Tutorial: Dynamic FPT Algorithms

Tuukka Korhonen

20 April 2026

Workshop on Modern Trends in Parameterized Complexity

Outline

- ⇒ Intro
- Technical showcase: Dynamic Baker's scheme
- Overview of the literature
- Technical showcase: Dynamic treewidth
- Future of dynamic FPT and a mini open problem / brainstorming session

Dynamic graph algorithms

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ *amortized* time, but no deletions [Tarjan'75]

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ *amortized* time, but no deletions [Tarjan'75] (also $\mathcal{O}(\log n)$ worst-case time)

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ *amortized* time, but no deletions [Tarjan'75] (also $\mathcal{O}(\log n)$ worst-case time)
3. Link/cut tree: $\mathcal{O}(\log n)$ amortized time when G is a forest [Sleator&Tarjan'81]

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(G): Initialize the data structure with a given n-vertex graph G
  2. Insert(u, v): Insert edge between u and v
  3. Delete(u, v): Delete edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ *amortized* time, but no deletions [Tarjan'75] (also $\mathcal{O}(\log n)$ worst-case time)
3. Link/cut tree: $\mathcal{O}(\log n)$ amortized time when G is a forest [Sleator&Tarjan'81]
4. [Henzinger&King'99]: $\mathcal{O}(\log^3 n)$ amortized time

Dynamic FPT

- Dynamic algorithms in the FPT setting

Dynamic FPT

- Dynamic algorithms in the FPT setting
- Update times:
  - $f(k) \cdot \log n$
  - $f(k) \cdot \text{polylog } n$
  - $f(k) \cdot n^{o(1)}$
  - $f(k) \cdot \sqrt{n}$
  - $2^{\mathcal{O}(k)} \cdot \log n$
  - 1.2738^k
  - ...

Dynamic FPT

- Dynamic algorithms in the FPT setting
- Update times:
  - $f(k) \cdot \log n$
  - $f(k) \cdot \text{polylog}\, n$
  - $f(k) \cdot n^{o(1)}$
  - $f(k) \cdot \sqrt{n}$
  - $2^{\mathcal{O}(k)} \cdot \log n$
  - 1.2738^k
  - ...
- Mostly makes sense for problems for which $f(k) \cdot n^{2-\varepsilon}$ time FPT algorithms known

Dynamic FPT

- Dynamic algorithms in the FPT setting
- Update times:
  - $f(k) \cdot \log n$
  - $f(k) \cdot \text{polylog}\, n$
  - $f(k) \cdot n^{o(1)}$
  - $f(k) \cdot \sqrt{n}$
  - $2^{\mathcal{O}(k)} \cdot \log n$
  - 1.2738^k
  - ...
- Mostly makes sense for problems for which $f(k) \cdot n^{2-\varepsilon}$ time FPT algorithms known
- Dynamic kernelization: After each update, output a kernel

Dynamic FPT

- Dynamic algorithms in the FPT setting
- Update times:
  - $f(k) \cdot \log n$
  - $f(k) \cdot \text{polylog}\, n$
  - $f(k) \cdot n^{o(1)}$
  - $f(k) \cdot \sqrt{n}$
  - $2^{\mathcal{O}(k)} \cdot \log n$
  - 1.2738^k
  - ...
- Mostly makes sense for problems for which $f(k) \cdot n^{2-\varepsilon}$ time FPT algorithms known
- Dynamic kernelization: After each update, output a kernel
  - Then can apply the best static algorithm on the kernel

Dynamic FPT

- Dynamic algorithms in the FPT setting
- Update times:
  - $f(k) \cdot \log n$
  - $f(k) \cdot \text{polylog}\, n$
  - $f(k) \cdot n^{o(1)}$
  - $f(k) \cdot \sqrt{n}$
  - $2^{\mathcal{O}(k)} \cdot \log n$
  - 1.2738^k
  - ...
- Mostly makes sense for problems for which $f(k) \cdot n^{2-\varepsilon}$ time FPT algorithms known
- Dynamic kernelization: After each update, output a kernel
  - Then can apply the best static algorithm on the kernel
- Observation: Dynamic XP not interesting
  - Sublinear time for every fixed $k \Rightarrow$ FPT algorithm

Dynamic vertex cover

Dynamic vertex cover

Assumptions:

- Initialized with an edgeless n-vertex graph G and a parameter k
- Support edge insertions/deletions and maintain whether $\text{vc}(G) \leq k$

Dynamic vertex cover

Assumptions:

- Initialized with an edgeless n-vertex graph G and a parameter k
- Support edge insertions/deletions and maintain whether $\text{vc}(G) \leq k$

Let's design an $f(k) \cdot \log n$ update time dynamic algorithm

Dynamic vertex cover

Assumptions:

- Initialized with an edgeless n-vertex graph G and a parameter k
- Support edge insertions/deletions and maintain whether $\text{vc}(G) \leq k$

Let's design an $f(k) \cdot \log n$ update time dynamic algorithm (think for a minute)

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$
  - Update changes the status of ≤ 2 vertices, a vertex whose status changes has $\leq k + 1$ neighbors, so $\leq 2(k + 1)$ edges updated

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$
  - Update changes the status of ≤ 2 vertices, a vertex whose status changes has $\leq k+1$ neighbors, so $\leq 2(k+1)$ edges updated
- Edges not incident to a vertex of degree $> k$ stored in a balanced binary search tree

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$
  - Update changes the status of ≤ 2 vertices, a vertex whose status changes has $\leq k + 1$ neighbors, so $\leq 2(k + 1)$ edges updated
- Edges not incident to a vertex of degree $> k$ stored in a balanced binary search tree

$\Rightarrow$ $\mathcal{O}(k \log n + k^2)$ update time to maintain kernel

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$
  - Update changes the status of ≤ 2 vertices, a vertex whose status changes has $\leq k+1$ neighbors, so $\leq 2(k+1)$ edges updated
- Edges not incident to a vertex of degree $> k$ stored in a balanced binary search tree

$\Rightarrow$ $\mathcal{O}(k \log n + k^2)$ update time to maintain kernel

$\Rightarrow$ $\mathcal{O}(k \log n + k^{\mathcal{O}(1)} f_{vc}(k))$ update time to decide whether $\mathrm{vc}(G) \leq k$, where $f_{vc}(k)$ the best FPT running time for vertex cover

Dynamic vertex cover: A solution

- Maintain a dynamic kernel $\Rightarrow$ run the best static FPT algorithm on it
- The $\mathcal{O}(k^2)$-edge kernel of Buss:
  1. All vertices of degree $> k$ must be in solution
  2. If yes-instance, at most k^2 edges not covered by them
- For each edge, maintain whether it is incident to a vertex of degree $> k$
  - Update changes the status of ≤ 2 vertices, a vertex whose status changes has $\leq k+1$ neighbors, so $\leq 2(k+1)$ edges updated
- Edges not incident to a vertex of degree $> k$ stored in a balanced binary search tree

$\Rightarrow$ $\mathcal{O}(k \log n + k^2)$ update time to maintain kernel

$\Rightarrow$ $\mathcal{O}(k \log n + k^{\mathcal{O}(1)} f_{vc}(k))$ update time to decide whether $\text{vc}(G) \leq k$, where $f_{vc}(k)$ the best FPT running time for vertex cover

- [Iwata & Oka '14]: $\mathcal{O}(k^2)$ update time $\mathcal{O}(k^2)$ kernel
- [Alman, Mnich & Vassilevska Williams '17]: $\mathcal{O}(1)$ amortized update time $\mathcal{O}(k^2)$ kernel

Outline

- Intro
- ⇒ Technical showcase: Dynamic Baker's scheme
- Overview of the literature
- Technical showcase: Dynamic treewidth
- Future of dynamic FPT and a mini open problem / brainstorming session

