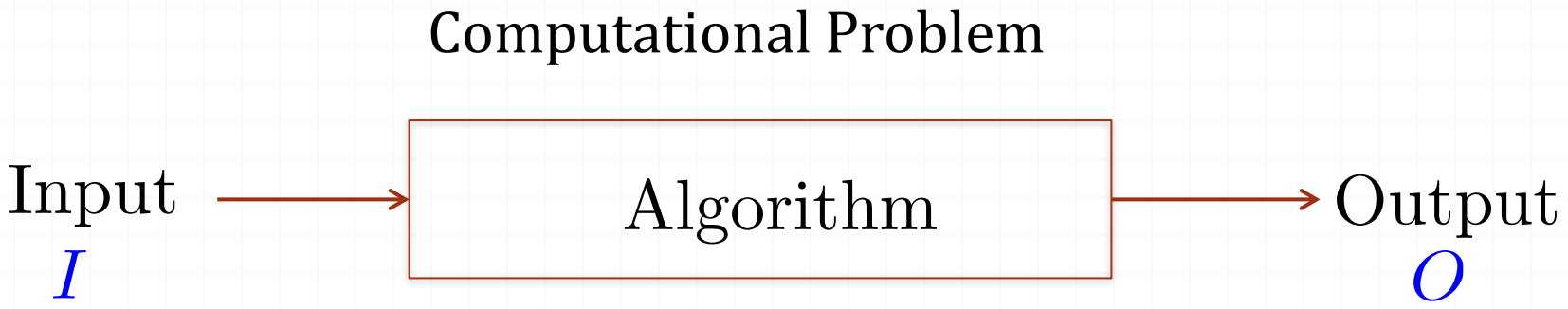
A survey of dynamic graph algorithms

Monika Henzinger
University of Vienna

universität wien

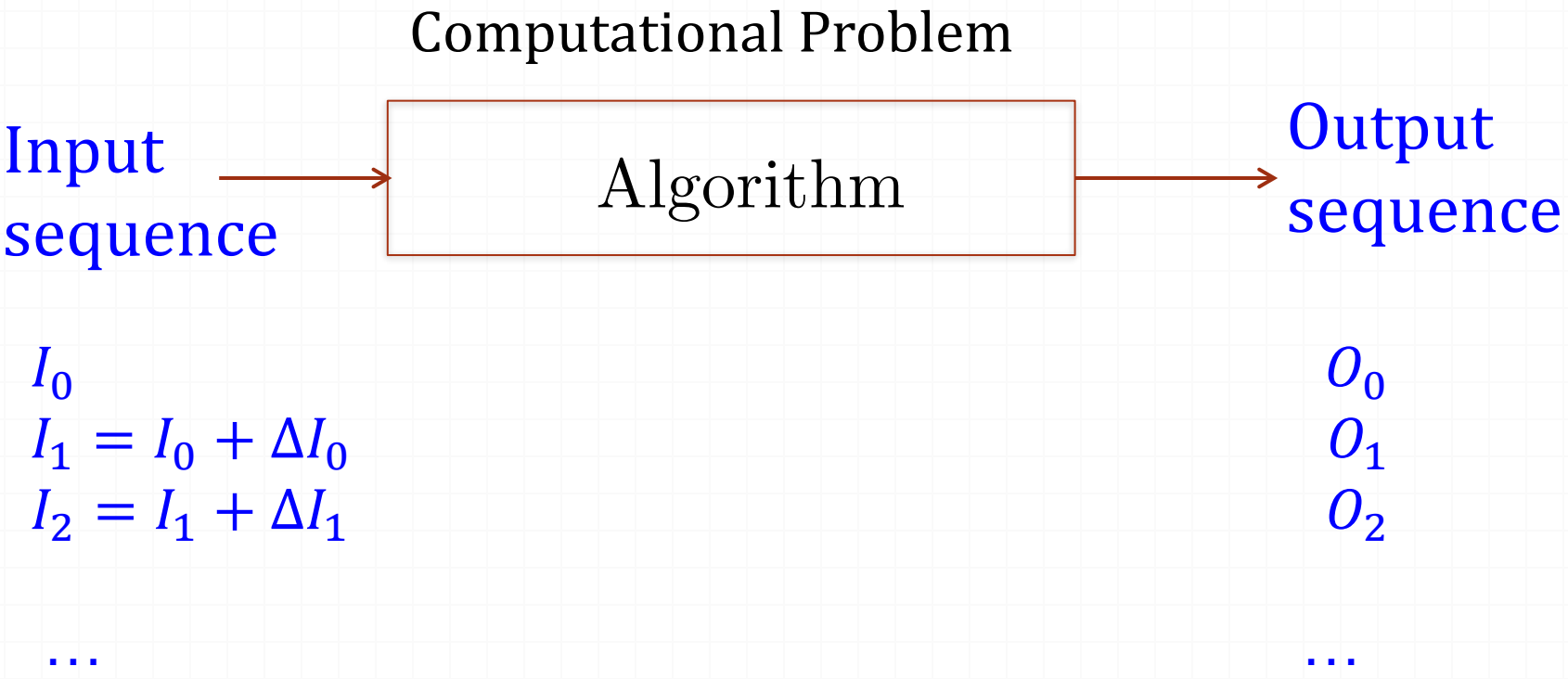
Static Algorithm



Performance measures:

1. Running Time
2. Space
3. Approximation ratio

What is a Dynamic Algorithm?



Do we need to recompute the output from scratch?

What is a *(Fully) Dynamic Graph Algorithm*?

Computational graph problem

Operations:

- Initialize(G)
- InsertEdge(u, v, *weight*)
- DeleteEdge(u,v)
- InsertNode(u)
- DeleteNode(u)
- Query(G) or Query(u) or Query(u, v)

What is a *(Fully) Dynamic Graph Algorithm*?

Computational graph problem

Operations:

- Initialize(G) — Preprocessing time
- InsertEdge(u, v, *weight*)
- DeleteEdge(u,v)
- (InsertNode(u))
- (DeleteNode(u))

  → Update time
- Query(G) or Query(u) or Query(u, v) — Query time

Input generation: Adversary Models

How is the input, i.e. the sequence of operations, generated?

Adversary Models

Adaptive adversary:

- knows the algorithm (but not its random choices)
- sees all the answers to the previous queries
- based on this information and unlimited computational power creates the next operation with *the goal of maximizing the total running time*

Adversary Models

Oblivious adversary:

- knows the algorithm (but not its random choices)
- has to fix the sequence of operations *before* it receives any output from the algorithm
- creates the sequence of operations with unlimited computational power and with *the goal of maximizing the total running time*

Adversary Models

Total running time for any sequence of operations s:

$$T_{adaptive} \geq T_{oblivious}$$

$T_{adaptive} = T_{oblivious}$ if

- algorithm is deterministic
- output is unique

Performance Measures

Space: Usually linear or small polynomial

Preprocessing time: Usually linear or small polynomial

Time per operation:

- **Trade-off:** query vs update time
- **Amortized vs worst-case**
- **Oblivious vs adaptive adversary**

Outline

1. Motivation
2. State of the art
3. Conditional lower bounds
4. Hierarchical graph decompositions with polylogarithmic time per operation
5. Hierarchical graph decompositions?

