Dynamic Graph Algorithms

Giuseppe F. Italiano

University of Rome "Tor Vergata"

italiano@disp.uniroma2.it

http://www.disp.uniroma2.it/users/italiano

Outline

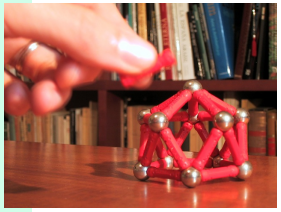
Dynamic Graph Problems

Methodology & State of the Art

Algorithmic Techniques & Experiments

Conclusions

Outline



Dynamic Graph Problems

Methodology & State of the Art

Algorithmic Techniques & Experiments

Conclusions

Graphs…

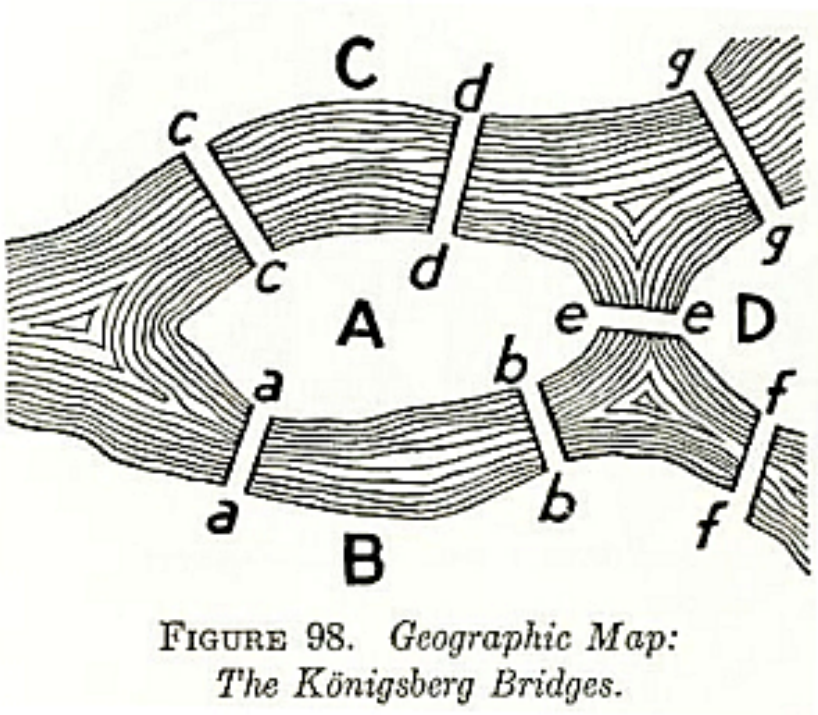
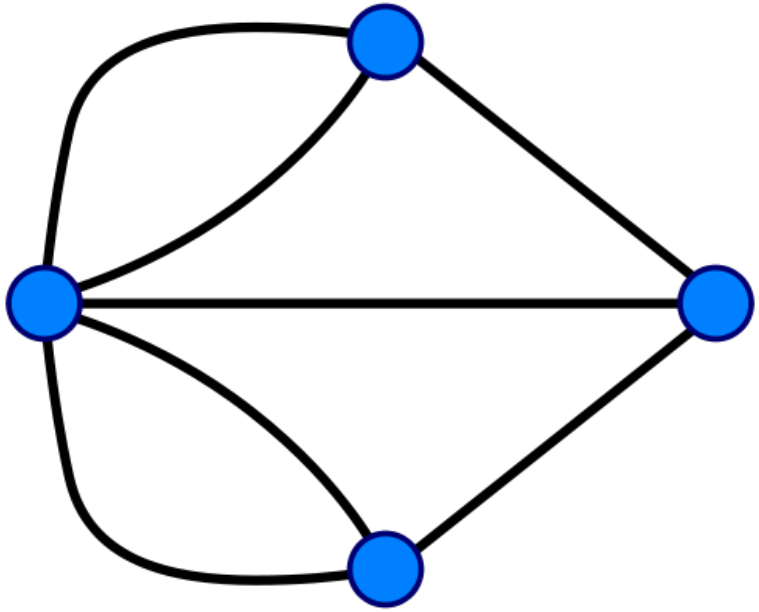
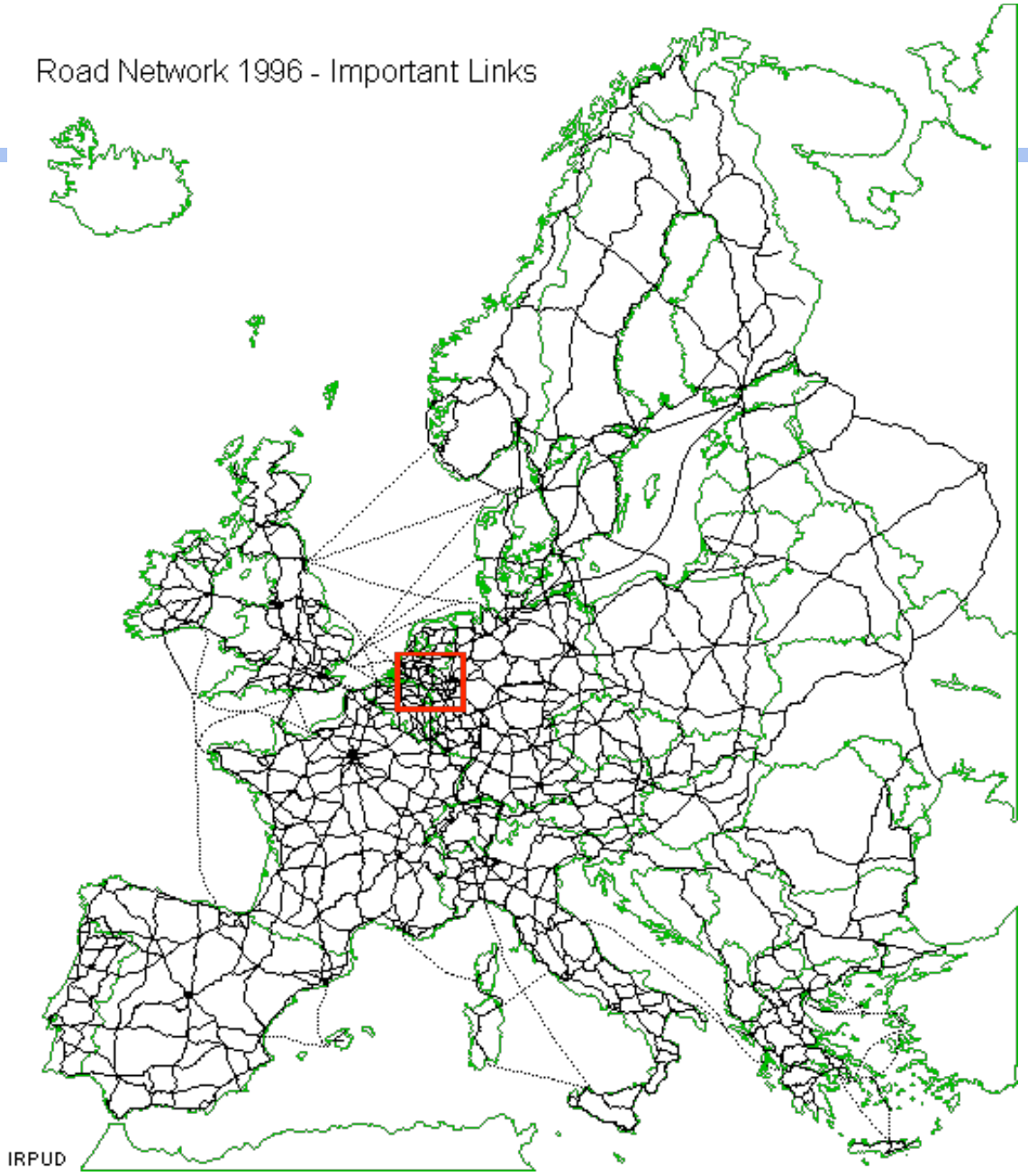


FIGURE 98. *Geographic Map: The Königsberg Bridges.*

1736



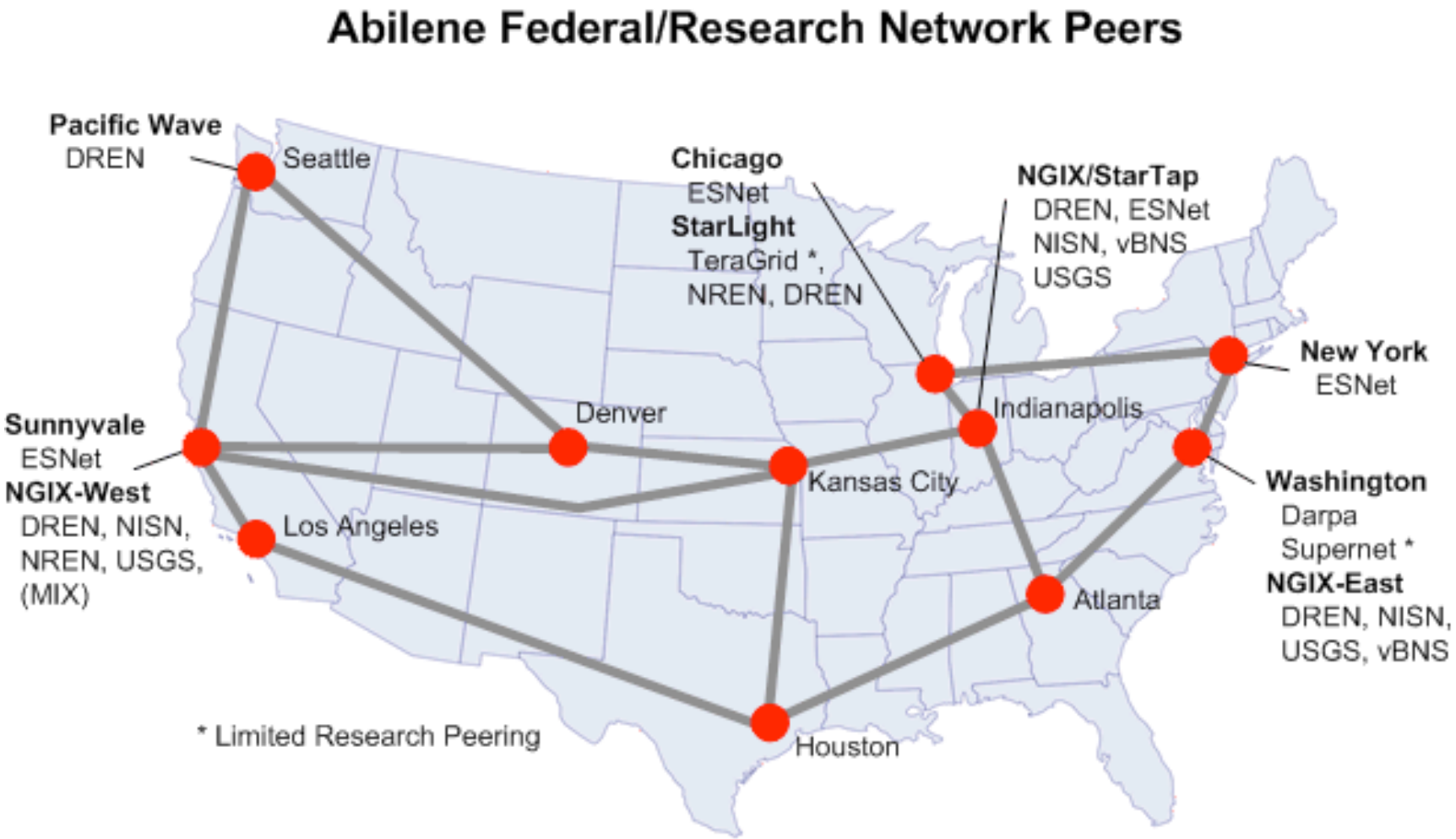
Graphs have been used for centuries to model relationships…


Road Network 1996 - Important Links
IRPUD


Detroit
Minneapolis
St Paul
Seattle
Portland
San Francisco
Los Angeles
Beijing
Seoul
Bangkok
Shanghai
Busan
Guangzhou
Hong
Kong
Taipei
Osaka
Nagoya
Tokyo
Singapore
Manila
Saipan
Guam
Honolulu
Kahului
Kona
Routes arranged by Airlineroutemaps.com


System Map
Legend
Commuter Rail
Parking
Glenmont
Wheaton
Forest Glen
Silver Spring
Takoma
Brookland
College Park-U of Md
Prince George's Plaza
West Hyattsville
Fort Totten
Rhode Island Ave
New Carrollton
Landover
Cheverly
Deanwood
Minnesota Ave
Capitol Heights
Addison Road
Morgan Blvd
Largo Town Center
Benning Road
Capitol Heaths
Stadium-Armory
Potomac Ave
Eastern Market
Navy Yard
Waterfront
Archives
L'Enfant Plaza
Smithsonian
Federal Triangle
Metro Center
McPherson Sq
Farragut West
Farragut North
Dupont Circle
Woodley Park-Zoo
Cleveland Park
Van Ness-UDC
Tenleytown-AU
Friendship Heights
Bethesda
Medical Center
Grosvenor
Twinbrook
Rockville
Shady Grove
Rosslyn
Court House
Clarendon
Virginia Square
Ballston
East Falls Church
West Falls Church
Dunn Loring-Merrifield
Vienna
ORANGE LINE
Arlington Cemetery
Pentagon
Pentagon City
Crystal City
National Airport
Braddock Road
King Street
Eisenhower Ave
Huntington
Franconia-Springfield
Van Dorn Street
Alexandria
Fairfax Co
Capitol Beltway
Potomac Yard
Columbia Heights
Georgia Ave-Petworth
U Street-Cardozo
Shaw-Howard U
Mt Vernon Sq/7th St-Convention Ctr
N
No Smoking
No Food or Drinks
No Animals
No Audio or Video Devices
No Litter or Spitting
No Dangerous or Flammable Materials

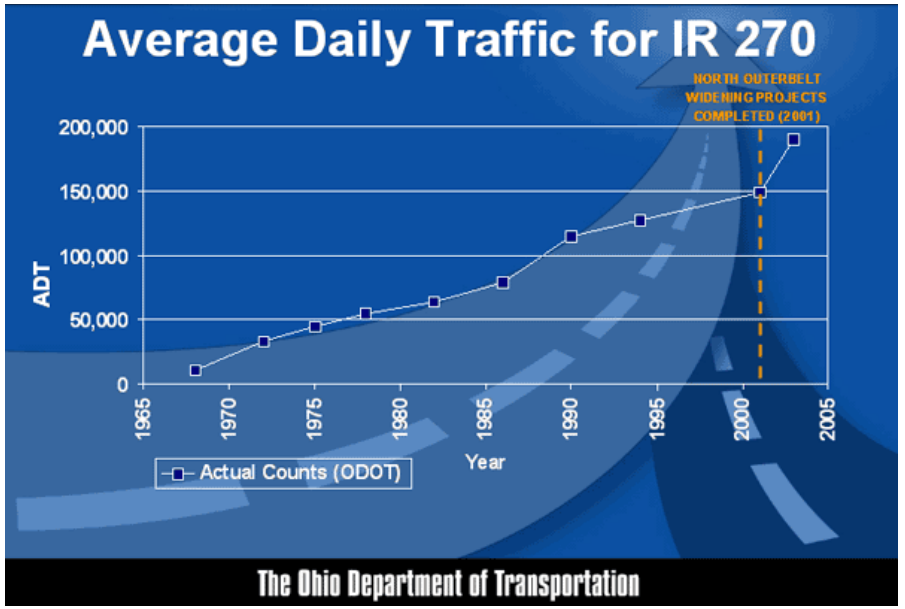
Graphs…


Abilene Federal/Research Network Peers
Pacific Wave
DREN
Seattle
Chicago
ESNet
StarLight
TeraGrid *,
NREN, DREN
NGIX/StarTap
DREN, ESNet
NISN, vBNS
USGS
New York
ESNet
Sunnyvale
ESNet
NGIX-West
DREN, NISN,
NREN, USGS,
(MIX)
Denver
Indianapolis
Kansas City
Los Angeles
Atlanta
Washington
Darpa
Supernet *
NGIX-East
DREN, NISN,
USGS, vBNS
* Limited Research Peering
Houston

Graphs…

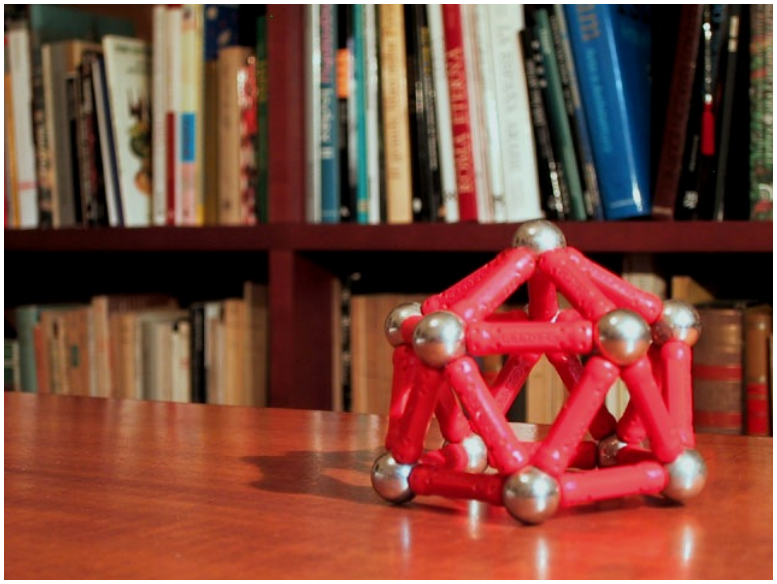
Dynamic Graphs

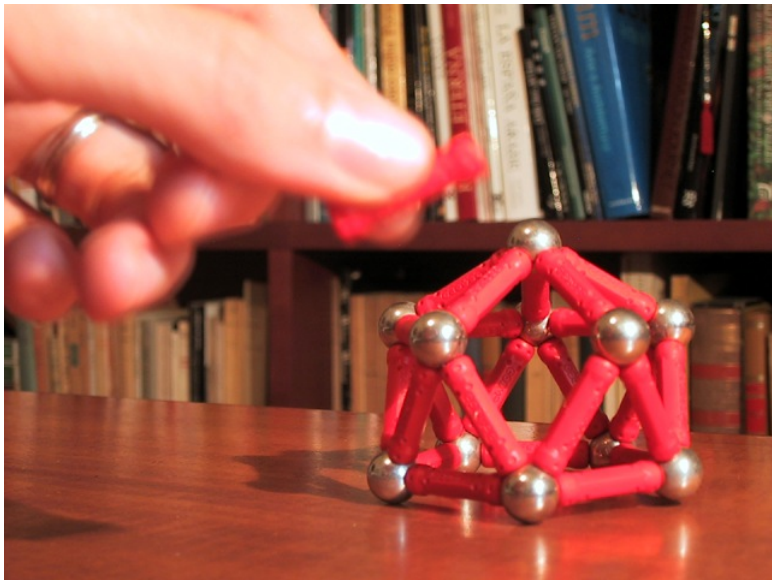
Sometimes, life looks a bit more dynamic …



Dynamic Graphs

Graphs subject to **update** operations





Typical updates:

```
Insert(u,v)
Delete(u,v)
SetWeight(u,v,w)
```

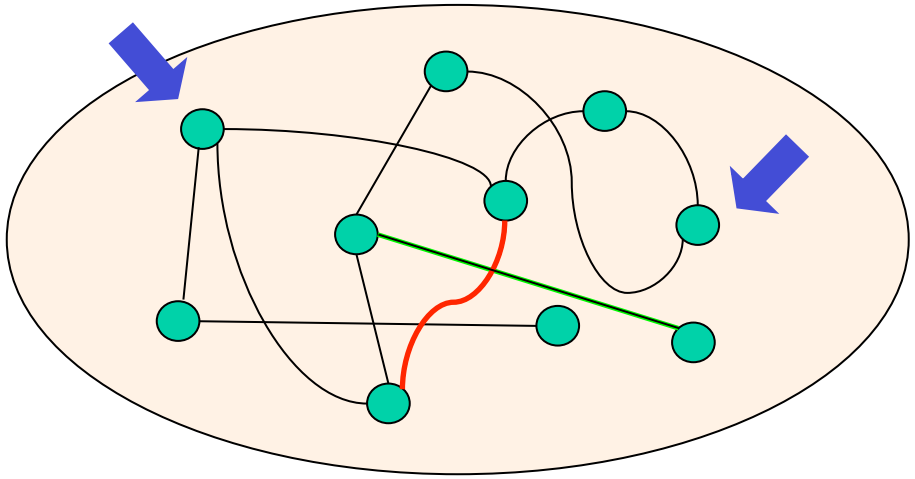
Dynamic Graphs

Initialize

Insert

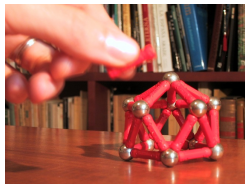
Delete

Query



A graph

Dynamic Graphs



Partially dynamic problems

Graphs subject to insertions only, or deletions only, but not both.

Fully dynamic problems

Graphs subject to intermixed sequences of insertions and deletions.

Dynamic Graph Problems

Support **query** operations about certain properties on a dynamic graph

- *Dynamic Connectivity (undirected graph G)*
  `Connected(): Connected(x,y):`
  Is G connected? Are x and y connected in G?

- *Dynamic Transitive Closure (directed graph G)*
  `Reachable(x,y):`
  Is y reachable from x in G?

Dynamic Graph Problems

- *Dynamic All Pairs Shortest Paths*

  `Distance(x,y):`

  What is the distance from x to y in G?