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

Baker's Scheme

Theorem (Baker '83)

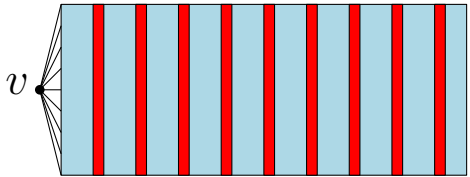
There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

- Pick a vertex v and do BFS-layering starting from v

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

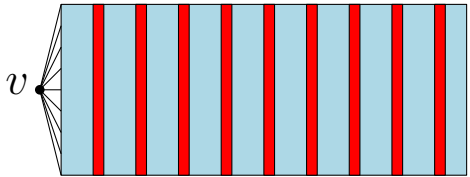


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

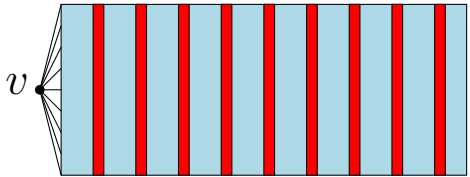


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

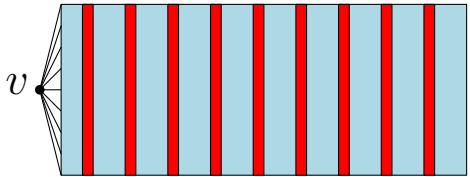


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$
- Try all offsets $\rightarrow$ For some offset, we delete $\leq 1/\varepsilon$ fraction of optimal solution

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

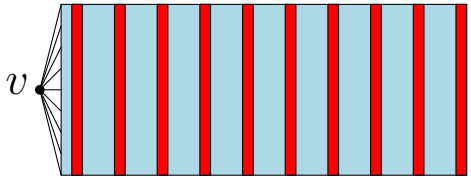


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$
- Try all offsets $\rightarrow$ For some offset, we delete $\leq 1/\varepsilon$ fraction of optimal solution

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

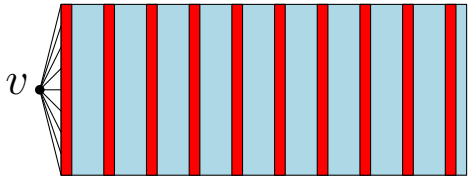


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$
- Try all offsets $\rightarrow$ For some offset, we delete $\leq 1/\varepsilon$ fraction of optimal solution

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.

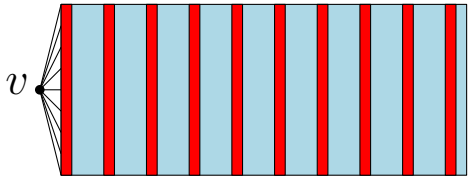


- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$
- Try all offsets $\rightarrow$ For some offset, we delete $\leq 1/\varepsilon$ fraction of optimal solution

Baker's Scheme

Theorem (Baker '83)

There is a $(1-\varepsilon)$-approximation for maximum weight independent set (MWIS) on planar graphs, with running time $2^{\mathcal{O}(1/\varepsilon)}n$.



- Pick a vertex v and do BFS-layering starting from v
- Choose $k = \lceil 1/\varepsilon \rceil$
- Delete every kth layer $\rightarrow$ treewidth $\mathcal{O}(k)$
- Try all offsets $\rightarrow$ For some offset, we delete $\leq 1/\varepsilon$ fraction of optimal solution
- Each can be solved exactly in time $2^{\mathcal{O}(k)}n = 2^{\mathcal{O}(1/\varepsilon)}n$ using treewidth

Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

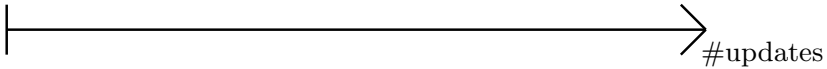
Technique: Periodical rebuilding

Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Technique: Periodical rebuilding

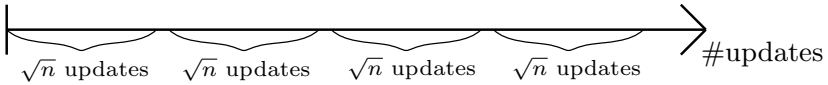


Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Technique: Periodical rebuilding

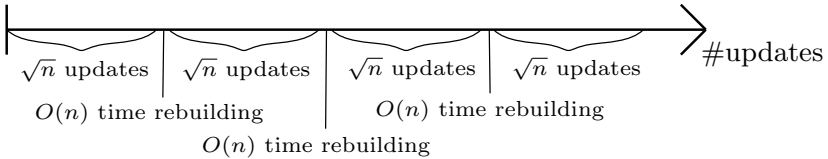


Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Technique: Periodical rebuilding

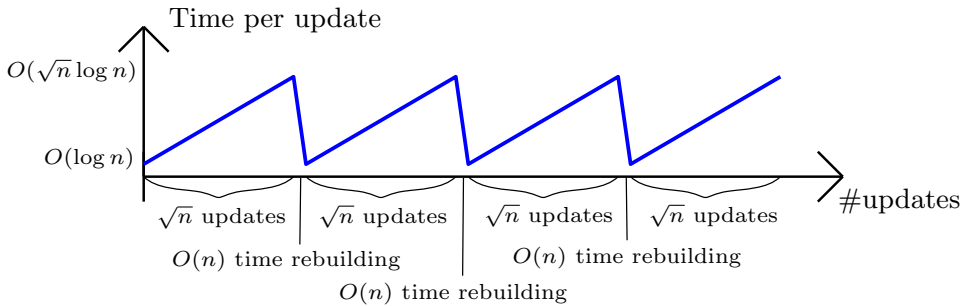


Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Technique: Periodical rebuilding

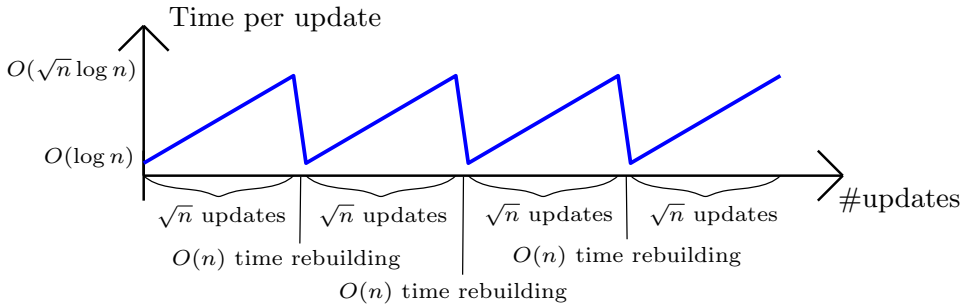


Dynamic Baker: Rebuilding technique

First goal: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

(edge insertions/deletions, vertex weight updates, maintain the value of solution)

Technique: Periodical rebuilding

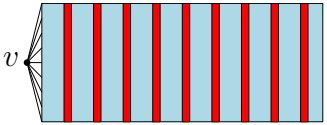


Suffices to design a data structure that:

- Can be initialized in $f(\varepsilon)n$ time
- Has update time $f(\varepsilon) \cdot u \log n$, where u is the number of updates so far

Dynamic Baker: Initialization

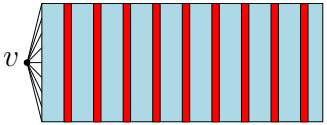
Initialization: BFS-layering to compute $k = \lceil 1/\varepsilon \rceil$ vertex sets $V_1, V_2, \ldots, V_k \subseteq V(G)$ so that



Dynamic Baker: Initialization

Initialization: BFS-layering to compute $k = \lceil 1/\varepsilon \rceil$ vertex sets $V_1, V_2, \ldots, V_k \subseteq V(G)$ so that

