Problems parameterized by treewidth tractable in single exponential time: a logical approach

Michał Pilipczuk

Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

23rd August 2011,
MFCS 2011: Mathematical Foundations of Computer Science

Motivation

- Problems expressible by a MS_2 formula φ admit a $f(|\varphi|, t)|G|^{O(1)}$ algorithm over tree decomposition. [Courcelle]

Motivation

- Problems expressible by a MS_2 formula φ admit a $f(|\varphi|, t)|G|^{O(1)}$ algorithm over tree decomposition. [Courcelle]
- Even for relatively simple fixed formulas φ, the function f can explode when treewidth varies.

Motivation

- Problems expressible by a MS_2 formula φ admit a $f(|\varphi|, t)|G|^{O(1)}$ algorithm over tree decomposition. [Courcelle]
- Even for relatively simple fixed formulas φ, the function f can explode when treewidth varies.
- In many applications running time of form $2^{O(t)}|G|^{O(1)}$ is essential.

Motivation

- Problems expressible by a MS_2 formula φ admit a $f(|\varphi|, t)|G|^{O(1)}$ algorithm over tree decomposition. [Courcelle]
- Even for relatively simple fixed formulas φ, the function f can explode when treewidth varies.
- In many applications running time of form $2^{O(t)}|G|^{O(1)}$ is essential.
  - When using bidimensionality technique, even $2^{O(t^2)}|G|^{O(1)}$ gives us no gain.

Motivation

- Problems expressible by a MS_2 formula φ admit a $f(|\varphi|, t)|G|^{O(1)}$ algorithm over tree decomposition. [Courcelle]
- Even for relatively simple fixed formulas φ, the function f can explode when treewidth varies.
- In many applications running time of form $2^{O(t)}|G|^{O(1)}$ is essential.
  - When using bidimensionality technique, even $2^{O(t^2)}|G|^{O(1)}$ gives us no gain.
- For many implicit problems there are such algorithms: Independent Set (2^t), Dominating Set (3^t), Odd Cycle Transversal (3^t)...

Motivation

- Lokshtanov et al. [SODA 2011]: the currently best bases of exponents are optimal under certain complexity assumptions.

Motivation

- Lokshtanov et al. [SODA 2011]: the currently best bases of exponents are optimal under certain complexity assumptions.
  - For instance, no $(2-\epsilon)^t|G|^{O(1)}$ for Independent Set...

Motivation

- Lokshtanov et al. [SODA 2011]: the currently best bases of exponents are optimal under certain complexity assumptions.
  - For instance, no $(2-\epsilon)^t|G|^{O(1)}$ for Independent Set...
- There is a natural class of 'connectivity' problems that admit a $2^{O(t \log t)}|G|^{O(1)}$.

Motivation

- Lokshtanov et al. [SODA 2011]: the currently best bases of exponents are optimal under certain complexity assumptions.
  - For instance, no $(2-\epsilon)^t|G|^{O(1)}$ for Independent Set...
- There is a natural class of 'connectivity' problems that admit a $2^{O(t \log t)}|G|^{O(1)}$.
  - Connected Vertex Cover, Hamiltonian Cycle, Disjoint Paths...

Motivation

- Lokshtanov et al. [SODA 2011]: the currently best bases of exponents are optimal under certain complexity assumptions.
  - For instance, no $(2-\epsilon)^t|G|^{O(1)}$ for Independent Set...
- There is a natural class of 'connectivity' problems that admit a $2^{O(t \log t)}|G|^{O(1)}$.
  - Connected Vertex Cover, Hamiltonian Cycle, Disjoint Paths...
- It seemed that this could not be beaten. For Disjoint Paths the lower bound has been already established.

Motivation

- **Surprise:** This is not true!

Motivation

- **Surprise:** This is not true!
- Cygan et al. [FOCS 2011] developed a technique, called CUT&COUNT, that yields $2^{O(t)}|G|^{O(1)}$ Monte-Carlo algorithms for such problems.

Motivation

- **Surprise:** This is not true!
- Cygan et al. [FOCS 2011] developed a technique, called CUT&COUNT, that yields $2^{O(t)}|G|^{O(1)}$ Monte-Carlo algorithms for such problems.
- A natural question arises:

Motivation

- **Surprise:** This is not true!
- Cygan et al. [FOCS 2011] developed a technique, called CUT&COUNT, that yields $2^{O(t)}|G|^{O(1)}$ Monte-Carlo algorithms for such problems.
- A natural question arises:

  What properties of a problem make it admit a single exponential algorithm in terms of treewidth?