  `ShortestPath(x,y):`

  What is the shortest path from x to y in G?

- *Dynamic Minimum Spanning Tree (undirected graph G)*

  Any property on a MST of G

Dynamic Graph Problems

- *Dynamic Min Cut*

  `MinCut():` Min cut?

  `Cut(x,y):` Are x and y on the same side of a min cut of G?

- *Dynamic Planarity Testing*

  `planar():` Is G planar?

- *Dynamic k-connectivity*

  `k-connected():` Is G k-connected?

  `k-connected(x,y):` Are x and y k-connected?

Dynamic Graph Algorithms

The goal of a dynamic graph algorithm is to support **query** and **update** operations as quickly as possible.

Notation:

$G = (V,E)$

$n = |V|$

$m = |E|$

We will sometimes use amortized analysis:

$$\frac{\text{Total worst-case time over sequence of ops}}{\text{\# operations}}$$

Amortized Analysis

Example: Counting in Binary
Cost is # bit flips

Total cost to count to n?

Worst-case cost to add 1: log(n + 1) + 1

011111…111  100000…000

Total cost to count to n: n logn ?

Amortized Analysis

Amortize:
To liquidate a debt by installment payments

Etymology from Vulgar Latin “admortire”:
To kill, to reduce to the point of death

In analysis of algorithms, analyze execution cost of algorithms over a sequence of operations
I.e., pay for the total cost of a sequence of operations by charging each
operation an equal (or appropriate) amount

Amortized Analysis

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
...

Example: Counting in Binary
Cost is # bit flips

Bit x_0 flips n times
Bit x_1 flips n/2 times
Bit x_2 flips n/4 times …

Total # bit flips to count to n is $\leq 2n$

Amortized # bit flips at each step is ≤ 2

Amortized Analysis

Another example: Array doubling
Dynamic array: double size each time fills up

Array reallocation may require an insertion to cost O(n)

However, sequence of n insertions can always be completed in O(n) time, since rest of insertions done in constant time

Amortized time per operation is therefore O(n) / n = O(1).

Running Example: Fully Dynamic APSP

Given a weighted directed graph G = (V,E,w), perform any intermixed sequence of the following operations:

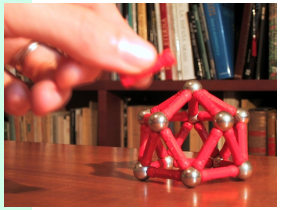
Update(v,w):	update edges incident to v [w( )]
Query(x,y):	return distance from x to y (or shortest path from x to y)

Some Terminology

- *APSP: All Pairs Shortest Paths*
- *SSSP: Single Source Shortest Paths*
- *SSSS: Single Source Single Sink Shortest Paths*
- *NAPSP, NSSP, NSSS: Shortest Paths on Non-negative weight graphs*

Outline

Dynamic Graph Problems



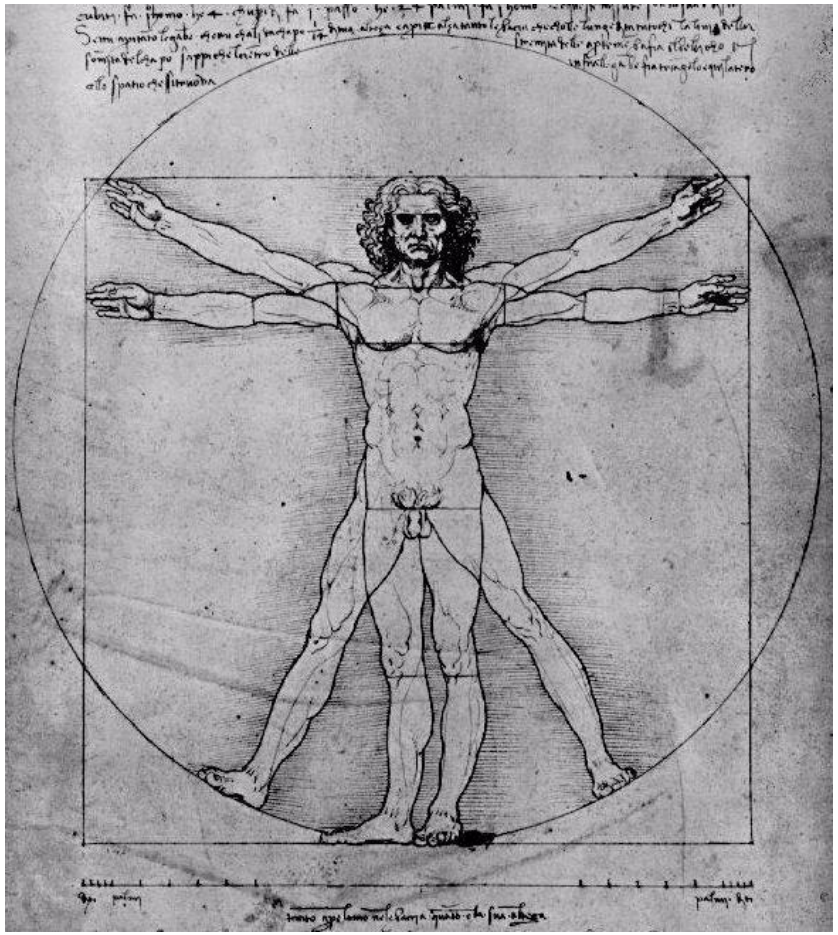
Methodology & State of the Art

Algorithmic Techniques & Experiments

Conclusions

Methodology: Algorithm Engineering

Theory



In theory, theory and practice are the same.

Why Algorithm Engineering?

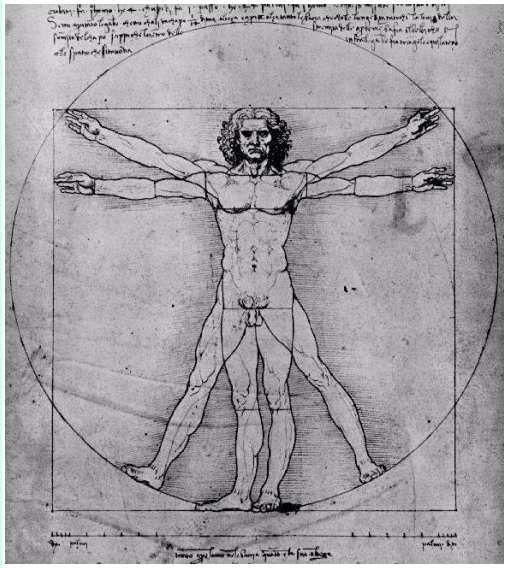
The real world out there...

In practice, theory and practice are different…

Programs are first class citizens as well

Theory 	Practice 
Number types: $\mathbb{N}$, $\mathbb{R}$	`int, float, double`
Only asymptotics matter	Seconds do matter
Abstract algorithm description	Non-trivial implementation decisions, error-prone
Unbounded memory, unit access cost	Memory hierarchy / bandwidth
Elementary operations take constant time	Multicore CPUs, Instruction pipelining, …

Bridging the Gap between Theory & Practice



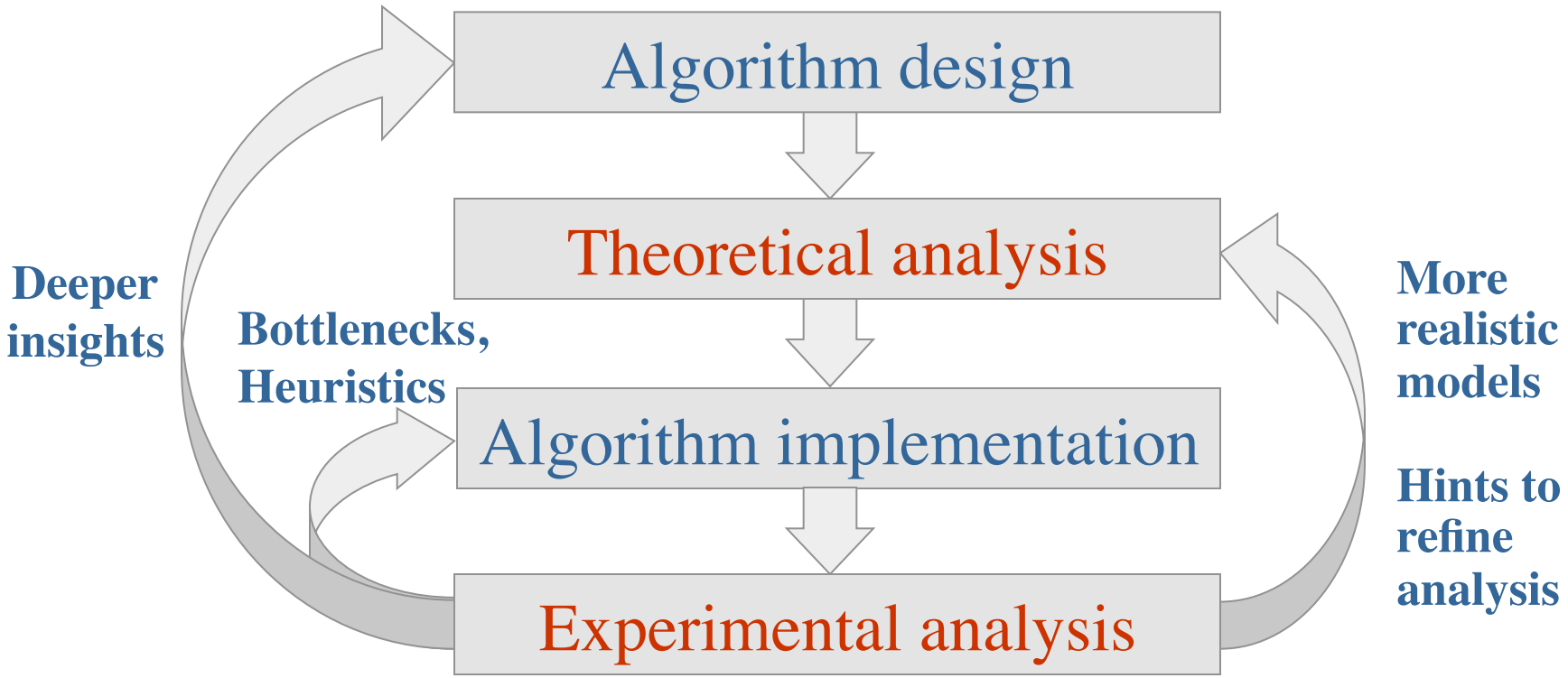
Theory is when we know something, but it doesn't work.

Wish to combine theory and practice…

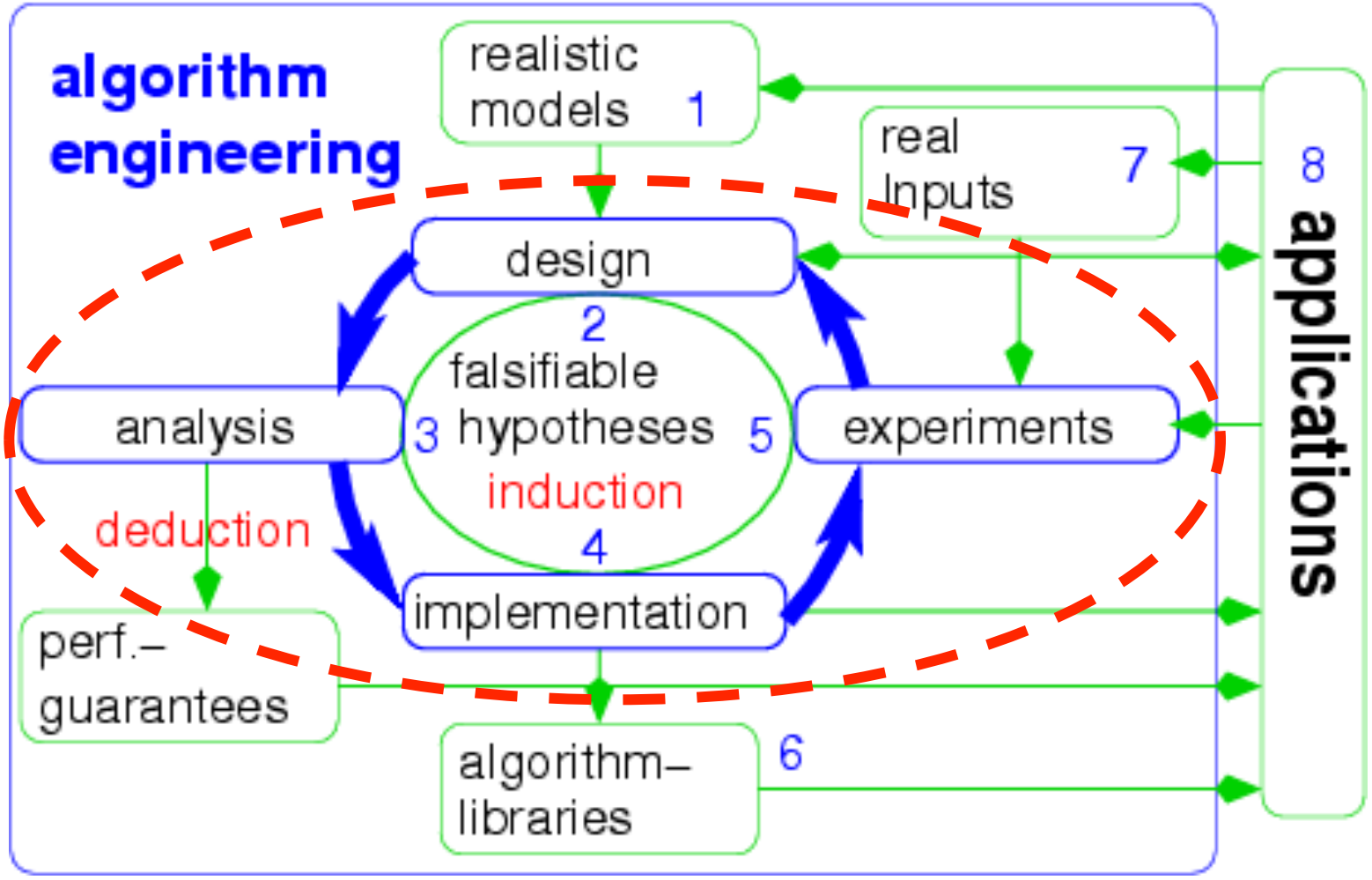
…i.e., nothing works and we don't know why.

Practice is when something works, but we don't know why.

The Algorithm Engineering cycle



Algorithm Engineering



Source: www.algorithm-engineering.de

Few Success Stories

New models of computation (Ladner et al, cache-aware analyses)

Huge speedups in scientific applications (Anderson, Bader, Moret & Warnow, Arge et al.)

New algorithms for shortest paths (Goldberg / Sanders)

Bast, Funke, Sanders, & Schultes. Fast routing in road networks with transit nodes. Science, 316:566, 2007.

Algorithms and data structures for specific classes (Johnson et al, TSP; DIMACS Challenges)

New conjectures, new theorems, new insights (Walsh & Gent Satisfiability, Bentley et al., Bin packing)

Algorithmic Libraries (LEDA / CGAL, Mehlhorn et al….)

Further Readings

Algorithm Engineering issues:

Bernard Moret: “Towards a Discipline of Experimental Algorithmics”

Richard Anderson: “The role of experiment in the theory of algorithms”

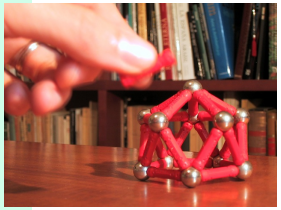
David Johnson: “A theoretician's guide to the experimental analysis of algorithms”

5th DIMACS Challenge Workshop:
Experimental Methodology Day.

Catherine McGeoch: “A Guide to Experimental Algorithmics”. Cambridge University Press

Outline

Dynamic Graph Problems



Methodology & **State of the Art**

Algorithmic Techniques & Experiments

Conclusions

Fully Dynamic APSP

Given a weighted directed graph G = (V,E,w), perform any intermixed sequence of the following operations:

Update(v,w): update edges incident to v [w( )]

Query(x,y): return distance from x to y
(or shortest path from x to y)

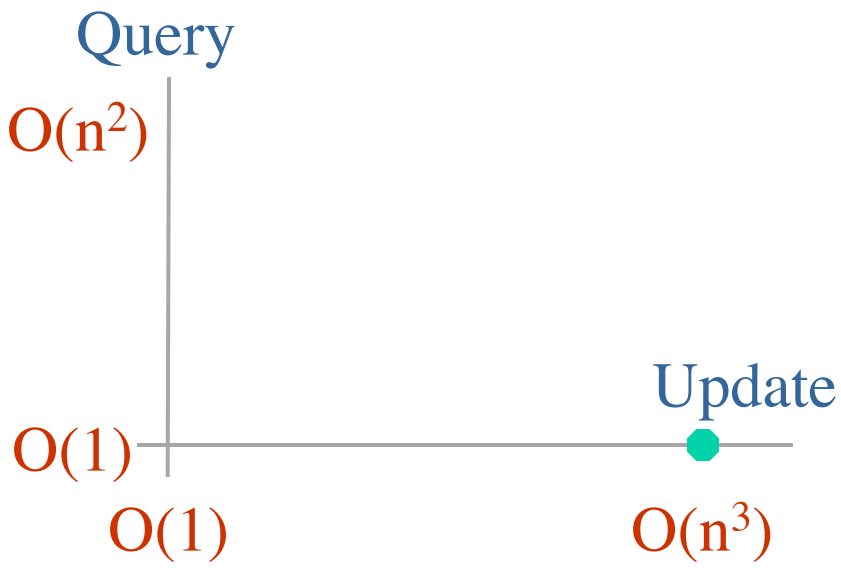
Simple-minded Approaches

Fast query approach	Fast update approach
Keep the solution up to date.	Do nothing on graph.
Rebuild it from scratch at each update.	Visit graph to answer queries.

Dynamic All-Pairs Shortest Paths

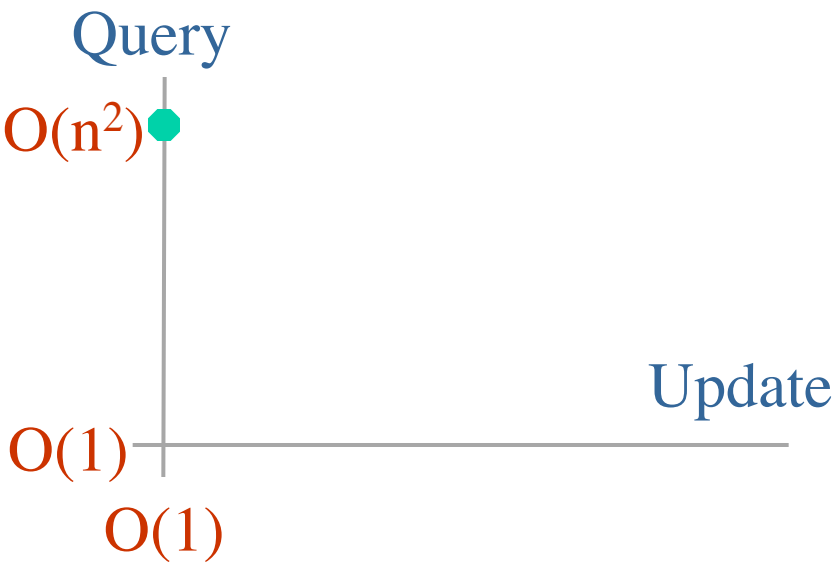
Fast query approach

Rebuild the distance matrix from scratch after each update.



Fast update approach

To answer a query about (x,y), perform a single-source computation from x.



State of the Art

First fully dynamic algorithms date back to the 60's

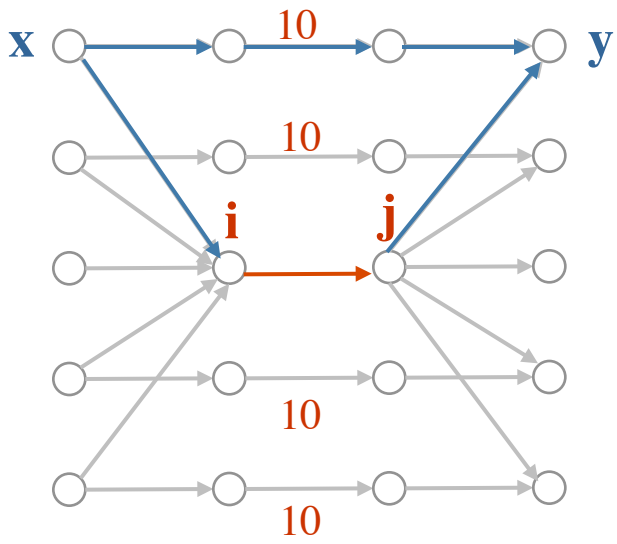
Until 1999, none of them was better in the worst case than recomputing APSP from scratch (~ cubic time!)