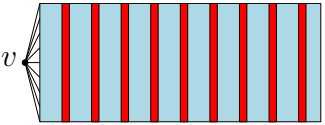
- Initially $\mathrm{tw}(G[V_i]) \leq \mathcal{O}(k)$



Dynamic Baker: Initialization

Initialization: BFS-layering to compute $k = \lceil 1/\varepsilon \rceil$ vertex sets $V_1, V_2, \ldots, V_k \subseteq V(G)$ so that

- Initially $\mathsf{tw}(G[V_i]) \leq \mathcal{O}(k)$
- After arbitrary edge updates $\exists i$ s.t. $\mathsf{OPT}(G[V_i]) \geq (1-\varepsilon) \cdot \mathsf{OPT}(G)$

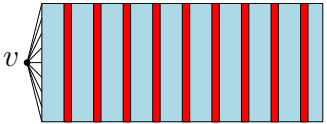


Dynamic Baker: Initialization

Initialization: BFS-layering to compute $k = \lceil 1/\varepsilon \rceil$ vertex sets $V_1, V_2, \ldots, V_k \subseteq V(G)$ so that

- Initially $\mathrm{tw}(G[V_i]) \leq \mathcal{O}(k)$
- After arbitrary edge updates $\exists i$ s.t. $\mathrm{OPT}(G[V_i]) \geq (1-\varepsilon) \cdot \mathrm{OPT}(G)$

$\Rightarrow$ After initialization, for each V_i a separate data structure that handles the induced subgraph $G[V_i]$

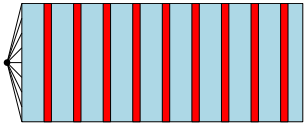


Dynamic Baker: Initialization

Initialization: BFS-layering to compute $k = \lceil 1/\varepsilon \rceil$ vertex sets $V_1, V_2, \ldots, V_k \subseteq V(G)$ so that

- Initially $\mathrm{tw}(G[V_i]) \leq \mathcal{O}(k)$
- After arbitrary edge updates $\exists i$ s.t. $\mathrm{OPT}(G[V_i]) \geq (1-\varepsilon) \cdot \mathrm{OPT}(G)$

$\Rightarrow$ After initialization, for each V_i a separate data structure that handles the induced subgraph $G[V_i]$



Suffices to design data structure that:

- Is initialized with a weighted planar graph of treewidth $\mathcal{O}(k)$ and a parameter ε in $f(k,\varepsilon) \cdot n$ time
- Maintains $(1-\varepsilon)$-approximation of MWIS (assuming the graph remains planar)
- Has update time $f(k,\varepsilon) \cdot u \log n$, where u the number of updates so far

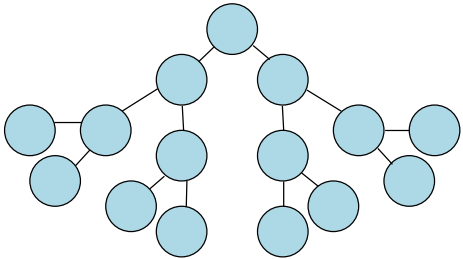
Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

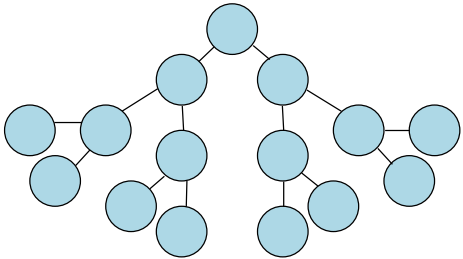
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

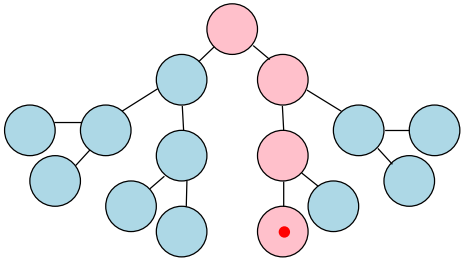
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

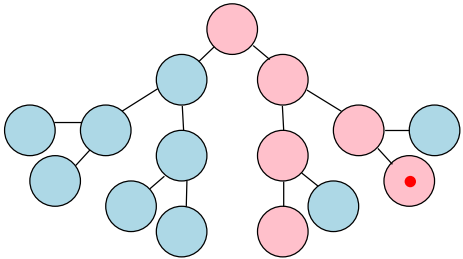
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

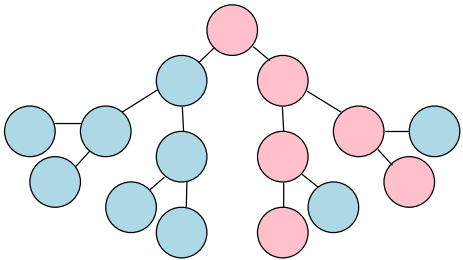
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

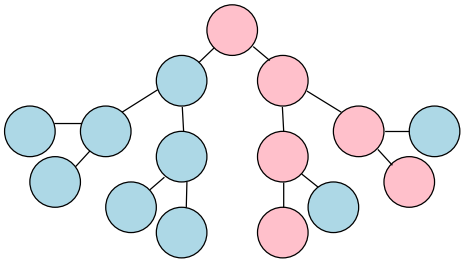
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors
  - After u updates, $\mathcal{O}(u \cdot \log n)$ bags marked



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

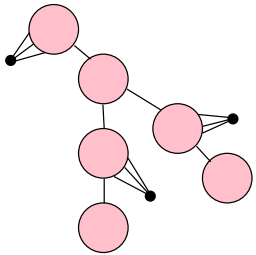
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors
  - After u updates, $\mathcal{O}(u \cdot \log n)$ bags marked
- Approximating MWIS:



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

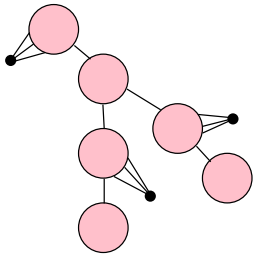
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors
  - After u updates, $\mathcal{O}(u \cdot \log n)$ bags marked
- Approximating MWIS:
  1. Compress unmarked subtrees into single vertices using pre-computed DP-tables, maintain planarity



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

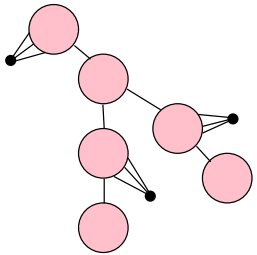
- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors
  - After u updates, $\mathcal{O}(u \cdot \log n)$ bags marked
- Approximating MWIS:
  1. Compress unmarked subtrees into single vertices using pre-computed DP-tables, maintain planarity
  2. Apply static Baker on the resulting graph with $\mathcal{O}(u \cdot \log n)$ vertices



Dynamic Baker: The main idea

Given: Planar graph of treewidth $\mathcal{O}(k)$

- Initialization: Compute a binary tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$
- Update: Mark the home bags of the involved vertices and their ancestors
  - After u updates, $\mathcal{O}(u \cdot \log n)$ bags marked
- Approximating MWIS:
  1. Compress unmarked subtrees into single vertices using pre-computed DP-tables, maintain planarity
  2. Apply static Baker on the resulting graph with $\mathcal{O}(u \cdot \log n)$ vertices
     $\Rightarrow$ $f(k, \varepsilon) \cdot u \cdot \log n$ update time



Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Can we improve?

Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Can we improve?

- Idea: Instead of static Baker, apply another dynamic Baker:

Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Can we improve?

- Idea: Instead of static Baker, apply another dynamic Baker:
  $\Rightarrow$ $f(k, \varepsilon) \cdot \sqrt{u \log n} \log n$ update time

Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

Can we improve?

