Logic and property testing on graphs of bounded degree

Isolde Adler

Joint work with Polly Fahey, Frederik Harwath, Noleen Köhler, Pan Peng

Panhellenic Logic Symposium
2.7.2024

CV overview

- Student & PhD in Mathematics: Göttingen, **Thessaloniki**, Freiburg
- Postdoc & Substitute Prof., *Logic in CS*: HU Berlin, Bergen
- Juniorprofessor *Theory of CS*: Goethe University Frankfurt
- Lecturer, Associate Prof. *Algorithms & Complexity*: Leeds
- Since 2022: Chair of *Algorithms and Complexity Theory*: Otto-Friedrich University Bamberg

Algorithms & Complexity @ Bamberg (Bavaria, Germany)

Contents

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark
All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex := number of its neighbours.
- Degree of a graph G := max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark

All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark
All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark

All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark

All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark

All techniques and results can be generalised to relational structures.

Graphs

- We consider finite, simple, undirected graphs $G = (V(G), E(G))$.
- Degree of a vertex $:=$ number of its neighbours.
- Degree of a graph $G :=$ max degree of its vertices.
- Class $\mathcal{C}$ of graphs has bounded degree, if there is a constant $d \in \mathbb{N}$ such that all graphs in $\mathcal{C}$ have degree $\leq d$.
- We use n to denote the number of vertices of G.
- All graph classes are closed under isomorphism, and a graph class is sometimes called a property.

Remark

All techniques and results can be generalised to relational structures.

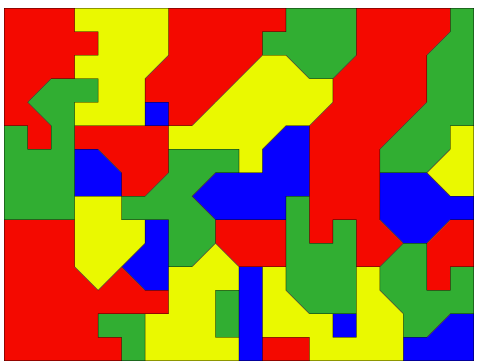
Hard problems on graphs

Many classical algorithmic problems are NP-hard in general.
However, they need to be solved in practice. [1]

Example

COL

Input: Graph G, $k \in \mathbb{N}$
Question: Does G have a k-colouring?



[1] E. g. in register allocation in compilers: Project by former PhD student P. Krause, Compiler Construction 2013, Discrete Applied Math. 2014

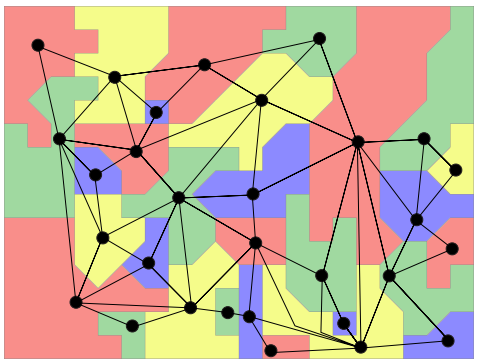
Hard problems on graphs

Many classical algorithmic problems are NP-hard in general.
However, they need to be solved in practice. [1]

Example

COL

Input: Graph G, $k \in \mathbb{N}$
Question: Does G have a k-colouring?



[1] E. g. in register allocation in compilers: Project by former PhD student P. Krause, Compiler Construction 2013, Discrete Applied Math. 2014

Complexity of the problems

Theorem (Karp 1972)

COL *is* NP*-complete.*

Observation

On trees, COL *and many other problems can be solved in linear time.*

Goal

Find large classes of input graphs, on which many problems can be solved efficiently!

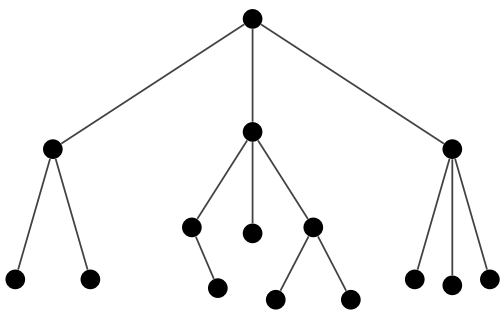
Complexity of the problems

Theorem (Karp 1972)

COL *is* NP*-complete.*

Observation

On trees, COL *and many other problems can be solved in linear time.*



Goal

Find large classes of input graphs, on which many problems can be solved efficiently!

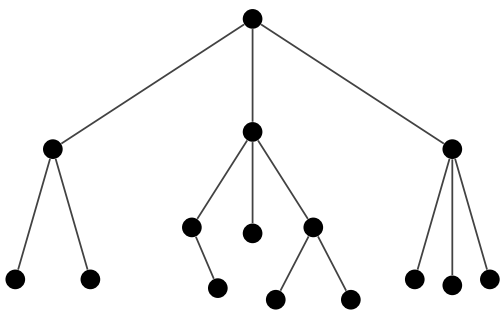
Complexity of the problems

Theorem (Karp 1972)

COL *is* NP*-complete.*

Observation

On trees, COL *and many other problems can be solved in linear time.*



Goal

Find large classes of input graphs, on which many problems can be solved efficiently!

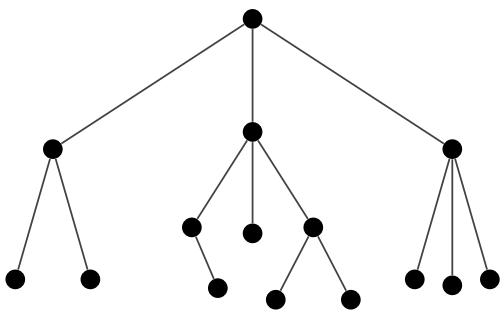
Complexity of the problems

Theorem (Karp 1972)

COL *is* NP*-complete.*

Observation

On trees, COL and many other problems can be solved in linear time.



Goal

Find large classes of input graphs, on which many problems can be solved efficiently!

Treewidth

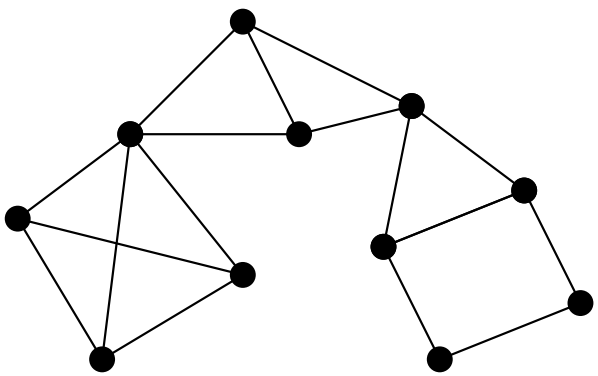
Intuition: Treewidth measures how close a graph G is to being a tree.

G has treewidth $\leq t$, if G can be pieced together from subgraphs of size $\leq t$ in a tree-like fashion:

Treewidth

Intuition: Treewidth measures how close a graph G is to being a tree.

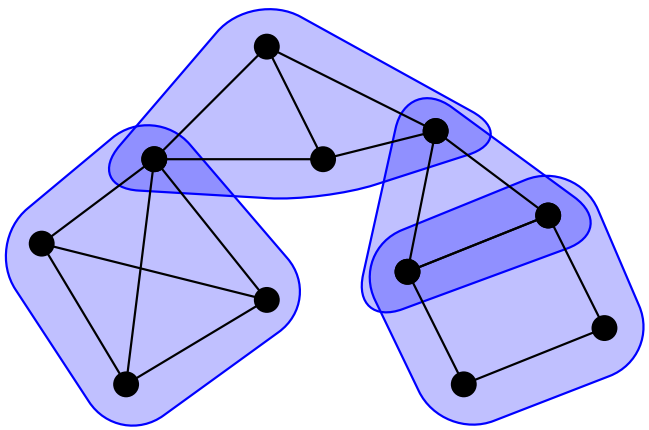
G has treewidth $\leq t$, if G can be pieced together from subgraphs of size $\leq t$ in a tree-like fashion:



Treewidth

Intuition: Treewidth measures how close a graph G is to being a tree.

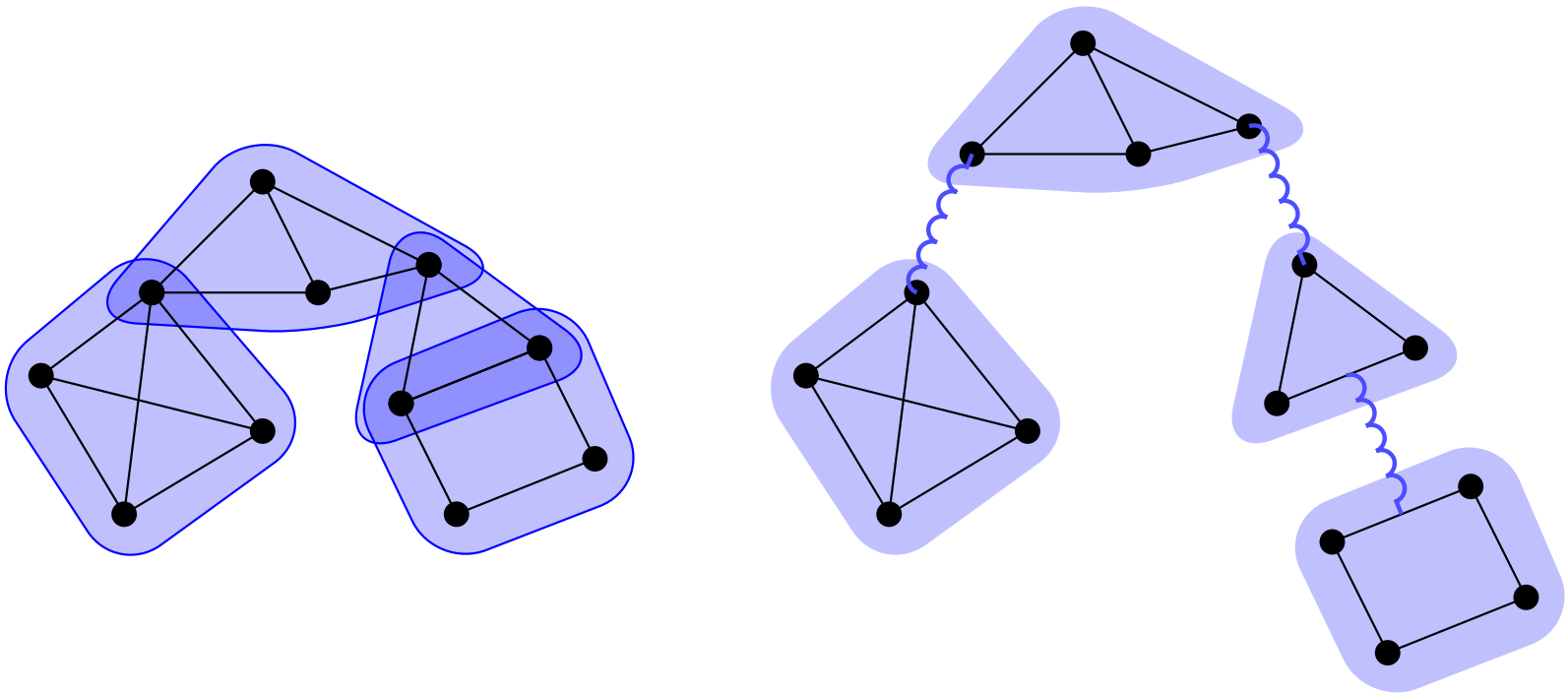
G has treewidth $\leq t$, if G can be pieced together from subgraphs of size $\leq t$ in a tree-like fashion:



Treewidth

Intuition: Treewidth measures how close a graph G is to being a tree.

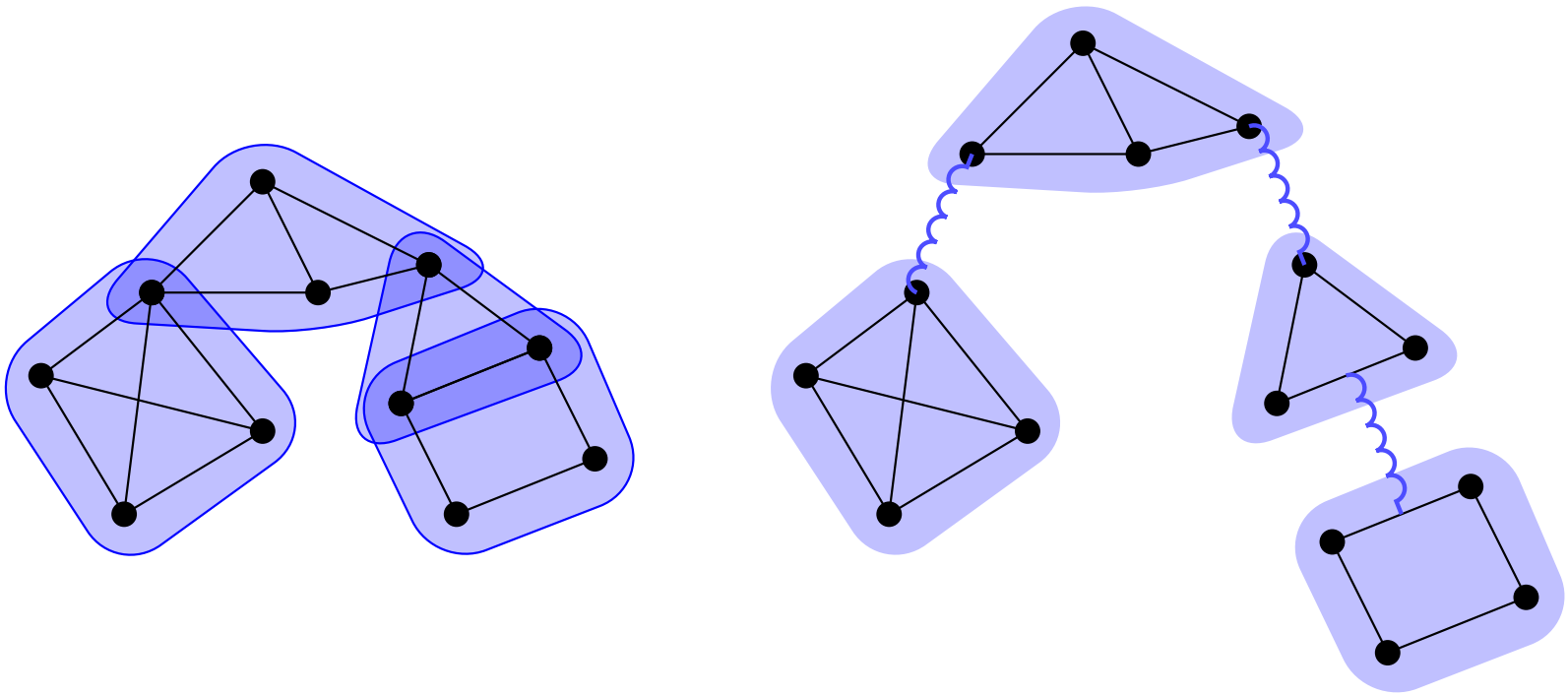
G has treewidth $\leq t$, if G can be pieced together from subgraphs of size $\leq t$ in a tree-like fashion:



Treewidth

Intuition: Treewidth measures how close a graph G is to being a tree.

G has treewidth $\leq t$, if G can be pieced together from subgraphs of size $\leq t$ in a tree-like fashion:



[Larisch, Salfelder, Implementation `https://www.algok.uni-bamberg.de/freetdi.html`, 1. Prize, PACE 2017]

Logics

FO := First-order logic

MSO := Monadic second-order logic =
first-order logic + quantification over subsets of the universe

CMSO = MSO + modulo counting $\quad$ '$\exists_{k \bmod m} x\, \varphi(x)$'

Example

Graph properties expressible in CMSO

- (non-)existence of: k-clique, k-vertex cover, k-independent set, k-dominating set, a fixed subgraph H
- planarity, bounded genus, excluded minor
- connectivity, transitive closure,
- colorability, hamiltonicity,
- even number of vertices, perfect, even-hole-freeness

On relational databases: relational core of SQL

Logics

FO := First-order logic

MSO := Monadic second-order logic =
first-order logic + quantification over subsets of the universe

CMSO = MSO + modulo counting '$\exists_{k \bmod m} X\, \varphi(X)$'

Example

Graph properties expressible in CMSO

- (non-)existence of: k-clique, k-vertex cover, k-independent set, k-dominating set, a fixed subgraph H
- planarity, bounded genus, excluded minor
- connectivity, transitive closure,
- colorability, hamiltonicity,
- even number of vertices, perfect, even-hole-freeness

On relational databases: relational core of SQL

Logics

FO := First-order logic

MSO := Monadic second-order logic =
first-order logic + quantification over subsets of the universe

CMSO = MSO + modulo counting $\quad$ '$\exists_{k \bmod m} X\, \varphi(X)$'

Example

Graph properties expressible in CMSO

- (non-)existence of: k-clique, k-vertex cover, k-independent set, k-dominating set, a fixed subgraph H
- planarity, bounded genus, excluded minor
- connectivity, transitive closure,
- colorability, hamiltonicity,
- even number of vertices, perfect, even-hole-freeness

On relational databases: relational core of SQL

MSO: example

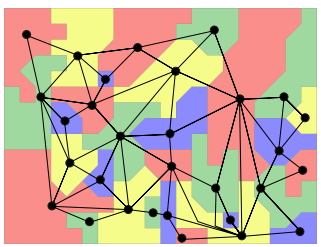
$$\varphi := \exists X_1 \exists X_2 \exists X_3 \forall y \forall z \Big(\bigvee_{1 \leq i \leq 3} (y \in X_i) \wedge \bigwedge_{1 \leq i < j \leq 3} \neg(y \in X_i \wedge y \in X_j) \wedge \bigwedge_{1 \leq i \leq 3} (Eyz \rightarrow \neg(y \in X_i \wedge z \in X_i)) \Big)$$

MSO: example

$$\varphi := \exists X_1 \exists X_2 \exists X_3 \forall y \forall z \Big(\bigvee_{1 \leq i \leq 3} (y \in X_i) \wedge \bigwedge_{1 \leq i < j \leq 3} \neg (y \in X_i \wedge y \in X_j) \wedge \bigwedge_{1 \leq i \leq 3} (Eyz \rightarrow \neg (y \in X_i \wedge z \in X_i)) \Big)$$

MSO: example

$$\varphi := \exists X_1 \exists X_2 \exists X_3 \forall y \forall z \Big(\bigvee_{1 \leq i \leq 3} (y \in X_i) \wedge \bigwedge_{1 \leq i < j \leq 3} \neg (y \in X_i \wedge y \in X_j) \wedge \bigwedge_{1 \leq i \leq 3} (Eyz \rightarrow \neg (y \in X_i \wedge z \in X_i)) \Big)$$



Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Applications:

- Model checking
- Graph algorithms
- Verification
- Database query evaluation
- Constraint satisfaction
- ...

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Applications:

- Model checking
- Graph algorithms
- Verification
- Database query evaluation
- Constraint satisfaction
- ...

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Applications:

- Model checking
- Graph algorithms
- Verification
- Database query evaluation
- Constraint satisfaction
- ...

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Theorem (Seese 1996)
$\mathcal{C}$ = bounded degree, $\varphi \in$ FO: in time $\mathcal{O}(n)$.

Theorem (Courcelle 1990)
$\mathcal{C}$ = bounded tree-width, $\varphi \in$ CMSO: in time $\mathcal{O}(n)$.

Theorem (Grohe, Kreutzer, Siebertz, 2014)
$\mathcal{C}$ = nowhere dense, $\varphi \in$ FO: f. a. $\varepsilon > 0$ in time $\mathcal{O}(n^{1+\varepsilon})$.

Theorem (Bonnet, Kim, Thomassé, Watrigant, 2020)
$\mathcal{C}$ = bounded twin-width, $\varphi \in$ FO:
in time $\mathcal{O}(n)$, provided a witness comes with input.