	Graph	Weight	Update	Query
Ramalin.&Reps 96	general	real	$O(n^3)$	O(1)
King 99	general	[0,C]	$O(n^{2.5} (C \log n)^{0.5})$	O(1)

- P. Loubal, A network evaluation procedure, ***Highway Research Record 205***, 96-109, **1967**.
- J. Murchland, The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph, REPORT LBS-TNT-26, Transport Network Theory Unit, London Business School, **1967**.
- V. Rodionov, A dynamization of the all-pairs least cost problem, ***USSR Comput. Math. And Math. Phys. 8***, 233-277, **1968**.
- …

Fully Dynamic APSP

Edge insertions (edge cost decreases)



For each pair x,y check whether

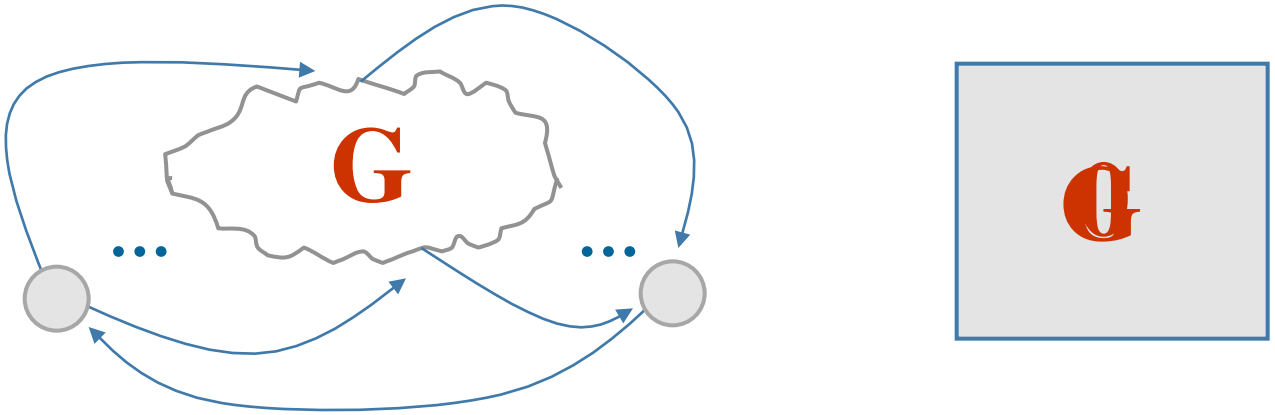
$$D(x,i) + w(i,j) + D(j,y) < D(x,y)$$

Quite easy: $O(n^2)$

Fully Dynamic APSP

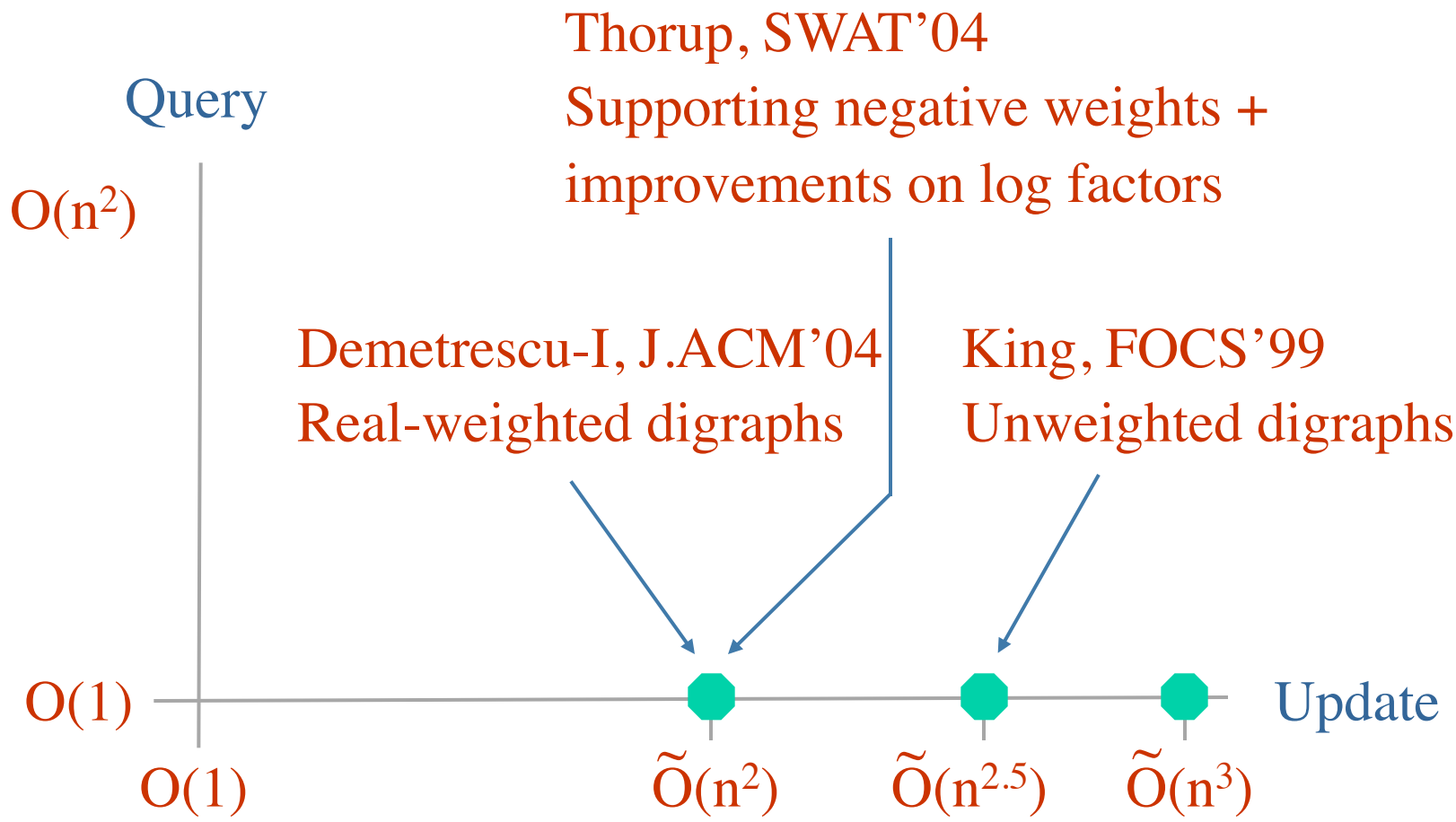
- Edge deletions (edge cost increases)

  Seem the hard operations. Intuition:



- When edge (shortest path) deleted: need info about second shortest path? (3rd, 4th, …)

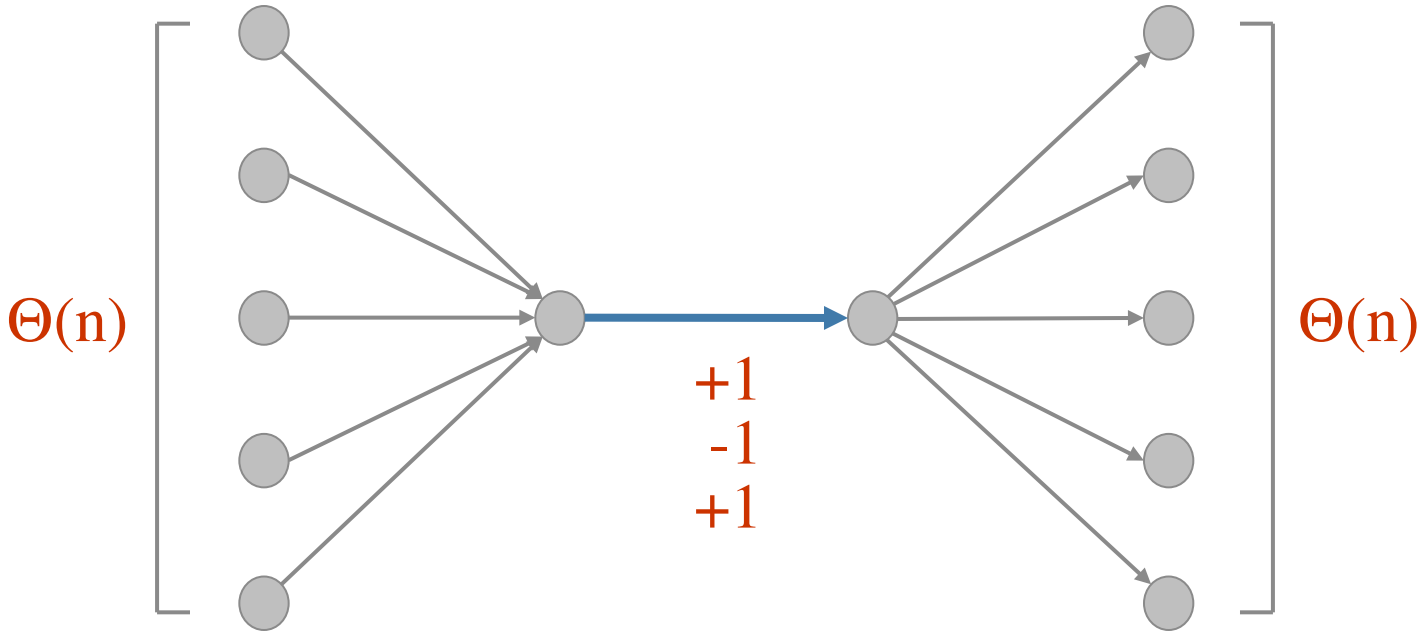
Dynamic APSP



Decremental bounds: Baswana, Hariharan, Sen J.Algs' 07

Approximate dynamic APSP: Roditty, Zwick FOCS' 04

Quadratic Update Time Barrier?



If distances are to be maintained explicitly, any algorithm must pay $\Omega(n^2)$ per update…

Related Problems

Dynamic Transitive Closure (directed graph G)

update	query	authors	notes
$O(n^2 \log n)$	$O(1)$	King, FOCS'99	
$O(n^2)$	$O(1)$	King-Sagert, JCSS '02	*DAGs*
		Demetrescu-I., Algorithmica'08	
		Sankowski, FOCS'04	worst-case
$O(n^{1.575})$	$O(n^{0.575})$	Demetrescu-I., J.ACM'05	*DAGs*
		Sankowski, FOCS'04	
$O(m\, n^{1/2})$	$O(n^{1/2})$	Roditty, Zwick, SIAM J. Comp.'08	
$O(m+n \log n)$	$O(n)$	Roditty, Zwick, FOCS'04	

Decremental bounds: Baswana, Hariharan, Sen, J.Algs.'07

Other Problems

Dynamic Connectivity (undirected graph G)

update	query	authors
$O(\log^3 n)$	$O(\log n/\log\log n)$	Henzinger, King, J. ACM '99 (randomized)
$O(n^{1/3} \log n)$	$O(1)$	Henzinger, King, SIAM J. Comp.' 01
$O(\log^2 n)$	$O(\log n/\log\log n)$	Holm, de Lichtenberg, Thorup, J.ACM' 01

Lower bounds:

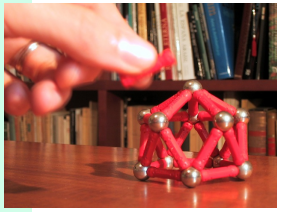
$\Omega(\log n)$ update Patrascu & Demaine, SIAM J.Comp' 06

$\Omega((\log n / \log\log n)^2)$ update Patrascu & Tarnita, TCS' 07

Outline

Dynamic Graph Problems

Methodology & State of the Art



Algorithmic Techniques & Experiments

Conclusions

Algorithmic Techniques

Will focus on techniques for path problems.

Running examples: shortest paths/transitive closure

Main Ingredients

Long paths property

Output bounded

Decremental BFS

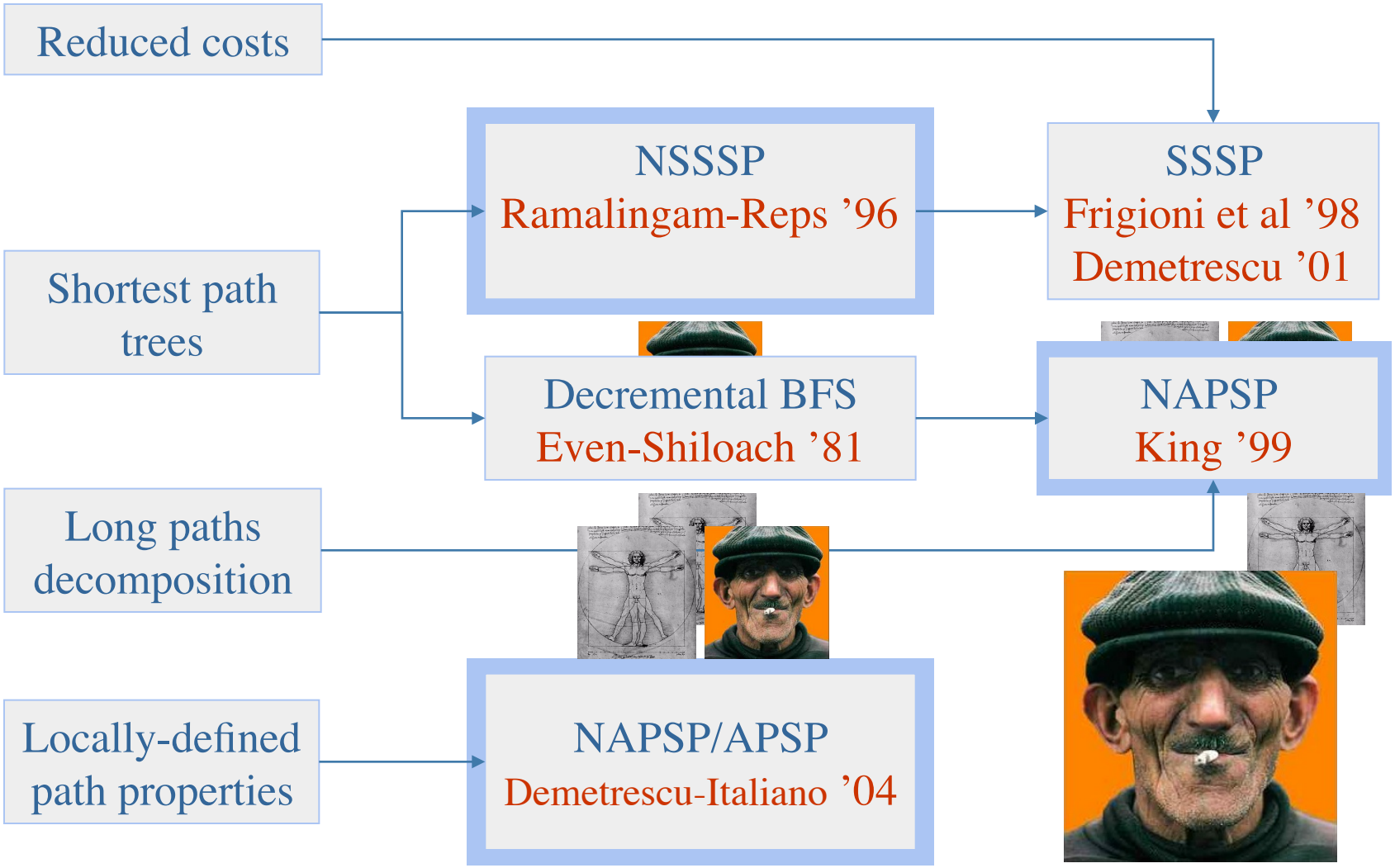
Path decompositions

Locally-defined path properties

Counting

Algebraic techniques

Dynamic shortest paths: roadmap



Main Ingredients

Long paths property

Output bounded

Decremental BFS

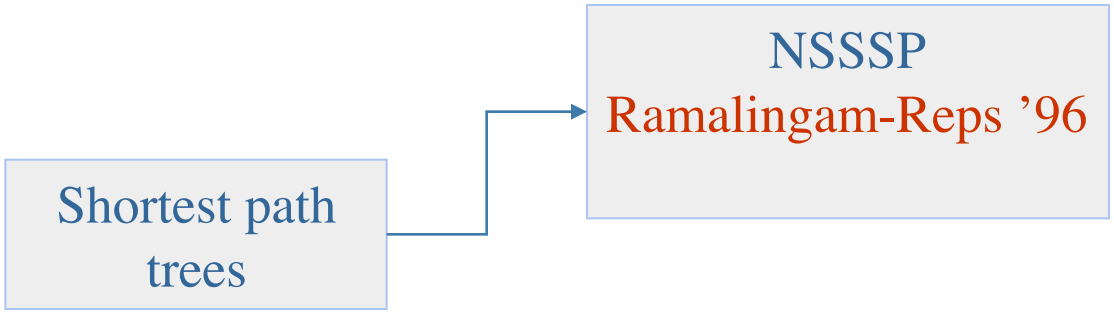
Path decompositions

Locally-defined path properties

Counting

Algebraic techniques

Dynamic shortest paths: roadmap



Fully Dynamic SSSP

Let: $G = (V,E,w)$ weighted directed graph

$w(u,v)$ weight of edge (u,v)

$s \in V$ source node

Perform intermixed sequence of operations:

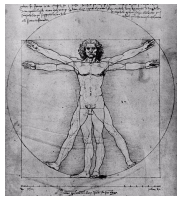
`Increase(u,v,ε):`	Increase weight w(u,v) by ε
`Decrease(u,v,ε):`	Decrease weight w(u,v) by ε
`Query(v):`	Return distance (or sh. path) from s to v in G

Ramalingam & Reps' approach

Maintain a shortest paths tree throughout the sequence of updates

Querying a shortest paths or distance takes optimal time

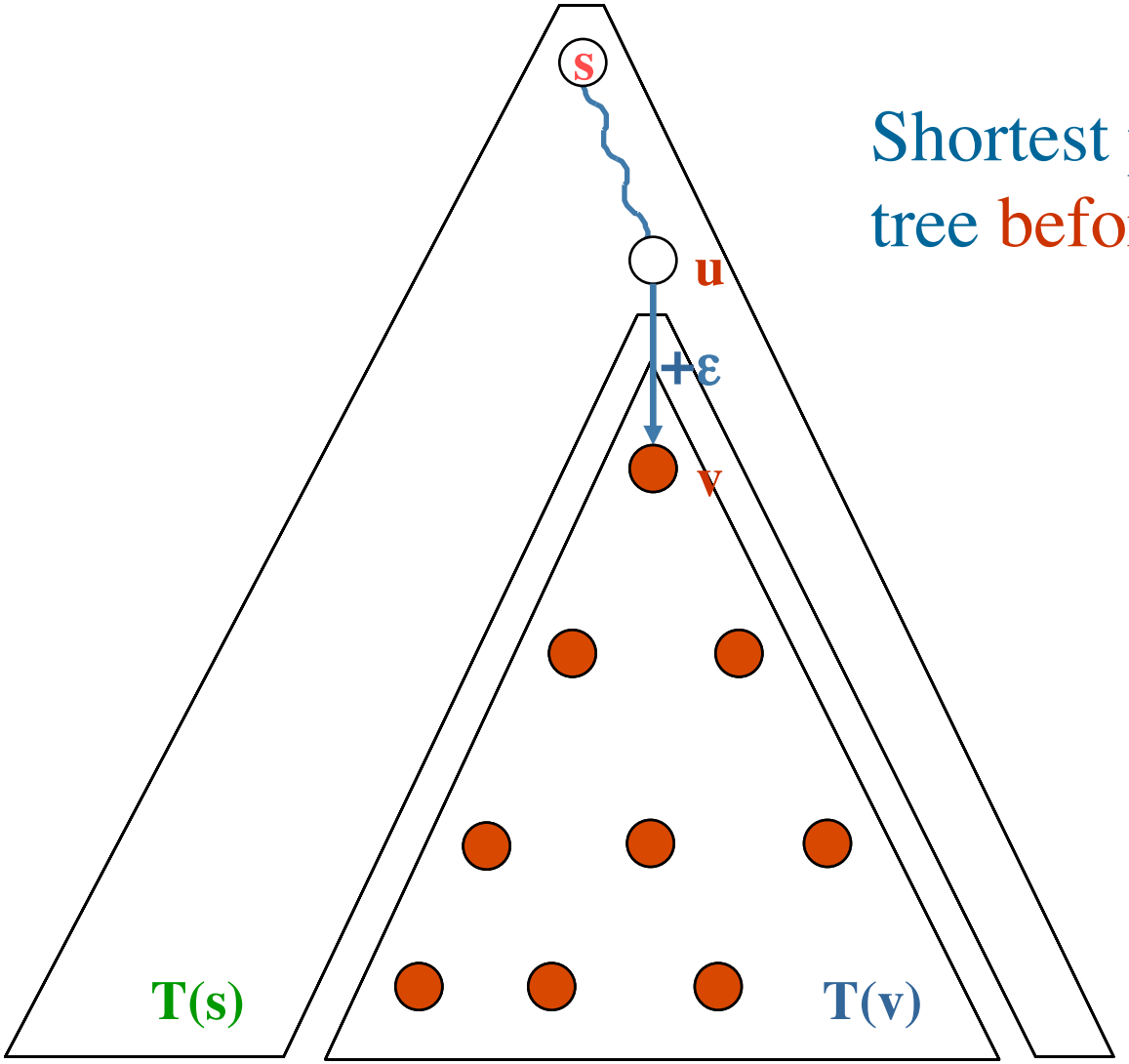
Update operations work only on the portion of tree affected by the update



Each update may take, in the worst case, as long as a static SSSP computation!