- Idea: Instead of static Baker, apply another dynamic Baker:
  - $\Rightarrow$ $f(k, \varepsilon) \cdot \sqrt{u \log n} \log n$ update time
  - $\Rightarrow$ rebuilding frequency $n^{2/3} \Rightarrow f(\varepsilon) \cdot n^{1/3} \log^{3/2} n$ update time

Dynamic Baker: Going deeper

Have: $f(\varepsilon) \cdot \sqrt{n} \log n$ amortized update time

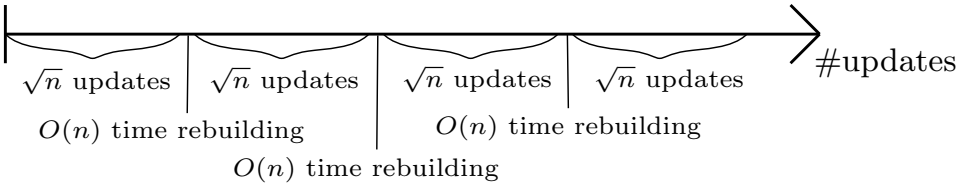
Can we improve?

- Idea: Instead of static Baker, apply another dynamic Baker:
  - $\Rightarrow$ $f(k, \varepsilon) \cdot \sqrt{u \log n} \log n$ update time
  - $\Rightarrow$ rebuilding frequency $n^{2/3} \Rightarrow f(\varepsilon) \cdot n^{1/3} \log^{3/2} n$ update time

- $\approx \log \log n$ levels of recursion $\Rightarrow f(\varepsilon) \cdot n^{o(1)}$ update time [K., Nadara, Pilipczuk & Sokołowski '24]

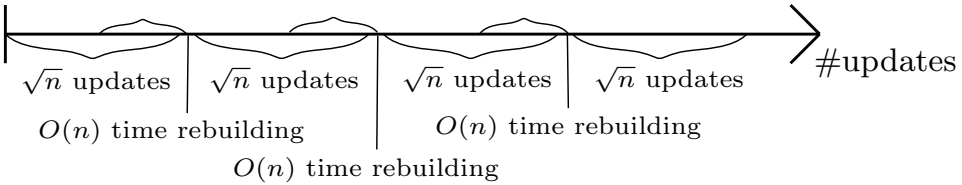
Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*



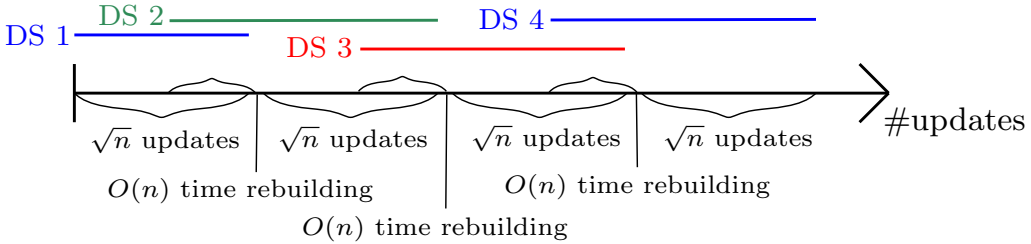
Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*
- Distribute the $\mathcal{O}(n)$-time rebuilding over multiple updates



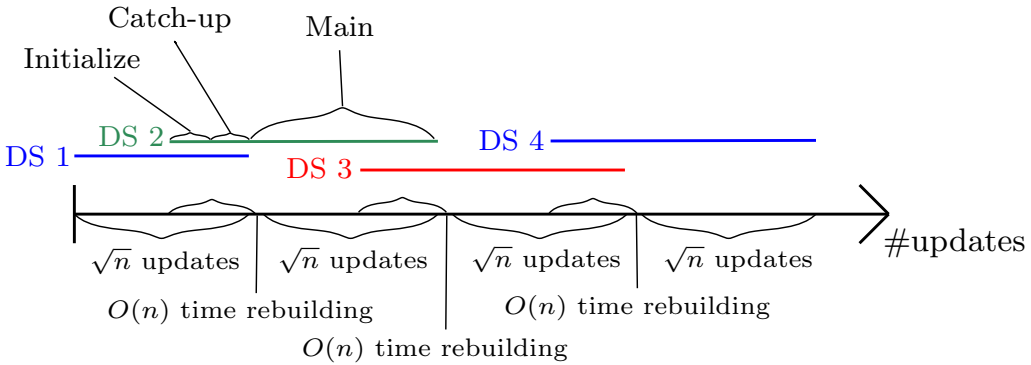
Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*
- Distribute the $\mathcal{O}(n)$-time rebuilding over multiple updates



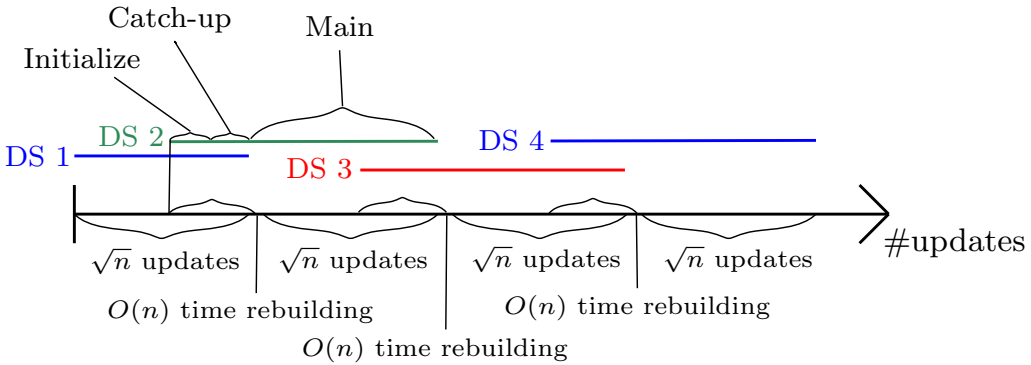
Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*
- Distribute the $\mathcal{O}(n)$-time rebuilding over multiple updates



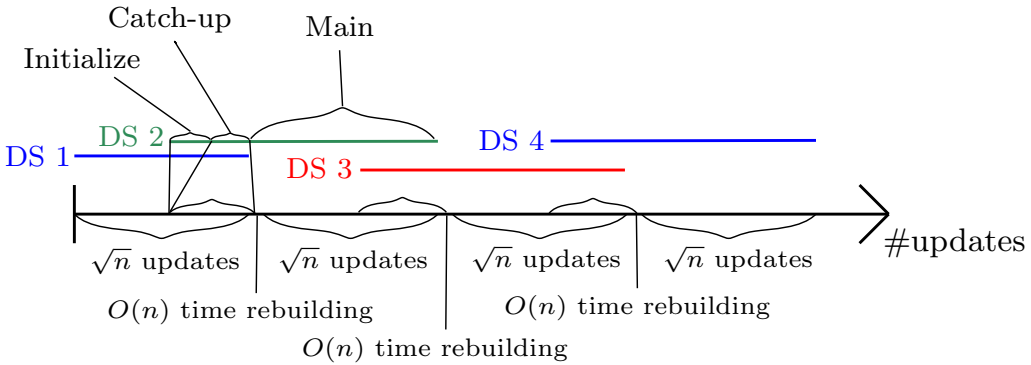
Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*
- Distribute the $\mathcal{O}(n)$-time rebuilding over multiple updates



Dynamic Baker: De-amortization

- Standard trick to de-amortize rebuilding, called *background rebuilding*
- Distribute the $\mathcal{O}(n)$-time rebuilding over multiple updates



Dynamic Baker: Conclusion

Theorem (K., Nadara, Pilipczuk & Sokołowski '24)

There is a dynamic $(1-\varepsilon)$-approximation algorithm for MWIS on planar graphs, supporting edge insertions and deletions, and vertex weight-updates, with worst-case update time $f(\varepsilon) \cdot n^{o(1)}$.