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Theorem (Seese 1996)

$\mathcal{C}$ = *bounded degree*, $\varphi \in$ FO*:* *in time* $\mathcal{O}(n)$.

Theorem (Courcelle 1990)

$\mathcal{C}$ = *bounded tree-width*, $\varphi \in$ CMSO*:* *in time* $\mathcal{O}(n)$.

Theorem (Grohe, Kreutzer, Siebertz, 2014)

$\mathcal{C}$ = *nowhere dense*, $\varphi \in$ FO*:* *f. a.* $\varepsilon > 0$ *in time* $\mathcal{O}(n^{1+\varepsilon})$.

Theorem (Bonnet, Kim, Thomassé, Watrigant, 2020)

$\mathcal{C}$ = *bounded twin-width*, $\varphi \in$ FO*:*
in time $\mathcal{O}(n)$*, provided a witness comes with input.*

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Theorem (Seese 1996)

$\mathcal{C}$ = *bounded degree*, $\varphi \in$ FO*:* *in time* $\mathcal{O}(n)$.

Theorem (Courcelle 1990)

$\mathcal{C}$ = *bounded tree-width*, $\varphi \in$ CMSO*:* *in time* $\mathcal{O}(n)$.

Theorem (Grohe, Kreutzer, Siebertz, 2014)

$\mathcal{C}$ = *nowhere dense*, $\varphi \in$ FO*:* *f. a.* $\varepsilon > 0$ *in time* $\mathcal{O}(n^{1+\varepsilon})$.

Theorem (Bonnet, Kim, Thomassé, Watrigant, 2020)

$\mathcal{C}$ = *bounded twin-width*, $\varphi \in$ FO*:*
in time $\mathcal{O}(n)$*, provided a witness comes with input.*

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Theorem (Seese 1996)

$\mathcal{C}$ = *bounded degree*, $\varphi \in$ FO*:* *in time* $\mathcal{O}(n)$.

Theorem (Courcelle 1990)

$\mathcal{C}$ = *bounded tree-width*, $\varphi \in$ CMSO*:* *in time* $\mathcal{O}(n)$.

Theorem (Grohe, Kreutzer, Siebertz, 2014)

$\mathcal{C}$ = *nowhere dense*, $\varphi \in$ FO*:* *f. a.* $\varepsilon > 0$ *in time* $\mathcal{O}(n^{1+\varepsilon})$.

Theorem (Bonnet, Kim, Thomassé, Watrigant, 2020)

$\mathcal{C}$ = *bounded twin-width*, $\varphi \in$ FO*:*
in time $\mathcal{O}(n)$*, provided a witness comes with input.*

Algorithmic Meta-Theorems

$\mathcal{C}$: Class of graphs
φ: Formula of some logic

φ-MODELCHECK($\mathcal{C}$)	
Input:	$G \in \mathcal{C}$.
Question:	Does G satisfy φ?

Theorem (Seese 1996)

$\mathcal{C}$ = *bounded degree*, $\varphi \in$ FO*:* *in time* $\mathcal{O}(n)$.

Theorem (Courcelle 1990)

$\mathcal{C}$ = *bounded tree-width*, $\varphi \in$ CMSO*:* *in time* $\mathcal{O}(n)$.

Theorem (Grohe, Kreutzer, Siebertz, 2014)

$\mathcal{C}$ = *nowhere dense*, $\varphi \in$ FO*:* *f. a.* $\varepsilon > 0$ *in time* $\mathcal{O}(n^{1+\varepsilon})$.

Theorem (Bonnet, Kim, Thomassé, Watrigant, 2020)

$\mathcal{C}$ = *bounded twin-width*, $\varphi \in$ FO*:*
in time $\mathcal{O}(n)$*, provided a witness comes with input.*

Can we be faster?

- Restrict the input class
- Restrict the logic
- Approximation
- Randomisation

Can we be faster?

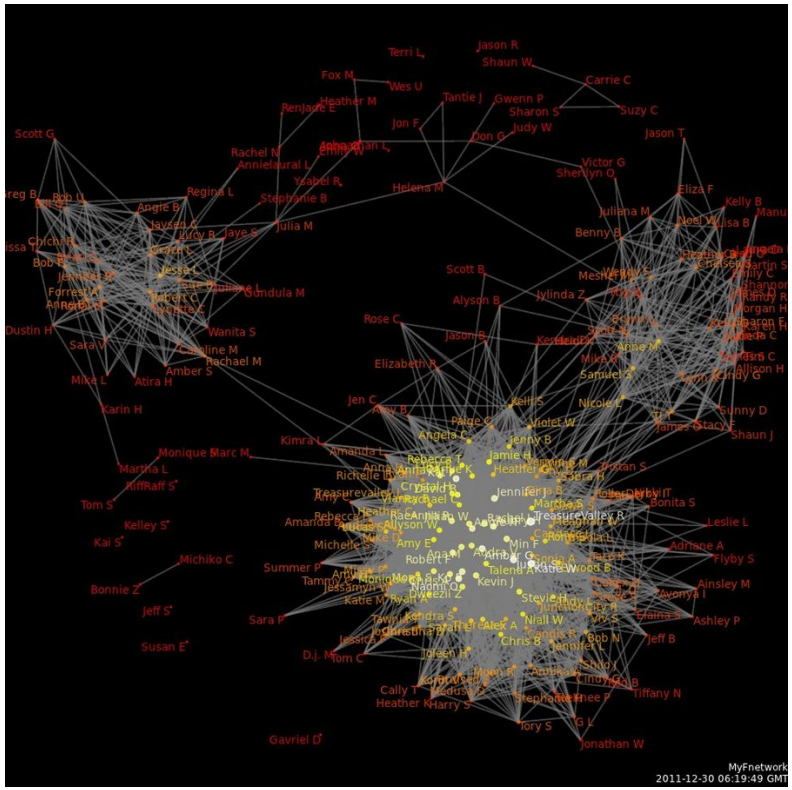
- Restrict the input class
- Restrict the logic
- Approximation
- Randomisation

Contents

Property testing: motivation

‘Efficiency’ when the data set is huge:

Even reading the whole input just once can be too expensive.



Data visualization of Facebook relationships

Author: Kencf0618, License: Creative Commons Attribution-Share Alike 3.0 Unported

Hence: Algorithms with local access to the input.

Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time $=$ running time w.r.t. n.
- The query complexity $=$ number of oracle queries w.r.t. n.

Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time $=$ running time w.r.t. n.
- The query complexity $=$ number of oracle queries w.r.t. n.

Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time = running time w.r.t. n.
- The query complexity = number of oracle queries w.r.t. n.

Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time = running time w.r.t. n.
- The query complexity = number of oracle queries w.r.t. n.

Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time $=$ running time w.r.t. n.
- The query complexity $=$ number of oracle queries w.r.t. n.

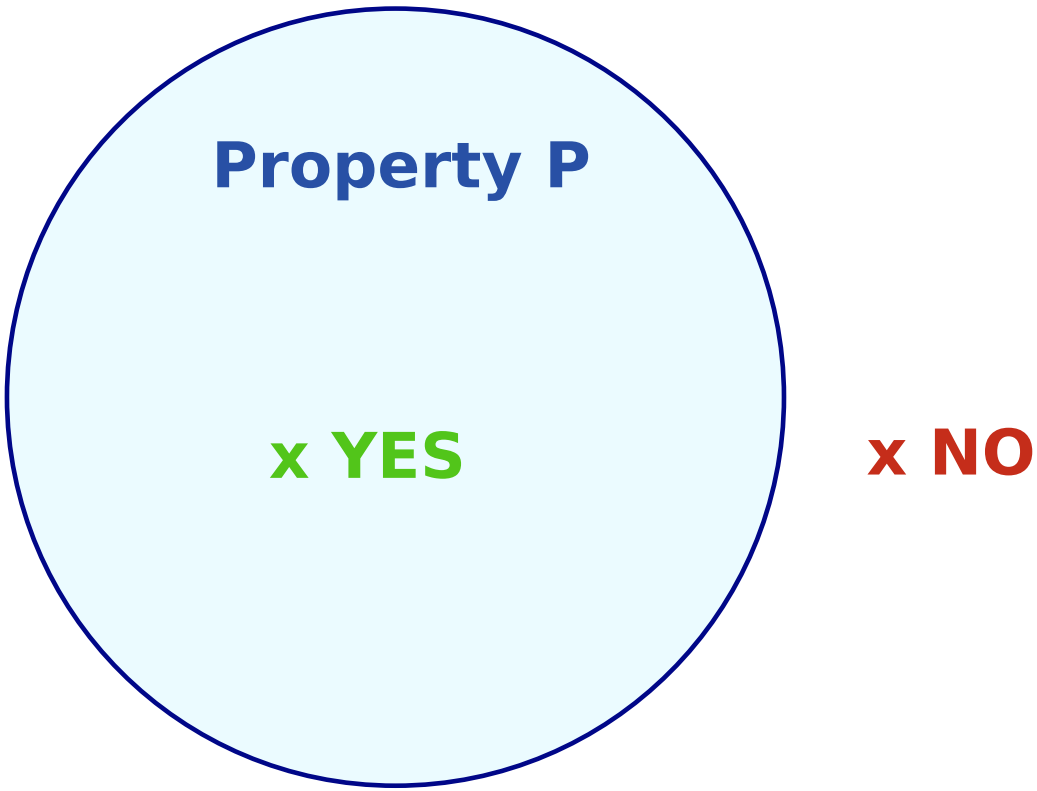
Algorithms with oracle access

From now on: All graphs have degree $\leq d$.

