The combinatorics of monadic stability, monadic dependence, and related notions

Algomanet, Warsaw, September 9-13, 2024

Jan Dreier, TU Wien

Today's Goal

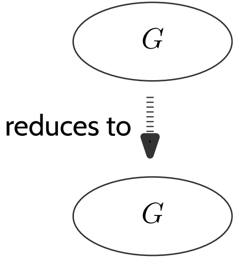
Today, we want to prove the following milestone.

D, Mählmann, Siebertz, 2023

D, Eleftheriadis, Mählmann, McCarty, Pilipczuk, Toruńczyk, 2024

Let $\mathcal{C}$ be a monadically stable graph class. There exists a function f such that for every FO formula φ and graph $G \in \mathcal{C}$ one can decide whether $G \models \varphi$ in time $f(|\varphi|)n^6$.

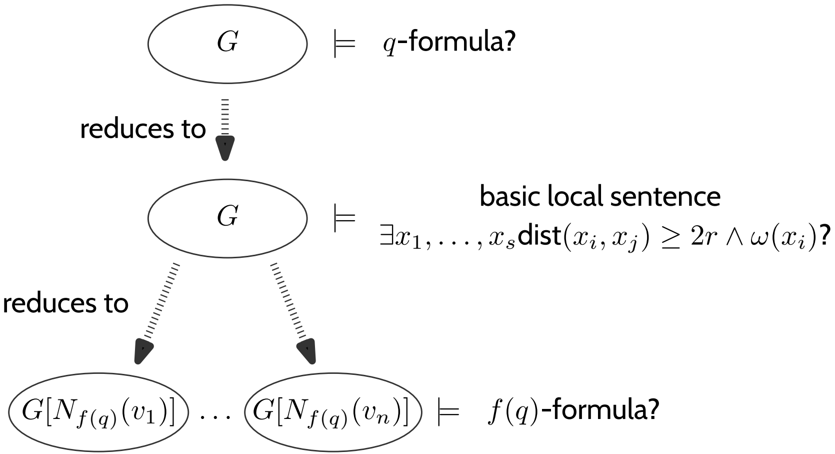
Gaifman–Approach



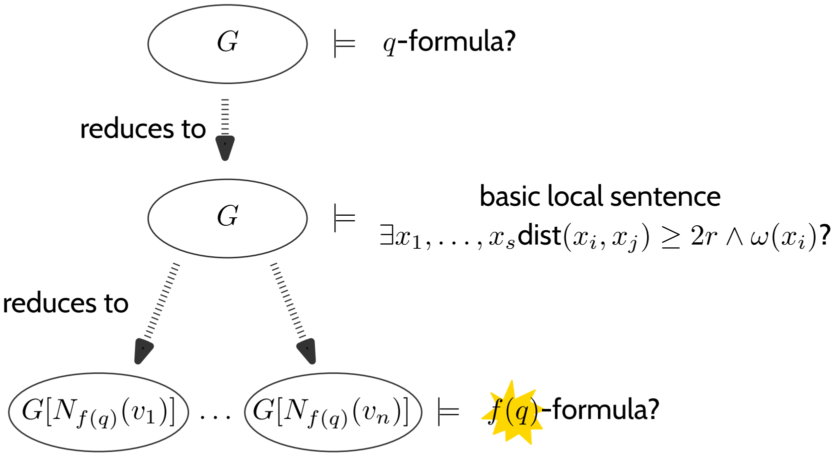
$G \models$ q-formula?

$G \models$ basic local sentence $\exists x_1, \ldots, x_s \text{dist}(x_i, x_j) \geq 2r \wedge \omega(x_i)$?

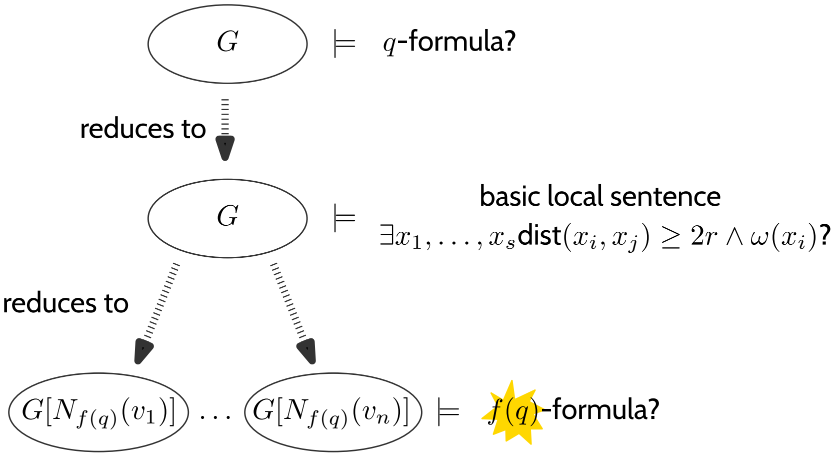
Gaifman–Approach



Gaifman–Approach



Gaifman–Approach



We want a stronger mechanism with $f(q) = q$.

q-types

Definition

Let q-type(G, v) be the set of all formulas φ of quantifier rank at most q with $G \models \varphi(v)$.

q-types

Definition

Let q-type(G, v) be the set of all formulas φ of quantifier rank at most q with $G \models \varphi(v)$.

The number of (normalized) formulas of quantifier rank q and c colors is bounded by some function $f(q, c)$.

q-types

Definition

Let q-type(G, v) be the set of all formulas φ of quantifier rank at most q with $G \models \varphi(v)$.

The number of (normalized) formulas of quantifier rank q and c colors is bounded by some function $f(q, c)$.

The number of q-types in graphs with c colors is bounded by $2^{f(q,c)}$.

Algorithm Idea

Idea of Gajarský, Gorsky, Kreutzer (2020): Assume we want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x \, \forall y \; \varphi(x, y)$ of quantifier rank q.

Algorithm Idea

Idea of Gajarský, Gorsky, Kreutzer (2020): Assume we want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \;\vee\; G \models \forall y \, \varphi(v_2, y) \;\vee\; \ldots \;\vee\; G \models \forall y \, \varphi(v_n, y).$$

Algorithm Idea

Idea of Gajarský, Gorsky, Kreutzer (2020): Assume we want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \;\lor\; G \models \forall y \, \varphi(v_2, y) \;\lor\; \ldots \;\lor\; G \models \forall y \, \varphi(v_n, y).$$

Assume q-type$(G, v_1) = q$-type(G, v_2). In other words, for all formulas $\psi(x)$ of quantifier rank q, $G \models \psi(v_1) \iff G \models \psi(v_2)$.

Algorithm Idea

Idea of Gajarský, Gorsky, Kreutzer (2020): Assume we want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \ \vee \ \ G \models \forall y \, \varphi(v_2, y) \ \ \vee \ \ \ldots \ \ \vee \ G \models \forall y \, \varphi(v_n, y).$$

Assume q-type$(G, v_1) = q$-type(G, v_2). In other words, for all formulas $\psi(x)$ of quantifier rank q, $G \models \psi(v_1) \iff G \models \psi(v_2)$.

In particular, $G \models \forall y \, \varphi(v_1, y) \iff G \models \forall y \, \varphi(v_2, y)$.

Algorithm Idea

Idea of Gajarský, Gorsky, Kreutzer (2020): Assume we want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x\, \forall y\, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x\, \forall y\, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y\, \varphi(v_1, y) \;\vee\; \cancel{G \models \forall y\, \varphi(v_2, y)} \;\vee\; \ldots \;\vee\; G \models \forall y\, \varphi(v_n, y).$$

Assume q-type$(G, v_1) = q$-type(G, v_2). In other words, for all formulas $\psi(x)$ of quantifier rank q, $G \models \psi(v_1) \iff G \models \psi(v_2)$.

In particular, $G \models \forall y\, \varphi(v_1, y) \iff G \models \forall y\, \varphi(v_2, y)$.
We only need to keep one “representative” of this type.

Local Types

Assume we have a fast blackbox algorithm to evaluate q-formulas on 2^q-balls of G. We can compute q-type$(G[N_{2^q}(v)], v)$ for all $v \in V(G)$,

Local Types

Assume we have a fast blackbox algorithm to evaluate q-formulas on 2^q-balls of G. We can compute q-type$(G[N_{2^q}(v)], v)$ for all $v \in V(G)$, but it does not give us global the global q-type(G, v).

Local Types

Assume we have a fast blackbox algorithm to evaluate q-formulas on 2^q-balls of G. We can compute q-type$(G[N_{2^q}(v)], v)$ for all $v \in V(G)$, but it does not give us global the global q-type(G, v).

But it lets us distinguish global q-types!

Local Types

Assume we have a fast blackbox algorithm to evaluate q-formulas on 2^q-balls of G. We can compute $q\text{-type}(G[N_{2^q}(v)], v)$ for all $v \in V(G)$, but it does not give us global the global $q\text{-type}(G, v)$.

But it lets us distinguish global q-types!