Notation:

- n = number of nodes, usually unchanged
- m = number of edges in current graph

Motivation

1. **Real-world applications:** huge dynamically changing graphs: 55 out of 89 in survey

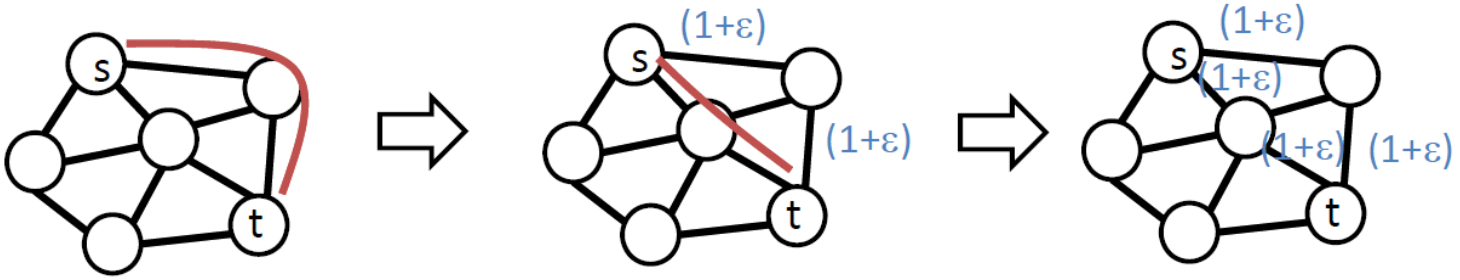
Graph	Number of nodes	Old number of nodes
WWW host names	1.3 billion (June 2019)	1.0 billion (May 2016)
WWW (indexed) web pages	5.7 billion (Sept 2019)	4.75 billion (July 2016)
Facebook	2.4 billion (July 2019)	2.1 billion (October 2016)

Motivation

1. Real-world applications
2. **Fundamental computational question**: How hard is it to find the answer after a small change in the input instance

Motivation

1. Real-world applications
2. Fundamental computational question
3. **Subroutines in static graph algorithms:**
   - Example: Max-flow and multi-commodity flow [Garg-Koenemann'98, Madry'10] algorithms with shortest augmenting paths need dynamic shortest-path algorithm



universität wien

Motivation

1. Real-world applications
2. Fundamental computational question
3. Subroutine in static graph algorithms
4. **Techniques are often reused** in other areas of algorithms such as *data stream algorithms, distributed algorithms*, and *parallel algorithms*

Update time for "classic" problems

with polylog query time

- **Undirected:** For any small constant $\epsilon > 0$
  - **Connectivity:** $\Omega(\log n)$ [Patrascu-Demaine '04], $\tilde{O}(1)$ [H-King '95, Holm-deLichtenberg-Thorup '98]
  - **MST:** $\Omega(\log n)$ [Patrascu-Demaine '04], $\tilde{O}(1)$ [Holm-deLichtenberg-Thorup '98]
  - **SSSP**: Exact: $\Omega(m^{1-\epsilon})$, $\tilde{O}(m)$
  - **APSP**: Exact: $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$ update, $\tilde{O}(n)$ path reporting query [Demetrescu-Italiano '03]
    - $O(n^{1.9})$ update, $O(n^{1.529})$ distance query, $O(n^{1.9})$ path reporting query [Bergamaschi, H, Gutenberg, Vassilevska Williams, Wein ' 20]
  - **Minimum Cut**:
    - $\Omega(n^{1-\epsilon})$ weighted exact, $\tilde{O}(m)$
    - $(1+\epsilon)$-approx: $\tilde{O}(\sqrt{n})$ [Thorup'01]
  - **Maximum cardinality matching**:
    - $\Omega(m^{1/2-\epsilon})$ exact, $O(m)$
    - $O(\sqrt{m})$ for (1+ε)-approx. [Gupta-Peng'13]

Update time for ``classic'' problems

with **polylog query time**

- **Directed:** For any small constant $\epsilon > 0$
  - **Reachability/SSSP:**
    - $\Omega(m^{1-\epsilon})$, $O(m)$
  - **SCC:** Is the graph strongly connected?
    - $\Omega(m^{1/2-\epsilon})$, $O(\min(m, n^{1.406}))$ [van den Brand, Nanongkai, Saranurak '19]
  - **Transitive closure:**
    - $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$
      - update time: $O(n^{1.406})$, query time $O(n^{1.406})$ [van den Brand, Nanongkai, Saranurak '19]
  - **APSP:**
    - $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$ update, $\tilde{O}(n)$ path reporting query Demetrescu-Italiano '03]

Update time for approximation algorithms

with polylog query time

- $(\Delta + 1)$ **− vertex coloring:**
  - $O(1)$ [H-Peng '19, Bhattacharya-Grandoni-Kulkarni-Liu '19]
- $(1 + \varepsilon)$**- approx min spanning forest value:**
  - $O(1)$ if max weight is $O(m^{1/3})$ [H-Peng'19]
- **Edge orientation with low outdegree:**
  - $O(1)$ for $O\left((\log n)^2\right)$ −approximation [Berglin-Brodal '17]
- $(1 + \varepsilon)$**- approx densest subgraph and** $(2 + \varepsilon)$**- approx degeneracy :**
  - $\tilde{O}(1)$ [Sawlani-Wang '19]
- $(4 + \varepsilon)$**- approx k-core decomposition:**
  - $\tilde{O}(1)$ [Sun-Chan-Sozio '20]
- $(1 + \varepsilon)$**- electrical flow:**
  - $O(\min(m^{3/4}, n^{5/6}))$ [Durfee-Gao-Goranci-Peng '19]

Update time for approximation algorithms

with polylog query time

- **Low-stretch probabilistic tree embedding:**
  - $n^{o(1)}$ for $n^{o(1)}$ stretch [Forster-H-Goranci '20]
- **Low-stretch spanner:**
  - $(2k-1)$ stretch with size $O(n^{1+1/k}\log n)$: $O(1)^k$ or $O(k(\log n)^2)$ [Baswana,-Khurana,-Sarkar '12, Forster-Goranci '19]
  - $(1+\varepsilon, n^{o(1)})$-spanner with size $O(n^{1+o(1)})$: $\Omega(m^{1-\epsilon})$ for combinatorial algorithms, $O(n^{1.529})$ [Bergamaschi, H, Gutenberg, Vassilevska Williams,Wein ' 20]
- **Diameter:**
  - $\Omega(n^{2-\epsilon})$ for $(\frac{3}{2}-\varepsilon)$-approx [Ancona, H, Roditty, Vassilevska Williams, Wein '19]

Outline

1. Motivation
2. State of the art
3. Conditional lower bounds
4. Hierarchical graph decompositions with polylogarithmic time per operation
5. Hierarchical graph decompositions?