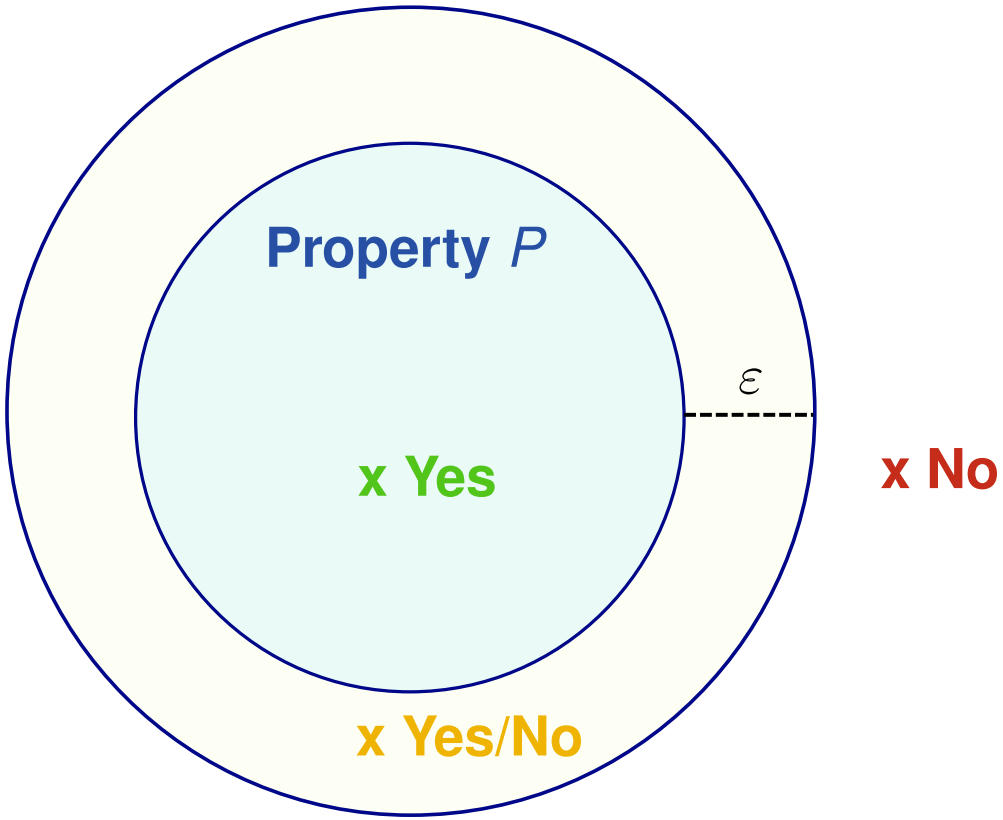
- Input: the number n of vertices of G, and
- Oracle access to G
  - *Query:* v, for $v \in V(G)$
  - *Answer:* the 1-neighbourhood of vertex v
- The running time $=$ running time w.r.t. n.
- The query complexity $=$ number of oracle queries w.r.t. n.

Decision Problems

Decision Problems



Property Testing = relaxation of decision problems

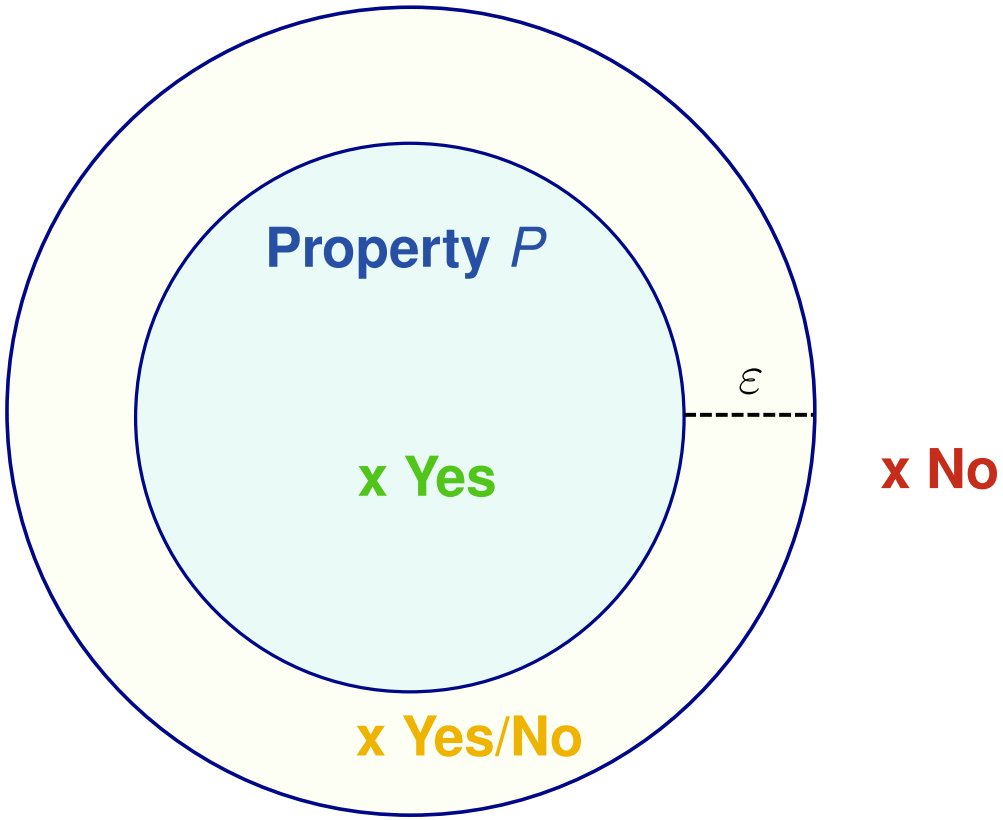


On inputs that have the property: YES with probability at least $2/3$.

On ε-far inputs: NO with probability at least $2/3$.

Aim: extremely efficient.

Property Testing = relaxation of decision problems

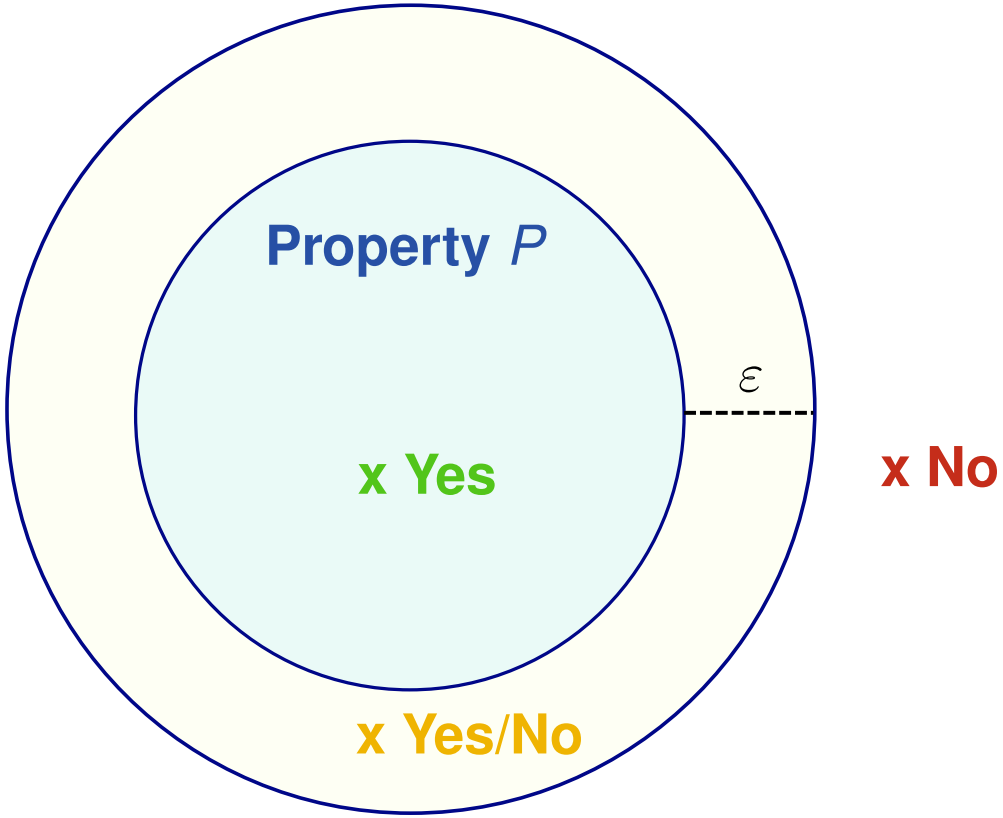


On inputs that have the property: YES with probability at least $2/3$.

On ε-far inputs: NO with probability at least $2/3$.

Aim: extremely efficient.

Property Testing = relaxation of decision problems

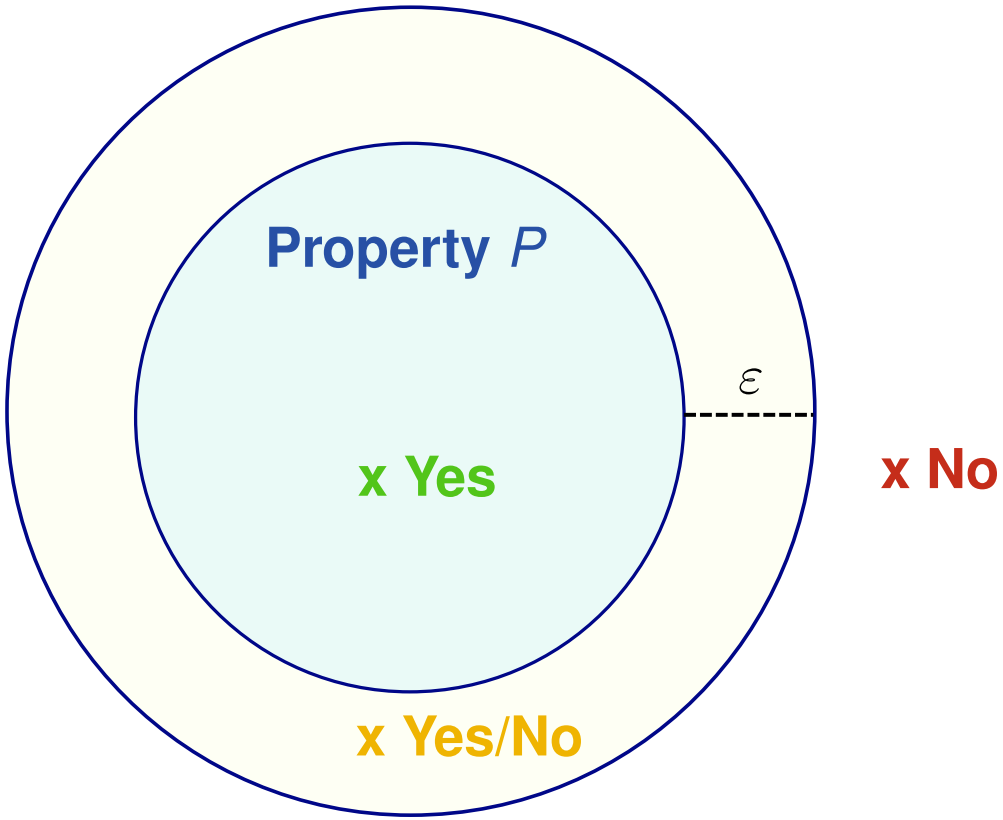


On inputs that have the property: YES with probability at least $2/3$.