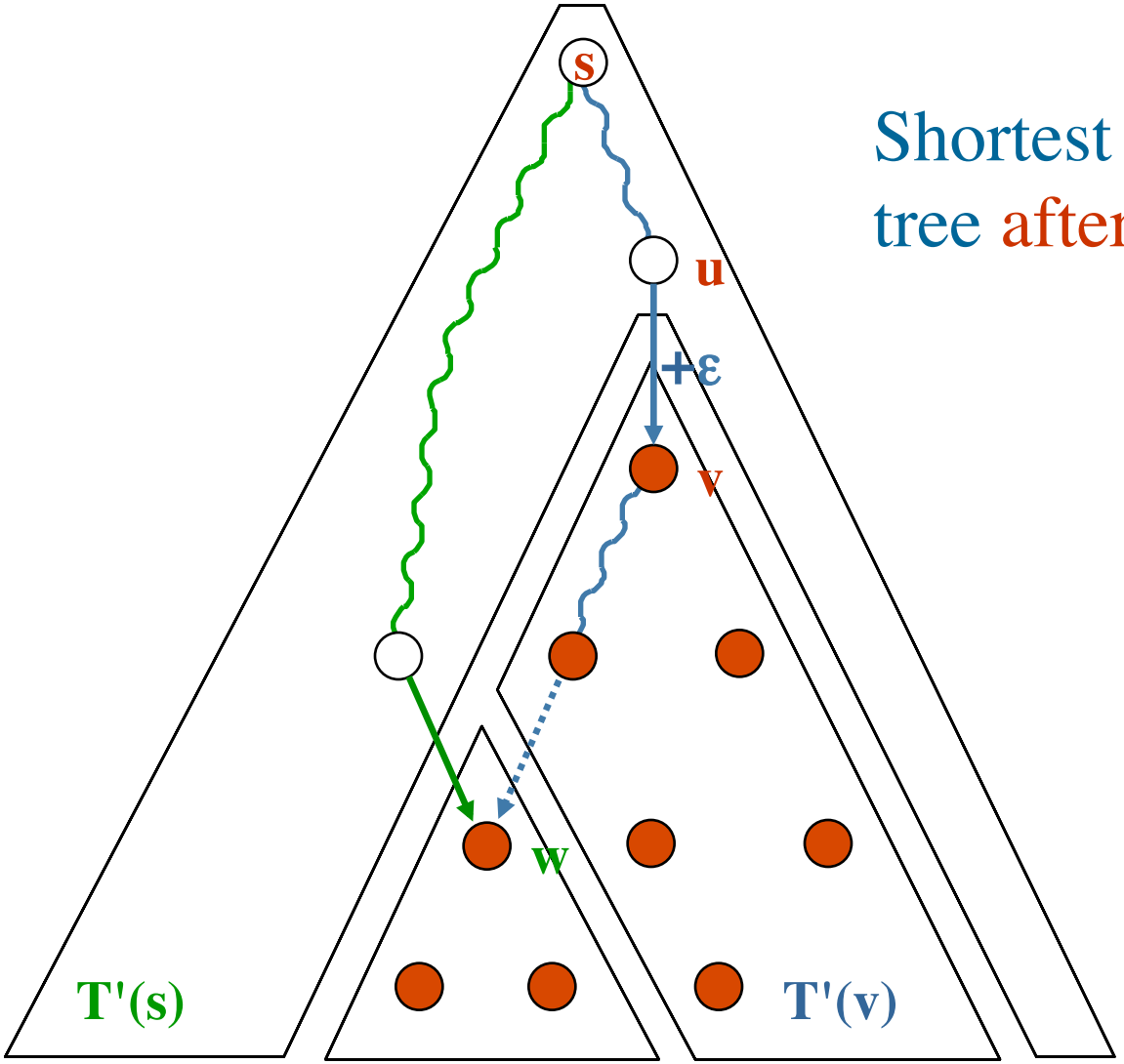
But very efficient in practice

Increase(u,v,ε)



Shortest paths
tree before the update

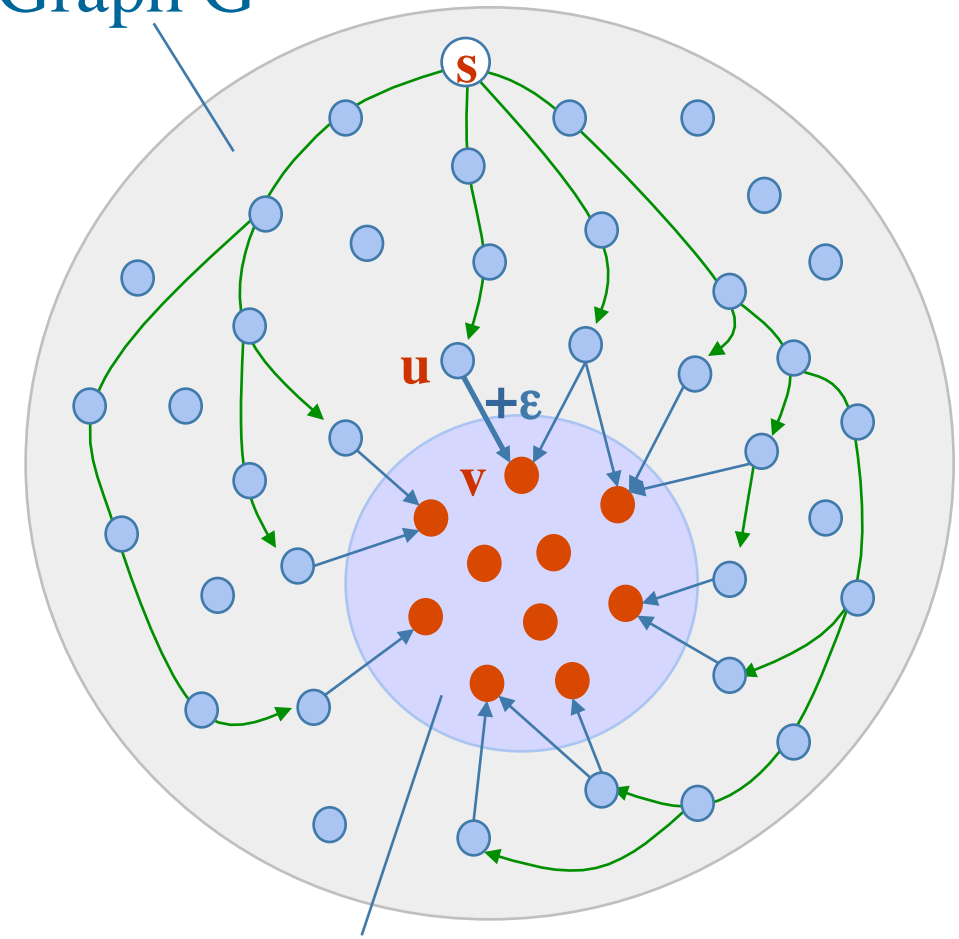
Increase(u,v,ε)



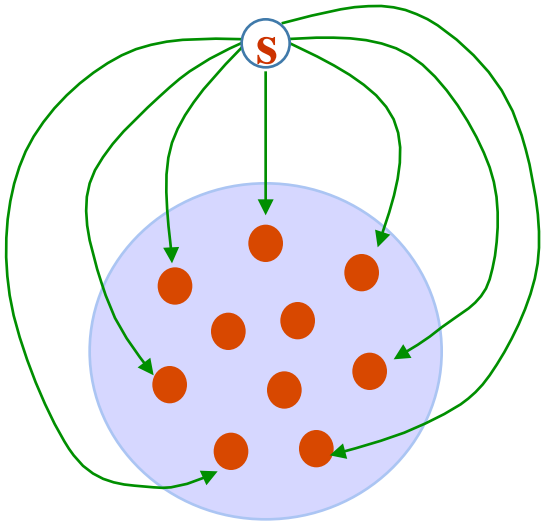
Shortest paths
tree after the update

Ramalingam & Reps' approach

Graph G



Perform SSSP
only on the subgraph
and source s



Subgraph induced by vertices in T(v)

Main Ingredients

Long paths property

Output bounded

Decremental BFS

Path decompositions

Locally-defined path properties

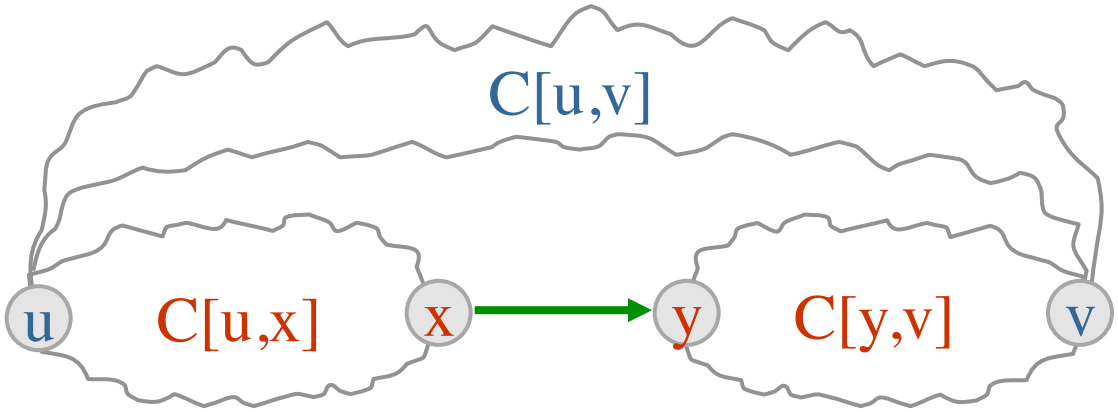
Counting

Algebraic techniques

Path Counting [King/Sagert, JCSS'02]

Dynamic Transitive Closure in a DAG

Idea: count # distinct paths for any vertex pair



$\forall$ u,v: $C[u,v] \leftarrow C[u,v] + C[u,x] \cdot C[y,v]$ $O(n^2)$

$\forall$ u,v: $C[u,v] \leftarrow C[u,v] - C[u,x] \cdot C[y,v]$ $O(n^2)$

Problem: counters as large as 2^n

Solution: use arithmetic modulo a random prime

Arithmetic mod primes

Reduce wordsize to 2c log n: pick random prime p between n^c and n^{c+1} and perform arithmetic mod p

O(1) time with wordsize O(log n)
But false 0s (x mod p = 0 but $x \neq 0$)

Lemma. If $O(n^k)$ arithmetic computations involving numbers $\leq 2^n$ are performed mod random p of value $\Theta(n^c)$, then probability of false 0 is $O(1/n^{c-k-1})$

(As # ops with particular prime increases, so does chance of getting false 0s)

Arithmetic mod primes

Lemma. If $O(n^k)$ arithmetic computations involving numbers $\leq 2^n$ are performed mod random p of value $\Theta(n^c)$, then probability of false 0 is $O(1/n^{c-k-1})$

Proof. Let $x \leq 2^n$. There are $O(n / \log n)$ prime divisors of value $\Theta(n^c)$ which divide x.
So, there are $O(n^{k+1}/ \log n)$ prime divisors of any of the numbers generated.
By Prime Number Theorem, approx. $\Theta(n^c / \log n)$ primes of value $\Theta(n^c)$.
Hence probability that random prime of value $\Theta(n^c)$ divides any of the numbers generated is $O(1/n^{c-k-1})$.

Arithmetic mod primes

Reduce wordsize to 2c log n: pick random prime p between n^c and n^{c+1} and perform arithmetic mod p

O(1) time with wordsize O(log n)
But false 0s (x mod p = 0 but $x \neq 0$)

Lemma. If $O(n^k)$ arithmetic computations involving numbers $\leq 2^n$ are performed mod random p of value $\Theta(n^c)$, then probability of false 0 is $O(1/n^{c-k-1})$

Choose new prime every n updates and reinitialize all data structures (k = 3, thus enough $c \geq 5$)

Dynamic Transitive Closure [King/Sagert, JCSS' 02]

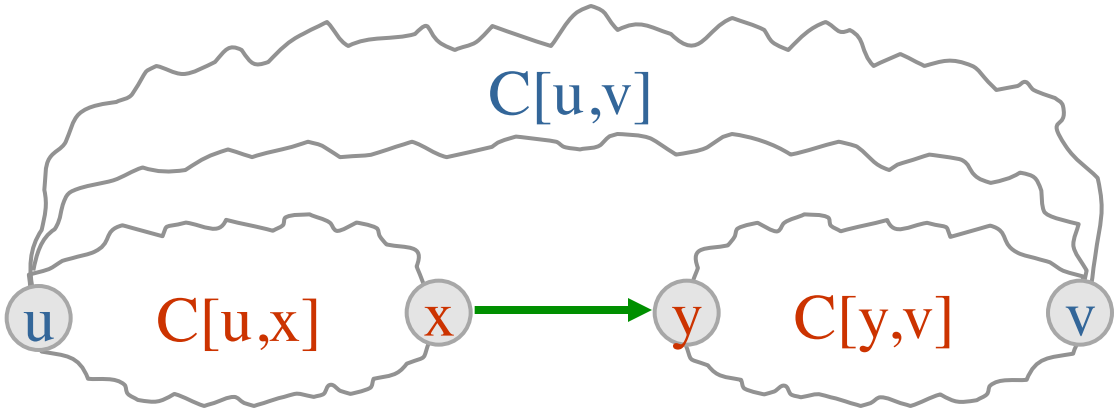
Update: $O(n^2)$ worst-case time

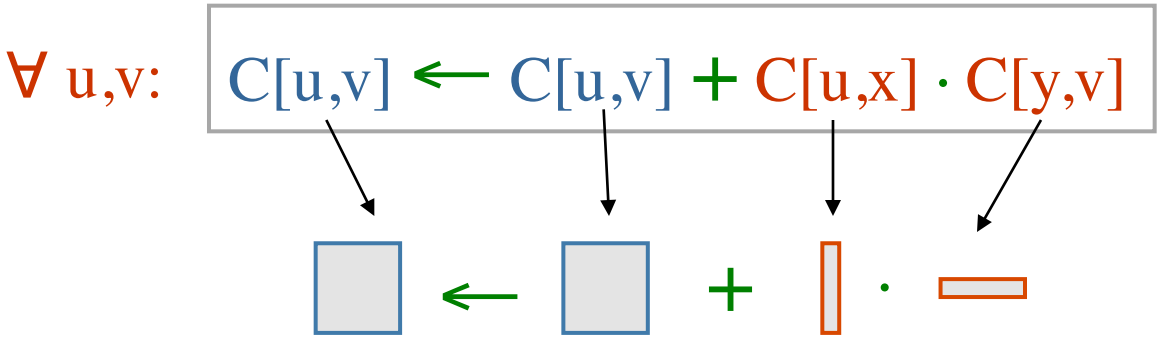
Query: $O(1)$ worst-case time

Works for directed acyclic graphs.
Randomized, one-sided error.

Can we trade off query time for update time?

Looking from the matrix viewpoint





Maintaining dynamic integer matrices

Given a matrix M of integers, perform any intermixed sequence of the following operations:

`Update`(J,I): $M \leftarrow M + J \cdot I$ $O(n^2)$

`Query`(i,j): return M[i,j] $O(1)$

Maintaining dynamic integer matrices

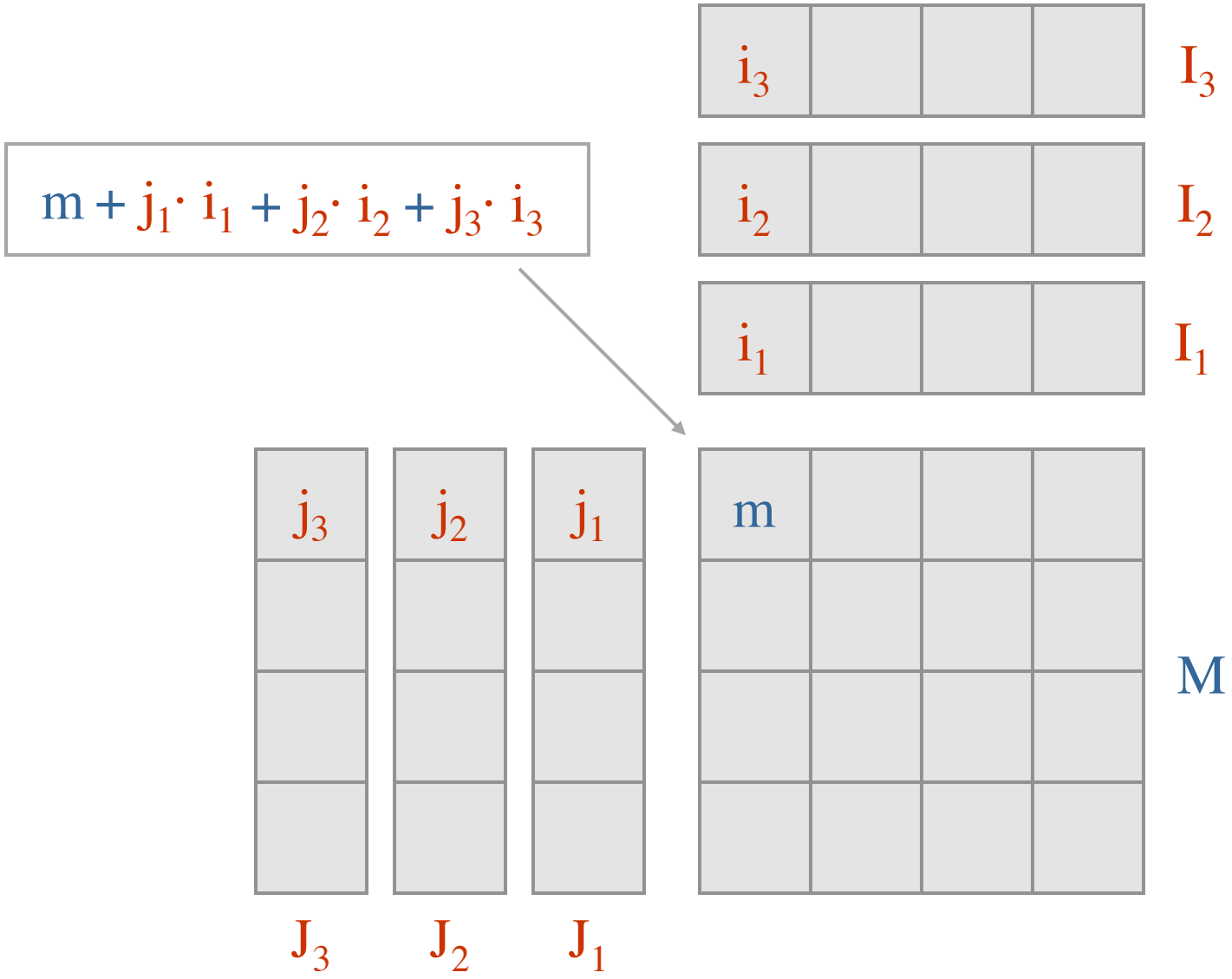
How can we trade off operations?

Lazy approach: buffer at most n^{ε} updates

Global rebuilding every n^{ε} updates

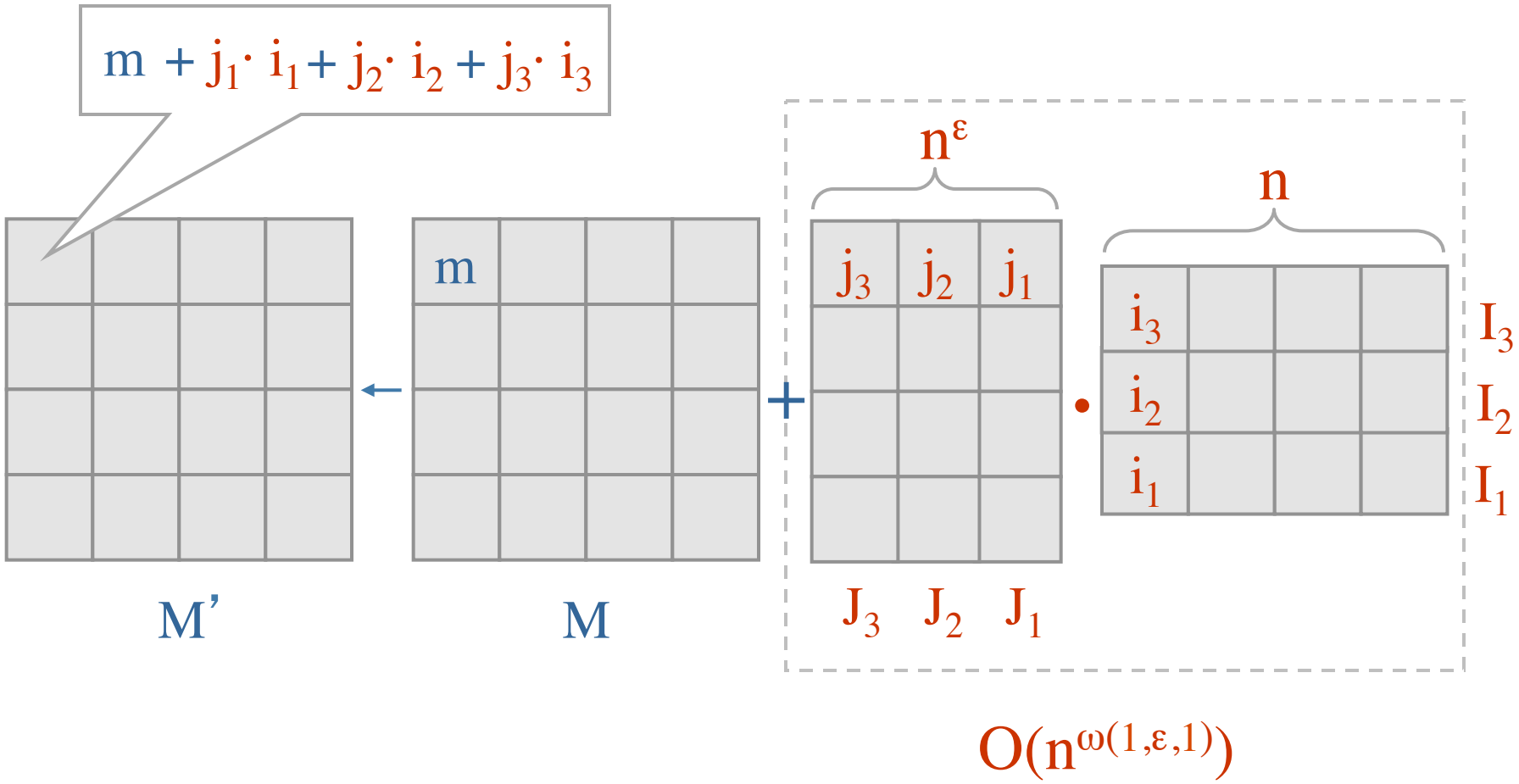
Rebuilding done via matrix multiplication