Motivation

- **Surprise:** This is not true!
- Cygan et al. [FOCS 2011] developed a technique, called Cut&Count, that yields $2^{O(t)}|G|^{O(1)}$ Monte-Carlo algorithms for such problems.
- A natural question arises:

  What properties of a problem make it admit a single exponential algorithm in terms of treewidth?

- **This paper:** we try to extract the common points of already known techniques, in order to give some intuition about the answer.

Results

- The goal is to have a theorem like this:

Results

- The goal is to have a theorem like this:

Meta-theorem

If problem P is expressible in logic L, then it admits a (randomized) algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

Results

- The goal is to have a theorem like this:

Meta-theorem

If problem P is expressible in logic L, then it admits a (randomized) algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- Deterministic $2^{O(t)}|G|^{O(1)}$ time: L is ECML.

Results

- The goal is to have a theorem like this:

Meta-theorem

If problem P is expressible in logic L, then it admits a (randomized) algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- Deterministic $2^{O(t)}|G|^{O(1)}$ time: L is ECML.
  - ECML =Existential Counting Modal Logic

Results

- The goal is to have a theorem like this:

Meta-theorem

If problem P is expressible in logic L, then it admits a (randomized) algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- Deterministic $2^{O(t)}|G|^{O(1)}$ time: L is ECML.
  - ECML =Existential Counting Modal Logic
- Randomized $2^{O(t)}|G|^{O(1)}$ time: L is $\mathrm{ECML}{+}\mathrm{C}$.

Results

- The goal is to have a theorem like this:

Meta-theorem

If problem P is expressible in logic L, then it admits a (randomized) algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- Deterministic $2^{O(t)}|G|^{O(1)}$ time: L is ECML.
  - ECML =Existential Counting Modal Logic
- Randomized $2^{O(t)}|G|^{O(1)}$ time: L is $\mathrm{ECML}{+}\mathrm{C}$.
  - $\mathrm{ECML}{+}\mathrm{C}$ =ECML+Connectivity

Example: Dominating Set

Before we introduce ECML, let us have an example of ECML formula.

Dominating Set

Input: graph $G = (V, E)$ and an integer k.
Question: Does there exist a subset X of vertices of cardinality at most k, such that every vertex is in X or has a neighbour in X?

Example: Dominating Set

- The problem consists of two parts:

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}$$

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}(|X| \leqslant k)$$

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}(|X| \leqslant k) \wedge \forall_v G, X, v \models$$

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}(|X| \leqslant k) \wedge \forall_v G, X, v \models X$$

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}(|X| \leqslant k) \wedge \forall_v G, X, v \models X \vee \Diamond X.$$

Example: Dominating Set

- The problem consists of two parts:
  - The outer part: 'Does there exist a subset X of vertices of cardinality at most k,'
  - The inner part: 'such that every vertex is in X or has a neighbour in X?'
- The formula:

$$\exists_{X \subseteq V}(|X| \leqslant k) \wedge \forall_v G, X, v \models X \vee \Diamond X.$$

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.
- The general form of ECML formula:

$$\exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v \left(G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v \right) \models \psi \right]$$

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.
- The general form of ECML formula:

$$\exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v \left(G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v \right) \models \psi \right]$$

- Here:

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.
- The general form of ECML formula:

$$\exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v \left(G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v \right) \models \psi \right]$$

- Here:
  - $\overline{X}$ and $\overline{Y}$ are vectors of quantified sets of vertices and edges, respectively;

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.
- The general form of ECML formula:

$$\exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v \left(G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v \right) \models \psi \right]$$

- Here:
  - $\overline{X}$ and $\overline{Y}$ are vectors of quantified sets of vertices and edges, respectively;
  - ϕ is an arithmetic formula over $\overline{k}$ and cardinalities of $V, E, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}$;

Syntax of ECML

- The formula in general is evaluated in a graph G supplied with a vector of parameters $\overline{k}$, a vector of fixed sets of vertices $\overline{FX}$ and a vector of fixed sets of edges $\overline{FY}$.
- The general form of ECML formula:

$$\exists_{\overline{X}} \exists_{\overline{Y}} \left[\phi \wedge \forall_v \left(G, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}, v \right) \models \psi \right]$$

- Here:
  - $\overline{X}$ and $\overline{Y}$ are vectors of quantified sets of vertices and edges, respectively;
  - ϕ is an arithmetic formula over $\overline{k}$ and cardinalities of $V, E, \overline{FX}, \overline{FY}, \overline{X}, \overline{Y}$;
  - ψ is a formula of **inner logic**, called CML, evaluated in a vertex of a graph.