Theorem (Siebertz, Toruńczyk)

Let G be a graph and a, b be two vertices with distance more than 2^q and

$$q\text{-type}(G[N_{2^q}(a)], a) = q\text{-type}(G[N_{2^q}(b)], b).$$

Then

$$q\text{-type}(G, a) = q\text{-type}(G, b).$$

Local Types

Assume we have a fast blackbox algorithm to evaluate q-formulas on 2^q-balls of G. We can compute q-type$(G[N_{2^q}(v)], v)$ for all $v \in V(G)$, but it does not give us global the global q-type(G, v).

But it lets us distinguish global q-types!

Theorem (Siebertz, Toruńczyk) (simplified, but wrong)

Let G be a graph and a, b be two vertices with

$$q\text{-type}(G[N_{2^q}(a)], a) = q\text{-type}(G[N_{2^q}(b)], b).$$

Then

$$q\text{-type}(G, a) = q\text{-type}(G, b).$$

Let us ignore the distance requirement.

Algorithm Idea

We want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \;\vee\; G \models \forall y \, \varphi(v_2, y) \;\vee\; \ldots \;\vee\; G \models \forall y \, \varphi(v_n, y).$$

Algorithm Idea

We want to evaluate on a graph G with vertices $v_1, \dots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\Longleftrightarrow$$

$$G \models \forall y \, \varphi(v_1, y) \;\vee\; G \models \forall y \, \varphi(v_2, y) \;\vee\; \dots \;\vee\; G \models \forall y \, \varphi(v_n, y).$$

Assume q-type$(G[N_{2^q}(v_1)], v_1) = q$-type$(G[N_{2^q}(v_2)], v_2)$. Then by the previous theorem q-type$(G, v_1) = q$-type(G, v_2).

Algorithm Idea

We want to evaluate on a graph G with vertices $v_1, \ldots, v_n$ a formula $\exists x\, \forall y\, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x\, \forall y\, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y\, \varphi(v_1, y) \;\vee\; G \models \forall y\, \varphi(v_2, y) \;\vee\; \ldots \;\vee\; G \models \forall y\, \varphi(v_n, y).$$

Assume q-type$(G[N_{2^q}(v_1)], v_1) = q$-type$(G[N_{2^q}(v_2)], v_2)$. Then by the previous theorem q-type$(G, v_1) = q$-type(G, v_2).

As argued before, $G \models \forall y\, \varphi(v_1, y) \iff G \models \forall y\, \varphi(v_2, y)$.

Algorithm Idea

We want to evaluate on a graph G with vertices $v_1, \dots, v_n$ a formula $\exists x \, \forall y \, \varphi(x, y)$ of quantifier rank q. Then

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \;\vee\; \cancel{G \models \forall y \, \varphi(v_2, y)} \;\vee\; \dots \;\vee\; G \models \forall y \, \varphi(v_n, y).$$

Assume q-type$(G[N_{2^q}(v_1)], v_1) = q$-type$(G[N_{2^q}(v_2)], v_2)$. Then by the previous theorem q-type$(G, v_1) = q$-type(G, v_2).

As argued before, $G \models \forall y \, \varphi(v_1, y) \iff G \models \forall y \, \varphi(v_2, y)$.
We only need to keep one "representative" of this local type.

Algorithm Idea

Build a small representative set of vertices S such that

$$\{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in S\} = \{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in V(G)\}.$$

Algorithm Idea

Build a small representative set of vertices S such that

$$\{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in S\} = \{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in V(G)\}.$$

As argued before, we can shorten our disjunction.

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$G \models \forall y \, \varphi(v_1, y) \;\vee\; \dots \;\vee\; G \models \forall y \, \varphi(v_n, y)$$

Algorithm Idea

Build a small representative set of vertices S such that

$$\{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in S\} = \{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in V(G)\}.$$

As argued before, we can shorten our disjunction.

$$G \models \exists x\, \forall y\, \varphi(x, y)$$

$$\iff$$

$$\bigvee_{v \in S} G \models \forall y\, \varphi(v, y)$$

Algorithm Idea

Build a small representative set of vertices S such that

$\{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in S\} = \{q\text{-type}(G[N_{2^q}(v)], v) \mid v \in V(G)\}.$

As argued before, we can shorten our disjunction.

$$G \models \exists x \, \forall y \, \varphi(x, y)$$

$$\iff$$

$$\bigvee_{v \in S} G \models \forall y \, \varphi(v, y)$$

The size of S is bounded by the number of q-types, which is bounded by a function of q and the number of colors of the graph.

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} G \models \forall y\, \varphi(w, y).$$

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} G \models \forall y\, \varphi(w, y).$$

For every $w \in S$ we rewrite

$$G \models \forall y\, \varphi(w, y)$$

$$\iff$$

$$G \models \varphi(w, v_1) \ \wedge \ \ldots \ \wedge \ G \models \varphi(w, v_n).$$

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} G \models \forall y \, \varphi(w, y).$$

For every $w \in S$ we rewrite

$$G \models \forall y \, \varphi(w, y)$$

$$\Longleftrightarrow$$

$$G \models \varphi(w, v_1) \ \wedge \ \ldots \ \wedge \ G \models \varphi(w, v_n).$$

As before, we construct a set S_w such that

$$\{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in S_w\} = \{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in V(G)\}.$$

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} G \models \forall y \, \varphi(w, y).$$

For every $w \in S$ we rewrite

$$G \models \forall y \, \varphi(w, y)$$

$$\iff$$

$$\bigwedge_{v \in S_w} G \models \varphi(w, v)$$

As before, we construct a set S_w such that

$$\{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in S_w\} = \{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in V(G)\}.$$

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} \bigwedge_{v \in S_w} G \models \varphi(w, v).$$

For every $w \in S$ we rewrite

$$G \models \forall y\, \varphi(w, y)$$

$$\iff$$

$$\bigwedge_{v \in S_w} G \models \varphi(w, v)$$

As before, we construct a set S_w such that

$$\{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in S_w\} = \{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in V(G)\}.$$

Algorithm Idea

We continue simplifying the formula

$$\bigvee_{w \in S} \bigwedge_{v \in S_w} G \models \varphi(w, v).$$

For every $w \in S$ we rewrite

$$G \models \forall y\, \varphi(w, y)$$

$$\iff$$

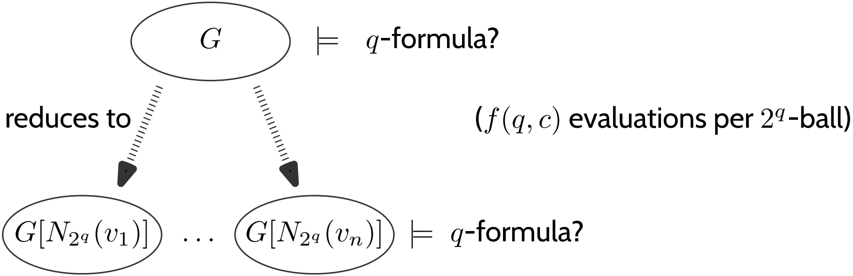
$$\bigwedge_{v \in S_w} G \models \varphi(w, v)$$

As before, we construct a set S_w such that

$$\{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in S_w\} = \{q\text{-type}(G[N_{2^r}(v)], wv) \mid v \in V(G)\}.$$

Continue like this until all quantifiers are replaced with constant-length conjunctions and disjunctions.

Quantifier-Rank Preserving Localization



Recursion

In monadically stable classes, the local parts are not yet simple enough to directly evaluate formulas.

Recursion

In monadically stable classes, the local parts are not yet simple enough to directly evaluate formulas. Instead, we

- modify each local part by flipping a vertex set,
- recurse into the modified local parts.

Recursion

In monadically stable classes, the local parts are not yet simple enough to directly evaluate formulas. Instead, we

- modify each local part by flipping a vertex set,
- recurse into the modified local parts.

For monadically stable classes, this terminates after a bounded number of steps with graphs consisting of single vertices, where model-checking is trivial.

Recursion

In monadically stable classes, the local parts are not yet simple enough to directly evaluate formulas. Instead, we

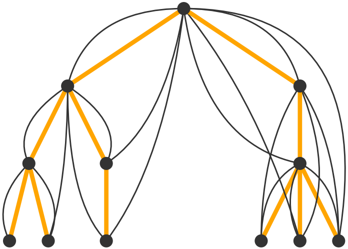
- modify each local part by flipping a vertex set,
- recurse into the modified local parts.

For monadically stable classes, this terminates after a bounded number of steps with graphs consisting of single vertices, where model-checking is trivial.

To show this, we need *pursuit-evasion games*.

∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

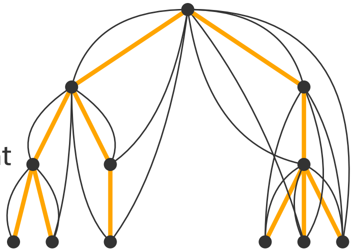


∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex



Splitter wins once a single vertex is reached.

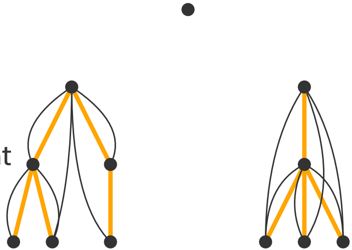
∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter.**

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.