Maintaining dynamic integer matrices



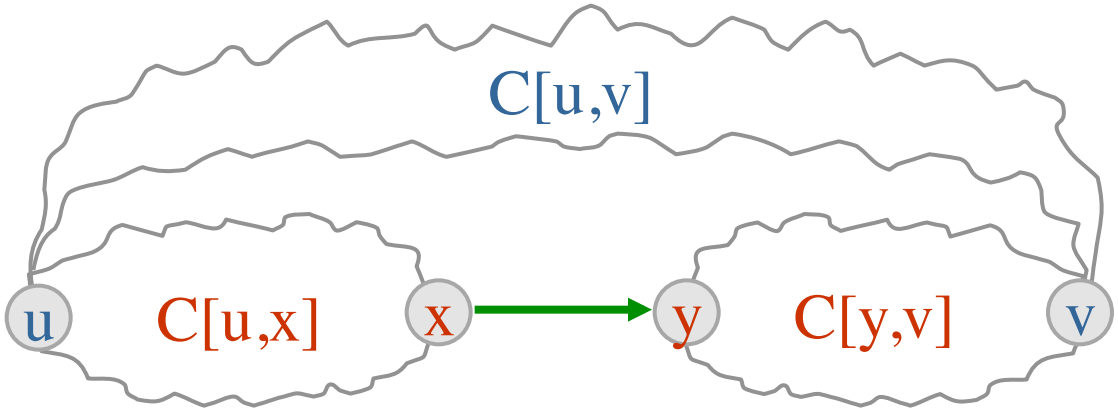
Maintaining dynamic integer matrices

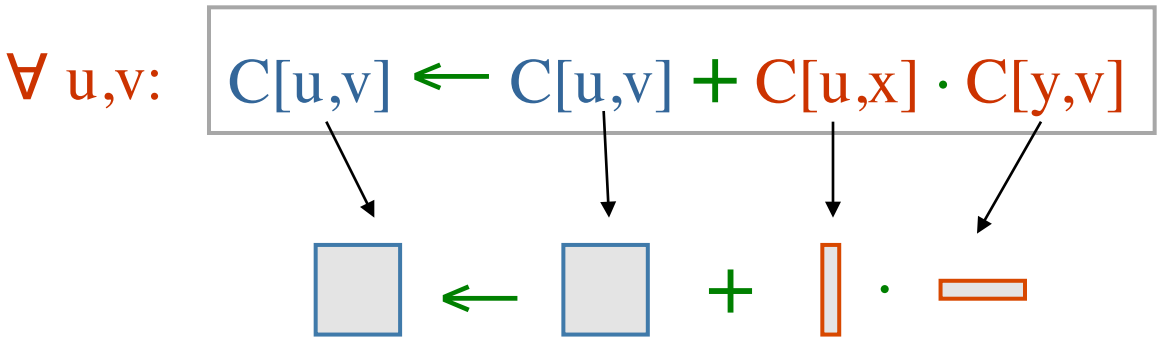
Global rebuilding every n^{ε} updates



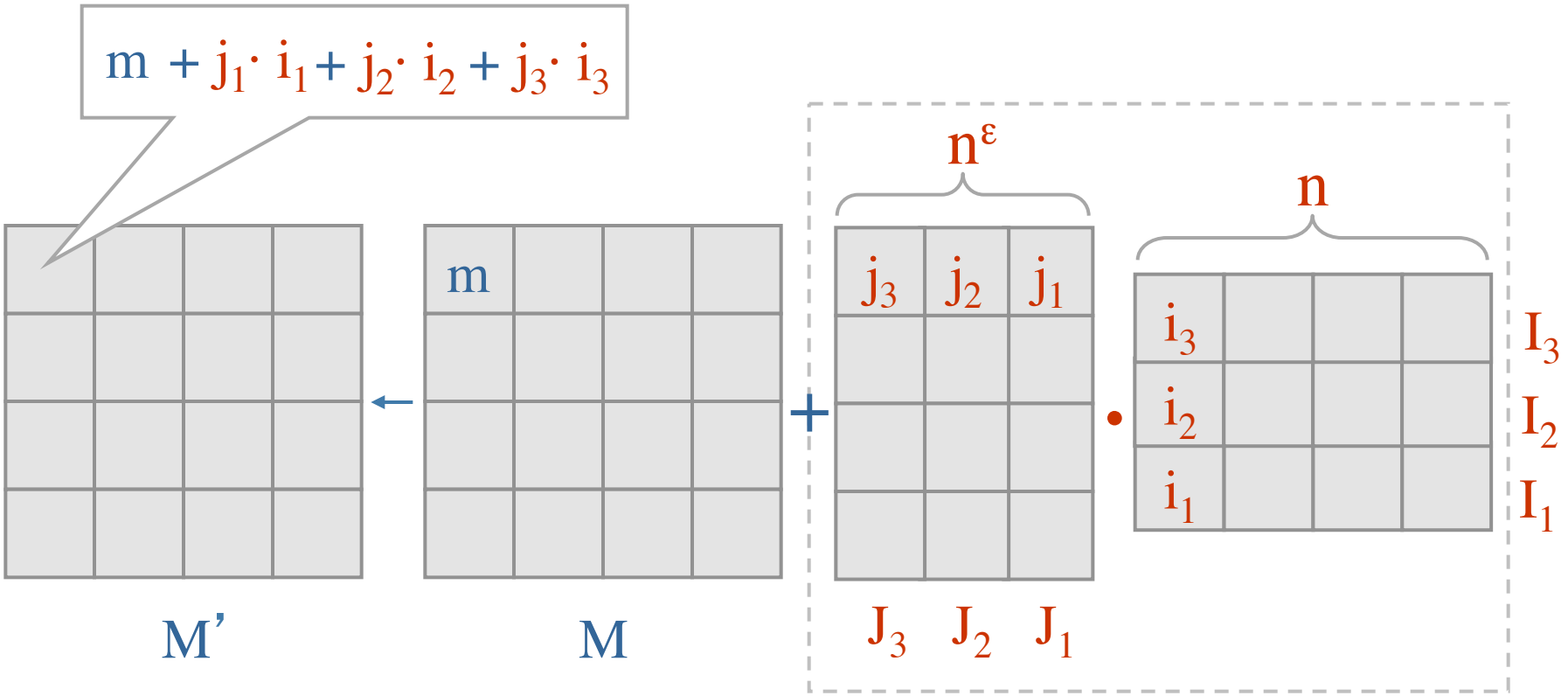
$O(n^{\omega(1,\varepsilon,1)})$

Back to Dynamic Transitive Closure





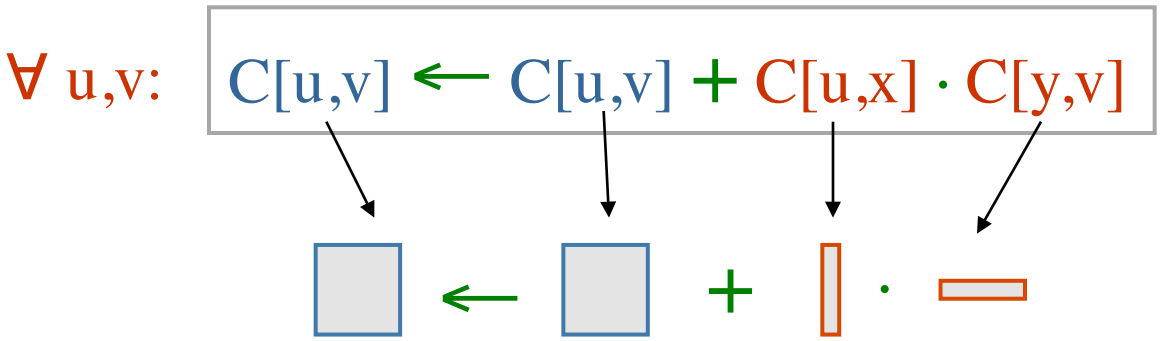
Query Time



Total Query time $O(n^{\varepsilon})$

Update Time

1. Compute C[u,x] and C[y,v] for any u,v

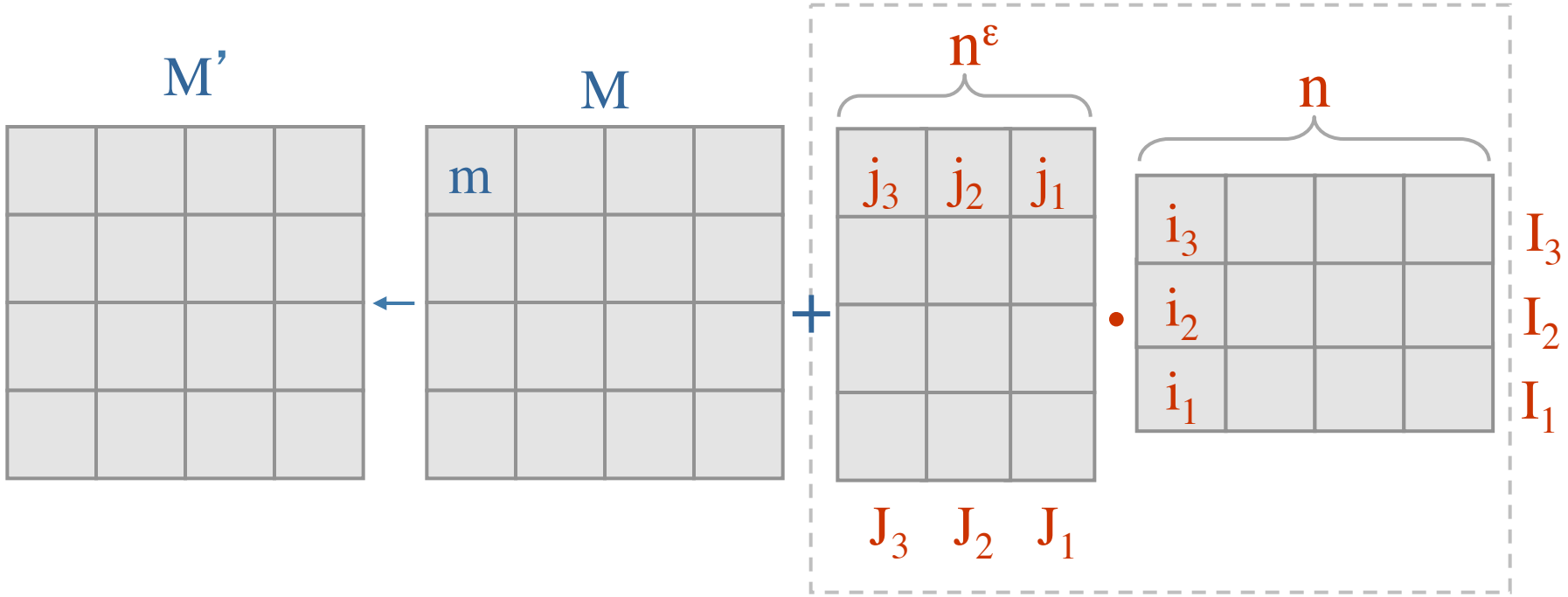


Carried out via O(n) queries

Time: $O(n^{1+\varepsilon})$

Update Time

2. Global rebuild every n^{ε} updates



Carried out via (rectangular) matrix multipl.

Amortized time: $O(n^{\omega(1,\varepsilon,1)} / n^{\varepsilon})$

Dynamic Transitive Closure [Demetrescu-I., J.ACM05]

Update: $O(n^{\omega(1,\varepsilon,1)-\varepsilon}+n^{1+\varepsilon})$

Query: $O(n^{\varepsilon})$

for any $0 < \varepsilon < 1$

Find ε such that $\omega(1,\varepsilon,1) = 1+2\varepsilon$

Best bound for rectangular matrix multiplication [Huang/Pan98]

$\varepsilon < 0.575$

Update: $O(n^{1.575})$ worst-case time

Query: $O(n^{0.575})$ worst-case time

Main Ingredients

Long paths property

Output bounded

Decremental BFS

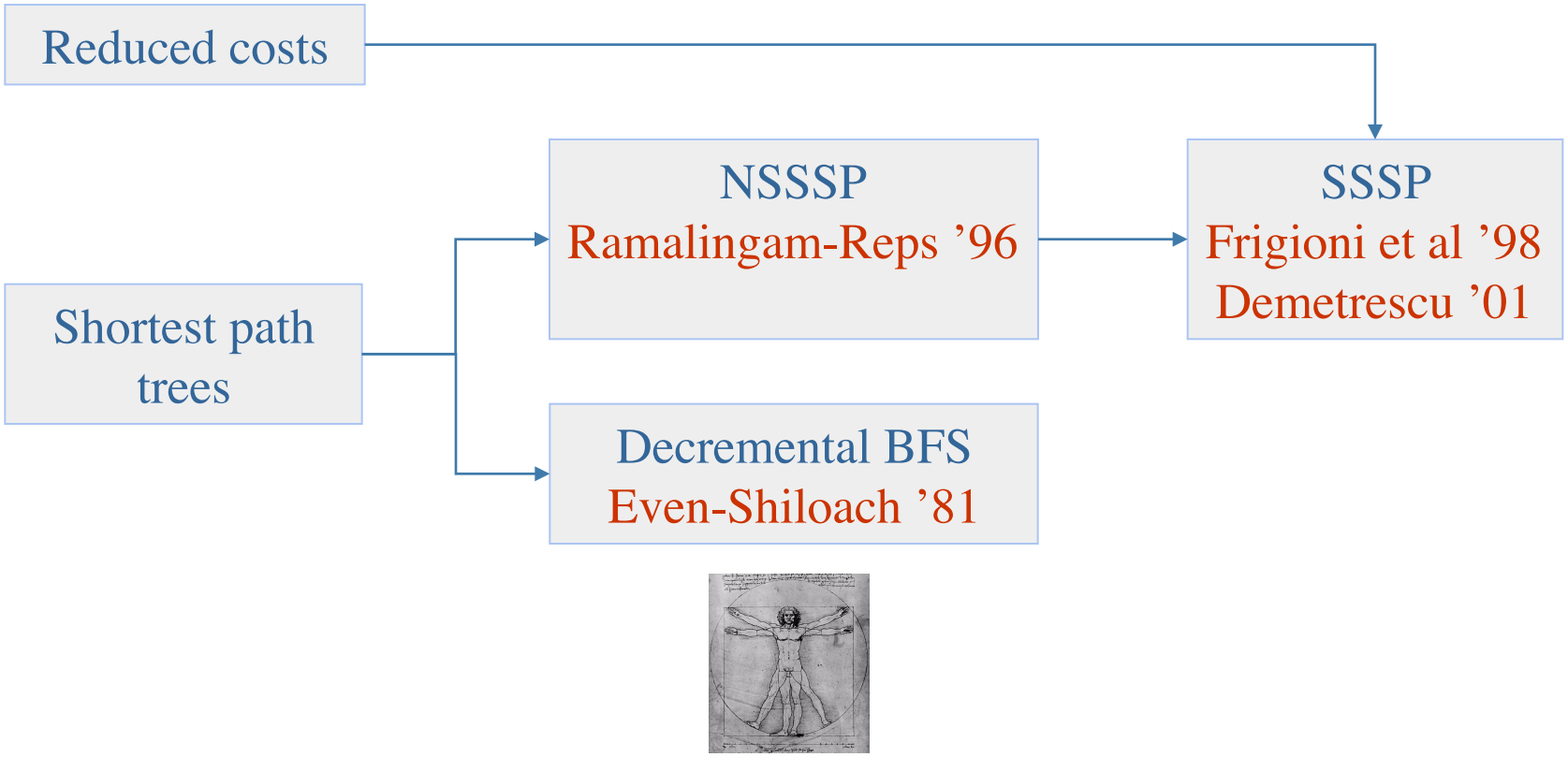
Path decompositions

Locally-defined path properties

Counting

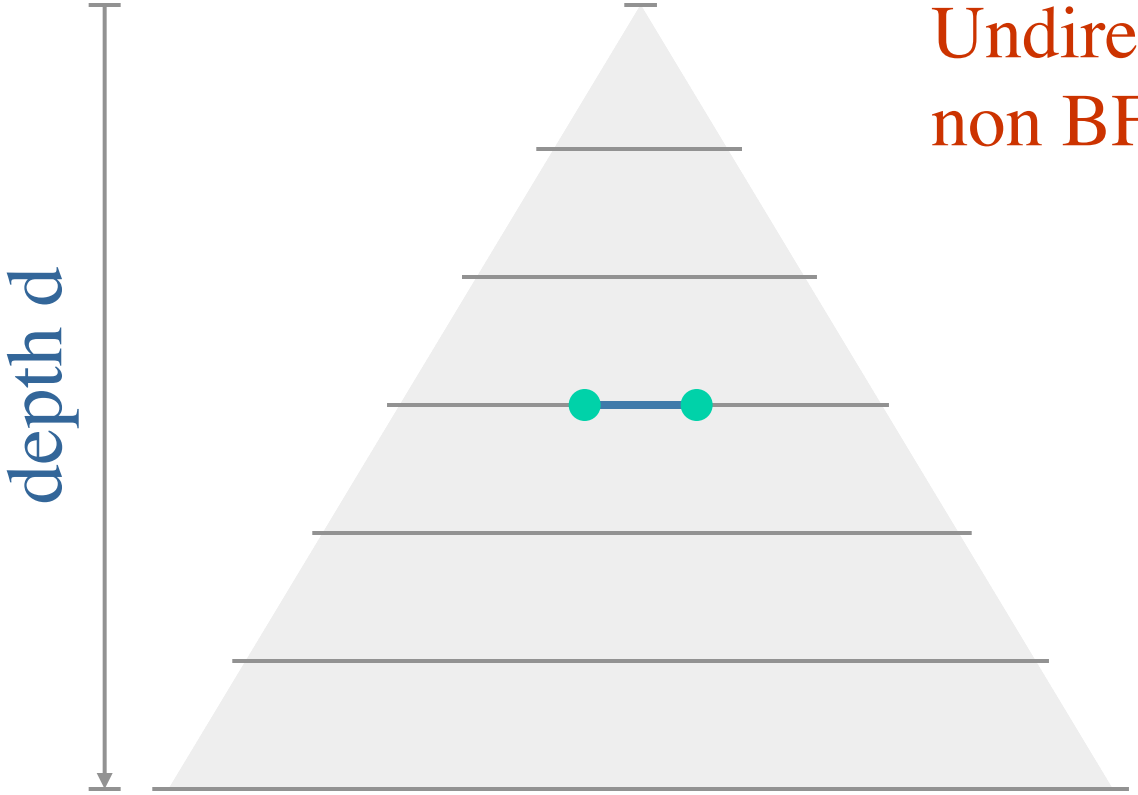
Algebraic techniques

Dynamic shortest paths: roadmap



Decremental BFS [Even-Shiloach, JACM'81]