Dynamic Baker: Conclusion

Theorem (K., Nadara, Pilipczuk & Sokołowski '24)

There is a dynamic $(1-\varepsilon)$-approximation algorithm for MWIS on planar graphs, supporting edge insertions and deletions, and vertex weight-updates, with worst-case update time $f(\varepsilon) \cdot n^{o(1)}$.

Lot of room for extensions/improvements:

Dynamic Baker: Conclusion

Theorem (K., Nadara, Pilipczuk & Sokołowski '24)

There is a dynamic $(1-\varepsilon)$-approximation algorithm for MWIS on planar graphs, supporting edge insertions and deletions, and vertex weight-updates, with worst-case update time $f(\varepsilon) \cdot n^{o(1)}$.

Lot of room for extensions/improvements:

- We tried the same for dominating set, but only managed in bounded-degree graphs...

Dynamic Baker: Conclusion

Theorem (K., Nadara, Pilipczuk & Sokołowski '24)

There is a dynamic $(1-\varepsilon)$-approximation algorithm for MWIS on planar graphs, supporting edge insertions and deletions, and vertex weight-updates, with worst-case update time $f(\varepsilon) \cdot n^{o(1)}$.

Lot of room for extensions/improvements:

- We tried the same for dominating set, but only managed in bounded-degree graphs...
- Good model for outputting the solution?

Dynamic Baker: Conclusion

Theorem (K., Nadara, Pilipczuk & Sokołowski '24)

There is a dynamic $(1-\varepsilon)$-approximation algorithm for MWIS on planar graphs, supporting edge insertions and deletions, and vertex weight-updates, with worst-case update time $f(\varepsilon) \cdot n^{o(1)}$.

Lot of room for extensions/improvements:

- We tried the same for dominating set, but only managed in bounded-degree graphs...
- Good model for outputting the solution?
- $f(\varepsilon) \cdot \operatorname{polylog} n$ update time?

Outline

- Intro
- Technical showcase: Dynamic Baker's scheme

⇒ Overview of the literature

- Technical showcase: Dynamic treewidth
- Future of dynamic FPT and a mini open problem / brainstorming session

Literature review

Literature review

- [Iwata, Oka '14], [Alman, Mnich, Vassilevska Williams '17]: Vertex cover, FVS, k-path, etc.
- [Olkowski, Pilipczuk, Rychlicki, Wegrzycki, Zych-Pawlewicz '23]: String problems
- [Majewski, Pilipczuk, Zych-Pawlewicz '24]: Split graph completion
- [Zych-Pawlewicz, Żochowski '24]: FAST, FVST
- [An, Cho, Jang, Jung, Lee, Oh, Shin, Shin, Song '24]: Problems on unit disk graphs

Literature review

- [Iwata, Oka '14], [Alman, Mnich, Vassilevska Williams '17]: Vertex cover, FVS, k-path, etc.
- [Olkowski, Pilipczuk, Rychlicki, Wegrzycki, Zych-Pawlewicz '23]: String problems
- [Majewski, Pilipczuk, Zych-Pawlewicz '24]: Split graph completion
- [Zych-Pawlewicz, Żochowski '24]: FAST, FVST
- [An, Cho, Jang, Jung, Lee, Oh, Shin, Shin, Song '24]: Problems on unit disk graphs

Structural (excluding dynamic treewidth):

Literature review

- [Iwata, Oka '14], [Alman, Mnich, Vassilevska Williams '17]: Vertex cover, FVS, k-path, etc.
- [Olkowski, Pilipczuk, Rychlicki, Wegrzycki, Zych-Pawlewicz '23]: String problems
- [Majewski, Pilipczuk, Zych-Pawlewicz '24]: Split graph completion
- [Zych-Pawlewicz, Żochowski '24]: FAST, FVST
- [An, Cho, Jang, Jung, Lee, Oh, Shin, Shin, Song '24]: Problems on unit disk graphs

Structural (excluding dynamic treewidth):

- [Bodlaender '93]: Treewidth-2 and Courcelle in that setting
- [Dvorak, Kupec & Tuma '14], [Chen, Czerwinski, Disser, Feldmann, Hermelin, Nadara, Pilipczuk, Pilipczuk, Sorge, Wroblewski & Zych-Pawlewicz '21]: Treedepth, with Courcelle
- [Majewski, Pilipczuk & Sokolowski '23]: FVS with Courcelle
- [Dvorak & Tuma '13]: Counting induced subgraphs on bounded-expansion and nowhere dense graphs

Dynamic treewidth: Literature

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs
- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem
- [K. '25]: $2^{\mathcal{O}(k)} \log n$ amortized update time, 9-approximate

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem

- [K. '25]: $2^{\mathcal{O}(k)} \log n$ amortized update time, 9-approximate
  - Based on a graph-theoretical insight that allows classical tree-balancing techniques (from splay-trees) to be applied

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem

- [K. '25]: $2^{\mathcal{O}(k)} \log n$ amortized update time, 9-approximate
  - Based on a graph-theoretical insight that allows classical tree-balancing techniques (from splay-trees) to be applied
  - $\mathcal{O}(\log n)$ optimal for dynamic Courcelle [Patrascu & Demaine '05]

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs
- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem
- [K. '25]: $2^{\mathcal{O}(k)} \log n$ amortized update time, 9-approximate
  - Based on a graph-theoretical insight that allows classical tree-balancing techniques (from splay-trees) to be applied
  - $\mathcal{O}(\log n)$ optimal for dynamic Courcelle [Patrascu & Demaine '05]
- [Bertram, Holm, Jensen, K. '26+]: $2^{\mathcal{O}(k)} \log^2 n$ worst-case update time, 9-approximate

Dynamic treewidth: Literature

- [Goranci, Räcke, Saranurak, Tan '21]: $n^{o(1)}$ amortized update time, $\Delta n^{o(1)}$-approximate, for graphs of max degree Δ
  - Directly from the expander hierarchy of [Goranci, Räcke, Saranurak, Tan '21]
  - No dynamic Courcelle even for bounded-degree graphs

- [K., Majewski, Nadara, Pilipczuk & Sokołowski '23]: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time, 6-approximate
  - Based on the local improvement technique developed in [K. '21], [K. & Lokshtanov '23]
  - Gives dynamic Courcelle's theorem

- [K. '25]: $2^{\mathcal{O}(k)} \log n$ amortized update time, 9-approximate
  - Based on a graph-theoretical insight that allows classical tree-balancing techniques (from splay-trees) to be applied
  - $\mathcal{O}(\log n)$ optimal for dynamic Courcelle [Patrascu & Demaine '05]

- [Bertram, Holm, Jensen, K. '26+]: $2^{\mathcal{O}(k)} \log^2 n$ worst-case update time, 9-approximate
  - Version of [K. '25] with different tree-balancing technique

Dynamic treewidth: Applications

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time
  - Main application: Static rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time
  - Main application: Static rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$
- [K., Pilipczuk & Stamoulis '24]: static k-disjoint paths and H-minor-testing in $f(k) \cdot n^{1+o(1)}$ time

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time
  - Main application: Static rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$
- [K., Pilipczuk & Stamoulis '24]: static k-disjoint paths and H-minor-testing in $f(k) \cdot n^{1+o(1)}$ time
  - Fast implementation of the irrelevant vertex technique using dynamic treewidth

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time
  - Main application: Static rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$
- [K., Pilipczuk & Stamoulis '24]: static k-disjoint paths and H-minor-testing in $f(k) \cdot n^{1+o(1)}$ time
  - Fast implementation of the irrelevant vertex technique using dynamic treewidth
- [Bertram, Haun, Jensen, K. '26]: Dynamic version of the meta-kernelization framework of [Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh & Thilikos '09]