Splitter isolates a vertex

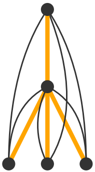
∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.



Connector picks connected component

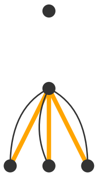
∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.



Splitter isolates a vertex

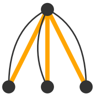
∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.



Connector picks connected component

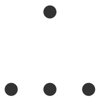
∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.



Splitter isolates a vertex

∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.

Connector picks connected component

∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.

Connector wins

∞-Splitter Game

Treedepth can be characterized by a game between **Connector** and **Splitter**.

∞-Splitter Game: In each round

- Connector picks connected component
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.

Connector wins

Characterization

A graph has treedepth $\leq d$ iff Splitter wins the ∞-Splitter game in $\leq d-1$ rounds.

r-Splitter Game

Here, the connected components get simpler over time, if we isolate vertices. But we only need the r-neighborhoods to get simpler!

r-Splitter Game

Here, the connected components get simpler over time, if we isolate vertices. But we only need the r-neighborhoods to get simpler!

r**-Splitter Game**: In each round

- Connector picks a subgraph with radius r (an r-ball)
- Splitter isolates a vertex

Splitter wins once a single vertex is reached.

r-Splitter Game

Here, the connected components get simpler over time, if we isolate vertices. But we only need the r-neighborhoods to get simpler!

r-**Splitter Game**: In each round

- Connector picks a subgraph with radius r (an r-ball)
- Splitter isolates a vertex

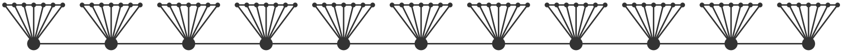
Splitter wins once a single vertex is reached.

Grohe, Kreuzer, Siebertz

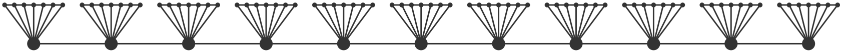
A class of graphs $\mathcal{C}$ is nowhere dense $\Leftrightarrow$

$\forall r \exists d$ such that Splitter wins the radius-r game on all graphs from $\mathcal{C}$ in d rounds.

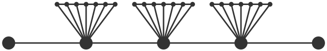
Example Play of 2-Splitter Game



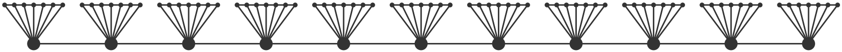
Example Play of 2-Splitter Game



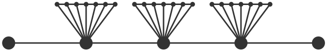
Connector picks 2-ball



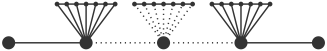
Example Play of 2-Splitter Game



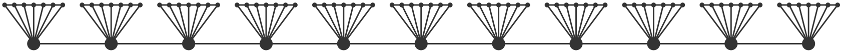
Connector picks 2-ball



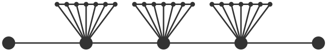
Splitter isolates vertex



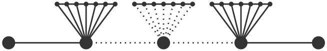
Example Play of 2-Splitter Game



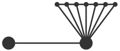
Connector picks 2-ball



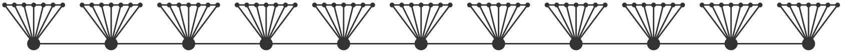
Splitter isolates vertex



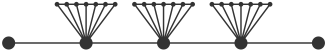
Connector picks 2-ball



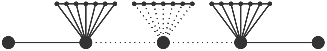
Example Play of 2-Splitter Game



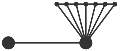
Connector picks 2-ball



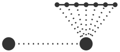
Splitter isolates vertex



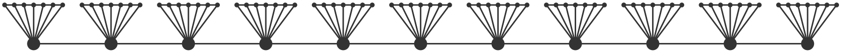
Connector picks 2-ball



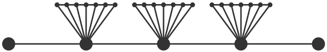
Splitter isolates vertex



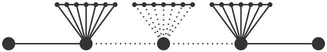
Example Play of 2-Splitter Game



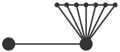
Connector picks 2-ball



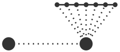
Splitter isolates vertex



Connector picks 2-ball

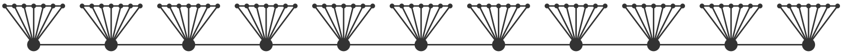


Splitter isolates vertex

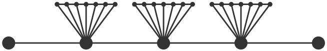


Connector picks 2-ball

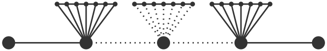
Example Play of 2-Splitter Game



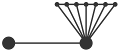
Connector picks 2-ball



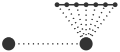
Splitter isolates vertex



Connector picks 2-ball



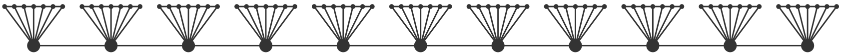
Splitter isolates vertex



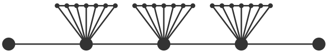
Connector picks 2-ball

[...]

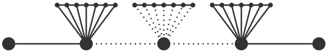
Example Play of 2-Splitter Game



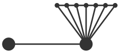
Connector picks 2-ball



Splitter isolates vertex



Connector picks 2-ball



Splitter isolates vertex

Connector picks 2-ball

[...]

Single vertex: Splitter wins

Cliques

How long does it take Flipper to win the radius-r game on a clique of size n?

Cliques

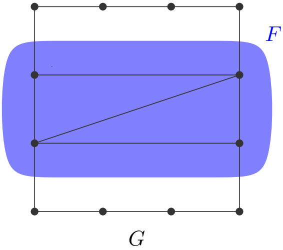
How long does it take Flipper to win the radius-r game on a clique of size n? Too long.

Flips

Denote by $G \oplus F$ the graph obtained from G by complementing edges between pairs of vertices from F.

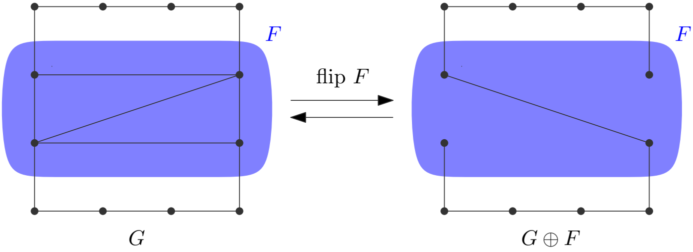
Flips

Denote by $G \oplus F$ the graph obtained from G by complementing edges between pairs of vertices from F.



Flips

Denote by $G \oplus F$ the graph obtained from G by complementing edges between pairs of vertices from F.



Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

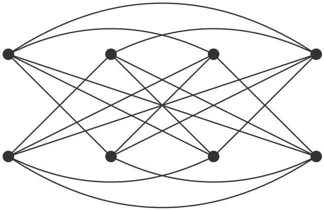
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



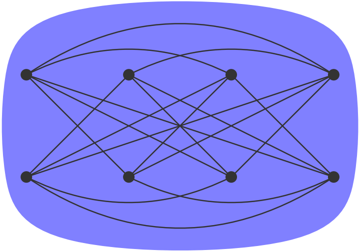
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



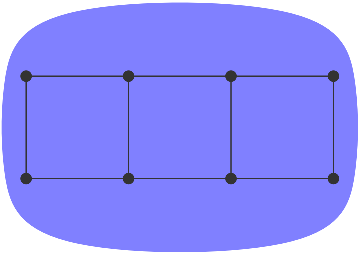
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



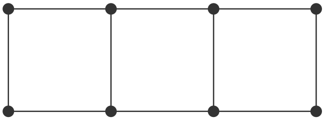
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



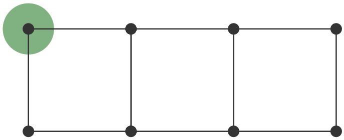
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



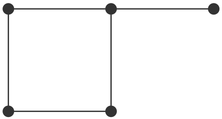
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



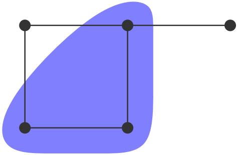
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



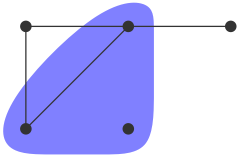
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



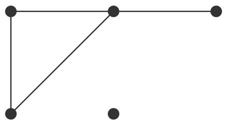
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



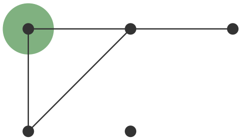
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



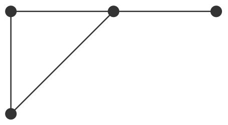
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



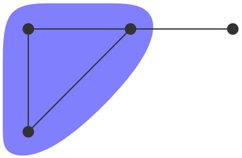
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



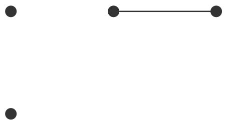
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



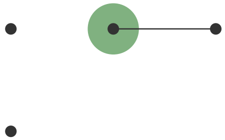
Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:



Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

Flipper Game

The radius-r Flipper game is played on a graph G_1. In round i

1. Flipper chooses a flip set F
2. Connector chooses G_{i+1} as a radius-r ball in $G_i \oplus F$.

Flipper wins once G_i has size 1.