Notation:

- n = number of nodes, usually unchanged
- m = number of edges in current graph

Conditional lower bounds

[Patrascu'10, Abboud, Vassilevska Williams'14, H-Krinninger-Nanongkai-Saranurak'15]

Lower bound $\Omega\left(m^{\frac{1}{2}-\varepsilon}\right)$, $\Omega(n^{1-\varepsilon})$or $\Omega(m^{1-\varepsilon})$ on time per update/query for fully dynamic based on common complexity assumptions

- Many apply to oblivious adversary
- Apply to amortized time

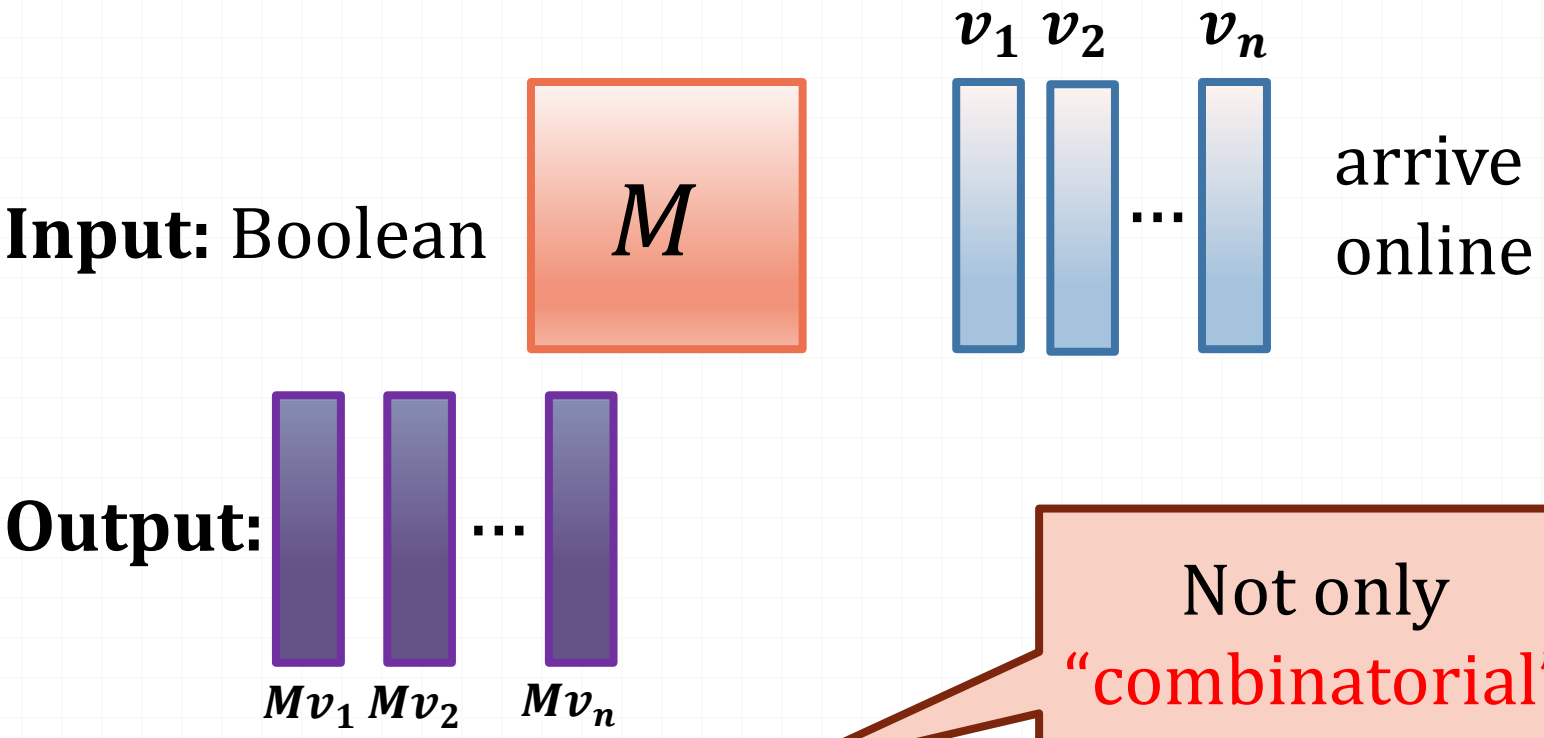
Conditional lower bounds

Conditional lower bounds [Patrascu '10, Abboud-VassilevskaWilliams'14,H-Krinninger-Nanongkai-Saranurak'15,] using these conjectures

- Assuming SETH (Strong Exponential Time Hypothesis): Satisfiability of a CNF formula with n variables takes time $\Omega\left(2^{(1-\varepsilon)n}\, poly(n)\right)$ or
- 3SUM takes time $\Omega(n^{2-\varepsilon})$ or
- Triangle detection takes time $\Omega(n^{3-\varepsilon})$ or
- AllPairsShortestPath or combinatorial Boolean matrix multiplication or online matrix-vector multiplications (OMv) takes time $\Omega(n^{3-\varepsilon})$

OMv Conjecture

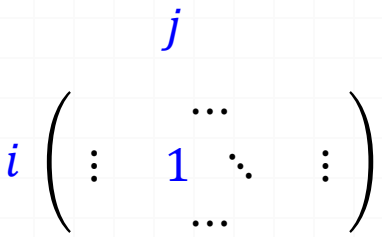
[H-Krinninger-Nanonkai-Saranurak'15]

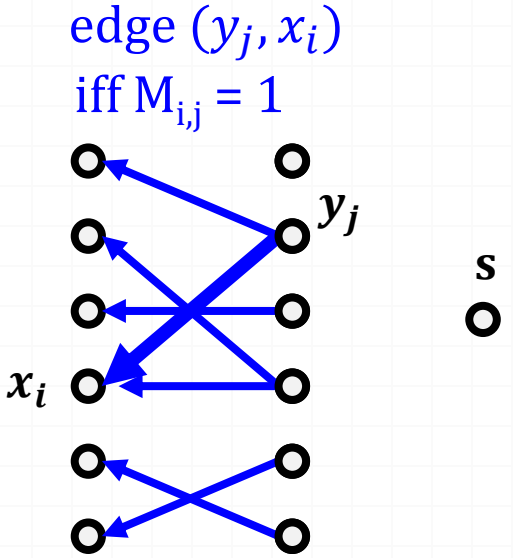


Conjecture: No algorithm with **total** time $O(n^{3-\epsilon})$

Dynamic ss-reachability takes $\Omega(n^{1-\epsilon})$ under OMv

- Given M construct bipartite graph $(X \cup Y, E)$ with n nodes in each column plus an isolated node s



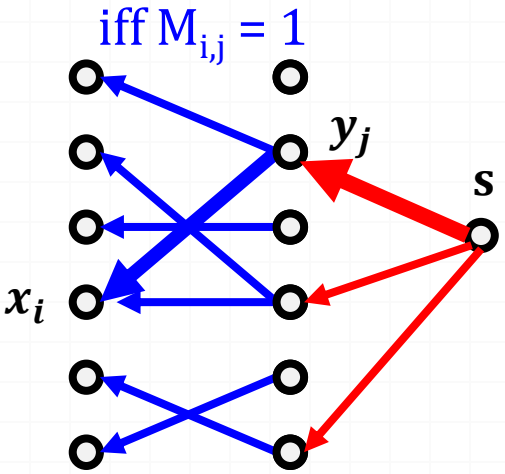


Dynamic ss-reachability takes $\Omega(n^{1-\epsilon})$ under OMv

- Given M construct bipartite graph $(X \cup Y, E)$with n nodes in each column plus an isolated node s
- Given vector v add edge (s, y_j) iff $v_j = 1$