Dynamic treewidth: Applications

- [K. & Sokołowski '24]: Dynamic rankwidth with $f(k) \cdot n^{o(1)}$ amortized update time
  - Main application: Static rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$
- [K., Pilipczuk & Stamoulis '24]: static k-disjoint paths and H-minor-testing in $f(k) \cdot n^{1+o(1)}$ time
  - Fast implementation of the irrelevant vertex technique using dynamic treewidth
- [Bertram, Haun, Jensen, K. '26]: Dynamic version of the meta-kernelization framework of [Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh & Thilikos '09]
  - Linear kernels on minor-free graphs, with amortized update time $\mathcal{O}(\log n)$, and worst-case $\mathcal{O}(1)$ updates to the kernel per update to the graph

Outline

- Intro
- Technical showcase: Dynamic Baker's scheme
- Overview of the literature
- ⇒ Technical showcase: Dynamic treewidth
- Future of dynamic FPT and a mini open problem / brainstorming session

Dynamic treewidth: Theorem statement

Theorem:

Dynamic treewidth: Theorem statement

Theorem:

There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\mathrm{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \mathrm{tw}(G) + 8$

Dynamic treewidth: Theorem statement

Theorem:

There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\mathrm{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \mathrm{tw}(G) + 8$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree decomposition automata)

Dynamic treewidth: Theorem statement

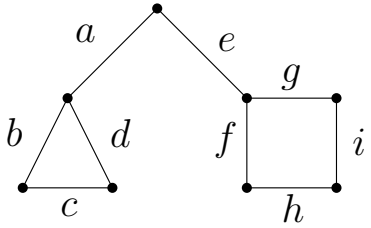
Theorem:

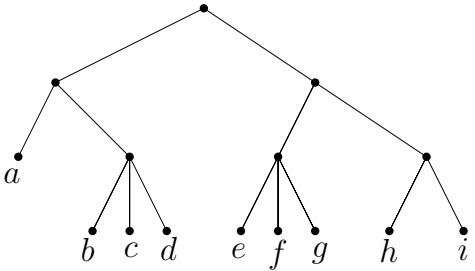
There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\mathrm{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \mathrm{tw}(G) + 8$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree decomposition automata)

$\Rightarrow$ Dynamic Courcelle's theorem in $f(k) \cdot \log n$ amortized update time

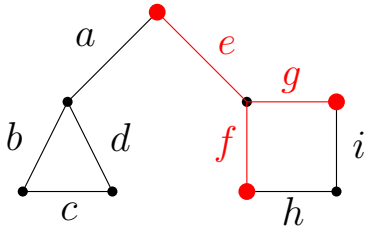
Dynamic treewidth: Maintained decomposition

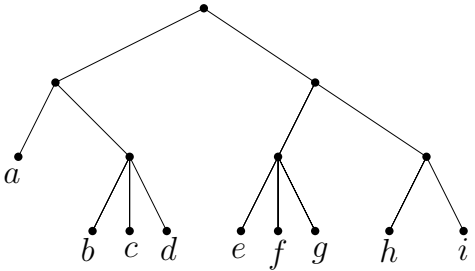




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph

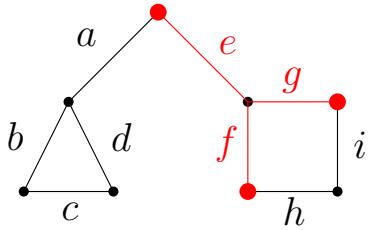
Dynamic treewidth: Maintained decomposition

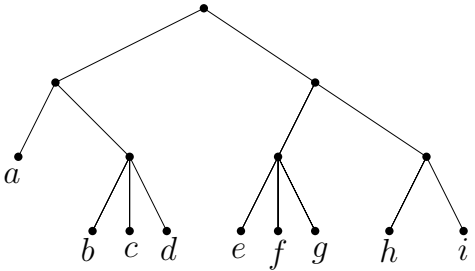




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.

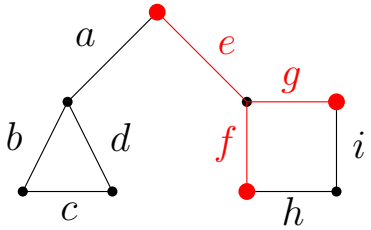
Dynamic treewidth: Maintained decomposition

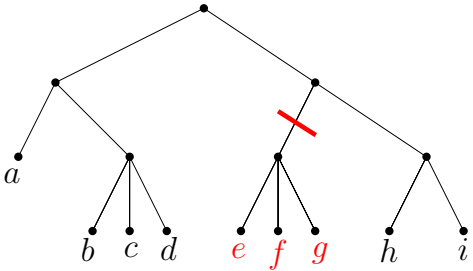




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$

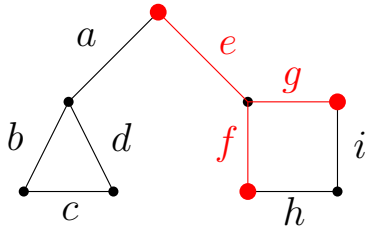
Dynamic treewidth: Maintained decomposition

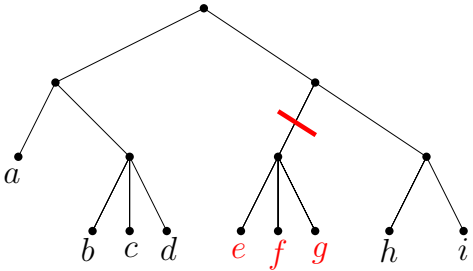




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"

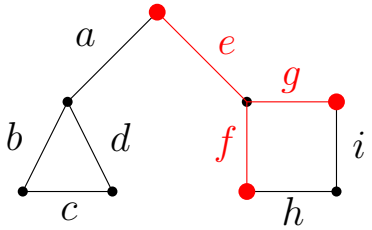
Dynamic treewidth: Maintained decomposition

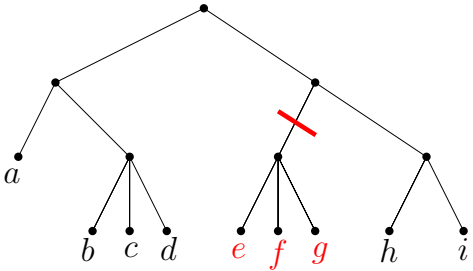




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$

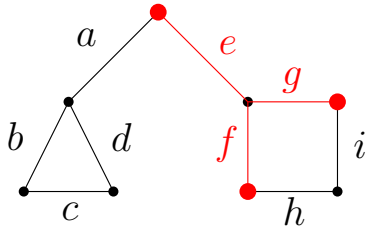
Dynamic treewidth: Maintained decomposition

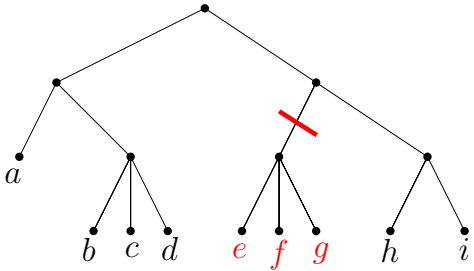




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$

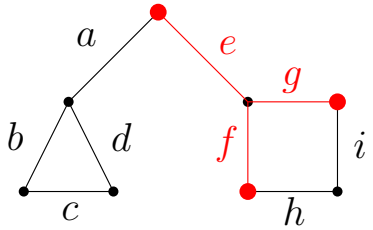
Dynamic treewidth: Maintained decomposition

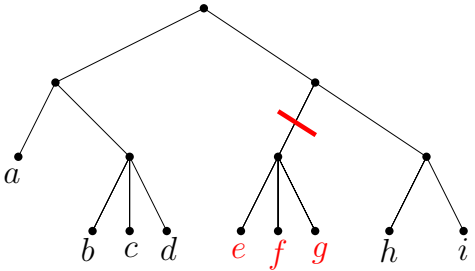




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$
  $\Rightarrow$ Corresponds to a tree decomposition of width $2^{\mathcal{O}(k)}$ (later optimize to $\mathcal{O}(k)$)