Example play of the radius-2 Flipper game:

Flipper Game and Monadic Stability

Gajarský, Mählmann, McCarty, Ohlmann, Pilipczuk, Przybyszewski, Siebertz, Sokołowski, Toruńczyk, 2023

A class of graphs $\mathcal{C}$ is monadically stable $\Leftrightarrow$

$\forall r \exists d$ such that Flipper wins the radius-r game on all graphs from $\mathcal{C}$ in d rounds.

Flipper Game and Monadic Stability

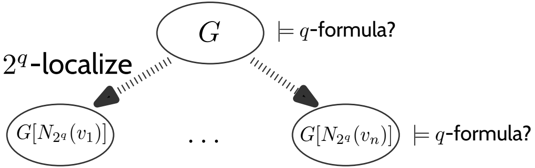
Gajarský, Mählmann, McCarty, Ohlmann, Pilipczuk, Przybyszewski, Siebertz, Sokołowski, Toruńczyk, 2023

A class of graphs $\mathcal{C}$ is monadically stable $\Leftrightarrow$

$\forall r \exists d$ such that Flipper wins the radius-r game on all graphs from $\mathcal{C}$ in d rounds.

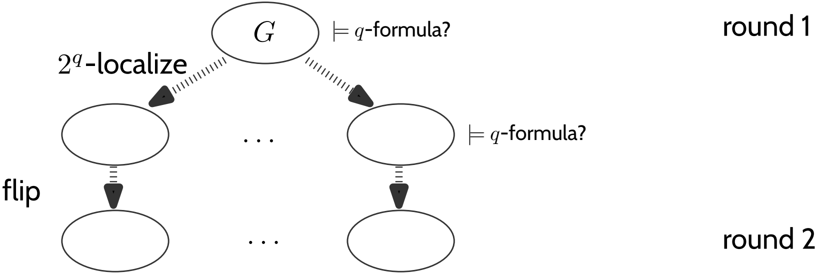
Moreover, Flipper's moves can be computed in time $O(n^2)$.

Guiding the Recursion with Splitter Games



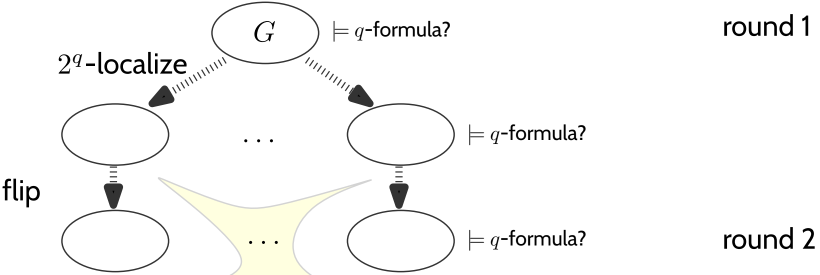
round 1

Guiding the Recursion with Splitter Games



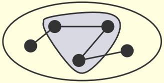
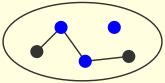
Nodes in our recursion tree are positions of the 2^q-splitter game!

Guiding the Recursion with Splitter Games



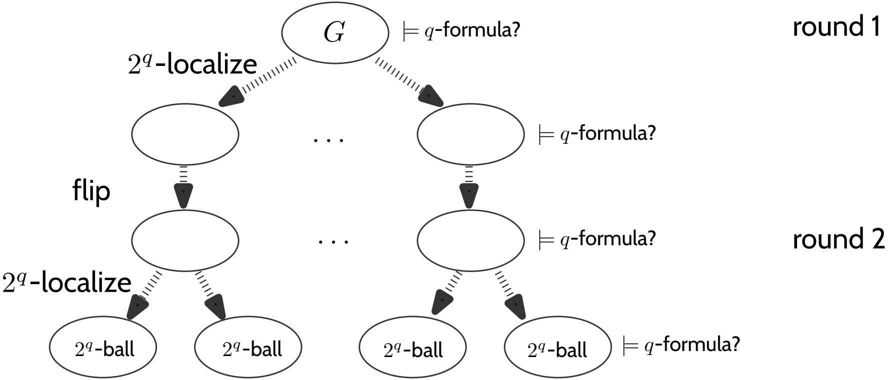
Give flip-set F a new color.

Update q-formulas by replacing each edge relation:

 $\models \; E(x, y) \quad \Longleftrightarrow$  $\models \; E(x,y) \oplus (\text{blue}(x) \land \text{blue}(y))$

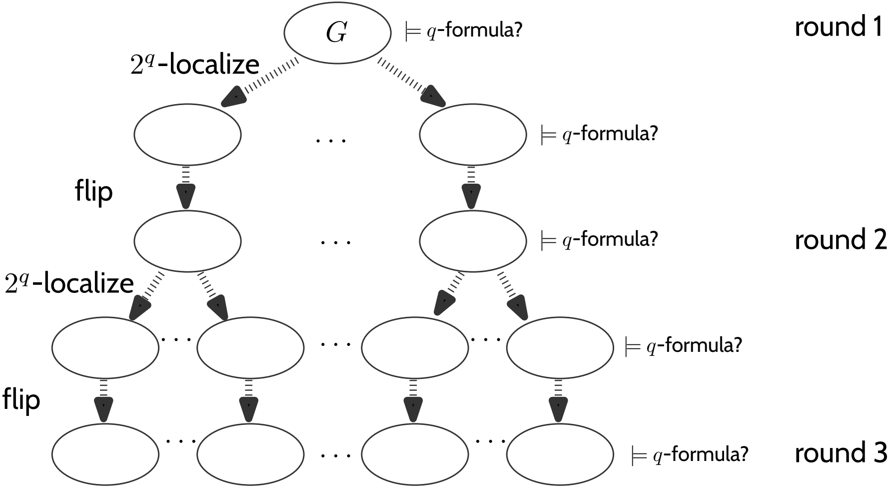
Nodes in our recursion tree are positions of the 2^q-splitter game!

Guiding the Recursion with Splitter Games



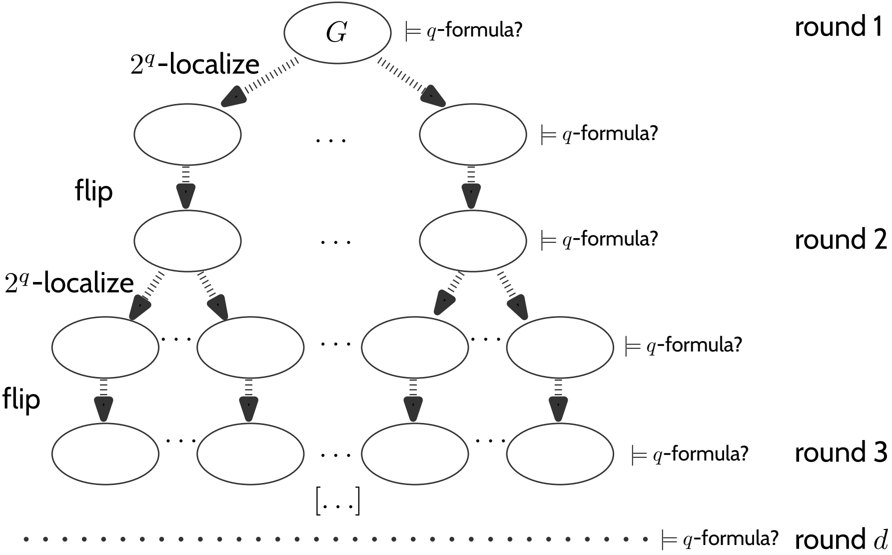
Nodes in our recursion tree are positions of the 2^q-splitter game!

Guiding the Recursion with Splitter Games



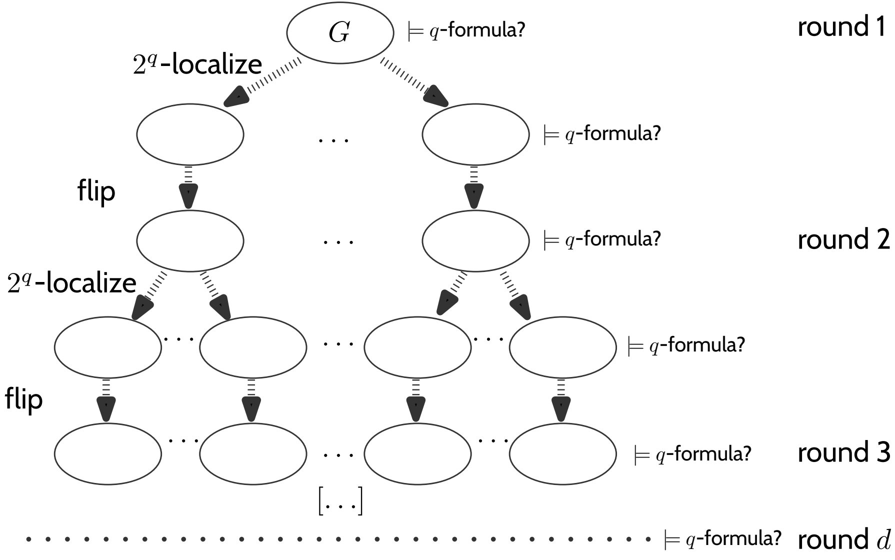
Nodes in our recursion tree are positions of the 2^q-splitter game!