On ε-far inputs: NO with probability at least $2/3$.

Aim: extremely efficient.

Property Testing = relaxation of decision problems



On inputs that have the property: YES with probability at least $2/3$.

On ε-far inputs: NO with probability at least $2/3$.

Aim: extremely efficient.

Property Testing

- First introduced in the context of programme checking
  [R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. SIAM J. Computing, 1996]
- Property testing of dense graphs
  [O. Goldreich, O. Goldwasser and D. Ron. Property Testing and its Connection to Learning and Approximation. Journal of the ACM, 1998]

  [N. Alon, E. Fischer, I. Newman, A. Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. SIAM J. Computing, 2009]
- Property testing of regular languages
  [N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. SIAM J. Computing, 2000.]
- Property testing of sparse graphs
  [O. Goldreich and D. Ron. Property Testing in Bounded Degree Graphs. Algorithmica, 2002]

  [Ito, Khoury, Newman. On the characterization of 1-sided error strongly testable graph properties for bounded-degree graphs. J. Comput. Complex. 2020]

  [O. Goldreich and L. Tauber. On Testing Isomorphism to a Fixed Graph in the Bounded-Degree Graph Model. ECCC, 2023]

Property Testing

- Constraint Satisfaction and Satisfiability
  [H. Chen, M. Valeriote, Y. Yoshida. Testing Assignments to Constraint Satisfaction Problems. FOCS 2016]
  [Y. Yoshida. Optimal constant-time approximation algorithms and (unconditional) inapproximability results for every bounded-degree CSP.]
- Machine Learning
  [K. Hayashi, Y. Yoshida. Fitting Low-Rank Tensors in Constant Time. NIPS 2017]
  [M. Grohe, M. Ritzert. Learning first-order definable concepts over structures of small degree. LICS 2017]
- Databases
  [H. Chen, Y. Yoshida. Testability of Homomorphism Inadmissibility: Property Testing Meets Database Theory. PODS 2019]
- Textbooks
  [Oded Goldreich. Introduction to Property Testing - Problems and Techniques. Cambridge University Press 2017]
  [Arnab Bhattacharyya, Yuichi Yoshida. Property Testing - Problems and Techniques. Springer 2022]

Distance (d-bounded degree model)

- Let $\varepsilon \in [0,1]$.
  Graphs G and H, both on n vertices, are ε-close, if we can make them isomorphic by modifying up to εdn edges of G or H.

  Edge modification = insertion/deletion

- If G, H are not ε-*close*, then they are ε-far.

- A graph G is ε-close to a class $\mathcal{C}$ if G is ε-close to some $H \in \mathcal{C}$.
  Otherwise, G is ε-far from $\mathcal{C}$.

Distance (d-bounded degree model)

- Let $\varepsilon \in [0, 1]$.
  Graphs G and H, both on n vertices, are ε-close, if we can make them isomorphic by modifying up to εdn edges of G or H.

  Edge modification = insertion/deletion

- If G, H are not ε-*close*, then they are ε-far.

- A graph G is ε-close to a class $\mathcal{C}$ if G is ε-close to some $H \in \mathcal{C}$.
  Otherwise, G is ε-far from $\mathcal{C}$.

Distance (d-bounded degree model)

- Let $\varepsilon \in [0, 1]$.
  Graphs G and H, both on n vertices, are ε-close, if we can make them isomorphic by modifying up to εdn edges of G or H.

  Edge modification $=$ insertion/deletion

- If G, H are not ε-*close*, then they are ε-far.

- A graph G is ε-close to a class $\mathcal{C}$ if G is ε-close to some $H \in \mathcal{C}$.
  Otherwise, G is ε-far from $\mathcal{C}$.

Distance (d-bounded degree model)

- Let $\varepsilon \in [0, 1]$.
  Graphs G and H, both on n vertices, are ε-close, if we can make them isomorphic by modifying up to εdn edges of G or H.

  Edge modification $=$ insertion/deletion

- If G, H are not ε-*close*, then they are ε-far.

- A graph G is ε-close to a class $\mathcal{C}$ if G is ε-close to some $H \in \mathcal{C}$.
  Otherwise, G is ε-far from $\mathcal{C}$.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,
2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,
2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,

2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,

2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,
2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Testability

Let $\mathcal{P}$ be a property.
An ε-tester for $\mathcal{P}$ is a probabilistic algorithm that, given oracle access to G and given $n := |V(G)|$ as input, does the following:

1. If $G \in \mathcal{P}$, then the tester accepts with probability $\geq \frac{2}{3}$,

2. if G is ε-far from $\mathcal{P}$, then the tester rejects with probability $\geq \frac{2}{3}$.

$\mathcal{P}$ is uniformly testable, if for each ε there is an ε-tester for $\mathcal{P}$ with constant query complexity.

$\mathcal{P}$ is (non-uniformly) testable, if for each ε and each n, there is a tester for $\mathcal{P}_n := \{G \in \mathcal{P} : |V(G)| = n\}$ with constant query complexity.

Remark
'Uniformly testable' is non-standard.

Example: induced subgraph-freeness

Fix H.
$\mathcal{P} :=$ all graphs that omit H as induced subgraph.

Theorem
$\mathcal{P}$ is testable with constant query complexity and running time.

Proof: Assume H is connected and $|V(H)| > 1$.
Let $r :=$ diameter of H.
Given ε and oracle access to input G on n vertices, do:

1. Uniformly and independently sample $\alpha := \log_{1-\varepsilon} 1/3$ vertices.
2. Explore the r-neighbourhood of each sampled vertex.
3. If an r-neighbourhood contains H as induced subgraph, reject; otherwise accept.

Running time: constant (r-neighbourhoods have constant size).

Example: induced subgraph-freeness

Fix H.
$\mathcal{P} :=$ all graphs that omit H as induced subgraph.

Theorem
$\mathcal{P}$ is testable with constant query complexity and running time.

Proof: Assume H is connected and $|V(H)| > 1$.
Let $r :=$ diameter of H.
Given ε and oracle access to input G on n vertices, do:

1. Uniformly and independently sample $\alpha := \log_{1-\varepsilon} 1/3$ vertices.
2. Explore the r-neighbourhood of each sampled vertex.
3. If an r-neighbourhood contains H as induced subgraph, reject; otherwise accept.

Running time: constant (r-neighbourhoods have constant size).

Example: induced subgraph-freeness

Fix H.
$\mathcal{P} :=$ all graphs that omit H as induced subgraph.

Theorem
$\mathcal{P}$ is testable with constant query complexity and running time.

Proof: Assume H is connected and $|V(H)| > 1$.
Let $r :=$ diameter of H.
Given ε and oracle access to input G on n vertices, do:

1. Uniformly and independently sample $\alpha := \log_{1-\varepsilon} 1/3$ vertices.
2. Explore the r-neighbourhood of each sampled vertex.
3. If an r-neighbourhood contains H as induced subgraph, reject; otherwise accept.

Running time: constant (r-neighbourhoods have constant size).

Example: induced subgraph-freeness

Fix H.
$\mathcal{P} :=$ all graphs that omit H as induced subgraph.

Theorem
$\mathcal{P}$ is testable with constant query complexity and running time.

Proof: Assume H is connected and $|V(H)| > 1$.
Let $r :=$ diameter of H.
Given ε and oracle access to input G on n vertices, do:

1. Uniformly and independently sample $\alpha := \log_{1-\varepsilon} 1/3$ vertices.
2. Explore the r-neighbourhood of each sampled vertex.
3. If an r-neighbourhood contains H as induced subgraph, reject; otherwise accept.

Running time: constant (r-neighbourhoods have constant size).

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.

2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Induced subgraph-freeness: correctness

1. If $G \in \mathcal{P}$ then the algorithm always accepts G.
2. If G is ε-far from $\mathcal{P}$:
   - Let $X \subseteq V(G)$ be the set of all vertices that belong to an induced copy of H in G.
   - Claim: $|X| > \varepsilon n$.
     Otherwise, remove all edges incident to X. Then the resulting graph G' contains no induced copy of H, so $G' \in \mathcal{P}$. But we removed $\leq \varepsilon dn$ edges, so G is ε-close to $\mathcal{P}$, a contradiction.
   - Hence the probability that out of the α elements sampled, no element of X appears, is less than $(1-\varepsilon)^{\alpha} = 1/3$.
   - Hence with probability at least $2/3$ the tester samples a vertex in X and thus finds an induced copy of H.
   - The tester rejects with probability at least $2/3$.

The case that H is not connected requires a bit more work. □

Examples

On bounded degree graphs:

Uniformly testable with constant query complexity and running time:

- k-(edge-)connectivity
- being Eulerian
- subgraph-freeness
- induced subgraph-freeness

Not testable with constant query complexity:

- Bipartiteness, colourability
- Expander graphs
- Hamiltonicity

[Goldreich and Ron 2002; Yoshida and Ito 2010]

Examples

On bounded degree graphs:

Uniformly testable with constant query complexity and running time:

- k-(edge-)connectivity
- being Eulerian
- subgraph-freeness
- induced subgraph-freeness

Not testable with constant query complexity:

- Bipartiteness, colourability
- Expander graphs
- Hamiltonicity

[Goldreich and Ron 2002; Yoshida and Ito 2010]

Examples

On bounded degree graphs:

Uniformly testable with constant query complexity and running time:

- k-(edge-)connectivity
- being Eulerian
- subgraph-freeness
- induced subgraph-freeness

Not testable with constant query complexity:

- Bipartiteness, colourability
- Expander graphs
- Hamiltonicity

[Goldreich and Ron 2002; Yoshida and Ito 2010]

Previous result: PT on hyperfinite graphs, I

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))
Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Let $0 \leq \varepsilon \leq 1$ and $k \in \mathbb{N}$.