Note: $(Mv)_i = 1$ iff s can reach x_i

- $$i \begin{pmatrix} & j & \\ \vdots & 1 & \vdots \\ & \cdots & \end{pmatrix} \begin{pmatrix} \vdots \\ 1 \\ \vdots \end{pmatrix} j$$



universität wien

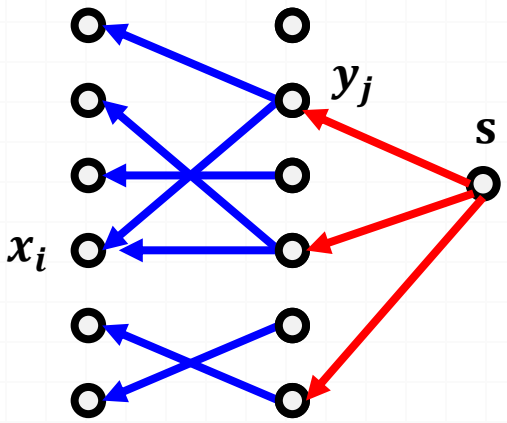
Dynamic ss-reachability takes $\Omega(n^{1-\epsilon})$ under OMv

- Given M construct bipartite graph with n nodes in each column plus an isolated node s
- Given vector v add edge (s, y_j) iff $v_i = 1$

Note: $(Mv)_i = 1$ iff s can reach x_i

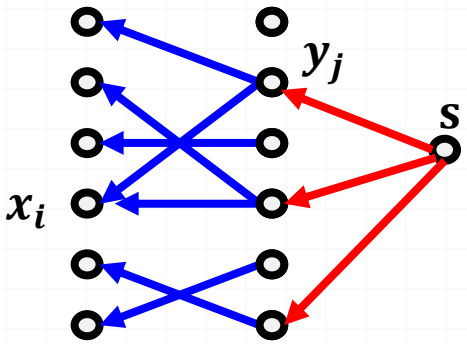
- **Algorithm for computing Mv**:
  - For j from 1 to n:
    - if $v_j = 1$ then insert(s, y_j)
  - For i from 1 to n:
    - if s can reach x_i return 1, else 0
  - For j from 1 to n:
    - if $v_j = 1$ then delete(s, y_j)

edge (y_j, x_i) iff $M_{i,j} = 1$



Dynamic ss-reachability takes $\Omega(n^{1-\epsilon})$ under OMv

- **Algorithm for computing Mv**:
  - For j from 1 to n:
    - if v_j= 1 then insert(s, y_j)
  - For i from 1 to n:
    - if s can reach x_i return 1 else 0
  - For j from 1 to n:
    - if v_j= 1 then delete(s, y_j)



$\Rightarrow$ n insertions, deletions and queries per vector v on a graph with $2n + 1$ nodes and $O(n^2)$ edges

$\Rightarrow$ n^2 insertions, deletions and queries for **all** vectors v on a graph with $2n + 1$ nodes and $O(n^2)$ edges take time $\Omega(n^{3-\varepsilon})$

Dynamic ss-reachability takes $\Omega(n^{1-\epsilon})$ under OMv

$\Rightarrow$ n^2 insertions, deletions and queries for all vectors v on a graph with $2n + 1$ nodes and $O(n^2)$ edges take time $\Omega(n^{3-\varepsilon})$

- No dynamic single source reachability algorithm can take time
  - $\Omega(n^{1-\varepsilon})$ per update **and**
  - $\Omega(n^{1-\varepsilon})$ per query

Lower bounds based on OMv

Lower bound of $\Omega(m^{1/2-\varepsilon})$ on time per update or query (sometimes even $\Omega(m^{1-\varepsilon})$) for fully dynamic

- <5/3-approximate SSP
- Densest subgraph
- $(2-\varepsilon)$-approximate weighted diameter
- Strong connectivity
- Bipartite perfect matching
- Size of bipartite maximum cardinality matching
- Electrical flow [Goranci-H-Peng'18]
- …

Outline

- Motivation
- State of the art
- Conditional lower bounds
- Hierarchical graph decompositions with polylogarithmic time per operation
  - Type 1: Monotone, global invariants
  - Type 2: Non-monotone, local invariants
- Are hierarchies necessary?

Hierarchical graph decompositions

with polylog time per operation

- Nodes and/or edges are partitioned into $O(\log n)$ levels dependent on
  - sequence of updates and
  - invariants
- **Goal:** Polylogarithmic time per update

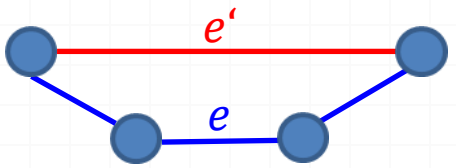
Hierarchical graph decomposition 1

- **Monotone hierarchy with global invariants** [H-King'95, Holm-de Lichtenberg-Thorup'98]:
  - Monotone: All nodes/edges initially and all newly inserted edges are on level 0 and only move up in the hierarchy.
  - Global invariants

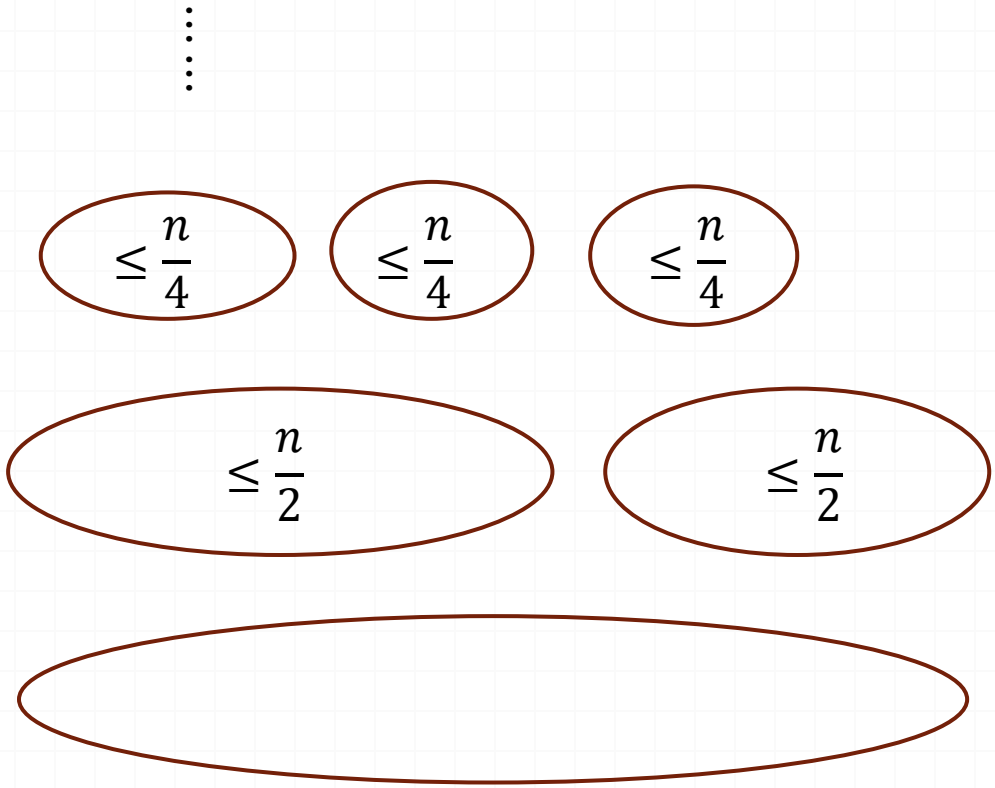
Hierarchical graph decomposition 1

Spanning tree [Holm et al '98]