Guiding the Recursion with Splitter Games



Once we reach single vertices, we are done.

Guiding the Recursion with Splitter Games

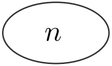


Why is it important that the quantifier-rank is preserved?

There is just one problem...

We evaluate a formula on a graph by recursing into all 2^q-balls of that graph. If we do it naively, the running time explodes:

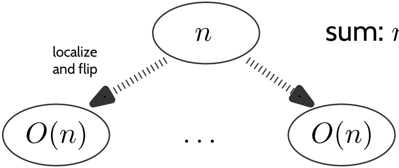
Recursion Trees are Large



sum: n vertices

Recursion into all 2^q-balls.

Recursion Trees are Large

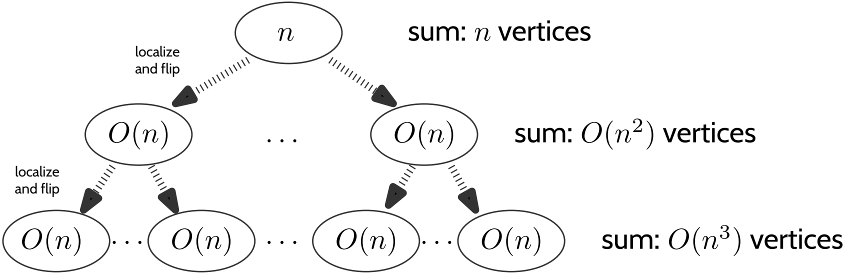


sum: n vertices

sum: $O(n^2)$ vertices

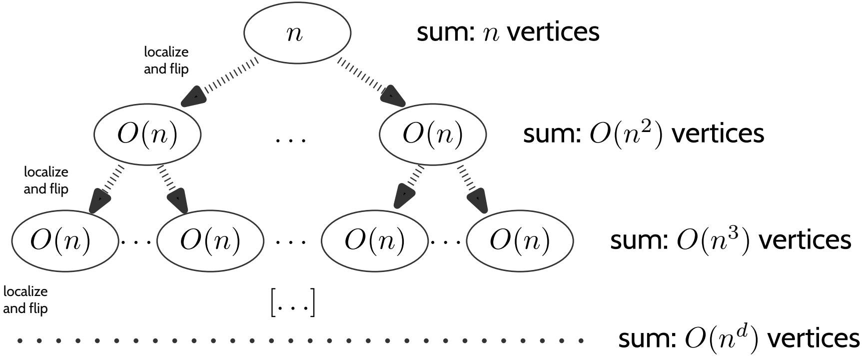
Recursion into all 2^q-balls. In a graph with n vertices, it may be that $\sum_{v \in V(G)} |N_{2^q}(v)| = n^2$.

Recursion Trees are Large



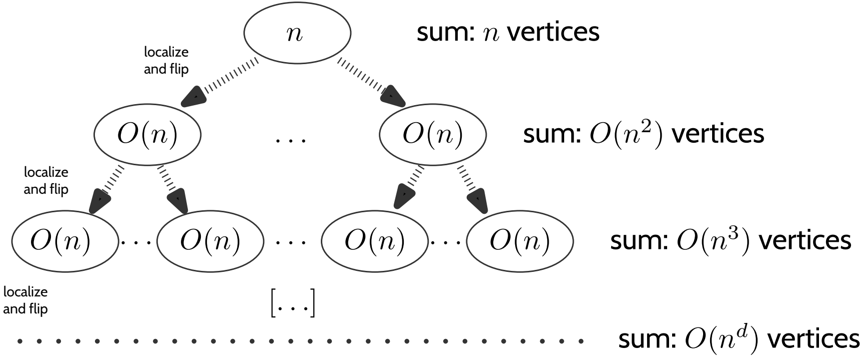
Recursion into all 2^q-balls. In a graph with n vertices, it may be that $\sum_{v \in V(G)} |N_{2^q}(v)| = n^2$. The recursion tree grows fast, even if we apply flips in between.

Recursion Trees are Large



Recursion into all 2^q-balls. In a graph with n vertices, it may be that $\sum_{v \in V(G)} |N_{2^q}(v)| = n^2$. The recursion tree grows fast, even if we apply flips in between. In the end, it may contain graphs whose number of vertices sum up to n^d. This does not lead to an FPT run time.

Recursion Trees are Large

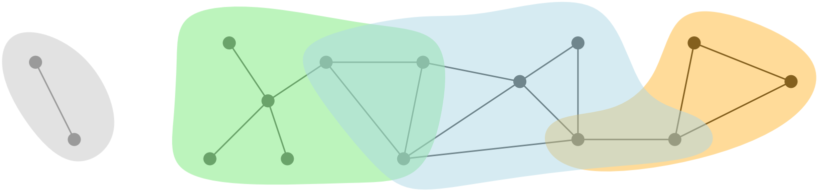


Recursion into all 2^q-balls. In a graph with n vertices, it may be that $\sum_{v \in V(G)} |N_{2^q}(v)| = n^2$. The recursion tree grows fast, even if we apply flips in between. In the end, it may contain graphs whose number of vertices sum up to n^d. This does not lead to an FPT run time. We instead group recursive calls together using *neighborhood covers*.

Neighborhood Covers

We say an r-ball is a subgraph with radius r. An *r-neighborhood cover* with overlap Δ in a graph G is a collection of sets $C_1, \ldots, C_l \subseteq V(G)$ such that

- every r-ball of G is contained in some C_i,
- every C_i is contained in some $4r$-ball of G,
- every vertex of G is contained in at most Δ many C_i.



1-neigbhorhood cover with degree 2

Neighborhood Covers

We say an r-ball is a subgraph with radius r. An r-*neighborhood cover* with overlap Δ in a graph G is a collection of sets $C_1, \ldots, C_l \subseteq V(G)$ such that

- every r-ball of G is contained in some C_i,
- every C_i is contained in some $4r$-ball of G,
- every vertex of G is contained in at most Δ many C_i.

D, Eleftheriadis, Mählmann, McCarty, Pilipczuk, Toruńczyk, 2024

Let $\mathcal{C}$ be a monadically stable graph class. For every $r \in \mathbb{N}$ and $\varepsilon > 0$ there exists c, such that every n-vertex $G \in \mathcal{C}$ has an r-neighborhood cover with overlap $c \cdot n^{\varepsilon}$.

1-neigbhorhood cover with degree 2

Neighborhood Covers

We say an r-ball is a subgraph with radius r. An r*-neighborhood cover* with overlap Δ in a graph G is a collection of sets $C_1, \dots, C_l \subseteq V(G)$ such that

- every r-ball of G is contained in some C_i,
- every C_i is contained in some $4r$-ball of G,
- every vertex of G is contained in at most Δ many C_i.

D, Eleftheriadis, Mählmann, McCarty, Pilipczuk, Toruńczyk, 2024

Let $\mathcal{C}$ be a monadically stable graph class. For every $r \in \mathbb{N}$ and $\varepsilon > 0$ there exists c, such that every n-vertex $G \in \mathcal{C}$ has an r-neighborhood cover with overlap $c \cdot n^\varepsilon$.

Then in particular, $\sum_{i=1}^{l} |C_i| \leq n \cdot c \cdot n^\varepsilon$.

1-neigbhorhood cover with degree 2

Algorithm Idea using Neighborhood Covers

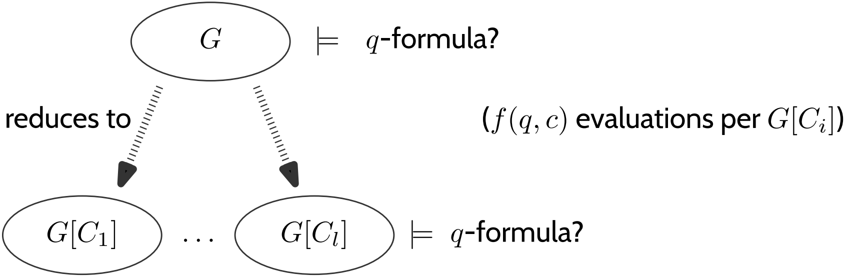
We just saw that recursing into the r-neighborhood of every vertex is too expensive. Instead, we will “aggregate” some computations by recursing only into r-neighborhood covers.

Algorithm Idea using Neighborhood Covers

We just saw that recursing into the r-neighborhood of every vertex is too expensive. Instead, we will "aggregate" some computations by recursing only into r-neighborhood covers.