Maintain BFS levels under deletion of edges

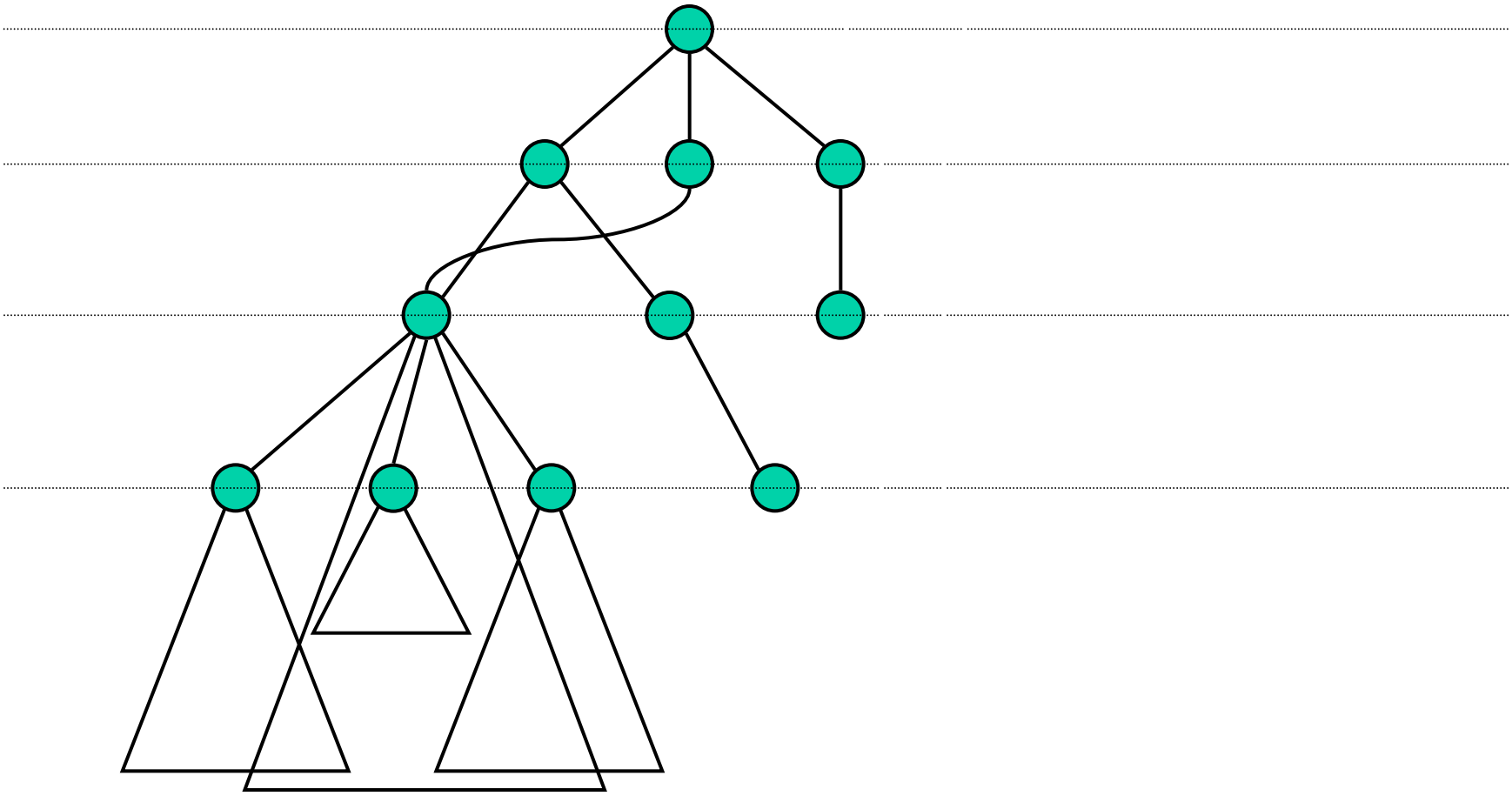


Undirected graphs:
non BFS-tree edges can be

either between two consecutive levels

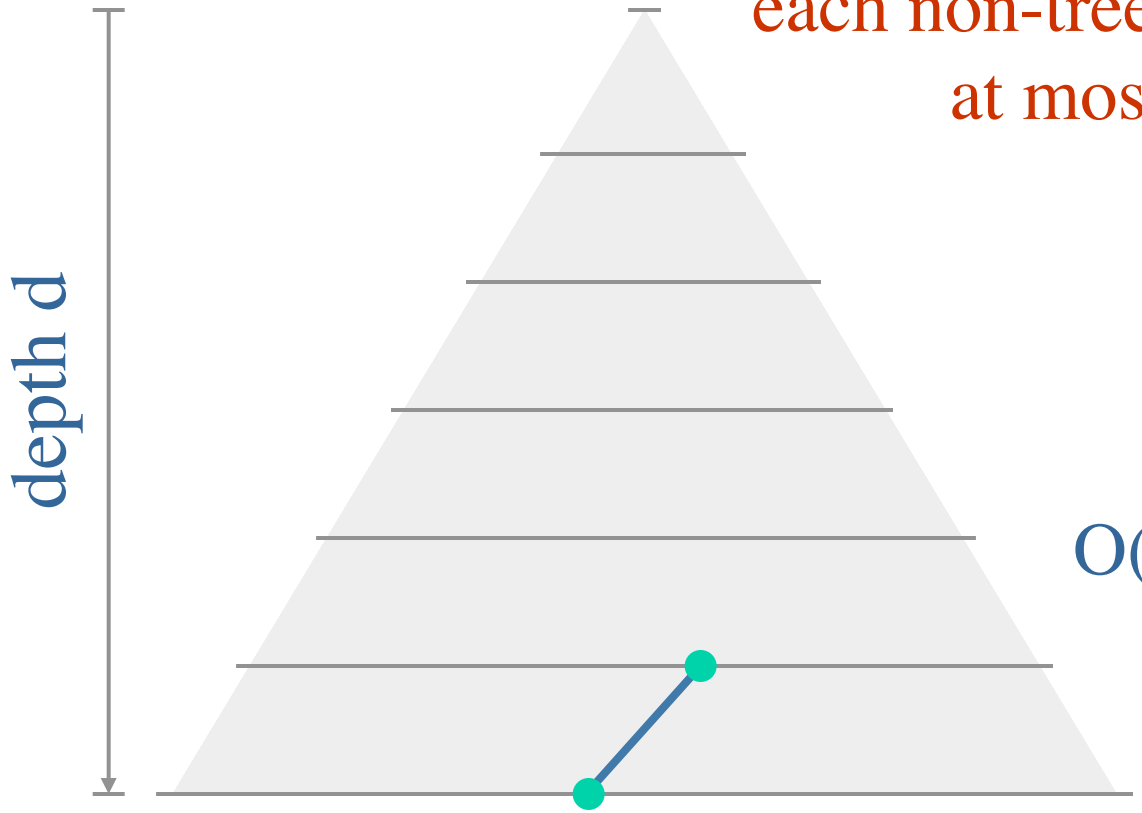
or at the same level

Decremental BFS [Even-Shiloach, JACM'81]



Decremental BFS [Even-Shiloach, JACM'81]

This implies that during deletion of edges

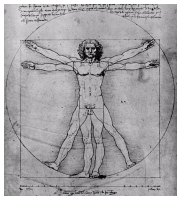


each non-tree edge can fall down
at most 2d times overall…

O(md) total time
over any sequence

O(d) time per deletion
(amortized over
Ω(m) deletions)

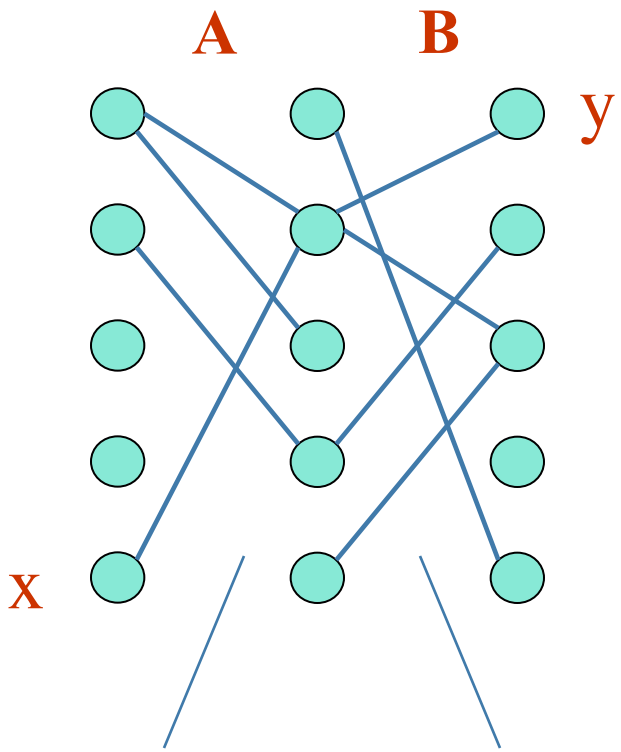
Can we do better than O(mn)?



Roditty and Zwick [2011] have shown two reductions:

Boolean matrix multiplication	⇨	(off-line) decremental undirected BFS
Weighted (static) undirected APSP	⇨	(off-line) decremental undirected SSSP

Matrix mult. ⇨ Decremental BFS



A and B Boolean matrices

Wish to compute C=A·B

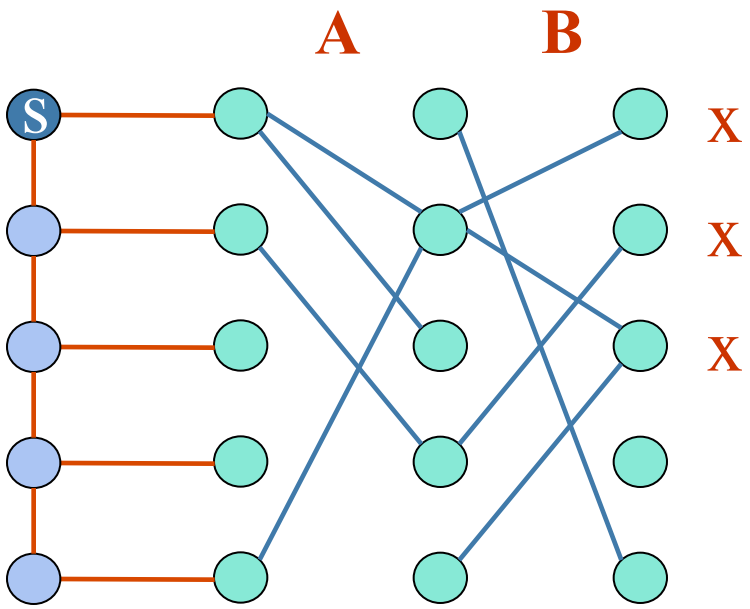
C[x,y]=1 iff there is z such that A[x,z]=1 and B[z,y]=1

C[x,y]=1 iff path of length 2 between x on first layer and y on last layer

Bipartite graph with an edge (x,y) for each A[x,y]=1

Bipartite graph with an edge (x,y) for each B[x,y]=1

Matrix mult. ⇨ Decremental BFS



C

1	0	1	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	0

n deletions and n^2 queries

Decremental BFS in o(mn) total time would imply Boolean matrix multiplication in o(mn)

More details in

Decremental BFS:
[Even-Shiloach'81]
S. Even and Y. Shiloach,
An On-line Edge Deletion Problem,
J. Assoc. Comput. Mach, Vol. 28, pp. 1-4, 1981

Reductions to decremental BFS:
[Roditty-Zwick'11]
Liam Roditty, Uri Zwick,
On dynamic shortest paths problems
Algorithmica 61(2): 389-401 (2011).

Main Ingredients

Long paths property

Output bounded

Decremental BFS

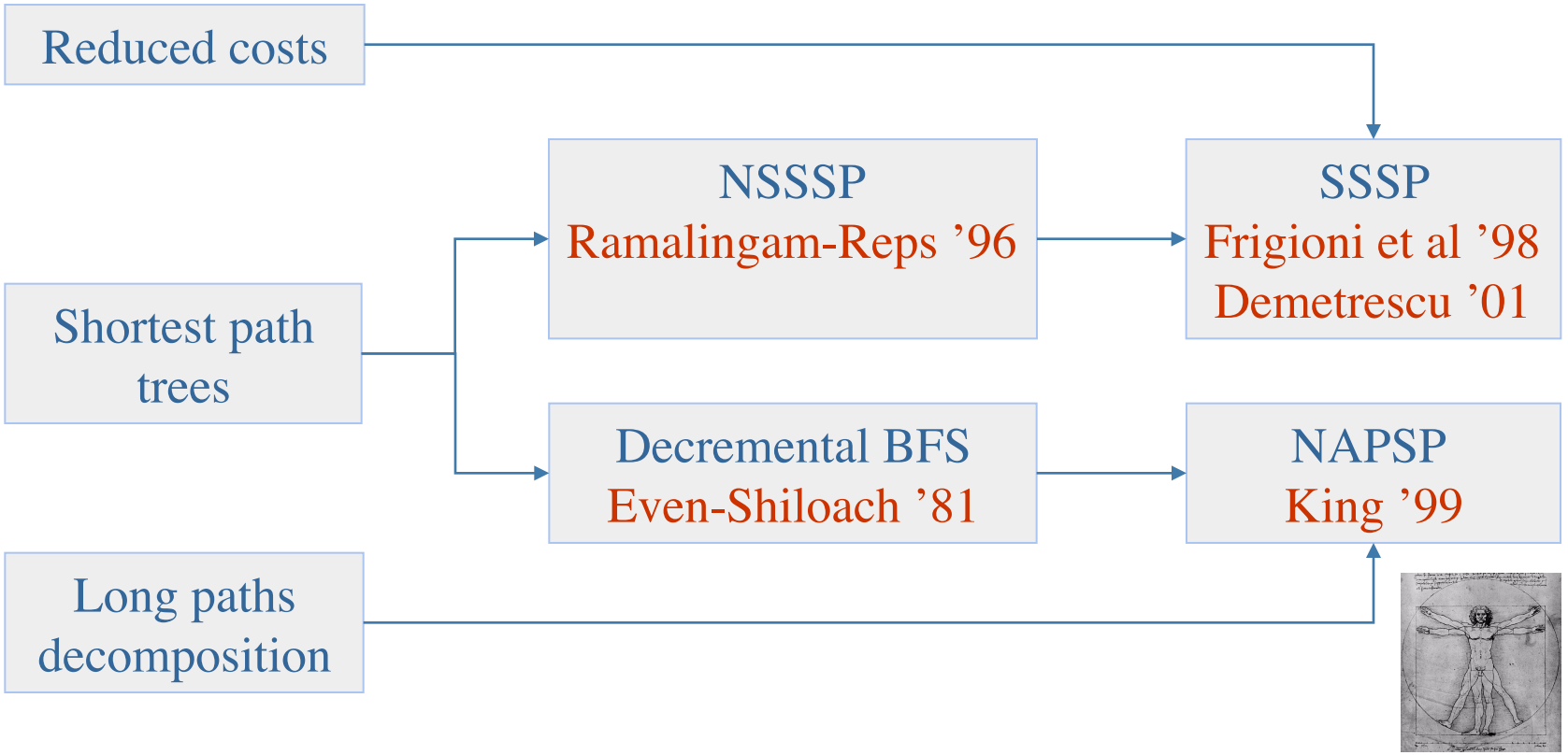
Path decompositions

Locally-defined path properties

Counting

Algebraic techniques

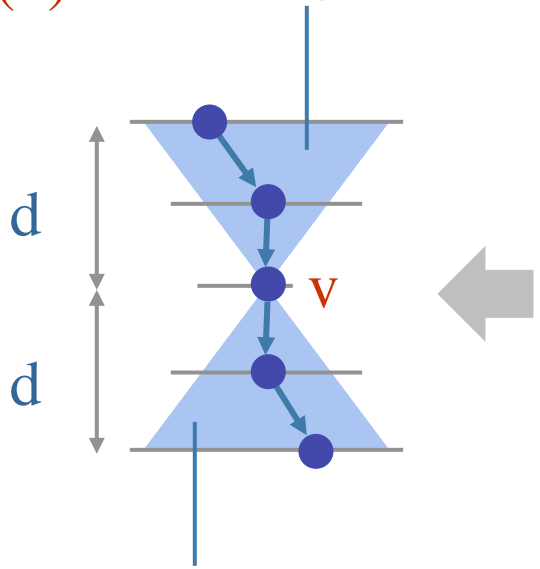
Dynamic shortest paths: roadmap



Make decremental fully dynamic

For each vertex v:

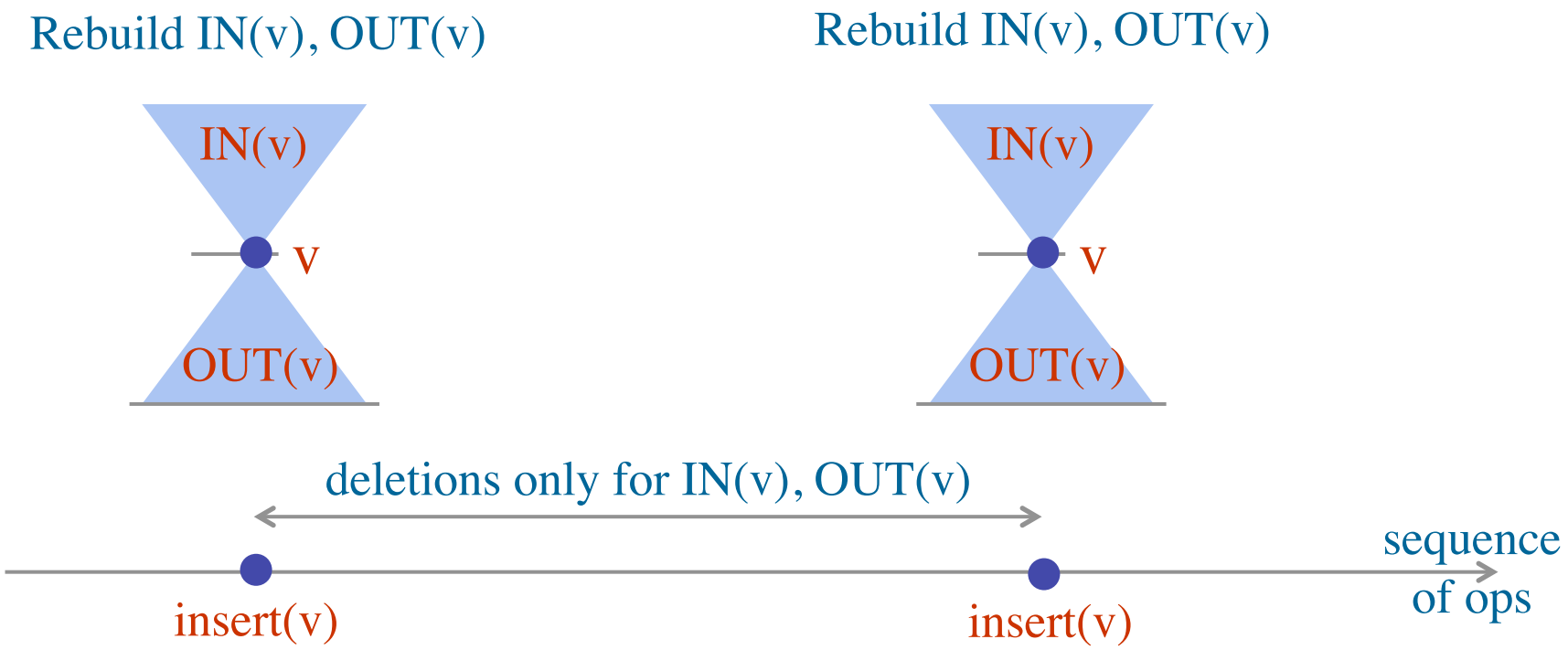
IN(v) maintained as a decremental BFS tree



Building block:
pair of IN/OUT trees
keeps track of all paths of length ≤ d passing through v

OUT(v) maintained as a decremental BFS tree

Make decremental fully dynamic

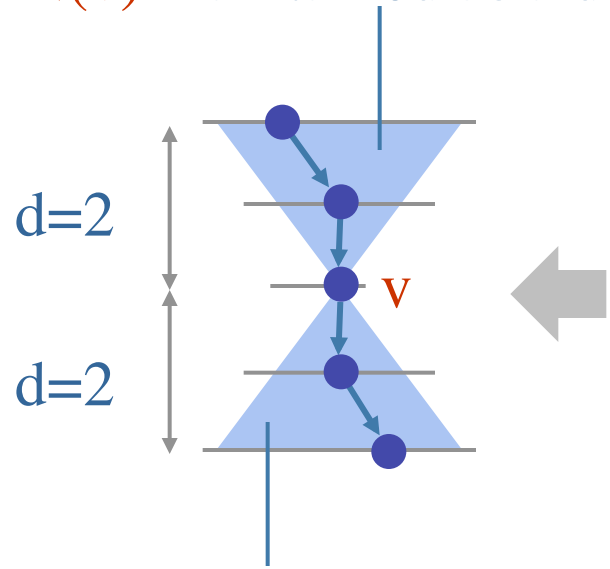


Total cost for rebuilding IN, OUT trees + deleting edges in between: O(md)
This is charged to insert(v)

Dynamic Transitive Closure [King, FOCS'99]

Ingredients: Decremental BFS + Doubling decomposition

IN(v) maintained as a decremental BFS tree



Building block:
pair of IN/OUT trees
keeps track of all paths of length ≤ 2 passing through v

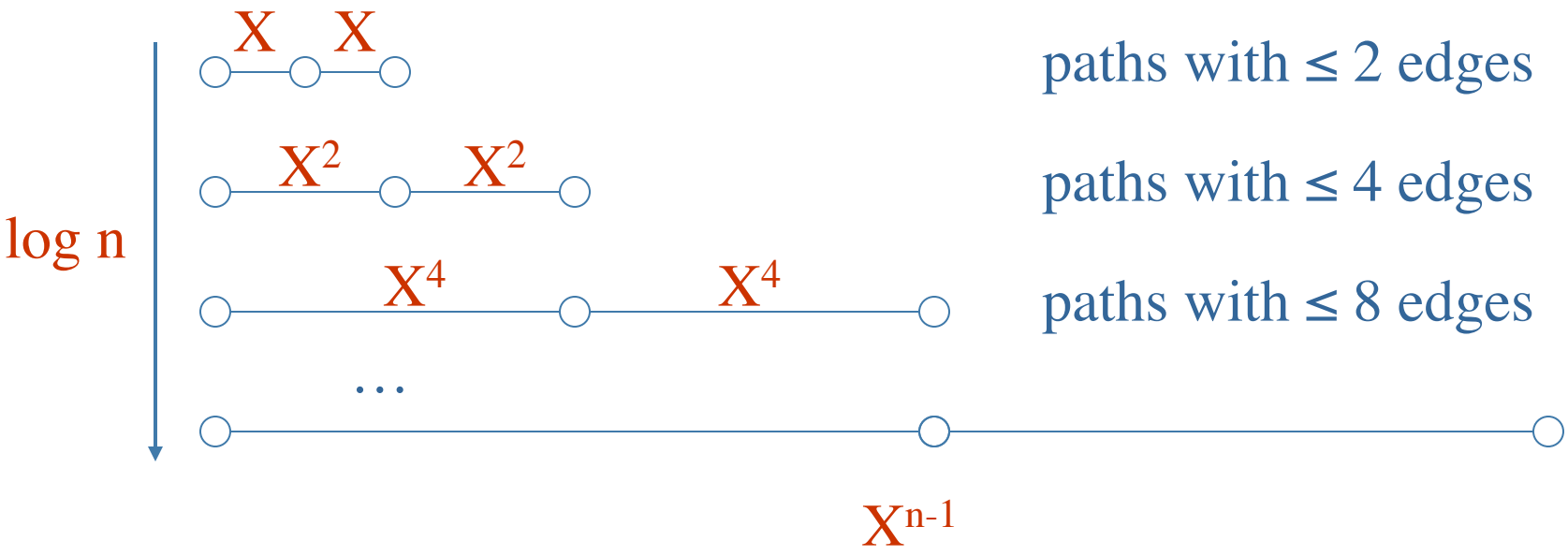
OUT(v) maintained as a decremental BFS tree

Total cost for building the two trees + deleting all edges: O(m)