- Nodes on all levels, each edge e has a level $l(e)$, initially 0
- Maintains spanning forest T
  - T_i subforest of T of edges of level $\geq i$, spans graph induced by all edges on level $\geq i$
- **Global invariants:**
  - Level invariant: Let e be a tree edge. All non-tree edges crossing the tree cut $T - \{ e \}$ must have level $\leq l(e)$
  - Size invariant: Number of nodes in a cc on level i is $O(\frac{n}{2^i})$



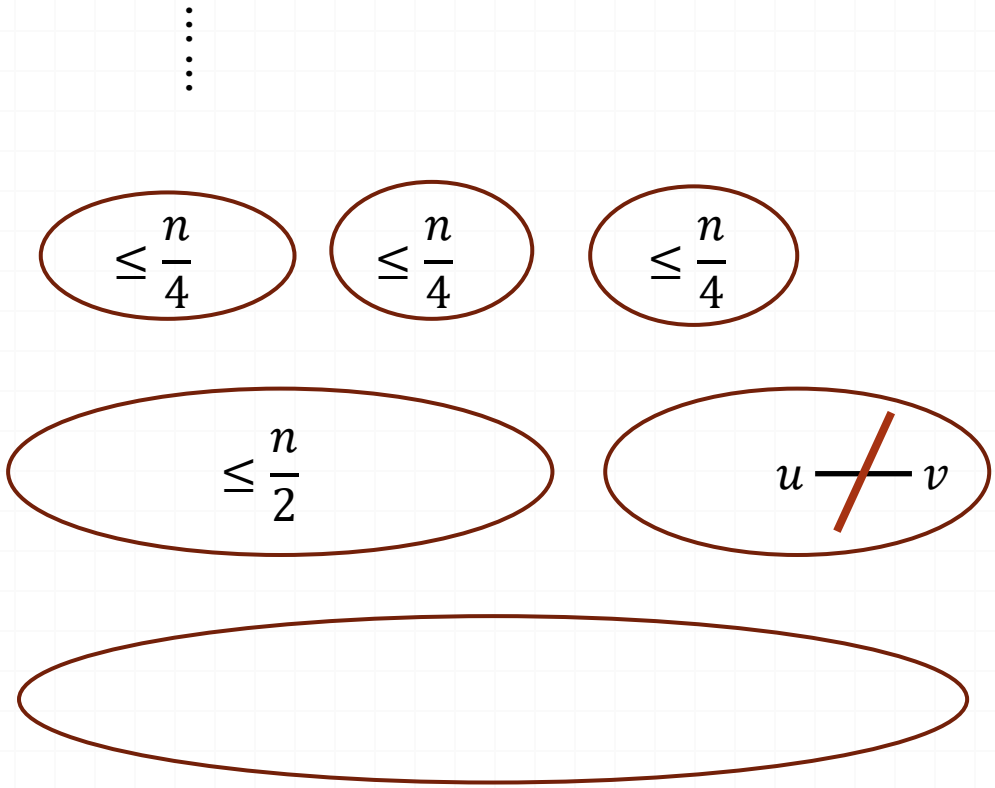
Hierarchical graph decomposition 1



$O(\log n)$ levels

Hierarchical graph decomposition 1

Delete(u, v)