- G is (ε, k)-hyperfinite if one can remove εn edges from G and obtain a graph whose connected components have size $\leq k$.
- Fix a function $\rho : \mathbb{R}^+ \to \mathbb{N}$.
  G is ρ-hyperfinite if G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite for every $\varepsilon > 0$.
- A graph class $\mathcal{C}$ is hyperfinite if there is a function ρ such that every $G \in \mathcal{C}$ is ρ-hyperfinite.

Previous result: PT on hyperfinite graphs, I

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))

Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Let $0 \leq \varepsilon \leq 1$ and $k \in \mathbb{N}$.

- G is (ε, k)-hyperfinite if one can remove εn edges from G and obtain a graph whose connected components have size $\leq k$.
- Fix a function $\rho : \mathbb{R}^+ \to \mathbb{N}$.
  G is ρ-hyperfinite if G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite for every $\varepsilon > 0$.
- A graph class $\mathcal{C}$ is hyperfinite if there is a function ρ such that every $G \in \mathcal{C}$ is ρ-hyperfinite.

Previous result: PT on hyperfinite graphs, I

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))
Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Let $0 \leq \varepsilon \leq 1$ and $k \in \mathbb{N}$.

- G is (ε, k)-hyperfinite if one can remove εn edges from G and obtain a graph whose connected components have size $\leq k$.
- Fix a function $\rho : \mathbb{R}^+ \to \mathbb{N}$.
  G is ρ-hyperfinite if G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite for every $\varepsilon > 0$.
- A graph class $\mathcal{C}$ is hyperfinite if there is a function ρ such that every $G \in \mathcal{C}$ is ρ-hyperfinite.

Previous result: PT on hyperfinite graphs, I

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))
Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Let $0 \leq \varepsilon \leq 1$ and $k \in \mathbb{N}$.

- G is (ε, k)-hyperfinite if one can remove εn edges from G and obtain a graph whose connected components have size $\leq k$.
- Fix a function $\rho : \mathbb{R}^+ \to \mathbb{N}$.
  G is ρ-hyperfinite if G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite for every $\varepsilon > 0$.
- A graph class $\mathcal{C}$ is hyperfinite if there is a function ρ such that every $G \in \mathcal{C}$ is ρ-hyperfinite.

Previous result: PT on hyperfinite graphs, I

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))
Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Let $0 \leq \varepsilon \leq 1$ and $k \in \mathbb{N}$.

- G is (ε, k)-hyperfinite if one can remove εn edges from G and obtain a graph whose connected components have size $\leq k$.
- Fix a function $\rho : \mathbb{R}^+ \to \mathbb{N}$.
  G is ρ-hyperfinite if G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite for every $\varepsilon > 0$.
- A graph class $\mathcal{C}$ is hyperfinite if there is a function ρ such that every $G \in \mathcal{C}$ is ρ-hyperfinite.

Previous result: PT on hyperfinite graphs, II

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))
Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Hyperfinite graph classes include
- bounded tree-width graphs,
- planar graphs,
- generally, graphs excluding a fixed minor.

Remark
The theorem does not say anything about the running time.
$\exists$ undecidable properties (of edgeless graphs) that are testable with constant query complexity by the theorem.

Previous result: PT on hyperfinite graphs, II

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))

Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Hyperfinite graph classes include
- bounded tree-width graphs,
- planar graphs,
- generally, graphs excluding a fixed minor.

Remark

The theorem does not say anything about the running time.
$\exists$ undecidable properties (of edgeless graphs) that are testable with constant query complexity by the theorem.

Previous result: PT on hyperfinite graphs, II

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))

Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Hyperfinite graph classes include
- bounded tree-width graphs,
- planar graphs,
- generally, graphs excluding a fixed minor.

Remark

The theorem does not say anything about the running time.

$\exists$ undecidable properties (of edgeless graphs) that are testable with constant query complexity by the theorem.

Previous result: PT on hyperfinite graphs, II

Theorem (Newman and Sohler, 2013 (Benjamini, Shramm, Shapira, Elek))

Let $\mathcal{C}$ be a hyperfinite class of graphs of bounded degree.

Every property $\mathcal{P} \subseteq \mathcal{C}$ is non-uniformly testable on $\mathcal{C}$ with constant query complexity.

Hyperfinite graph classes include
- bounded tree-width graphs,
- planar graphs,
- generally, graphs excluding a fixed minor.

Remark

The theorem does not say anything about the running time.
$\exists$ undecidable properties (of edgeless graphs) that are testable with constant query complexity by the theorem.

Contents

CMSO *on bounded tw with sublinear running time*

Theorem (A., Harwath, STACS 2018)

Let $\mathcal{C}_d^t$ be the class of all t-bounded tree-width graphs of degree $\leq d$.

Every CMSO-definable property $\mathcal{P} \subseteq \mathcal{C}_d^t$ is uniformly testable with constant query complexity and polylogarithmic running time.

- First algorithmic meta-theorem with sublinear running time.
- Open: can it be improved to constant running time?

CMSO *on bounded tw with sublinear running time*

Theorem (A., Harwath, STACS 2018)

Let $\mathcal{C}_d^t$ be the class of all t-bounded tree-width graphs of degree $\leq d$.

Every CMSO-definable property $\mathcal{P} \subseteq \mathcal{C}_d^t$ is uniformly testable with constant query complexity and polylogarithmic running time.

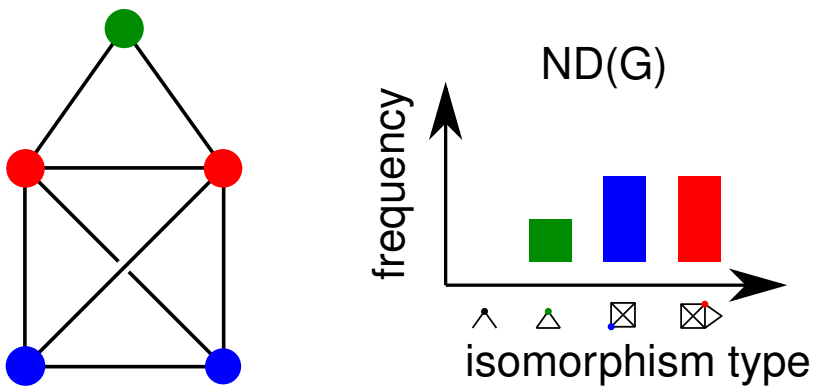
- First algorithmic meta-theorem with sublinear running time.
- Open: can it be improved to constant running time?

Algorithm and proof idea

Theorem (A., Harwath, STACS 2018)

$\mathcal{C}_d^t :=$ *all* t*-bounded tree-width graphs of degree* $\leq d$*.*

Every CMSO*-definable property* $\mathcal{P} \subseteq \mathcal{C}_d^t$ *is uniformly testable with constant query complexity and polylogarithmic running time.*



Algorithm: Fix ε. Given oracle access to $G \in \mathcal{C}_d^t$, and $n = |V(G)|$.

- $\bar{x} =$ estimation of neighbourhood distribution $\mathrm{ND}(G)$, via sampling
- Accept, if $\|\bar{x} - \mathrm{ND}(H)\|_1 \leq \lambda = \lambda(\varepsilon)$ is small, for some $H \in \mathcal{P}$ on n vertices, otherwise reject.

Correctness: Characterisation of testability; theorem from [Newman, Sohler, 2013].

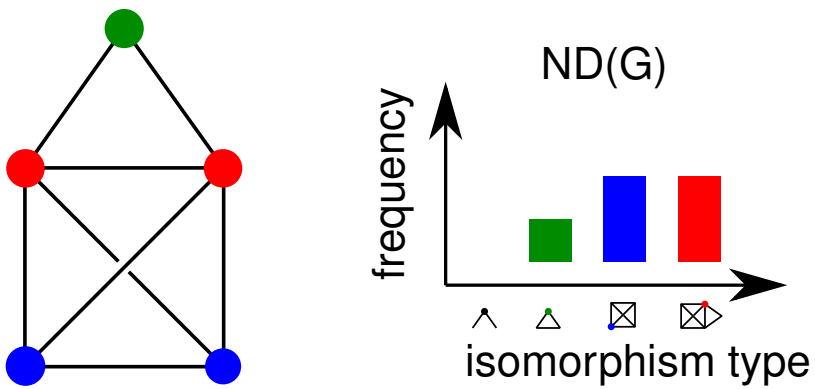
Runtime: Show: histogram vectors of CMSO properties are semilinear, via [Fischer, Makowsky, 2003]; use [Lenstra, IP with a fixed number of variables, 1983]. □

Algorithm and proof idea

Theorem (A., Harwath, STACS 2018)

$\mathcal{C}_d^t :=$ *all* t*-bounded tree-width graphs of degree* $\leq d$*.*

Every CMSO*-definable property* $\mathcal{P} \subseteq \mathcal{C}_d^t$ *is uniformly testable with constant query complexity and polylogarithmic running time.*



Algorithm: Fix ε. Given oracle access to $G \in \mathcal{C}_d^t$, and $n = |V(G)|$.

- $\bar{x} =$ estimation of neighbourhood distribution $\mathrm{ND}(G)$, via sampling
- Accept, if $\|\bar{x} - \mathrm{ND}(H)\|_1 \leq \lambda = \lambda(\varepsilon)$ is small, for some $H \in \mathcal{P}$ on n vertices, otherwise reject.

Correctness: Characterisation of testability; theorem from [Newman, Sohler, 2013].