Syntax of CML

- CML is a modal logic — a formula is evaluated in a vertex of a graph supplied with vectors of vertex sets $\overline{X}$ and edge sets $\overline{Y}$.

Syntax of CML

- CML is a modal logic — a formula is evaluated in a vertex of a graph supplied with vectors of vertex sets $\overline{X}$ and edge sets $\overline{Y}$.
- Defined by grammar:

$$\psi := \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbb{X} \mid \mathbb{Y} \mid \Diamond^S \psi \mid \Box^S \psi$$
$$\mathbb{X} := X_1 \mid X_2 \mid \ldots \mid X_p$$
$$\mathbb{Y} := Y_1 \mid Y_2 \mid \ldots \mid Y_q$$

Syntax of CML

- CML is a modal logic — a formula is evaluated in a vertex of a graph supplied with vectors of vertex sets $\overline{X}$ and edge sets $\overline{Y}$.
- Defined by grammar:

$$\psi := \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbb{X} \mid \mathbb{Y} \mid \Diamond^S \psi \mid \Box^S \psi$$
$$\mathbb{X} := X_1 \mid X_2 \mid \ldots \mid X_p$$
$$\mathbb{Y} := Y_1 \mid Y_2 \mid \ldots \mid Y_q$$

- Boolean operators act as usual,

Syntax of CML

- CML is a modal logic — a formula is evaluated in a vertex of a graph supplied with vectors of vertex sets $\overline{X}$ and edge sets $\overline{Y}$.
- Defined by grammar:

$$\psi := \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbb{X} \mid \mathbb{Y} \mid \Diamond^S \psi \mid \Box^S \psi$$
$$\mathbb{X} := X_1 \mid X_2 \mid \ldots \mid X_p$$
$$\mathbb{Y} := Y_1 \mid Y_2 \mid \ldots \mid Y_q$$

- Boolean operators act as usual,
- $\Diamond^S$ and $\Box^S$ are quantifiers,

Syntax of CML

- CML is a modal logic — a formula is evaluated in a vertex of a graph supplied with vectors of vertex sets $\overline{X}$ and edge sets $\overline{Y}$.
- Defined by grammar:

$$\psi := \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbb{X} \mid \mathbb{Y} \mid \Diamond^S \psi \mid \Box^S \psi$$
$$\mathbb{X} := X_1 \mid X_2 \mid \ldots \mid X_p$$
$$\mathbb{Y} := Y_1 \mid Y_2 \mid \ldots \mid Y_q$$

- Boolean operators act as usual,
- $\Diamond^S$ and $\Box^S$ are quantifiers,
- $\mathbb{X}$ and $\mathbb{Y}$ are operators used to test belonging to certain sets.

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.
- S has to be a semilinear (ultimately periodic) set over $\mathbb{N}$, i.e., such that there exist N, k such that for all $n \geqslant N$ it holds that $n \in S \Leftrightarrow n + k \in S$.

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.
- S has to be a semilinear (ultimately periodic) set over $\mathbb{N}$, i.e., such that there exist N, k such that for all $n \geqslant N$ it holds that $n \in S \Leftrightarrow n + k \in S$.
- $\Box^S$ is a syntactic sugar: $\Box^S \psi$ is equivalent to $\neg \Diamond^S \neg \psi$.

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.
- S has to be a semilinear (ultimately periodic) set over $\mathbb{N}$, i.e., such that there exist N, k such that for all $n \geqslant N$ it holds that $n \in S \Leftrightarrow n + k \in S$.
- $\Box^S$ is a syntactic sugar: $\Box^S \psi$ is equivalent to $\neg \Diamond^S \neg \psi$.
- We use shortened forms:

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.
- S has to be a semilinear (ultimately periodic) set over $\mathbb{N}$, i.e., such that there exist N, k such that for all $n \geqslant N$ it holds that $n \in S \Leftrightarrow n + k \in S$.
- $\Box^S$ is a syntactic sugar: $\Box^S \psi$ is equivalent to $\neg \Diamond^S \neg \psi$.
- We use shortened forms:
  - $\Diamond \psi \equiv \Diamond^{\mathbb{N}^+} \psi$: 'there is a neighbour satisfying ψ';