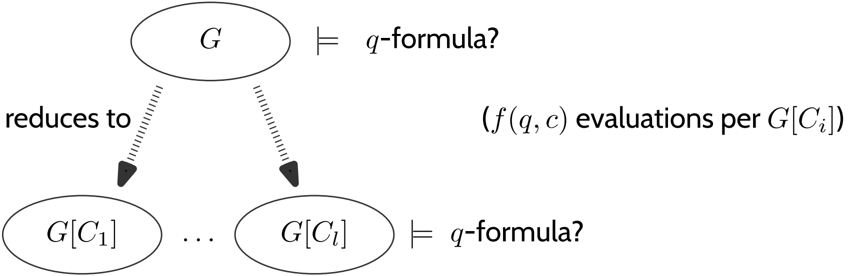
We aim for the following recursive calls:

Algorithm Idea using Neighborhood Covers



where $C_1, \dots, C_l$ is an r-neighborhood cover of G with $r := 12q \cdot (2^q + 1)^2$

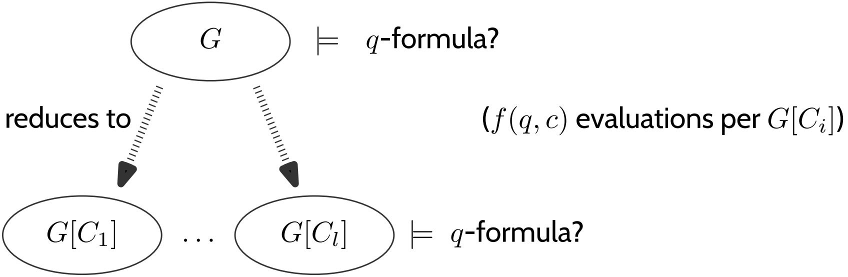
Algorithm Idea using Neighborhood Covers



where $C_1, \dots, C_l$ is an r-neighborhood cover of G with $r := 12q \cdot (2^q + 1)^2$

Each $G[C_i]$ has radius at most $4r$, so these still are "localization moves" in the $4r$-Flipper game, bounding the recursion depth.

Algorithm Idea using Neighborhood Covers

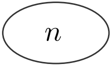


where $C_1, \dots, C_l$ is an r-neighborhood cover of G with $r := 12q \cdot (2^q + 1)^2$

Each $G[C_i]$ has radius at most $4r$, so these still are “localization moves” in the $4r$-Flipper game, bounding the recursion depth.

To get an idea of the run time of such a recursion, let us count the summed number of vertices per level.

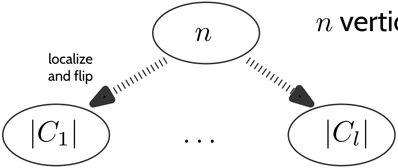
Small Recursion Trees using Neighborhood Covers



n vertices

Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers

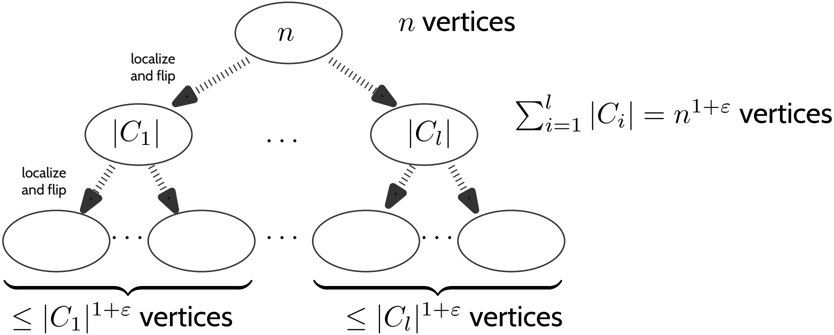


n vertices

$\sum_{i=1}^{l} |C_i| = n^{1+\varepsilon}$ vertices

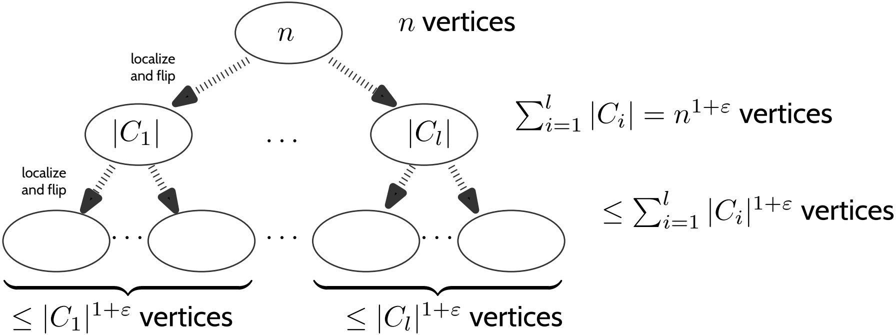
Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers



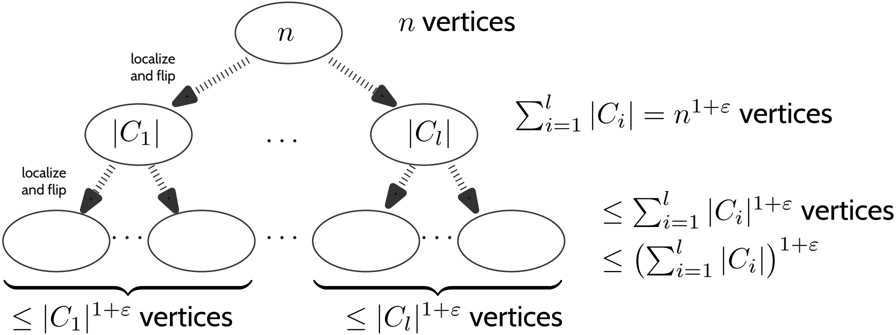
Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers



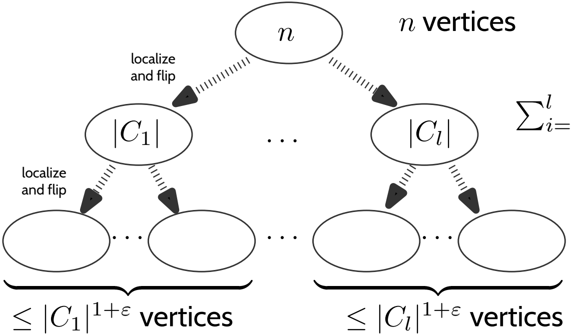
Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers



Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers

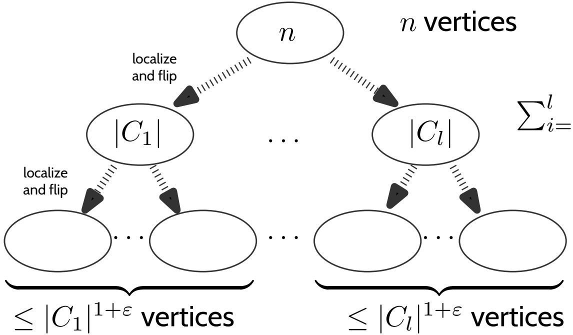


$\sum_{i=1}^{l} |C_i| = n^{1+\varepsilon}$ vertices

$$\leq \sum_{i=1}^{l} |C_i|^{1+\varepsilon} \text{ vertices}$$
$$\leq \left(\sum_{i=1}^{l} |C_i|\right)^{1+\varepsilon}$$
$$\leq \left(n^{1+\varepsilon}\right)^{1+\varepsilon}$$

Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.

Small Recursion Trees using Neighborhood Covers



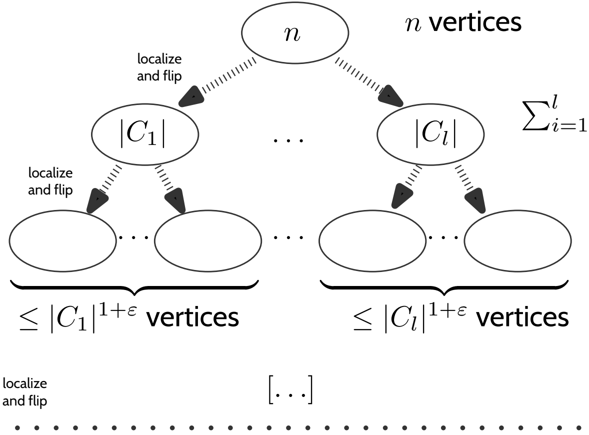
n vertices

$\sum_{i=1}^{l} |C_i| = n^{1+\varepsilon}$ vertices

$\le \sum_{i=1}^{l} |C_i|^{1+\varepsilon}$ vertices
$\le \left(\sum_{i=1}^{l} |C_i|\right)^{1+\varepsilon}$
$\le \left(n^{1+\varepsilon}\right)^{1+\varepsilon}$
$\le n^{(1+\varepsilon)^2}$ vertices

Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.
The vertices in level i sum up to $n^{(1+\varepsilon)^i}$.