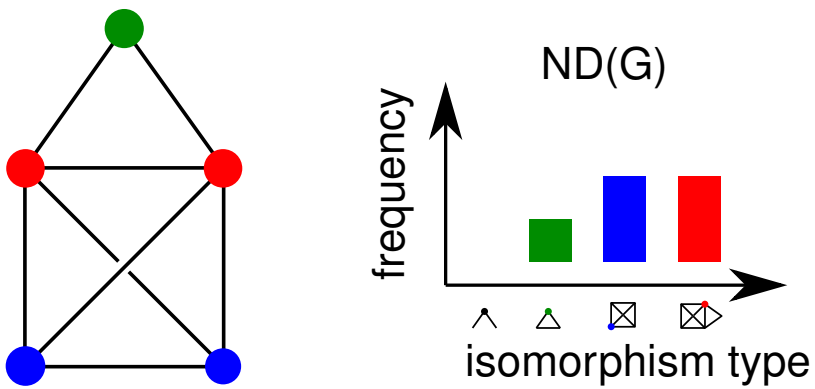
Runtime: Show: histogram vectors of CMSO properties are semilinear, via [Fischer, Makowsky, 2003]; use [Lenstra, IP with a fixed number of variables, 1983]. □

Algorithm and proof idea

Theorem (A., Harwath, STACS 2018)

$\mathcal{C}_d^t :=$ *all* t*-bounded tree-width graphs of degree* $\leq d$*.*

Every CMSO*-definable property* $\mathcal{P} \subseteq \mathcal{C}_d^t$ *is uniformly testable with constant query complexity and polylogarithmic running time.*



Algorithm: Fix ε. Given oracle access to $G \in \mathcal{C}_d^t$, and $n = |V(G)|$.

- $\bar{x} =$ estimation of neighbourhood distribution $\mathrm{ND}(G)$, via sampling
- Accept, if $\|\bar{x} - \mathrm{ND}(H)\|_1 \leq \lambda = \lambda(\varepsilon)$ is small, for some $H \in \mathcal{P}$ on n vertices, otherwise reject.

Correctness: Characterisation of testability; theorem from [Newman, Sohler, 2013].

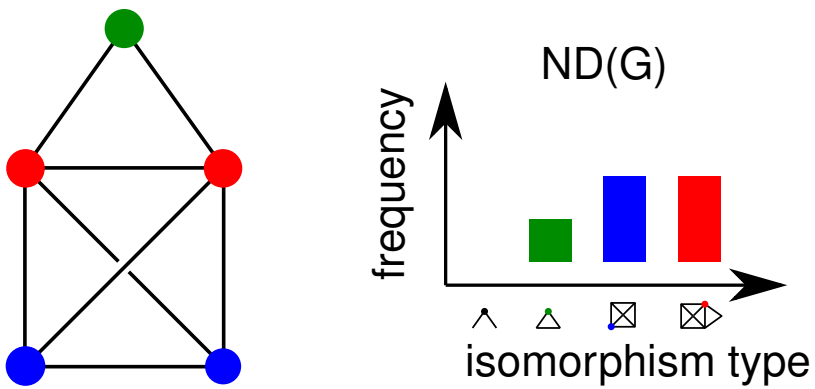
Runtime: Show: histogram vectors of CMSO properties are semilinear, via [Fischer, Makowsky, 2003]; use [Lenstra, IP with a fixed number of variables, 1983]. □

Algorithm and proof idea

Theorem (A., Harwath, STACS 2018)

$\mathcal{C}_d^t :=$ *all* t*-bounded tree-width graphs of degree* $\leq d$*.*

Every CMSO*-definable property* $\mathcal{P} \subseteq \mathcal{C}_d^t$ *is uniformly testable with constant query complexity and polylogarithmic running time.*



Algorithm: Fix ε. Given oracle access to $G \in \mathcal{C}_d^t$, and $n = |V(G)|$.

- $\bar{x} =$ estimation of neighbourhood distribution $\mathrm{ND}(G)$, via sampling
- Accept, if $\|\bar{x} - \mathrm{ND}(H)\|_1 \leq \lambda = \lambda(\varepsilon)$ is small, for some $H \in \mathcal{P}$ on n vertices, otherwise reject.

Correctness: Characterisation of testability; theorem from [Newman, Sohler, 2013].

Runtime: Show: histogram vectors of CMSO properties are semilinear, via [Fischer, Makowsky, 2003]; use [Lenstra, IP with a fixed number of variables, 1983]. □

Logic & PT on bounded degree

[Courcelle, 1990]
$\mathcal{C}$:= graph class of bounded treewidth

Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO
Question: Does G satisfy φ?

Decidable in FPT: time $f(|\varphi|) \cdot \mathcal{O}(n)$

Logic & PT on bounded degree

[Courcelle, 1990] $\mathcal{C}$:= graph class of bounded treewidth Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Decidable in FPT: time $f(\lvert\varphi\rvert) \cdot \mathcal{O}(n)$	[Adler, Harwath, STACS 2018] $\mathcal{C}$:= bounded treewidth+degree Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Testable in time $f(\lvert\varphi\rvert) \cdot o(n)$

Logic & PT on bounded degree

[Courcelle, 1990] $\mathcal{C}$:= graph class of bounded treewidth Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Decidable in FPT: time $f(\lvert\varphi\rvert) \cdot \mathcal{O}(n)$	[Adler, Harwath, STACS 2018] $\mathcal{C}$:= bounded treewidth+degree Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Testable in time $f(\lvert\varphi\rvert) \cdot o(n)$
[Alon, Fischer, Krivelevich, Szegedy, 2000] Dense graphs: • $\exists^*\forall^*$-fragment of FO is testable • non-testable $\forall^*\exists^*$-property	

Logic & PT on bounded degree

[Courcelle, 1990] $\mathcal{C}$:= graph class of bounded treewidth Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Decidable in FPT: time $f(\lvert\varphi\rvert) \cdot \mathcal{O}(n)$	[Adler, Harwath, STACS 2018] $\mathcal{C}$:= bounded treewidth+degree Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Testable in time $f(\lvert\varphi\rvert) \cdot o(n)$
[Alon, Fischer, Krivelevich, Szegedy, 2000] Dense graphs: • $\exists^*\forall^*$-fragment of FO is testable • non-testable $\forall^*\exists^*$-property	[Adler, Köhler, Peng, SODA 2021] Bounded degree: • $\exists^*\forall^*$-fragment is testable • non-testable $\forall^*\exists^*$-property

Testability of FO*?*

Theorem (A., Köhler, Peng, SODA 2021, SIAM J. Computing 2024)

In the bounded degree model:

- *Every property that can be expressed by an* FO*-formula with quantifier prefix type* $\exists^*\forall^*$ *is testable with constant query complexity.*
- *There is an* FO*-property of quantifier prefix type* $\forall^*\exists^*$ *that is not testable.*

Every $\exists^*\forall^*$-property is testable (proof idea)

- On bounded-degree graphs, every $\exists^*\forall^*$-property is 'indistinguishable' from a $\forall^*$-property.

- $\forall^*$-properties are testable with constant query complexity:

  - $\forall\overline{x}\varphi(\overline{x}) \equiv \neg\exists\overline{x}\psi(\overline{x})$, for $\psi(\overline{x}) := \neg\varphi(\overline{x})$.

  - '$\psi(\overline{x})$-freeness' is testable with constant query complexity and running time similar to induced subgraph freeness.

Every $\exists^*\forall^*$-property is testable (proof idea)

- On bounded-degree graphs, every $\exists^*\forall^*$-property is 'indistinguishable' from a $\forall^*$-property.

- $\forall^*$-properties are testable with constant query complexity:

  - $\forall\overline{x}\varphi(\overline{x}) \equiv \neg\exists\overline{x}\psi(\overline{x})$, for $\psi(\overline{x}) := \neg\varphi(\overline{x})$.

  - '$\psi(\overline{x})$-freeness' is testable with constant query complexity and running time similar to induced subgraph freeness.

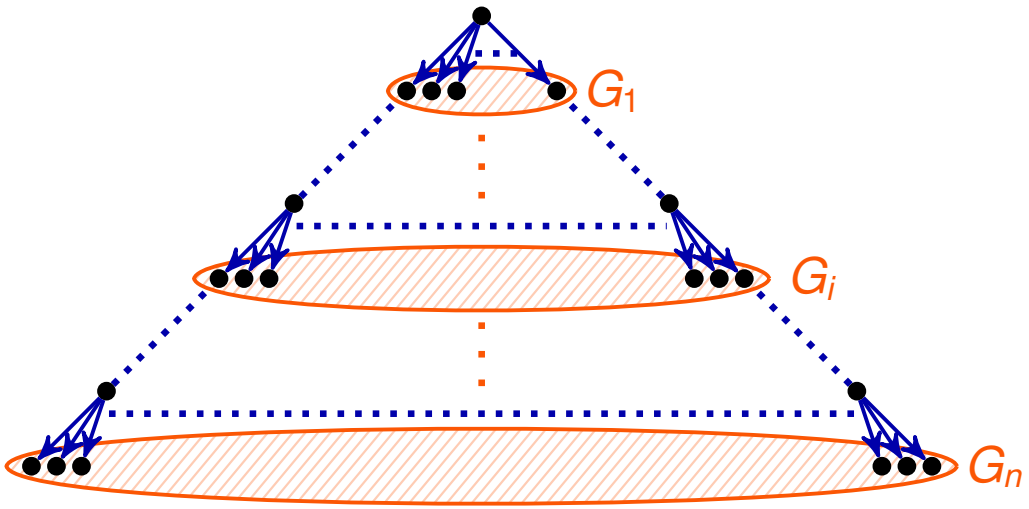
Non-testable $\forall^*\exists^*$-property (proof idea)

Find $\forall^*\exists^*$-formula φ whose models are expander graphs.

$(G_i)_{i\in\mathbb{N}}$: zig-zag construction [Rheingold, Vadhan, Wigdersen, Ann. of Math., 2002]

Non-testable $\forall^*\exists^*$-property (proof idea)

Find $\forall^*\exists^*$-formula φ whose models are expander graphs.



