}}$

TANGLE LEARNING



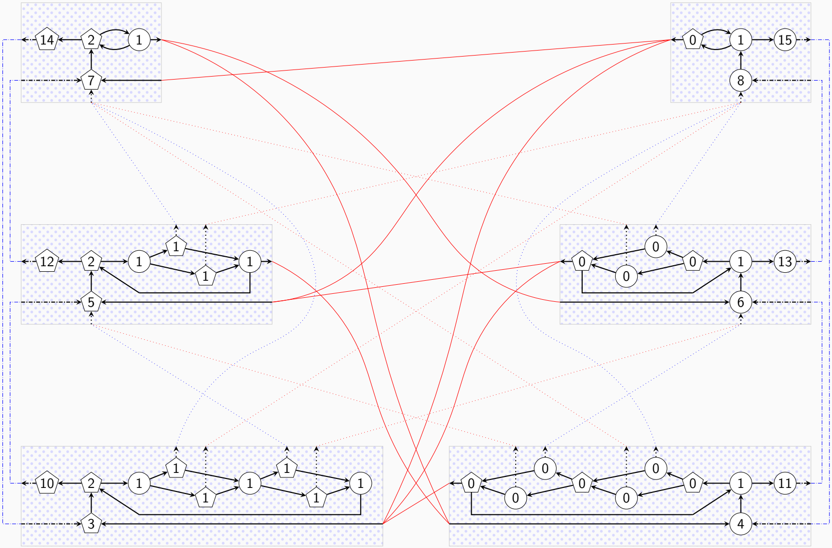
Learned tangles: $\{6, 4\to 6\}$, $\{5, 3\to 5\}$, $\{2, 1\to 2\}$, $\underline{\{4\to 2, 2, 1\to 2\}}$

TANGLE LEARNING

Tangle learning (two-player basic variation)

- Problem! This algorithm has an exponential lower bound.
- Defeated by the two binary counters game

EXAMPLE 3-BIT GAME



EXAMPLE 3-BIT GAME

Iteration	Even	Odd	Event
	000	000	Initial state
1	001	000	Set Even 3
2	001	001	Set Odd 3
3	011	000	Set Even 2 (reset Odd 3)
4	010	010	Set Odd 2 (reset Even 3)
5	011	010	Set Even 3
6	011	011	Set Odd 3
7	111	000	Set Even 1 (reset Odd 2, 3)
8	100	100	Set Odd 1 (reset Even 2, 3)
9	101	100	Set Even 3
10	101	101	Set Odd 3
11	111	100	Set Even 2 (reset Odd 3)
12	110	110	Set Odd 2 (reset Even 3)
13	111	110	Set Even 3
14	111	111	Set Odd 3

RECURSIVE TANGLE LEARNING

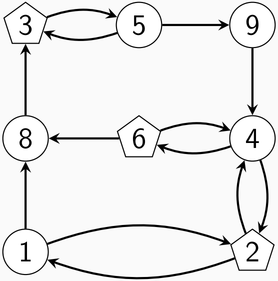
Recursive tangle learning (two-player variation)

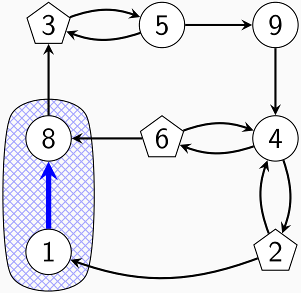
- Same as tangle learning, and...
- ...partition every (open) region recursively:
    - “We now want to avoid the top vertex instead”
    - Remove the opponent attractor (inside the region) to the top vertex
    - Partition the remainder of the region

Recursive tangle learning (one-player variation)

- Only consider the even priority vertices as attractor targets (or only odd priority vertices)

RECURSIVE TANGLE LEARNING

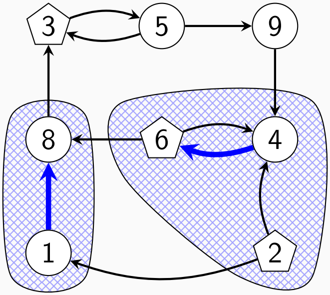




Regions:

- 8 1 (open)

  recursive: -

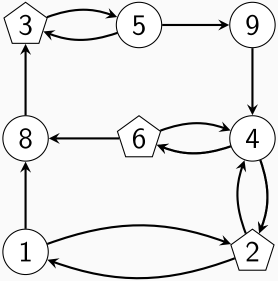


Regions:

- 8 1 (open)
  recursive: -
- 6 4 2 (closed)

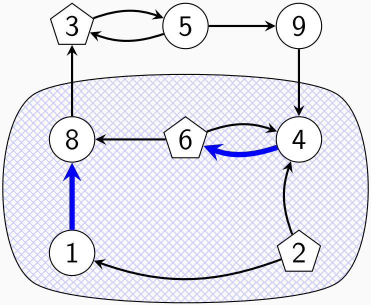
Learned tangles: $\{6, 4 \to 6\}$

RECURSIVE TANGLE LEARNING



Learned tangles: $\{6, 4 \to 6\}$

RECURSIVE TANGLE LEARNING

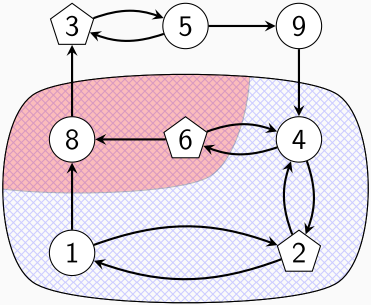


Regions:

- 8 6 4 1 2 (open)
  recursive: 4 2 1

Learned tangles: $\{6, 4 \to 6\}$

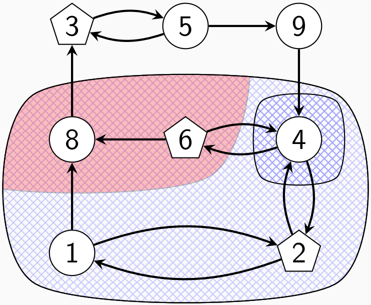
RECURSIVE TANGLE LEARNING



Regions:

- 8 6 4 1 2 (open)
  recursive: 4 2 1

Learned tangles: $\{6, 4 \rightarrow 6\}$

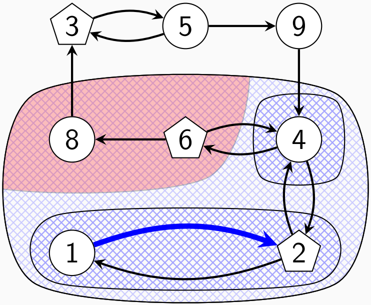


Regions:

- 8 6 4 1 2 (open)
  recursive: 4 2 1
- 4 (open)
  recursive: -

Learned tangles: $\{6, 4 \to 6\}$

RECURSIVE TANGLE LEARNING

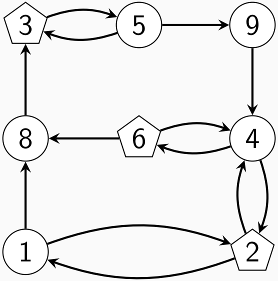


Regions:

- 8 6 4 1 2 (open)
  recursive: 4 2 1
- 4 (open)
  recursive: -
- 2 1 (closed)

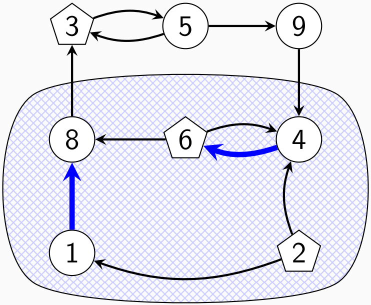
Learned tangles: $\{6, 4 \rightarrow 6\}$, $\{2, 1 \rightarrow 2\}$

RECURSIVE TANGLE LEARNING



Learned tangles: $\{6, 4 \to 6\}$, $\{2, 1 \to 2\}$

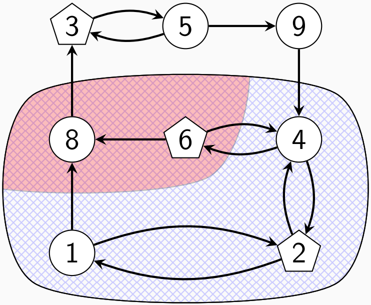
RECURSIVE TANGLE LEARNING



Regions:

- 8 6 4 2 1 (open)
  recursive: 4 2 1

Learned tangles: $\{6, 4 \to 6\}$, $\{2, 1 \to 2\}$

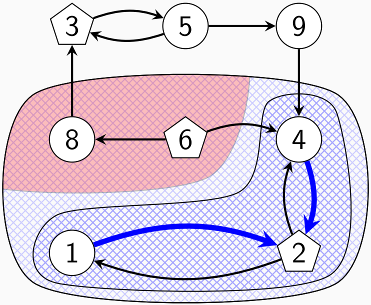


Regions:

- 8 6 4 2 1 (open)
  recursive: 4 2 1

Learned tangles: $\{6, 4\to 6\}$, $\{2, 1\to 2\}$

RECURSIVE TANGLE LEARNING



Regions:

- 8 6 4 2 1 (open)
  recursive: 4 2 1
- 4 2 1 (closed)

Learned tangles: $\{6, 4{\to}6\}$, $\{2, 1{\to}2\}$, $\underline{\{4{\to}2, 2, 1{\to}2\}}$

DISTRACTIONS

- Vertex 8 was a distraction
- Two-player tangle learning simply attracts 8 via $\{3, 5\}$ to 9
- Recursive tangle learning “avoids” 8 in the recursion

DISTRACTIONS

- Vertex 8 was a distraction
- Two-player tangle learning simply attracts 8 via $\{3, 5\}$ to 9
- Recursive tangle learning “avoids” 8 in the recursion
- Recursive tangle learning fixes TBC!
- But recursive tangle learning is easily defeated too (see paper)

PERSPECTIVES

- TL and RTL have worst-case exponential behavior
- Ongoing work: combining TL with universal tree values (PMTL)
- Ongoing work: heuristics for "currently distractions" (DF*TL)
- Ideas to prove that TL cannot solve in polynomial time
  - Learning tangles moves vertices along the universal tree
    - Every algorithm orders vertices with a universal tree
  - Maybe: the maximal "knowledge" (tangles and path values) per iteration can be characterized, such that the number of steps in the worst case is quasi-polynomial?
  - Could be generalized to classes of algorithms?