Quantifiers

- $\Diamond^S \psi$ for $S \subseteq \mathbb{N}$ means that the number of neighbours of the current vertex, in which ψ is satisfied, belongs to S.
- S has to be a semilinear (ultimately periodic) set over $\mathbb{N}$, i.e., such that there exist N, k such that for all $n \geqslant N$ it holds that $n \in S \Leftrightarrow n + k \in S$.
- $\Box^S$ is a syntactic sugar: $\Box^S \psi$ is equivalent to $\neg \Diamond^S \neg \psi$.
- We use shortened forms:
  - $\Diamond \psi \equiv \Diamond^{\mathbb{N}^+} \psi$: 'there is a neighbour satisfying ψ';
  - $\Box \psi \equiv \Box^{\mathbb{N}^+} \psi$: 'all the neighbours satisfy ψ'.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.
- $\mathbb{Y}$ operators are a bit confusing.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.
- $\mathbb{Y}$ operators are a bit confusing.
  - The process of evaluation of a formula can be viewed as a walk on a graph.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.
- $\mathbb{Y}$ operators are a bit confusing.
  - The process of evaluation of a formula can be viewed as a walk on a graph.
  - Operator Y_j is true in a vertex iff the edge used to access this vertex belongs to Y_j.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.
- $\mathbb{Y}$ operators are a bit confusing.
  - The process of evaluation of a formula can be viewed as a walk on a graph.
  - Operator Y_j is true in a vertex iff the edge used to access this vertex belongs to Y_j.
  - Therefore, we allow usage of $\mathbb{Y}$ **only under some quantification**.

$\mathbb{X}$ and $\mathbb{Y}$ operators

- $\mathbb{X}$ operators are simple: X_i is true in a vertex v iff $v \in X_i$.
- $\mathbb{Y}$ operators are a bit confusing.
  - The process of evaluation of a formula can be viewed as a walk on a graph.
  - Operator Y_j is true in a vertex iff the edge used to access this vertex belongs to Y_j.
  - Therefore, we allow usage of $\mathbb{Y}$ **only under some quantification**.
- Extending to directed graphs: adding two more $\mathbb{Y}$ operators.

Main result #1

ECML model checking

If problem P is expressible in ECML, then it admits an algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t. Moreover, the algorithm can also count the number of vectors $\overline{X}$, $\overline{Y}$ satisfying the formula.

- The proof:

Main result #1

ECML model checking

If problem P is expressible in ECML, then it admits an algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t. Moreover, the algorithm can also count the number of vectors $\overline{X}$, $\overline{Y}$ satisfying the formula.

- The proof:
  - Dynamic programming on nice tree decomposition

Main result #1

ECML model checking

If problem P is expressible in ECML, then it admits an algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t. Moreover, the algorithm can also count the number of vectors $\overline{X}$, $\overline{Y}$ satisfying the formula.

- The proof:
  - Dynamic programming on nice tree decomposition
  - *Prediction* technique: in a vertex we store the set of subformulas of ψ that are **predicted** to be true in the whole graph.

Main result #1

ECML model checking

If problem P is expressible in ECML, then it admits an algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t. Moreover, the algorithm can also count the number of vectors $\overline{X}$, $\overline{Y}$ satisfying the formula.

- The proof:
  - Dynamic programming on nice tree decomposition
  - *Prediction* technique: in a vertex we store the set of subformulas of ψ that are **predicted** to be true in the whole graph.
  - Role of semilinearity of S: we represent the number of neighbours of a vertex having a certain CML formula satisfied as an element of a finite monoid.

Examples

- Independent Set:

Examples

- Independent Set:

$$\exists_{X \subseteq V}(|X| \geqslant k) \wedge \forall_v G, X, v \models (X \Rightarrow \Box\neg X).$$

Examples

- r-Dominating Set:

Examples

- r-Dominating Set:

$$\exists_{X \subseteq V}(|X| \leqslant k) \wedge \forall_v G, X, v \models \underbrace{X \vee \Diamond(X \vee \ldots \Diamond(X \vee \Diamond X)\ldots)}_{r \text{ quantifications}}.$$

Examples

- #Perfect Matchings:

Examples

- #Perfect Matchings:

$$\exists_{M \subseteq E} \forall_v \, G, M, v \models \Diamond^{\{1\}} M.$$

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.
  - $|\mathrm{cc}(X_i)|$ is the number of connected components of $G[X_i]$;

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.
  - $|\mathrm{cc}(X_i)|$ is the number of connected components of $G[X_i]$;
  - $|\mathrm{cc}(Y_j)|$ is the number of connected components of $(V(Y_j), Y_j)$, where $V(Y_j)$ is the set of endpoints of Y_j (we throw out the isolated vertices).