Dynamic treewidth: Maintained decomposition



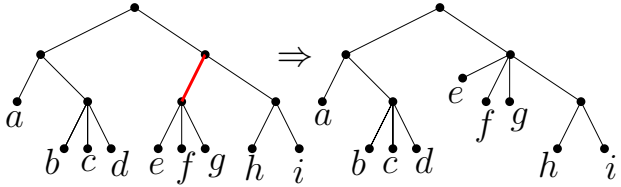


- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$
  $\Rightarrow$ Corresponds to a tree decomposition of width $2^{\mathcal{O}(k)}$ (later optimize to $\mathcal{O}(k)$)
- Depth at most $2^{\mathcal{O}(k)} \log n$

Dynamic treewidth: Local rotations

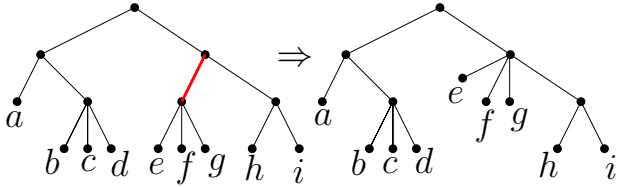
Dynamic treewidth: Local rotations

1. **Contraction**: Given an edge uv of the decomposition, contract it.



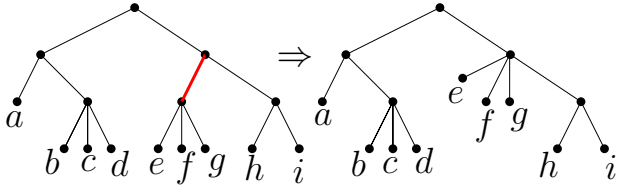
Dynamic treewidth: Local rotations

1. **Contraction**: Given an edge uv of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree

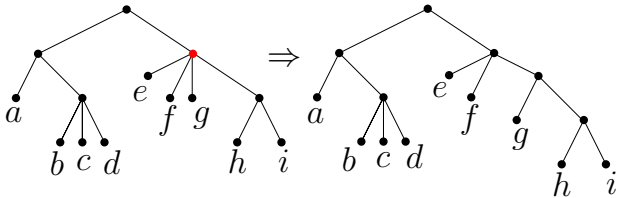


Dynamic treewidth: Local rotations

1. **Contraction**: Given an edge uv of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree

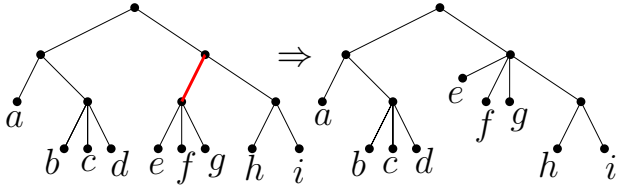


2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes

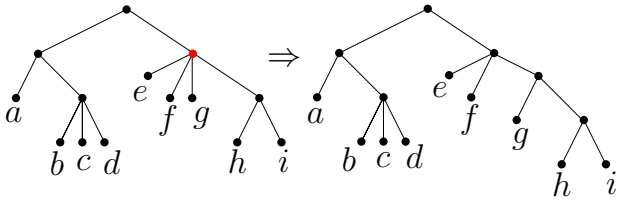


Dynamic treewidth: Local rotations

1. **Contraction**: Given an edge uv of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree

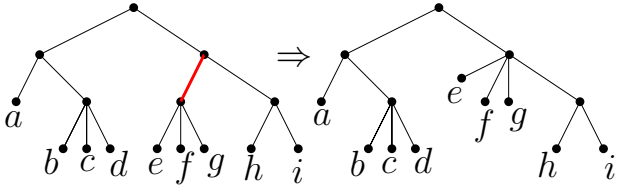


2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes
   - Lemma: Can be done so that downwards well-linkedness is maintained

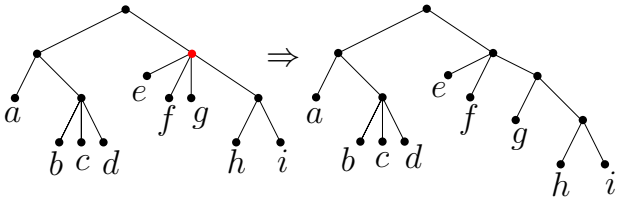


Dynamic treewidth: Local rotations

1. **Contraction**: Given an edge uv of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree



2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes
   - Lemma: Can be done so that downwards well-linkedness is maintained
   - Can choose a set of ≤ 3 children that will not drop deeper



Dynamic treewidth: Inserting and deleting edges

Dynamic treewidth: Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex

Dynamic treewidth: Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:

Dynamic treewidth: Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update

Dynamic treewidth: Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update
- To delete an edge uv:

Dynamic treewidth: Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update
- To delete an edge uv:
  1. Rotate uv and the self-loops u and v to be children of the root
  2. Delete uv
  3. Decomposition is easy to update

Dynamic treewidth: Controlling the depth

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)/2.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then $\text{size}(b) < \text{size}(a)/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Lemma

If a pair a, b does not satisfy the invariant, then can decrease the value of $\Phi(T)$ in time proportional to the decrease

Dynamic treewidth: Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)/2.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Lemma

If a pair a, b does not satisfy the invariant, then can decrease the value of $\Phi(T)$ in time proportional to the decrease

Lemma

Edge insertion and deletion increase $\Phi(T)$ by $2^{\mathcal{O}(k)} \log n$

Outline

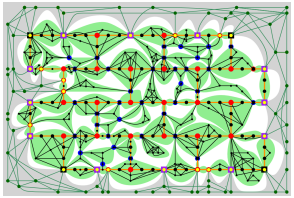
- Intro
- Technical showcase: Dynamic Baker's scheme
- Overview of the literature
- Technical showcase: Dynamic treewidth

$\Rightarrow$ Future of dynamic FPT and a mini open problem / brainstorming session

Dynamic irrelevant vertex technique

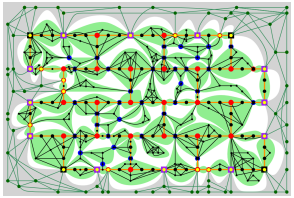
Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution



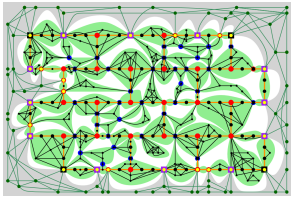
Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors



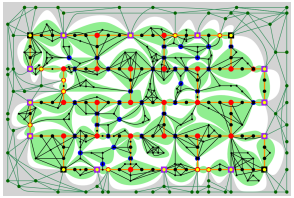
Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth



Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth

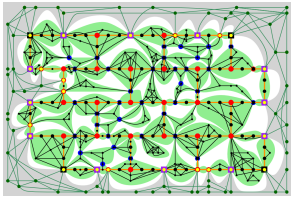


Question

Dynamic implementation of the irrelevant vertex technique?

Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth



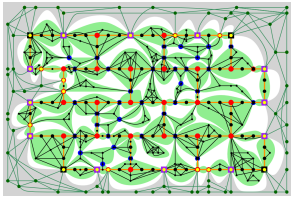
Question

Dynamic implementation of the irrelevant vertex technique?

- Dynamic k-Disjoint Paths on planar graphs? $f(k) \cdot \log n$, or $f(k) \cdot n^{o(1)}$, or $f(k) \cdot \sqrt{n}$ update time?

Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth



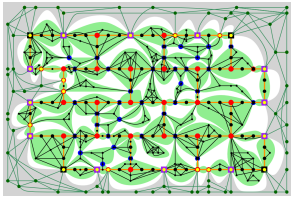
Question

Dynamic implementation of the irrelevant vertex technique?

- Dynamic k-Disjoint Paths on planar graphs? $f(k) \cdot \log n$, or $f(k) \cdot n^{o(1)}$, or $f(k) \cdot \sqrt{n}$ update time?
  - Seems challenging even in the simplest settings, i.e., fixed embedding and fixed terminals

Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth



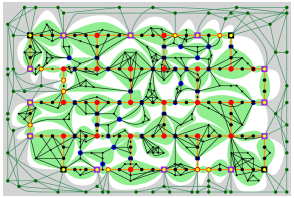
Question

Dynamic implementation of the irrelevant vertex technique?

- Dynamic k-Disjoint Paths on planar graphs? $f(k) \cdot \log n$, or $f(k) \cdot n^{o(1)}$, or $f(k) \cdot \sqrt{n}$ update time?
  - Seems challenging even in the simplest settings, i.e., fixed embedding and fixed terminals
- Dynamic H-minor-containment on general graphs?

Dynamic irrelevant vertex technique

- If a graph has high treewidth, we can find a vertex that can be removed without changing the solution
- Introduced by [Robertson & Seymour, Graph Minors XIII], used in various problems related to graph minors
- [K. Pilipczuk, Stamoulis '24]: Can be implemented in $f(k) \cdot n \log n$ time on apex-minor-free graphs using dynamic treewidth



Question

Dynamic implementation of the irrelevant vertex technique?

- Dynamic k-Disjoint Paths on planar graphs? $f(k) \cdot \log n$, or $f(k) \cdot n^{o(1)}$, or $f(k) \cdot \sqrt{n}$ update time?
  - Seems challenging even in the simplest settings, i.e., fixed embedding and fixed terminals
- Dynamic H-minor-containment on general graphs?
- Your favorite problem whose solution requires irrelevant vertex?

Parameterized cell-probe lower bounds

Parameterized cell-probe lower bounds

- Cell-probe lower bounds: Framework for unconditional lower bounds for (amortized) update times

Parameterized cell-probe lower bounds

- Cell-probe lower bounds: Framework for unconditional lower bounds for (amortized) update times
- Examples:
  - $\Omega(\log n / \log \log n)$ for prefix sums mod-2 [Fredman & Saks '89]
  - $\Omega(\log n)$ for prefix sums [Patrascu & Demaine '05]
  - $\Omega(\log n)$ for connectivity on forests [Patrascu & Demaine '05]
  - $\Omega(\log^2 n / \log^2 \log n)$ for 2D range queries [Green Larsen '11]

Parameterized cell-probe lower bounds

- Cell-probe lower bounds: Framework for unconditional lower bounds for (amortized) update times
- Examples:
  - $\Omega(\log n / \log \log n)$ for prefix sums mod-2 [Fredman & Saks '89]
  - $\Omega(\log n)$ for prefix sums [Patrascu & Demaine '05]
  - $\Omega(\log n)$ for connectivity on forests [Patrascu & Demaine '05]
  - $\Omega(\log^2 n / \log^2 \log n)$ for 2D range queries [Green Larsen '11]

Question

Interesting lower bounds in the dynamic FPT setting?

Parameterized cell-probe lower bounds

- Cell-probe lower bounds: Framework for unconditional lower bounds for (amortized) update times
- Examples:
  - $\Omega(\log n / \log \log n)$ for prefix sums mod-2 [Fredman & Saks '89]
  - $\Omega(\log n)$ for prefix sums [Patrascu & Demaine '05]
  - $\Omega(\log n)$ for connectivity on forests [Patrascu & Demaine '05]
  - $\Omega(\log^2 n / \log^2 \log n)$ for 2D range queries [Green Larsen '11]

Question

Interesting lower bounds in the dynamic FPT setting?

- [Patrascu '11] implies that no $f(k)$ update time dynamic algorithm for DFVS

Parameterized cell-probe lower bounds

- Cell-probe lower bounds: Framework for unconditional lower bounds for (amortized) update times
- Examples:
  - $\Omega(\log n / \log \log n)$ for prefix sums mod-2 [Fredman & Saks '89]
  - $\Omega(\log n)$ for prefix sums [Patrascu & Demaine '05]
  - $\Omega(\log n)$ for connectivity on forests [Patrascu & Demaine '05]
  - $\Omega(\log^2 n / \log^2 \log n)$ for 2D range queries [Green Larsen '11]

Question

Interesting lower bounds in the dynamic FPT setting?

- [Patrascu '11] implies that no $f(k)$ update time dynamic algorithm for DFVS
- Can we separate $f(k) + \mathcal{O}(\log n)$ from $f(k) \cdot \log n$?

Open problems/questions/directions/ideas?

Conclusion

Techniques and tools:

Conclusion

Techniques and tools:

- Dynamic kernelization
- Rebuilding
- Dynamic treewidth

Conclusion

Techniques and tools:

- Dynamic kernelization
- Rebuilding
- Dynamic treewidth

Future directions:

Conclusion

Techniques and tools:

- Dynamic kernelization
- Rebuilding
- Dynamic treewidth

Future directions:

- Dynamic irrelevant vertex technique?
- Parameterized cell-probe lower bounds?
- Dynamic FPT and structural graph theory?

Conclusion

Techniques and tools:

- Dynamic kernelization
- Rebuilding
- Dynamic treewidth

Future directions:

- Dynamic irrelevant vertex technique?
- Parameterized cell-probe lower bounds?
- Dynamic FPT and structural graph theory?

Thank you!

Dynamic FPT and structural graph theory

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

- Minor-closed class with polynomial algorithms [Robertson & Seymour '86]
- Decidability of monadic second-order logic (MSO_2) [Seese '91]
- Polynomial algorithms for CSP [Grohe, Schwentick & Segoufin '01], [Marx '07]

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

- Minor-closed class with polynomial algorithms [Robertson & Seymour '86]
- Decidability of monadic second-order logic (MSO_2) [Seese '91]
- Polynomial algorithms for CSP [Grohe, Schwentick & Segoufin '01], [Marx '07]

Question

Similar role of graph structure in the dynamic setting?

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

- Minor-closed class with polynomial algorithms [Robertson & Seymour '86]
- Decidability of monadic second-order logic (MSO_2) [Seese '91]
- Polynomial algorithms for CSP [Grohe, Schwentick & Segoufin '01], [Marx '07]

Question

Similar role of graph structure in the dynamic setting?

- Bounded treewidth/rankwidth the most general class with $\mathcal{O}(\log n)$ update time?

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

- Minor-closed class with polynomial algorithms [Robertson & Seymour '86]
- Decidability of monadic second-order logic (MSO_2) [Seese '91]
- Polynomial algorithms for CSP [Grohe, Schwentick & Segoufin '01], [Marx '07]

Question

Similar role of graph structure in the dynamic setting?

- Bounded treewidth/rankwidth the most general class with $\mathcal{O}(\log n)$ update time?
- Bounded treedepth/shrubdepth/rankdepth the most general class with $\mathcal{O}(1)$ update time?

Dynamic FPT and structural graph theory

Results saying bounded treewidth is the most general graph class for something:

- Minor-closed class with polynomial algorithms [Robertson & Seymour '86]
- Decidability of monadic second-order logic (MSO_2) [Seese '91]
- Polynomial algorithms for CSP [Grohe, Schwentick & Segoufin '01], [Marx '07]

Question

Similar role of graph structure in the dynamic setting?

- Bounded treewidth/rankwidth the most general class with $\mathcal{O}(\log n)$ update time?
- Bounded treedepth/shrubdepth/rankdepth the most general class with $\mathcal{O}(1)$ update time?
- Bounded expansion?