Small Recursion Trees using Neighborhood Covers



n vertices

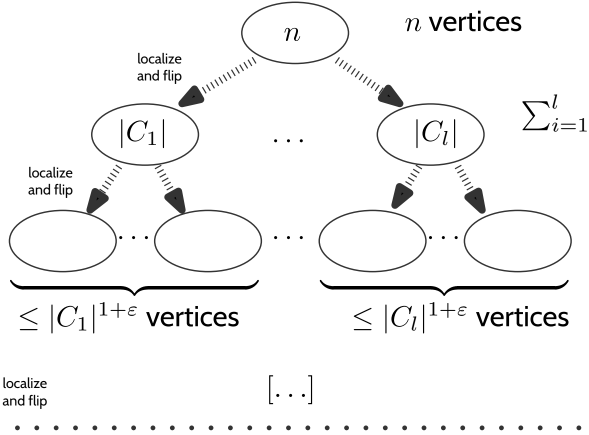
$\sum_{i=1}^{l} |C_i| = n^{1+\varepsilon}$ vertices

$\leq \sum_{i=1}^{l} |C_i|^{1+\varepsilon}$ vertices
$\leq \left(\sum_{i=1}^{l} |C_i|\right)^{1+\varepsilon}$
$\leq \left(n^{1+\varepsilon}\right)^{1+\varepsilon}$
$\leq n^{(1+\varepsilon)^2}$ vertices

$\leq n^{(1+\varepsilon)^d}$ vertices

Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.
The vertices in level i sum up to $n^{(1+\varepsilon)^i}$.

Small Recursion Trees using Neighborhood Covers



n vertices

$\sum_{i=1}^{l} |C_i| = n^{1+\varepsilon}$ vertices

$\leq \sum_{i=1}^{l} |C_i|^{1+\varepsilon}$ vertices
$\leq \left(\sum_{i=1}^{l} |C_i|\right)^{1+\varepsilon}$
$\leq \left(n^{1+\varepsilon}\right)^{1+\varepsilon}$
$\leq n^{(1+\varepsilon)^2}$ vertices

$\leq n^{(1+\varepsilon)^d}$ vertices

Assume we recurse into r-neighborhood covers with $\sum_{i=1}^{l} |C_l| = n^{1+\varepsilon}$.
The vertices in level i sum up to $n^{(1+\varepsilon)^i}$.
Set $\varepsilon > 0$ for example such that $n^{(1+\varepsilon)^d} \leq n^{1.0001}$.

Updating the Localization Routine

We need to update our localization routine to be faster.

Updating the Localization Routine

We need to update our localization routine to be faster.

We previously evaluated a formula by

- recursing into the 2^q-neighborhoods of every vertex,
- and picking a small representative set of vertices.

Updating the Localization Routine

We need to update our localization routine to be faster.

We previously evaluated a formula by

- recursing into the 2^q-neighborhoods of every vertex,
- and picking a small representative set of vertices.

Instead, we

- recurse into the neighborhood covers only,
- and pick a small representative set of neighborhood covers.

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \ldots, C_l$ the formula $\exists x\ \varphi(x)$ of quantifier rank q.

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \dots, C_l$ the formula $\exists x\, \varphi(x)$ of quantifier rank q. Let V_i be all vertices v with $N_{2^q}(v) \subseteq C_i$.

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \ldots, C_l$ the formula $\exists x\, \varphi(x)$ of quantifier rank q. Let V_i be all vertices v with $N_{2^q}(v) \subseteq C_i$. Since $V_1 \cup \cdots \cup V_l = V(G)$,

$$G \models \exists x\, \varphi(x)$$

$$\iff$$

$$G \models \exists x \in V_1\, \varphi(x) \vee \; G \models \exists x \in V_2\, \varphi(x) \; \vee \ldots \vee G \models \exists x \in V_l\, \varphi(x).$$

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \dots, C_l$ the formula $\exists x\, \varphi(x)$ of quantifier rank q. Let V_i be all vertices v with $N_{2^q}(v) \subseteq C_i$. Since $V_1 \cup \cdots \cup V_l = V(G)$,

$$G \models \exists x\, \varphi(x)$$

$$\iff$$

$$G \models \exists x \in V_1\, \varphi(x) \lor \ G \models \exists x \in V_2\, \varphi(x) \ \lor \dots \lor G \models \exists x \in V_l\, \varphi(x).$$

Assume we know $G \models \exists x \in V_1\, \varphi(x) \iff G \models \exists x \in V_2\, \varphi(x)$.

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \ldots, C_l$ the formula $\exists x\, \varphi(x)$ of quantifier rank q. Let V_i be all vertices v with $N_{2^q}(v) \subseteq C_i$. Since $V_1 \cup \cdots \cup V_l = V(G)$,

$$G \models \exists x\, \varphi(x)$$

$$\iff$$

$$G \models \exists x \in V_1\, \varphi(x) \lor \; G \models \exists x \in V_2\, \varphi(x) \;\lor \ldots \lor G \models \exists x \in V_l\, \varphi(x).$$

Assume we know $G \models \exists x \in V_1\, \varphi(x) \iff G \models \exists x \in V_2\, \varphi(x)$.
We only need to keep one "representative".

Algorithm Idea using Neighborhood Covers

Assume we want to evaluate on a graph G with a 2^q-neighborhood cover $C_1, \ldots, C_l$ the formula $\exists x\, \varphi(x)$ of quantifier rank q. Let V_i be all vertices v with $N_{2^q}(v) \subseteq C_i$. Since $V_1 \cup \cdots \cup V_l = V(G)$,

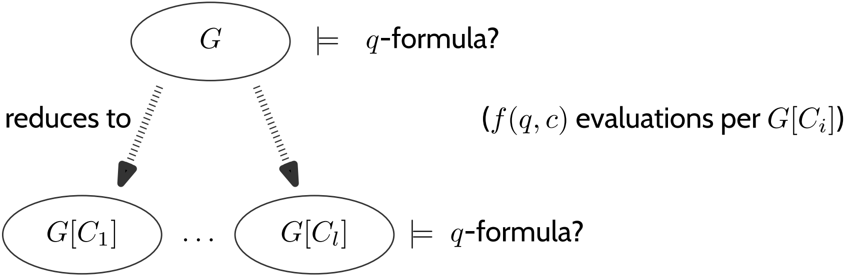
$$G \models \exists x\, \varphi(x)$$

$$\Longleftrightarrow$$

$$G \models \exists x \in V_1\, \varphi(x) \vee \sout{G \models \exists x \in V_2\, \varphi(x)} \vee \ldots \vee G \models \exists x \in V_l\, \varphi(x).$$

Assume we know $G \models \exists x \in V_1\, \varphi(x) \iff G \models \exists x \in V_2\, \varphi(x)$.
We only need to keep one "representative".

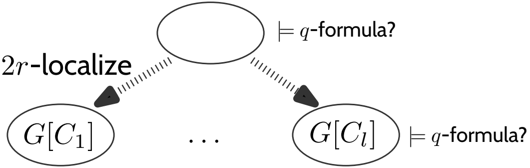
Algorithm Idea using Neighborhood Covers



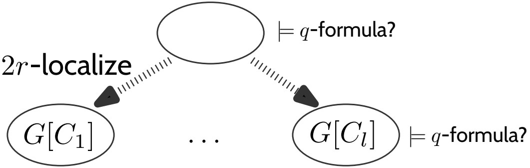
where $C_1, \dots, C_l$ is an r-neighborhood cover of G with $r := 12q \cdot (2^q + 1)^2$

Updated Branching

round 1



Updated Branching

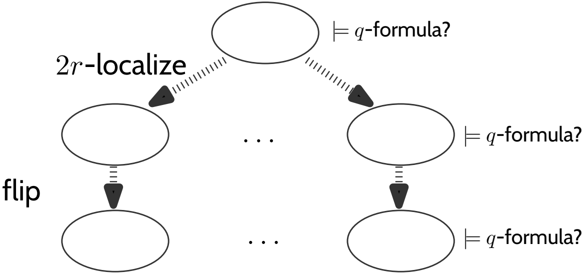


round 1

evaluations per $G[C_i]$:
$f(q, c)$

vertex sum:
$f(q, c) \cdot n^{1+\varepsilon}$

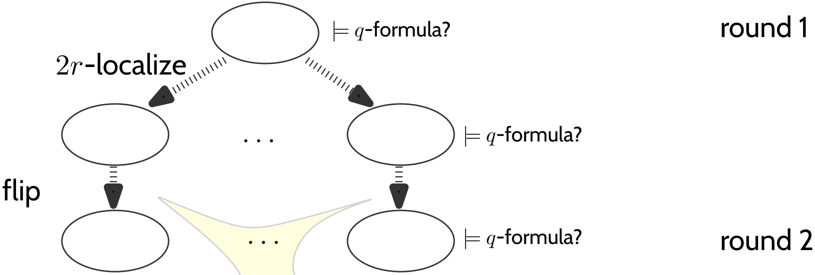
Updated Branching



round 1

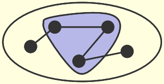
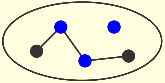
round 2

Updated Branching

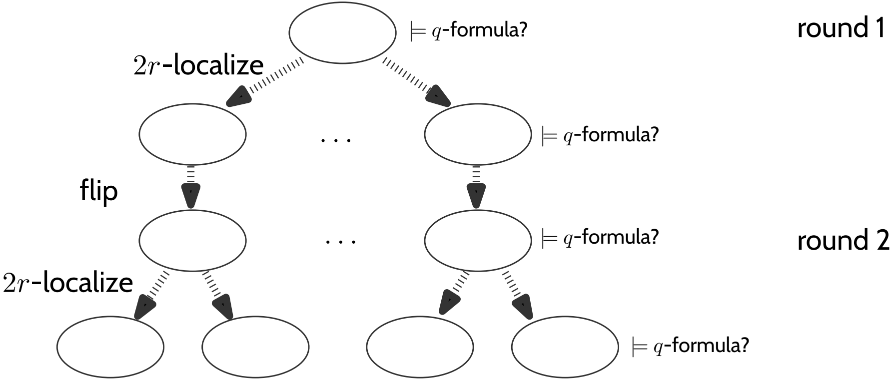


Give flip-set F a new color.