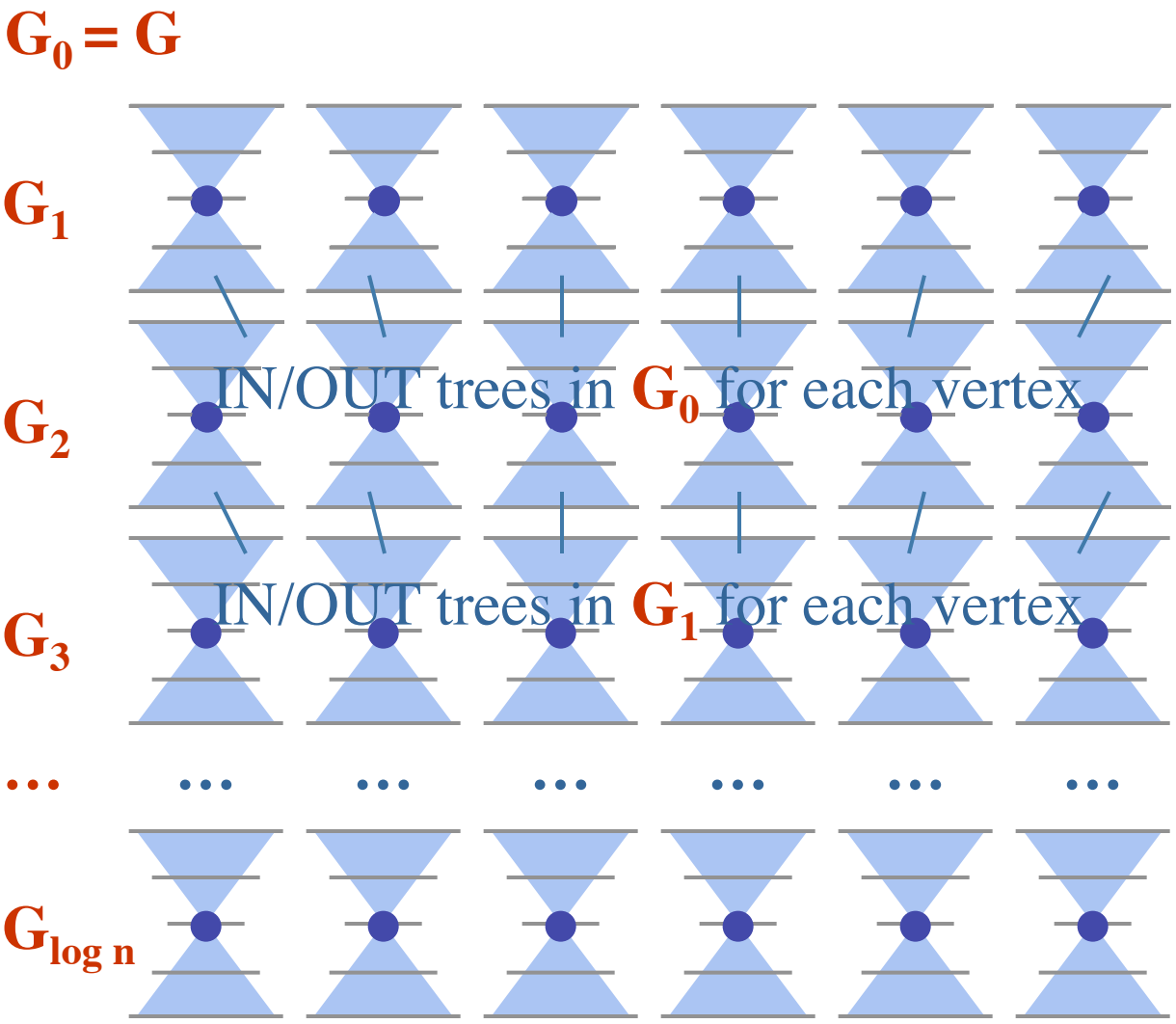
Doubling Decomposition [folklore]

Transitive closure can be computed with O(log n) products of Boolean matrices

X = adjacency matrix + I X^{n-1} = transitive closure



Dynamic Transitive Closure [King, FOCS'99]



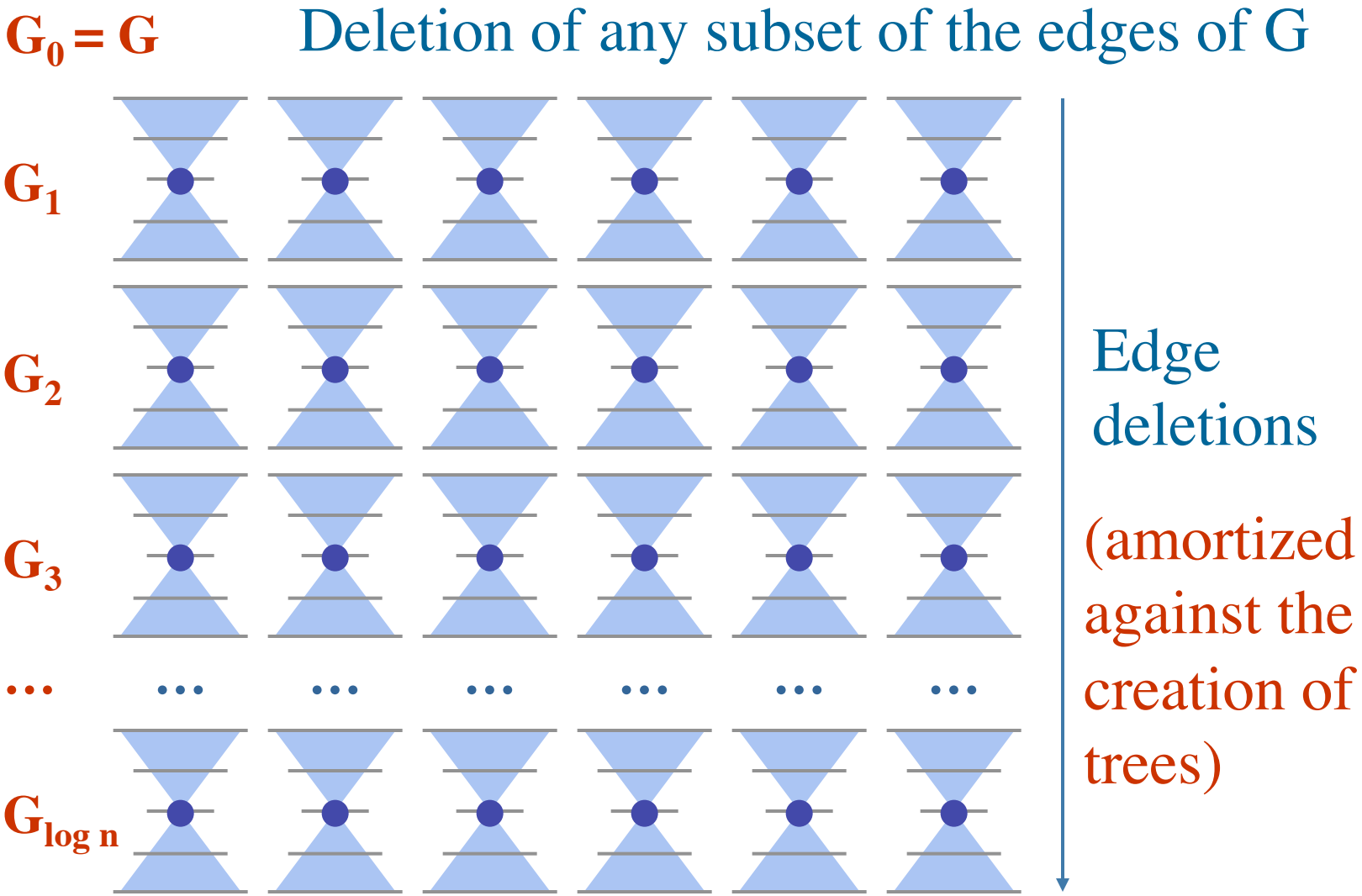
$(x,y) \in G_1$ iff
$x \in IN(v)$ and
$y \in OUT(v)$ for
some v in G_0

$(x,y) \in G_2$ iff
$x \in IN(v)$ and
$y \in OUT(v)$ for
some v in G_1

from x to y in G
of length $\leq k$, then
there is an edge
(x,y) in $G_{\lceil \log k \rceil}$

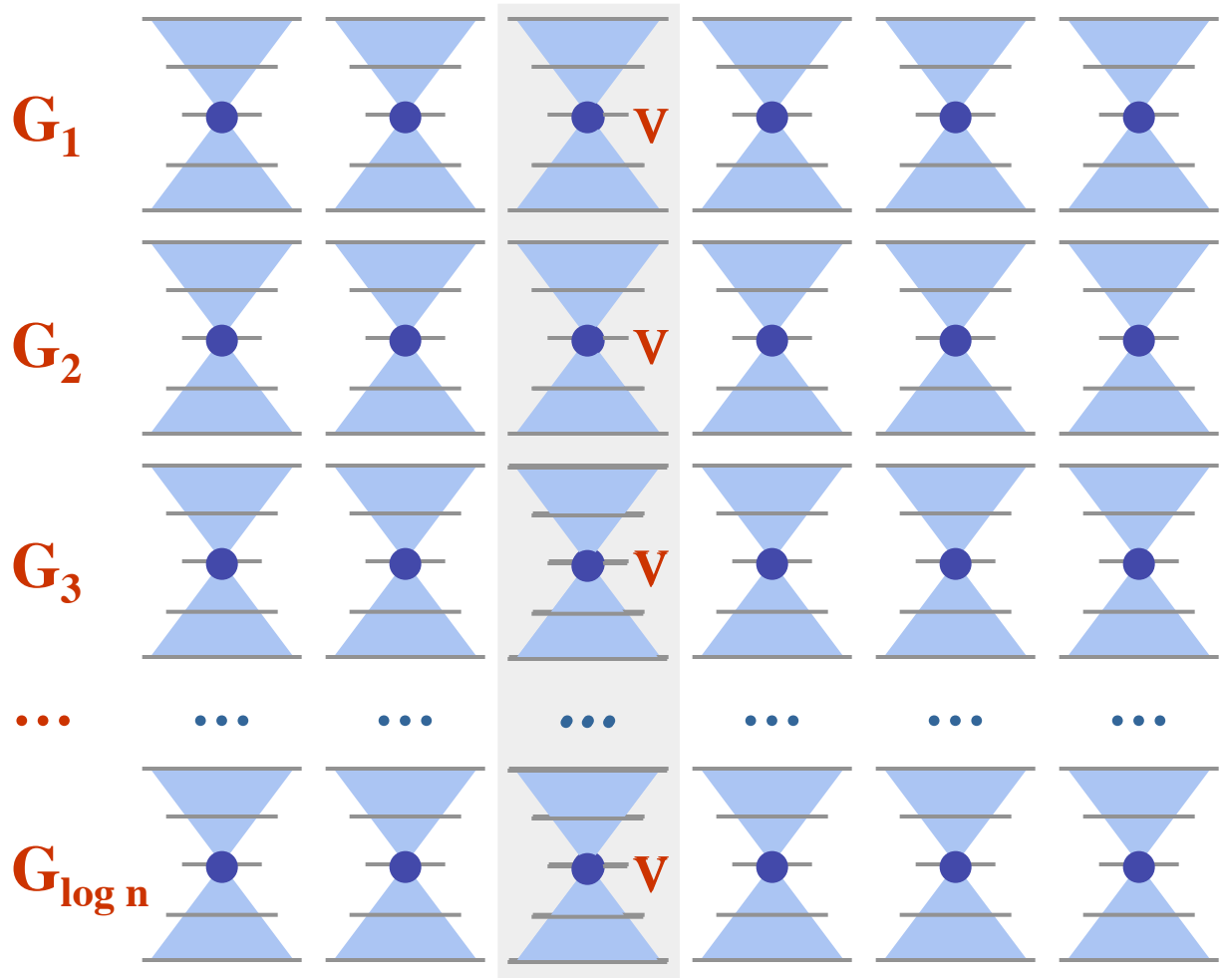
Reachability
queries in $G_{\lceil \log n \rceil}$

Dynamic Transitive Closure [King, FOCS' 99]



Dynamic Transitive Closure [King, FOCS'99]

$G_0 = G$ Insertion of edges incident to a vertex v



IN(v) and OUT(v) rebuilt from scratch on each level…

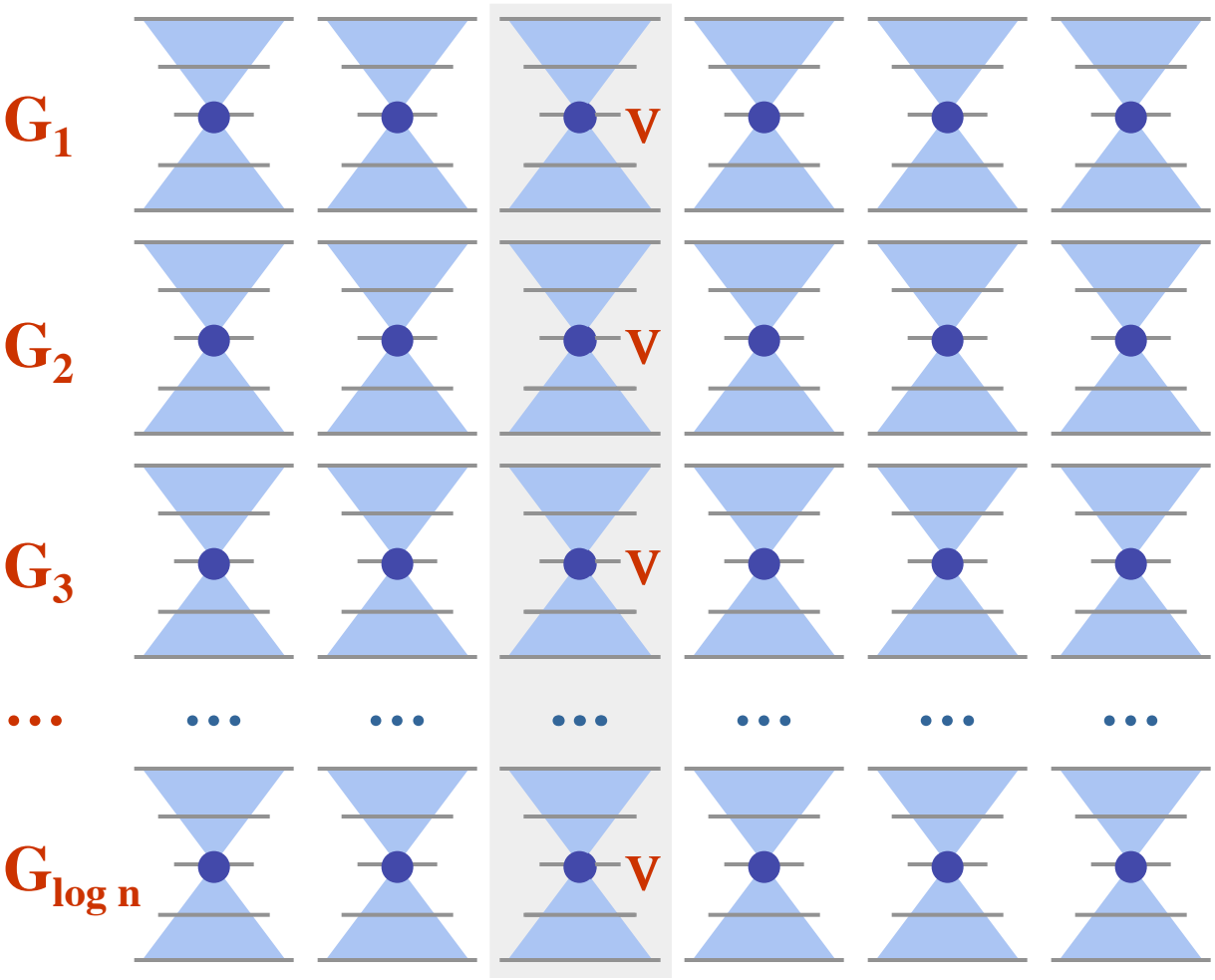
Each level has $O(n^2)$ edges

$O(n^2 \log n)$ total time

Dynamic Transitive Closure [King, FOCS'99]

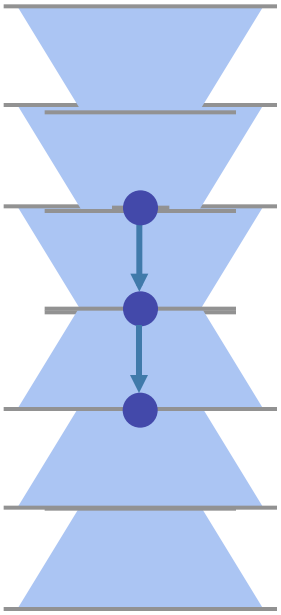
$G_0 = G$

Insertion of edges incident to a vertex v



Correctness?

Path a,b,c in G_{i-1}
$\Rightarrow$ (a,c) in G_i ?