$(G_i)_{i\in\mathbb{N}}$: zig-zag construction [Rheingold, Vadhan, Wigdersen, Ann. of Math., 2002]

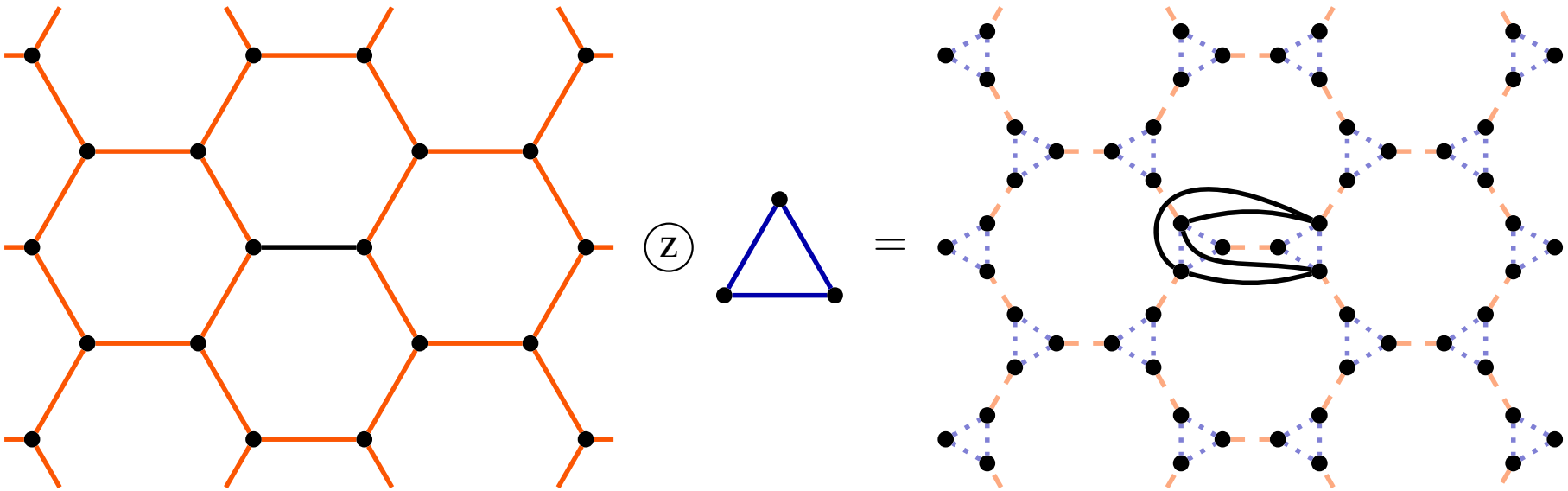
Avi Wigdersen: Turing Award recipient 2023

*1965: Israeli computer scientist and mathematician, school of mathematics, Princeton, USA

Avi Wigdersen

"For reshaping our understanding of the role of randomness in computation, and for decades of intellectual leadership in theoretical computer science."

Example: Zig-zag product of a 3-regular grid with a triangle



Non-testable $\forall^*\exists^*$-property (proof idea), cont'd

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

- Corollary: Tester cannot distinguish expander G from $H \dot\cup \ldots \dot\cup H$.
- $G \models \varphi$ and $H \dot\cup \ldots \dot\cup H$ is far from satisfying φ.

□

Non-testable $\forall^*\exists^*$-property (proof idea), cont'd

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

- Corollary: Tester cannot distinguish expander G from $H \dot\cup \ldots \dot\cup H$.
- $G \models \varphi$ and $H \dot\cup \ldots \dot\cup H$ is far from satisfying φ.

□

Non-testable $\forall^*\exists^*$-property (proof idea), cont'd

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

- Corollary: Tester cannot distinguish expander G from $H \dot{\cup} \ldots \dot{\cup} H$.

- $G \models \varphi$ and $H \dot{\cup} \ldots \dot{\cup} H$ is far from satisfying φ.

Non-testable $\forall^*\exists^*$-property (proof idea), cont'd

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

- Corollary: Tester cannot distinguish expander G from $H \dot\cup \ldots \dot\cup H$.
- $G \models \varphi$ and $H \dot\cup \ldots \dot\cup H$ is far from satisfying φ.

□

Logic & PT on bounded degree

[Courcelle, 1990] $\mathcal{C}$:= graph class of bounded treewidth Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Decidable in FPT: time $f(\lvert\varphi\rvert) \cdot \mathcal{O}(n)$	[A., Harwath, STACS 2018] $\mathcal{C}$:= bounded treewidth+degree Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Testable in time $f(\lvert\varphi\rvert) \cdot o(n)$
[Alon, Fischer, Krivelevich, Szegedy, 2000] Dense graphs: • $\exists^*\forall^*$-fragment of FO is testable • non-testable $\forall^*\exists^*$-property	[A., Köhler, Peng, SODA 2021] Bounded degree: • $\exists^*\forall^*$-fragment is testable • non-testable $\forall^*\exists^*$-property

[A., Köhler, Peng, CCC 2021]: Answers to open questions on

- Characterisation of POTs in [Goldreich and Ron, STOC 2009; SICOMP 2011].
- Charact. of 1-sided testability in [Ito, Khoury, Newman, J. Comput. Complex. 2020].

Journal version to appear in [A., Köhler, Peng, SIAM J. on Computing, 2024].

Logic & PT on bounded degree

[Courcelle, 1990] $\mathcal{C}$:= graph class of bounded treewidth Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Decidable in FPT: time $f(\lvert\varphi\rvert) \cdot \mathcal{O}(n)$	[A., Harwath, STACS 2018] $\mathcal{C}$:= bounded treewidth+degree Input: $G \in \mathcal{C}$, $\varphi \in$ CMSO Question: Does G satisfy φ? Testable in time $f(\lvert\varphi\rvert) \cdot o(n)$
[Alon, Fischer, Krivelevich, Szegedy, 2000] Dense graphs: • $\exists^*\forall^*$-fragment of FO is testable • non-testable $\forall^*\exists^*$-property	[A., Köhler, Peng, SODA 2021] Bounded degree: • $\exists^*\forall^*$-fragment is testable • non-testable $\forall^*\exists^*$-property

[A., Köhler, Peng, CCC 2021]: Answers to open questions on

- Characterisation of POTs in [Goldreich and Ron, STOC 2009; SICOMP 2011].
- Charact. of 1-sided testability in [Ito, Khoury, Newman, J. Comput. Complex. 2020].

Journal version to appear in [A., Köhler, Peng, SIAM J. on Computing, 2024].

Contents

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

Outlook

Open problem: can constant running time be achieved on bounded degree+tree-width in the classical bounded degree model (without vertex modifications)?

Open problem: characterisation of testability in the bounded degree model

Open problem: characterisation of *uniform* testability in the bounded degree model

Extension to sparse graphs of unbounded degree, and beyond

Testability of other logics, including temporal logics

Testability in graph databases

[A., Harwath. Property Testing for Bounded Degree Databases, STACS 2018]
http://eprints.whiterose.ac.uk/126769/8/LIPIcs-STACS-2018-6.pdf

[A., Fahey. Faster Property Testers in a Variation of the Bounded Degree Model, FSTTCS 2020]
https://arxiv.org/pdf/2009.07770.pdf

[A., Köhler, Peng. On Testability of First-Order Properties in Bounded-Degree Graphs, SODA 2021]
https://arxiv.org/pdf/2008.05800.pdf

[Aboulker, A., Kim, Sintiari, Trotignon. On the tree-width of even-hole-free graphs, Europ. J. Combinatorics, 2021.]
https://arxiv.org/pdf/2008.05504.pdf

[A., Köhler, Peng. GSF-locality is not sufficient for proximity-oblivious testing, CCC, 2021.]
https://drops.dagstuhl.de/opus/volltexte/2021/14308/

[A., Köhler. On graphs of bounded degree that are far from being Hamiltonian, DMTCS 2022.]
https://dmtcs.episciences.org/8997/pdf

[A., Köhler, Peng. On testability of first-order properties in bounded-degree graphs and connections to proximity-oblivious testing, SIAM J. on Computing, accepted for publication 2024.]

Thank you!

CMSO *on bounded tw with constant running time?*

Theorem (A., Fahey, FSTTCS 2020)

Let $\mathcal{C}_d^t$ be the class of all t-bounded tree-width graphs of degree $\leq d$.

If we allow vertex deletions/additions in addition to edge modifications:
Every CMSO-definable property $\mathcal{P} \subseteq \mathcal{C}_d^t$ is uniformly testable with constant running time.

The proof uses a constructive version (for bounded tree-width) of:

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

CMSO *on bounded tw with constant running time?*

Theorem (A., Fahey, FSTTCS 2020)

Let $\mathcal{C}_d^t$ be the class of all t-bounded tree-width graphs of degree $\leq d$.

If we allow vertex deletions/additions in addition to edge modifications:
Every CMSO*-definable property $\mathcal{P} \subseteq \mathcal{C}_d^t$ is uniformly testable with constant running time.*

The proof uses a constructive version (for bounded tree-width) of:

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

CMSO *on bounded tw with constant running time?*

Theorem (A., Fahey, FSTTCS 2020)

Let $\mathcal{C}_d^t$ be the class of all t-bounded tree-width graphs of degree $\leq d$.

If we allow vertex deletions/additions in addition to edge modifications:
Every CMSO*-definable property $\mathcal{P} \subseteq \mathcal{C}_d^t$ is uniformly testable with constant running time.*

The proof uses a constructive version (for bounded tree-width) of:

Theorem (Alon 2011)

On graphs of bounded degree:
For any graph G, radius r and ε, there exists a graph H whose size is independent of G such that $||ND_r(G) - ND_r(H)||_1 \leq \varepsilon$.

Implementation

- Test connectivity, 2- and 3-edge-connectivity in constant running time
  `http://zshg.sourceforge.net`

- A graph format and library that allow loading a graph from a file in constant time (where the file is loaded into memory space and the OS loads data when accessed).
  `http://slgraph.sourceforge.net`

With Dr. Philipp Krause, University of Freiburg.

Implementation

- Test connectivity, 2- and 3-edge-connectivity in constant running time
  `http://zshg.sourceforge.net`

- A graph format and library that allow loading a graph from a file in constant time (where the file is loaded into memory space and the OS loads data when accessed).
  `http://slgraph.sourceforge.net`

With Dr. Philipp Krause, University of Freiburg.