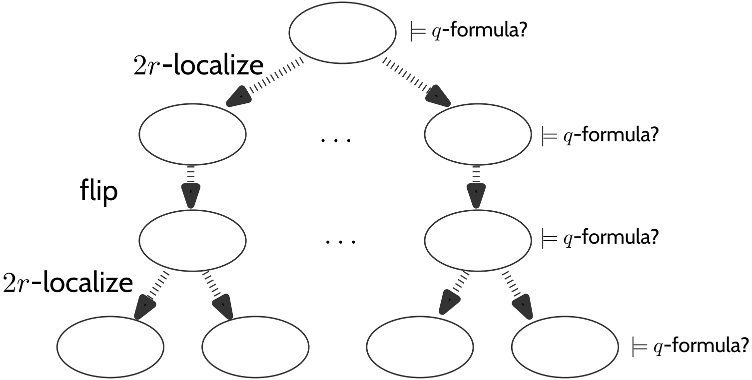
Update q-formulas by replacing each edge relation:

 $\models E(x, y) \iff$  $\models E(x,y) \oplus (\text{blue}(x) \wedge \text{blue}(y))$

Updated Branching



Updated Branching

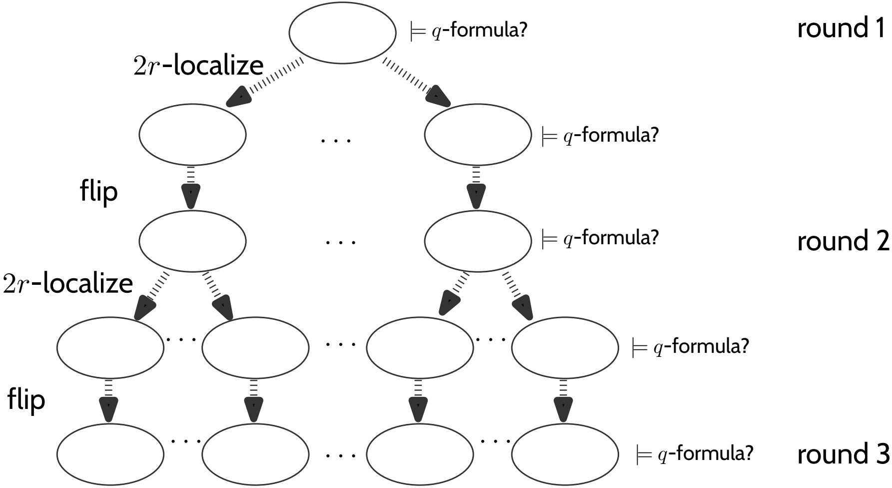


round 1

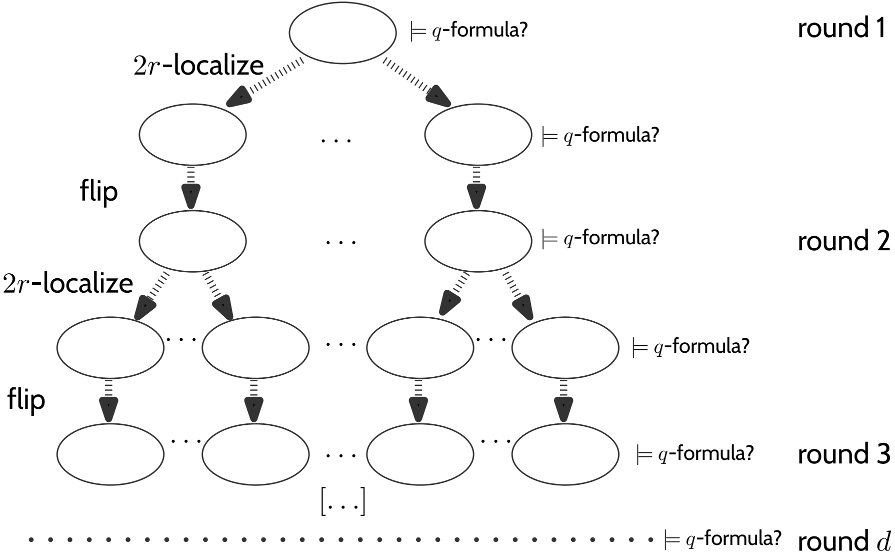
round 2

vertex sum:
$f'(q,c) \cdot n^{(1+\varepsilon)^2}$

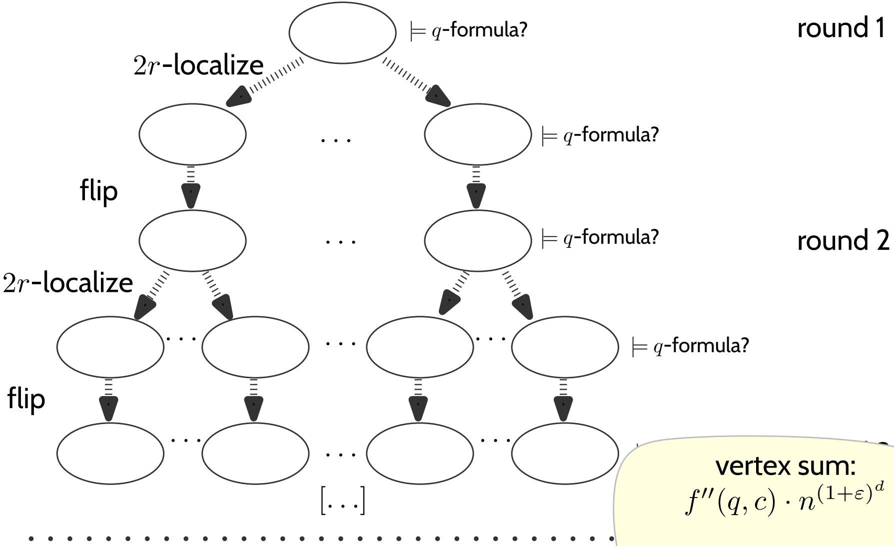
Updated Branching



Updated Branching



Updated Branching



vertex sum:
$f''(q,c) \cdot n^{(1+\varepsilon)^d}$

pick ε such that this is
$f''(q,c) \cdot n^{1.001}$

Summary

This completes the proof (sketch) of the theorem.

D, Mählmann, Siebertz, 2023

D, Eleftheriadis, Mählmann, McCarty, Pilipczuk, Toruńczyk, 2024

Let $\mathcal{C}$ be a monadically stable graph class. There exists a function f such that for every FO formula φ and graph $G \in \mathcal{C}$ one can decide whether $G \models \varphi$ in time $f(|\varphi|)n^6$.

Summary

In summary, we used three ingredients.

Summary

In summary, we used three ingredients.

- The local types help use localize while preserving the quantifier-rank (and thus the radius of the Flipper game).

Summary

In summary, we used three ingredients.

- The local types help use localize while preserving the quantifier-rank (and thus the radius of the Flipper game).
- The Flipper game bounds the depth of the recursion tree.

Summary

In summary, we used three ingredients.

- The local types help use localize while preserving the quantifier-rank (and thus the radius of the Flipper game).
- The Flipper game bounds the depth of the recursion tree.
- The neighborhood covers bound the size of the recursion tree.

Prove that the following are functionally equivalent.

- A constant number of flips.
- A constant number of pairwise flips.
- A flip based on a partition into a constant number of parts.

Prove that every nowhere dense graph class is monadically stable.

Prove that every nowhere dense graph class is monadically stable.

Prove: If Splitter can win the radius-r Splitter game in d rounds, then Flipper can win the radius-r Flipper game in $3d$ rounds.

Prove directly: First-order model-checking is fpt on the class of $\log(n)$ subdivisions of graphs of size n.

Prove that Connector has a winning strategy for the radius-2 Flipper game to play for $\Theta(\log(n))$ rounds on ladders of length n.

- Show that every class of bounded degree has neighborhood covers with bounded overlap.

- Show that every class of bounded degree has neighborhood covers with bounded overlap.
- Show that every tree has radius-1 neighborhood covers with overlap at most three.

- Show that every class of bounded degree has neighborhood covers with bounded overlap.

- Show that every tree has radius-1 neighborhood covers with overlap at most three.

- A graph class has *subpolynomial degree* if the degree of every n-vertex graph is bounded by $f(\epsilon)n^{\epsilon}$ for every $\epsilon > 0$. Prove that such a class has neighborhood covers with subpolynomial overlap.

Normalize first-order formulas such that the number of formulas with quantifier rank q and c colors is bounded by some function $f(q, c)$.