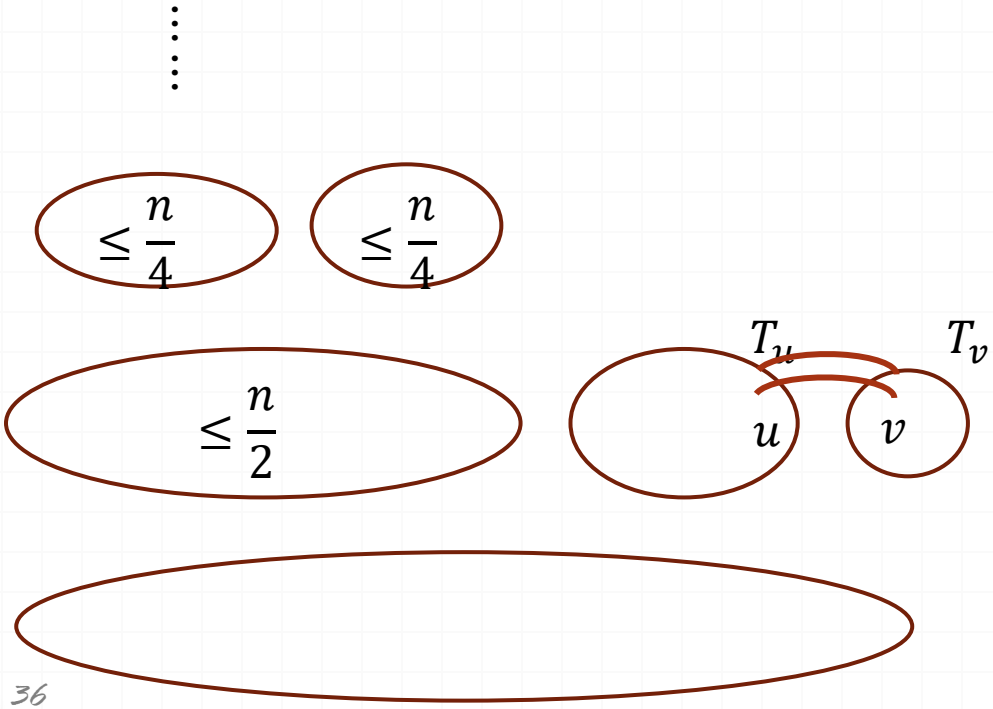
$O(\log n)$ levels

Hierarchical graph decomposition 1

Delete(u, v)**:**

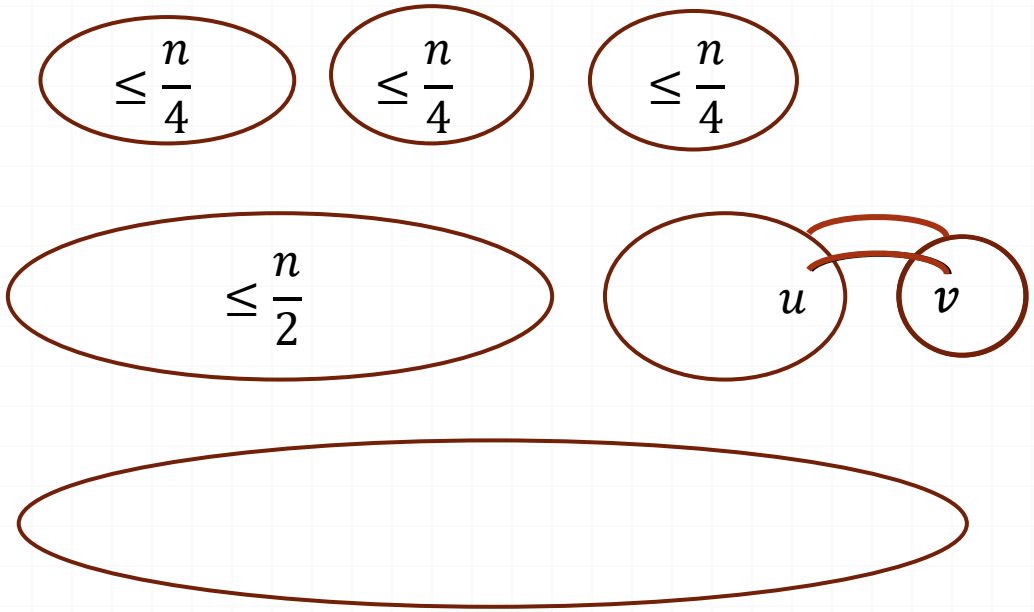
If (u, v) is a tree edge on level i then

1. Let T' := smaller of T_u and T_v
2. …



Hierarchical graph decomposition 1

2. Move all tree edges of T' that are on level i to level i +1
3. Test all non-tree edges incident to T' until one is found that reconnects T_u and T_v (=replacement edge) or all have been tested.
4. Move all the tested non-tree edges to level i +1
5. If no replacement edge found on level i, repeat on level $i-1$ else replacement edge becomes a level-i tree edge



$O(\log n)$ levels

Running time analysis

- Level invariant implies: # of levels $\leq \log n$
- With suitable data structures: work of delete is O(# of edges whose level has increased * polylog n)
  - Charge O(polylog n) to each edge that has its level increased to account for the work
  - Each edge can be charged only $\log n$ times
  - Total charge per edge O(polylog n)
  - Total work O(m polylog n)
  - Amortized time O(polylog n) per deletion

Hierarchical graph decomposition 1

- **Monotone hierarchy with global invariants**:
  - Monotone edge movements
  - Global invariants
  - Simple running time analysis, longer proof of global invariants

Usage: Connectivity, MST, 2-edge connectivity, 2-vertex connectivity

Almost tight: $\Omega(\log n)$ cell probe lower bound [Patrascu-Demaine 04]

No other approaches achieve same performance for these problems

Hierarchical graph decomposition 2

- **Hierarchies with local invariants [Bhattacharya-H-Italiano'15]:**
  - Bi-directional movements: Edges and nodes move up and down deterministically
  - Local invariants: Condition on neighborhood of each node, directly enforced by algorithm
  - Combination with a second algorithm, e.g., primal-dual method

Hierarchical graph decomposition 2

for matching and vertex cover

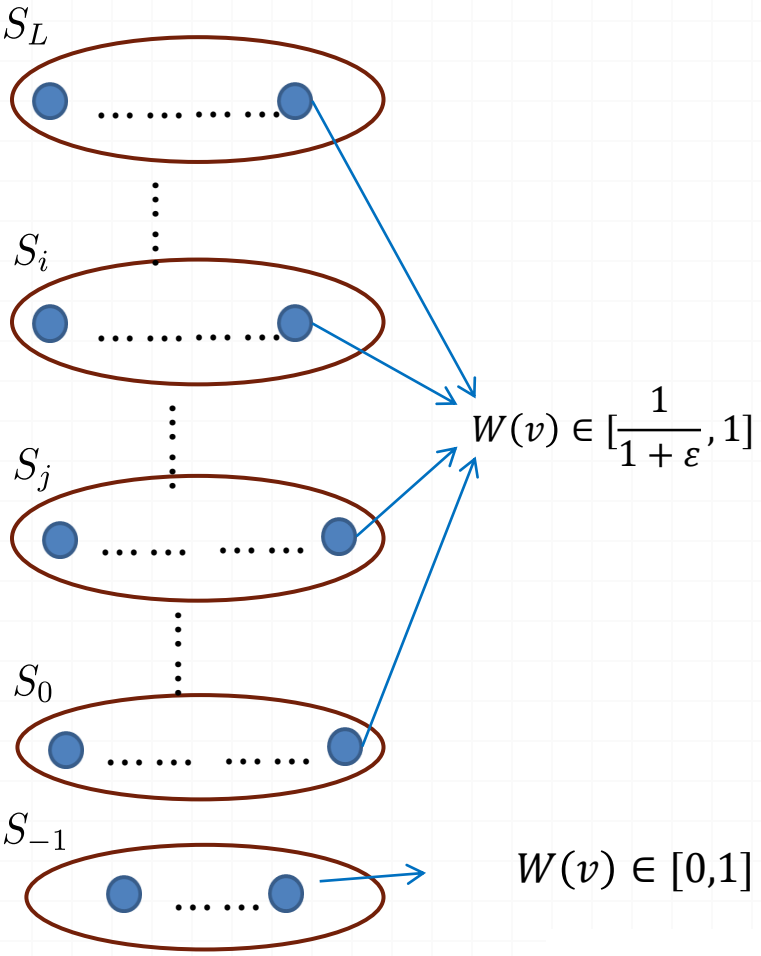
- Each node v has a level $l(v) \in [-1, \log n]$
- Each edge e has a weight $w(e) = {}^{1}/_{(1+\varepsilon)^{i}}$, where i is the level of the higher end point
- Each node v has a weight $W(v) = \sum_{(u,v)\in E} w(u,v)$

- **Local invariants:**
  - $\forall v \in V - S_{-1}: W(v) \in \left[\frac{1}{1+\varepsilon}, 1\right]$
  - $\forall v \in S_{-1}: W(v) \in [0,1]$

$\Rightarrow$ No edge between 2 vertices of S_{-1}

$\Rightarrow$ $V - S_{-1}$ is vertex cover, S_{-1} is not in vertex cover

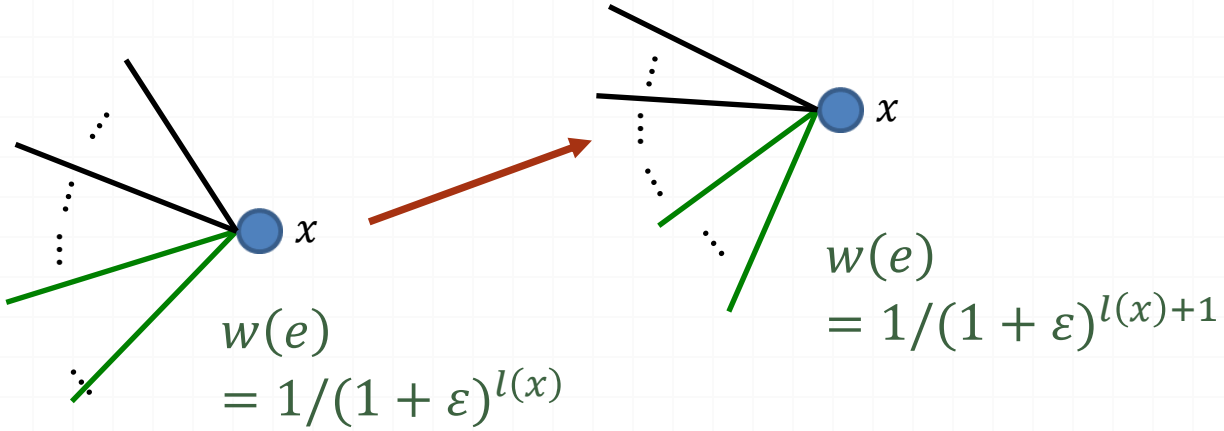
$\Rightarrow$ Local invariants = relaxed complementary slackness conditions for matching LP