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.
  - $|\mathrm{cc}(X_i)|$ is the number of connected components of $G[X_i]$;
  - $|\mathrm{cc}(Y_j)|$ is the number of connected components of $(V(Y_j), Y_j)$, where $V(Y_j)$ is the set of endpoints of Y_j (we throw out the isolated vertices).
- We restrict ourselves only to formulas ϕ that are monotonous with respect to the numbers of connected components.

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.
  - $|\mathrm{cc}(X_i)|$ is the number of connected components of $G[X_i]$;
  - $|\mathrm{cc}(Y_j)|$ is the number of connected components of $(V(Y_j), Y_j)$, where $V(Y_j)$ is the set of endpoints of Y_j (we throw out the isolated vertices).
- We restrict ourselves only to formulas ϕ that are monotonous with respect to the numbers of connected components.
  - If ϕ is true, then for a smaller number of components it is true as well.

ECML+C

- In $\mathrm{ECML{+}C}$ we allow the arithmetic formula ϕ to depend also on the numbers of connected components of subgraphs induced by sets X_i, Y_j.
  - $|\mathsf{cc}(X_i)|$ is the number of connected components of $G[X_i]$;
  - $|\mathsf{cc}(Y_j)|$ is the number of connected components of $(V(Y_j), Y_j)$, where $V(Y_j)$ is the set of endpoints of Y_j (we throw out the isolated vertices).
- We restrict ourselves only to formulas ϕ that are monotonous with respect to the numbers of connected components.
  - If ϕ is true, then for a smaller number of components it is true as well.
- This cannot be avoided: a number of lower bounds under ETH for maximization problems in the Cut&Count paper.

Main result #2

$\mathrm{ECML{+}C}$ model checking

If problem P is expressible in $\mathrm{ECML{+}C}$, then it admits a Monte-Carlo algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

Main result #2

$\mathrm{ECML}+\mathrm{C}$ model checking

If problem P is expressible in $\mathrm{ECML}+\mathrm{C}$, then it admits a Monte-Carlo algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- The proof: transformation of the algorithm for ECML to the language of $\textsc{Cut\&Count}$.

Main result #2

ECML+C model checking

If problem P is expressible in ECML+C, then it admits a Monte-Carlo algorithm running in $2^{O(t)}|G|^{O(1)}$ time, working on the given tree decomposition of width t.

- The proof: transformation of the algorithm for ECML to the language of Cut&Count.
- The theorem subsumes all the tractability results of the Cut&Count paper, but with much worse bases of exponents.

Examples

- Steiner Tree (T is the fixed set of terminals):

Examples

- Steiner Tree (T is the fixed set of terminals):

$$\exists_{X \subseteq V}(|\mathsf{cc}(X)| \leqslant 1 \wedge |X| \leqslant k + |T|) \wedge \forall_v G, T, X, v \models (T \Rightarrow X)$$

Examples

- Hamiltonian Cycle:

Examples

- Hamiltonian Cycle:

$$\exists_{Y \subseteq E}(|\mathsf{cc}(Y)| \leqslant 1) \wedge \forall_v G, Y, v \models \Diamond^{\{2\}} Y$$

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?
  - Yes, C_5-Vertex Deletion does not admit a $2^{o(t^2)}|G|^{O(1)}$ solution unless ETH fails.

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?
  - Yes, C_5-Vertex Deletion does not admit a $2^{o(t^2)}|G|^{O(1)}$ solution unless ETH fails.
- There are problems not expressible in ECML, admitting a single exponential algorithm.

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?
  - Yes, C_5-Vertex Deletion does not admit a $2^{o(t^2)}|G|^{O(1)}$ solution unless ETH fails.
- There are problems not expressible in ECML, admitting a single exponential algorithm.
  - K_l-Vertex Deletion,

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?
  - Yes, C_5-Vertex Deletion does not admit a $2^{o(t^2)}|G|^{O(1)}$ solution unless ETH fails.
- There are problems not expressible in ECML, admitting a single exponential algorithm.
  - K_l-Vertex Deletion,
  - r-Packing.

Conclusions and open problems

- Is the modality/navigationality/acyclicity necessary?
  - Yes, C_5-Vertex Deletion does not admit a $2^{o(t^2)}|G|^{O(1)}$ solution unless ETH fails.
- There are problems not expressible in ECML, admitting a single exponential algorithm.
  - K_l-Vertex Deletion,
  - r-Packing.
- Can we extend ECML in an **elegant** way in order to capture also these problems?

Thank you for your attention

Questions?