universität wien

Making it dynamic

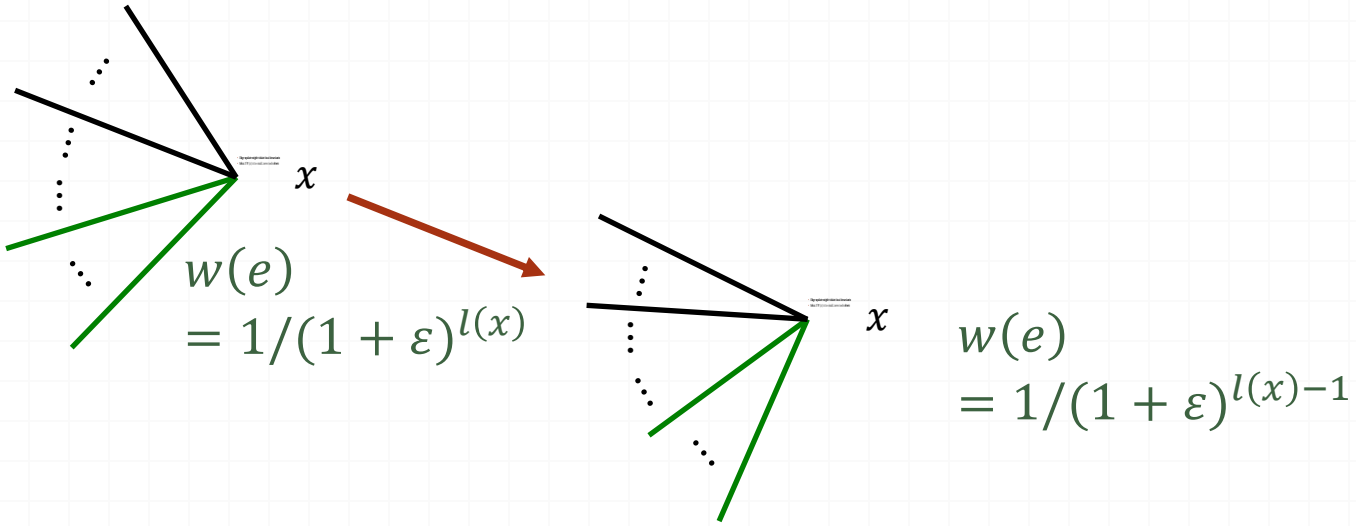
- **Edge update might violate local invariants**
- **Idea:** If $W(x)$ is too large, move node **up**



$\Rightarrow W(x)$ might drop, at most by multiplicative factor $1/(1+\varepsilon)$

Making it dynamic

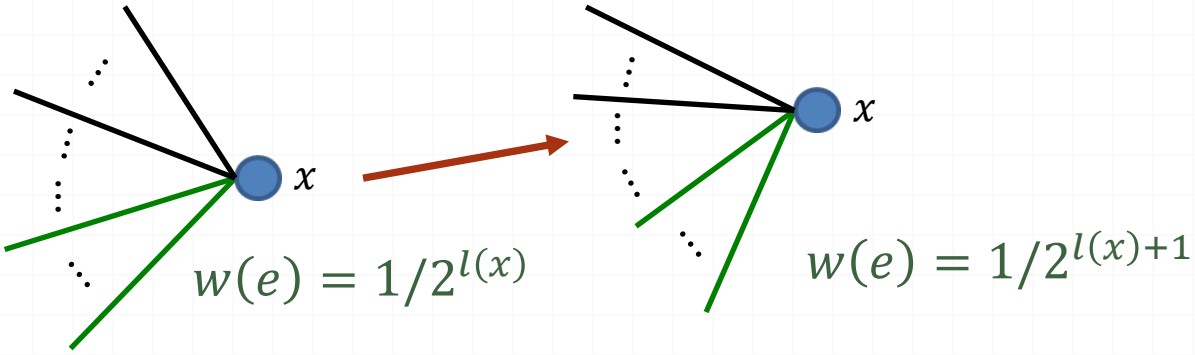
- **Edge update might violate local invariants**
- **Idea:** If $W(x)$ is too small, move node **down**



$\Rightarrow W(x)$ might increase, at most by multiplicative factor $(1+\varepsilon)$

Making it dynamic

- **Idea:** If $W(x)$ is too large/small, move node



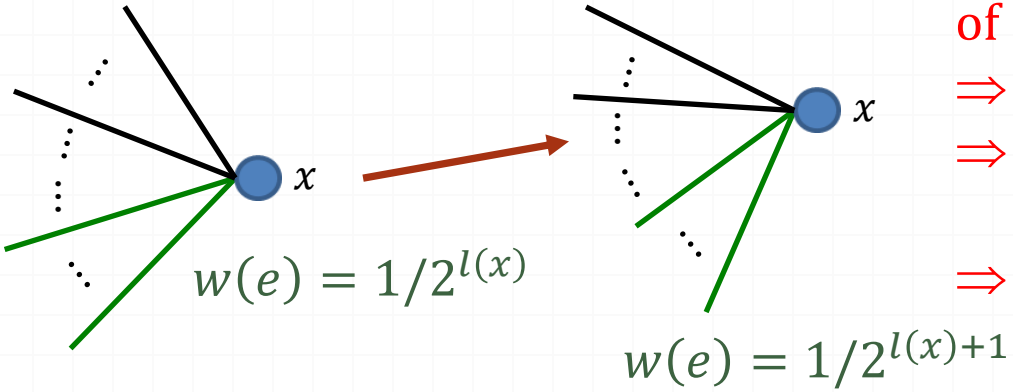
FIX-NODE-WEIGHTS

WHILE there is a node x with $W(x)$ too large or too small
  If $W(x)$ is too large then
    move x up to the lowest level such that $W(x) \leq 1$
  If $W(x)$ is too small then
    move x down ...

Maintains relaxed complementary slackness conditions

Making it dynamic

- If $W(x)$ is too large, move node up:



might decrease weight of neighbors
⇒ neighbors might drop
⇒ their neighbors might rise
⇒ AVALANCHE !

FIX-NODE-WEIGHTS

WHILE there is a node x with $W(x)$ too large or too small
  If $W(x)$ is too large then
    move x up to the lowest level such that $W(x) \leq 1$
  If $W(x)$ is too small then
    move x down ...

Hierarchical graph decomposition 2

- **Hierarchies with local invariants:**
  - Simple correctness proof
  - Running time analysis:
    - With carefully chosen potential function:
      - $O(\log n)$ [Bhattacharya-H-Italiano '15]
      - $O(1)$ [Bhattacharya-Chakrabarty-H'17, Bhattacharya-Kulkarni'18]
  - Combination with second alg, e.g. primal-dual method

Hierarchical graph decomposition 2

- **Hierarchies with local invariants [Bhattacharya-H-Italiano'15]**
  - Bi-directional movements
  - Local invariants
  - Combination with second algorithm
  - Correctness proof follows directly, running time analysis is sophisticated

Usage:

- $(2+\varepsilon)$-approximate vertex cover & fractional matching
- $O(f^2)$-approximate set cover, f = number of sets one element can belong to
- $(2+\varepsilon)$-approximate densest subgraph
- $(\Delta+1)$-vertex coloring
- $(4+\varepsilon)$- approx k-core decomposition

Outline

- Motivation
- State of the art
- Hierarchical graph decompositions with polylogarithmic time per operation
  - Type 1: Monotone, global invariants
  - Type 2: Non-monotone, local invariants
- Are hierarchies necessary?

Outline

- Motivation
- State of the art
- Hierarchical graph decompositions with polylogarithmic time per operation
  - Type 1: Monotone, global invariants
  - Type 2: Non-monotone, local invariants
- Are dynamic hierarchies necessary?

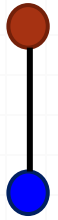
Dynamic hierarchical graph decompositions?

Disadvantages:

- Hard to implement
- Large constants, slow in practice
- Sometimes bottleneck in theory:
  - **Example:** $(\Delta + 1)$-vertex coloring
    - worst-case $O(\log n)$ time to maintain hierarchy
    - $O(1)$ expected time to maintain coloring with hierarchy

Is dynamic hierarchical graph decomposition always needed?

Dynamic $(\Delta + 1)$- vertex coloring



- A k-vertex coloring is an assignment of a number $c(u)$ from $\{1, \dots, k\}$ to every vertex u such that $c(u) \neq c(v)$ if $(u, v) \in E$
- **Fact:** Every graph with maximum degree Δ has a $(\Delta + 1)$-vertex coloring
- **Dynamic version:** Given graph G with maximum degree $\leq \Delta$ and a sequence of edge insertions and deletions
  - Goal: Maintain $(\Delta + 1)$-vertex coloring

Dynamic $(\Delta + 1)$- vertex coloring

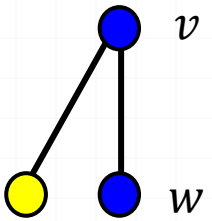
- **Random ranks** [H – Pan Peng '19]:
  - Initial graph is empty
  - Each vertex v samples a random rank $r(v) \in [0,1]$ and a random color $c(v)$
  - Maintain
    - L_v = all neighbors of v with ranks $< r(v)$
    - H_v = all neighbors of v with ranks $\geq r(v)$
    - B_v = colors used by no neighbor of v
    - U_v = colors used by exactly one neighbor of L_v and no neighbor in H_v

Dyn. $(\Delta + 1)$- vertex coloring

- Maintain
  - L_v = all neighbors of v with ranks $< r(v)$
  - H_v = all neighbors of v with ranks $\geq r(v)$
  - B_v = colors used by no neighbor of v
  - U_v = colors used by exactly one neighbor of L_v and no neighbor in H_v
- Insert(u, v):
  - if no conflict: c unchanged, update L_v, H_v, B_v, U_v
  - else: suppose v was colored last, call Recolor(v)

Dyn. $(\Delta + 1)$- vertex coloring

- Maintain
  - L_v = all neighbors of v with ranks $< r(v)$
  - H_v = all neighbors of v with ranks $\geq r(v)$
  - B_v = colors used by no neighbor of v
  - U_v = colors used by exactly one neighbor of L_v and no neighbor in H_v
- Recolor(v):
  - if $|B_v|$ large: sample color from B_v, done
  - else:
    - sample color c^* from $B_v \cup U_v$ and set $c(v) = c^*$
    - if $c^* \in B_v$, done
    - else:
      - find unique conflicting neighbor w and call Recolor(w)

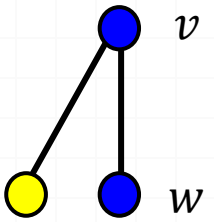


Dyn. $(\Delta + 1)$- vertex coloring

- Maintain
  - L_v = all neighbors of v with ranks $< r(v)$
  - H_v = all neighbors of v with ranks $\geq r(v)$
  - B_v = colors used by no neighbor of v
  - U_v = colors used by exactly one neighbor of L_v and no neighbor in H_v
- Recolor(v):
  - if $|B_v|$ large: sample color from B_v, done
  - else:
    - sample color c^* from $B_v \cup U_v$ (*) and set $c(v) = c^*$
    - if $c^* \in B_v$, done
    - else:
      - find unique conflicting neighbor w and call Recolor(w)

$\Longrightarrow$ Leads to recoloring path

(*) sampling has various subcases



universität wien

Constant time analysis

Theorem: A $(\Delta + 1)$-vertex coloring can be maintained dynamically in amortized expected $O(1)$ time.

Proof idea: Expensive case if recoloring path is long.

- Partition vertices on path in 2 groups, A and B.
- Potential function Φ
  - $\Phi = 0$ initially, $\Phi \geq 0$ always

    $\Longrightarrow$ sum of increases $\geq$ sum of decreases
  - Case A increases or decreases Φ, Case B only decreases Φ
  - $|\Delta\Phi|$ proportional to running time of Case A and Case B
  - Total running time in case A $\geq$ sum of increases $\geq$ sum of decreases $\geq$ total running time in case B
- Total running time in case A can be analyzed by probabilistic argument

Constant time analysis

Theorem: A $(\Delta + 1)$-vertex coloring can be maintained dynamically in amortized expected $O(1)$ time.

Note:

- Implies O(1) amortized expected bound on number of color changes
- Independent, different approach: Bhattacharya et al.'19

Random Rank

Random Rank [Behnezhad et al, Chechik-Zhang, H-Peng '19]:

- No movements of vertices
- Oblivious adversary: does not learn the ranks
- Might use additional randomness

Usage:

- $(\Delta + 1)$-vertex coloring in $O(1)$ time [H-Peng '19]
- Maximal independent set and maximal matching in $O(\log^4 n)$ time [Behnezhad et al, Chechik-Zhang '19]

Comparison

Hierarchy

- Partial ordering
- Ordering of vertices is dynamic
- Maintenance overhead
- Deterministic
- Potential function argument

Random ranks

- Full ordering
- Ordering of vertices is fixed
- No maintenance overhead
- Randomized against oblivious adversary
- Sophisticated analysis: potential function and probabilistic analysis

No other techniques achieve polylog n update and query times

Summary

- Dynamic graph algorithms are well-studied
  - Strong conditional lower bounds, but still some interesting gaps (fully dynamic APSP)
- Many interesting techniques
  - Classic: Hierarchical decompositions in dynamic graph algorithms
  - New: Vertices ordered by random ranking
  - Sparsification

Open questions

- Is picking a random permutation of the vertices also useful for other dynamic problems?
- Experimental evaluation
  - See upcoming survey
- Close gaps:
  - between upper and lower bounds
  - adaptive vs oblivious adversary