Main Ingredients

Long paths property

Output bounded

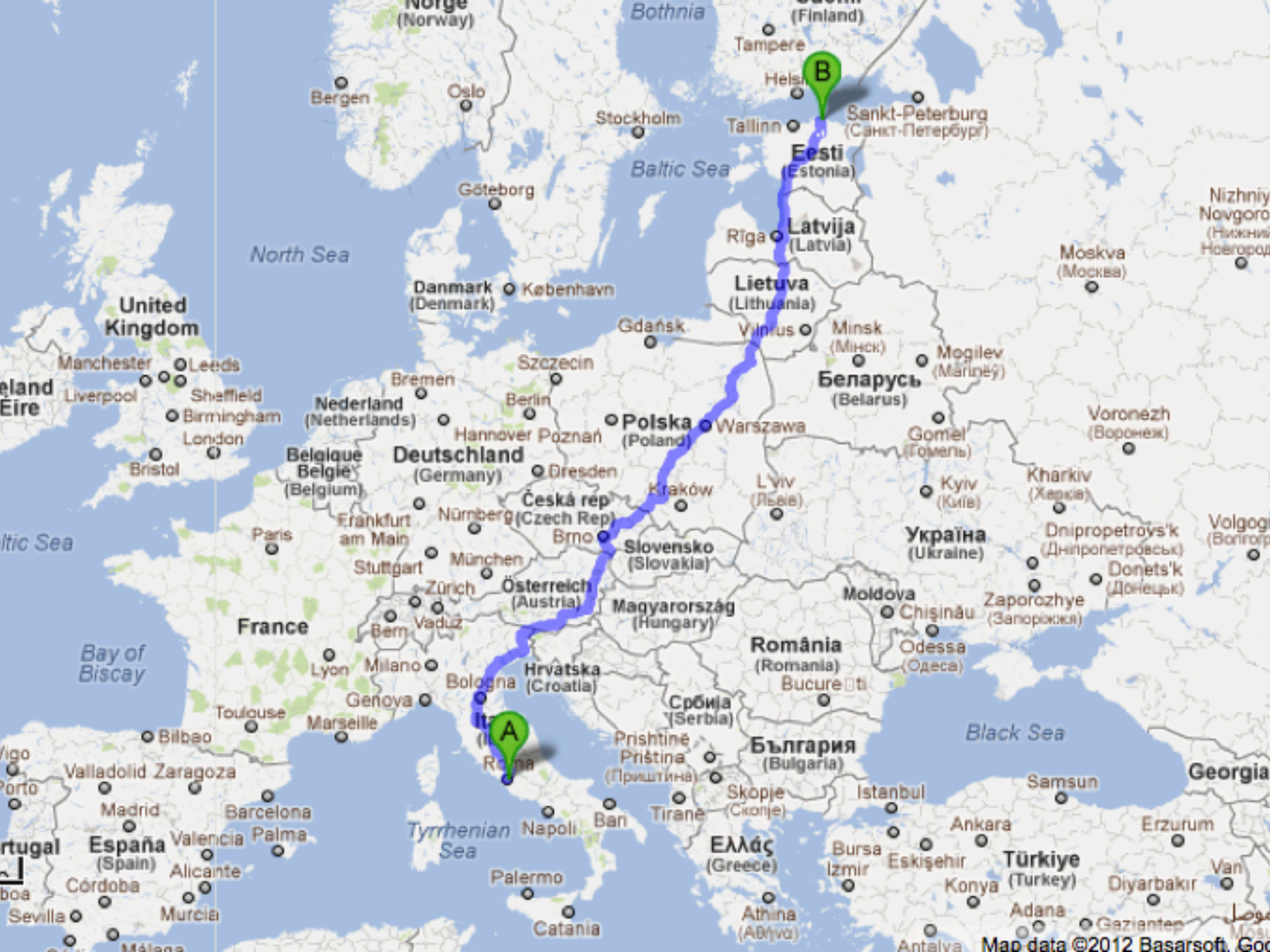
Decremental BFS

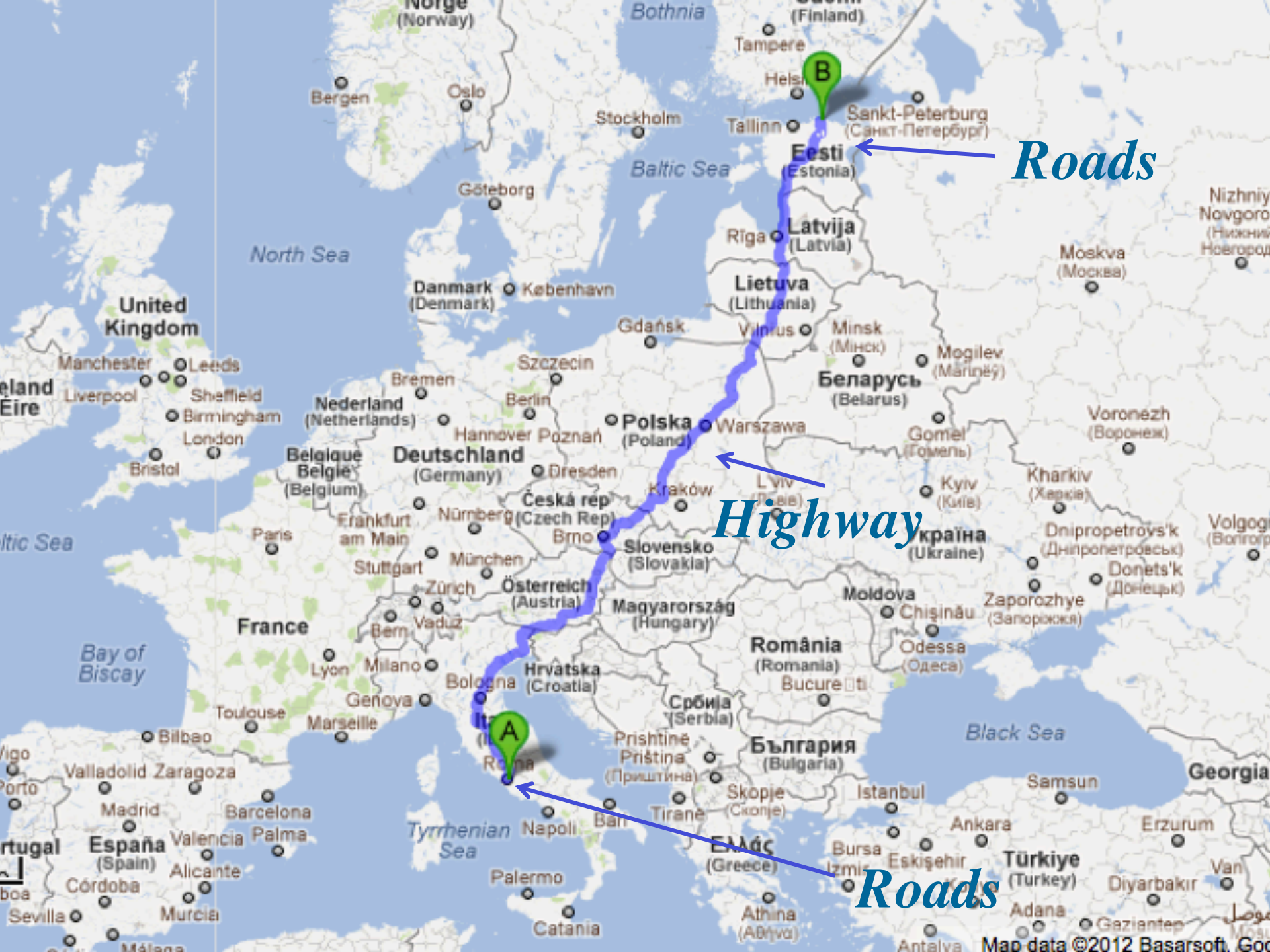
Path decompositions

Locally-defined path properties

Counting

Algebraic techniques


B
A
Bothnia
Tampere
Helsinki
Sankt-Peterburg
(Санкт-Петербург)
Tallinn
Eesti
(Estonia)
Baltic Sea
Stockholm
Oslo
Bergen
Göteborg
North Sea
Rīga
Latvija
(Latvia)
Moskva
(Москва)
Danmark
(Denmark)
København
Lietuva
(Lithuania)
Vilnius
Minsk
(Минск)
Mogilev
(Магілёў)
Беларусь
(Belarus)
Gdańsk
Szczecin
Bremen
Berlin
Nederland
(Netherlands)
Hannover
Poznań
Polska
(Poland)
Warszawa
Gomel
(Гомель)
Voronezh
(Воронеж)
United
Kingdom
Manchester
Leeds
Liverpool
Sheffield
Birmingham
London
Bristol
Belgique
België
(Belgium)
Deutschland
(Germany)
Dresden
Česká rep
(Czech Rep)
Kraków
Lviv
(Львів)
Kyiv
(Київ)
Kharkiv
(Харків)
Україна
(Ukraine)
Dnipropetrovs'k
(Дніпропетровськ)
Donets'k
(Донецьк)
Zaporozhye
(Запорожжя)
Frankfurt
am Main
Nürnberg
Brno
Slovensko
(Slovakia)
Paris
Stuttgart
München
Österreich
(Austria)
Zürich
Magyarország
(Hungary)
Moldova
Chişinău
Odessa
(Одеса)
France
Bern
Vaduz
Milano
Lyon
Bologna
Hrvatska
(Croatia)
România
(Romania)
Bucureşti
Genova
Toulouse
Marseille
Србија
(Serbia)
Prishtinë
Priština
(Приштина)
България
(Bulgaria)
Black Sea
Bilbao
Valladolid
Zaragoza
Madrid
Barcelona
Palma
Valencia
España
(Spain)
Alicante
Córdoba
Sevilla
Murcia
Tyrrhenian
Sea
Napoli
Bari
Tirane
Skopje
(Скопје)
Istanbul
Samsun
Georgia
Ankara
Erzurum
Bursa
Izmir
Eskişehir
Türkiye
(Turkey)
Konya
Diyarbakır
Van
Adana
Gaziantep
Ελλάς
(Greece)
Palermo
Catania
Athina
(Αθήνα)
Antalya
Map data ©2012 Basarsoft, Goo


Roads
Highway
Roads
A
B
Map data ©2012 Basarsoft, Goo

Are there roads and highways in graphs?

Long Paths Property
[Ullman-Yannakakis '91]

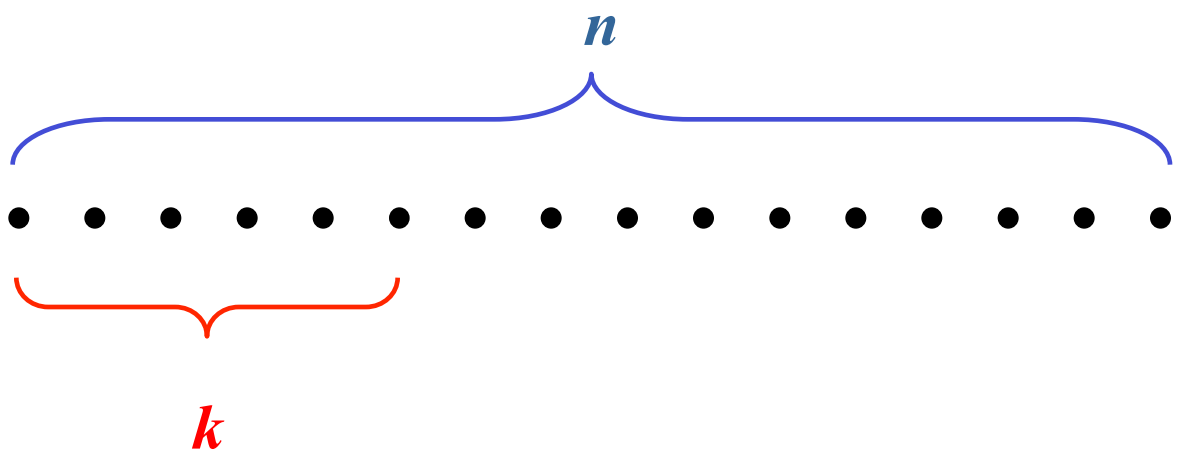
Let P be a path of length at least k.

Let S be a random subset of vertices of size $(c\, n \ln n) / k$.

Then with high probability $P \cap S \neq \emptyset$.

Probability $\geq 1 - (1 / n^c)$ (depends on c)

Long Paths Property [Ullman-Yannakakis '91]



Select each element independently with probability

$$p = \frac{c \ln n}{k}$$

The probability that a given set of ***k*** elements is not hit is

$$(1-p)^k = \left(1 - \frac{c \ln n}{k}\right)^k < n^{-c}$$

Long Paths Property

Can prove stronger property:

Let P be a path of length at least k.

Let S be a random subset of vertices of size $(c\, n \ln n) / k$.

Then with high probability *there is no subpath of* P *of length* k *with no vertices in* S ($P \cap S \neq \varnothing$).

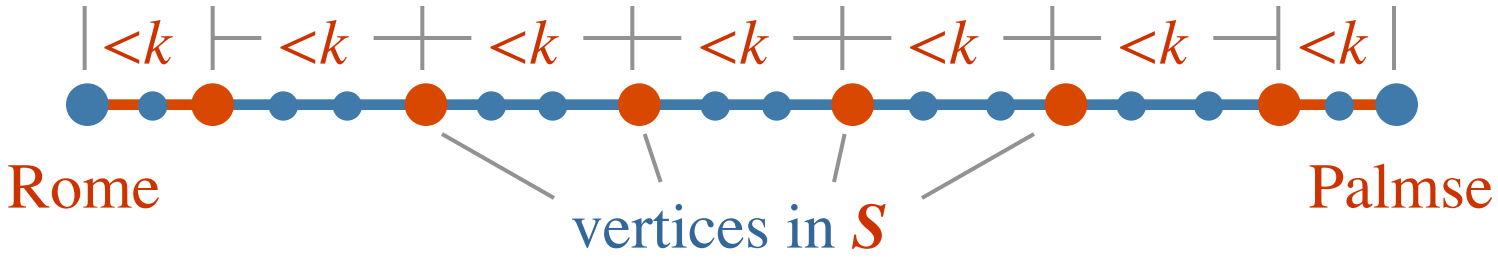
Probability $\geq 1 - (1 / n^{\alpha c})$ for some $\alpha > 0$.

Exploit Long Paths Property

Randomly pick a set S of vertices in the graph

$$|S| = \frac{c\, n \log n}{k} \qquad c, k > 0$$

Then on any path in the graph
every k vertices there is a vertex in S,
with probability $\geq 1 - (1 / n^{\alpha c})$

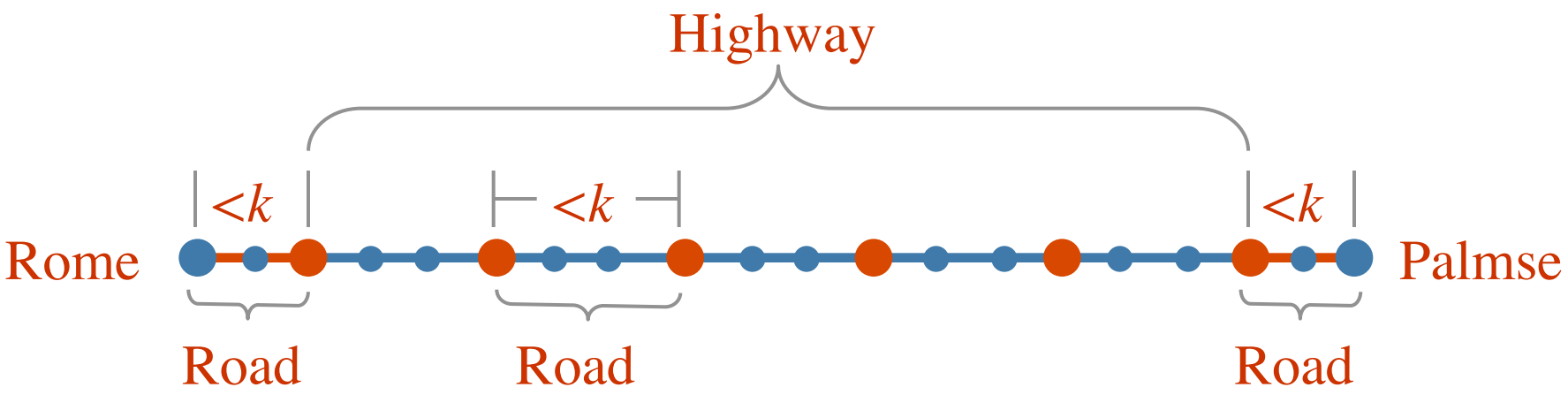


Roads and Highways in Graphs

Highway entry points = vertices in ***S***

Road = shortest path using at most ***k*** edges

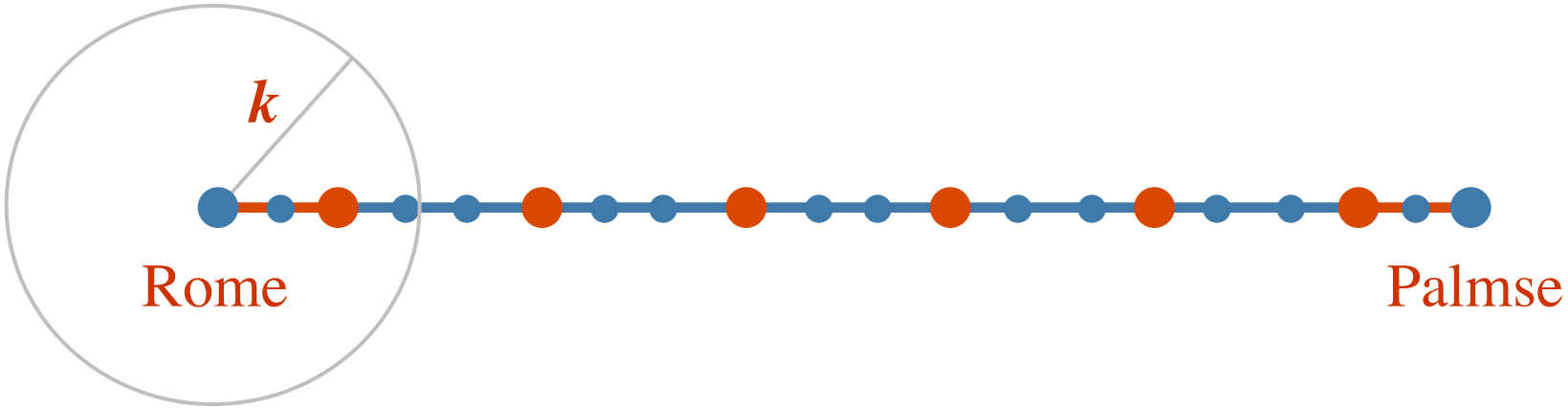
Highway = shortest path between two vertices in ***S***



Computing Shortest Paths 1/3

1

Compute roads
(shortest paths using at most ***k*** edges)

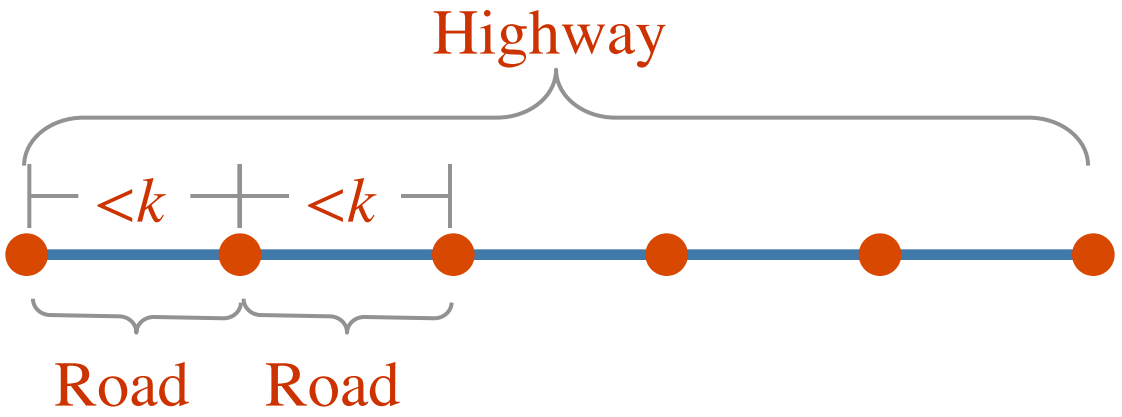


Even & Shiloach BFS trees may become handy…

Computing Shortest Paths 2/3

2

Compute highways
(by stitching together roads)

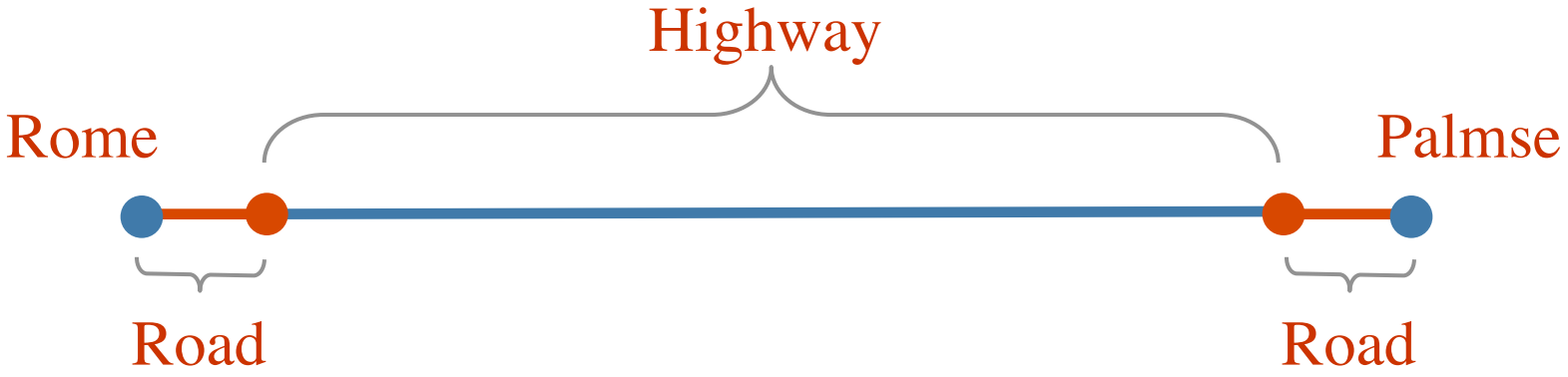


…essentially an all pairs shortest paths computation on a contracted graph with vertex set ***S***, and edge set = roads

Computing Shortest Paths 3/3

3

Compute shortest paths (longer than ***k*** edges)
(by stitching together roads + highways + roads)



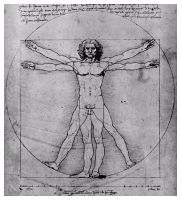
Used (for dynamic graphs) by King [FOCS'99], Demetrescu-I. [JCSS'06], Roditty-Zwick [FOCS'04], …

Fully Dynamic APSP

Given a weighted directed graph G=(V,E,w), perform any intermixed sequence of the following operations:

Update(u,v,w):	update weight of edge (u,v) to w
Query(x,y):	return distance from x to y (or shortest path from x to y)

King's algorithm [King' 99]

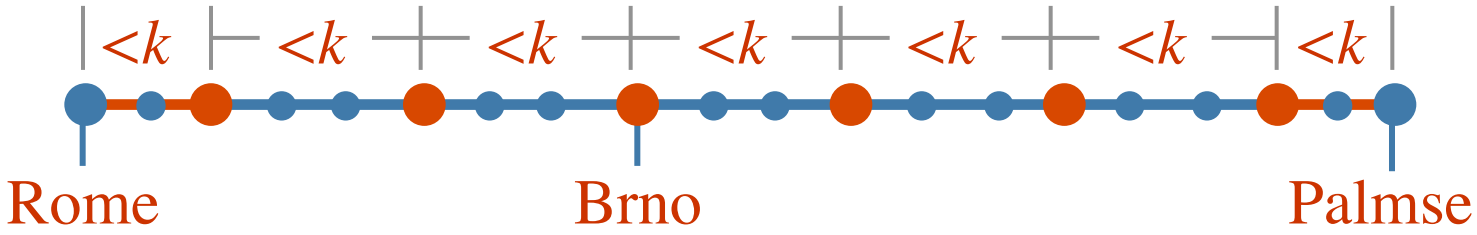


Directed graphs with integer edge weights in [0,C]

$\tilde{O}(n^{2.5}\sqrt{C})$ update time	$O(1)$ query time	$\tilde{O}(n^{2.5}\sqrt{C})$ space

Approach:

1. Maintain dynamically shortest paths up to length $k = (nC\log n)^{0.5}$ using variant of decremental data structure by Even-Shiloach. Amortized cost per update is $O(n^2(nC\log n)^{0.5})$ (details in the paper)
2. Stitch together short paths from scratch to form long paths exploiting long paths decomposition



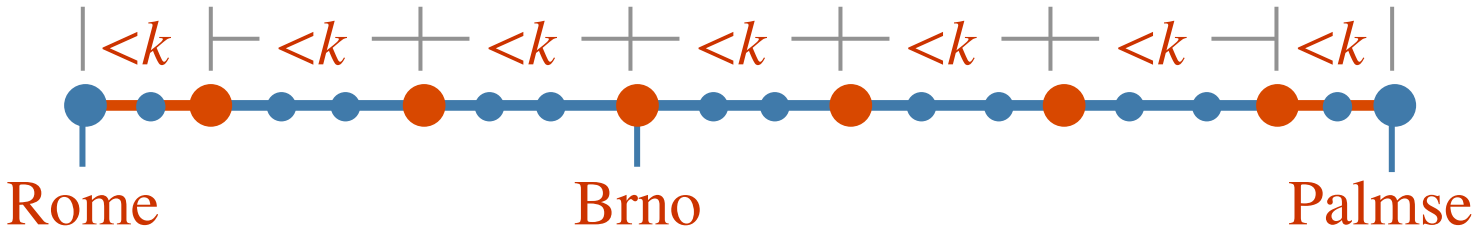
More details on stitching

Always distances up to $k=(Cn\log n)^{1/2}$ (IN e OUT trees)

Perform the following tasks at each update:

1. Build S deterministically, $|S|=(Cn\log n)^{1/2}$: $O(n^2)$
2. Compute APSP in S: $O(|S|^3) = O((Cn\log n)^{3/2})$
3. For each v in V, s in S, update distance by considering $\min_{s'}\{D(v,s')+D(s',s)\}$: $O(n|S|^2) = O(Cn^2\log n)$
4. For each u,v in V, update distance by considering $\min_{s'}\{D(u,s')+D(s',v)\}$: $O(n^2|S|) = O(n^{5/2}(C\log n)^{1/2})$

$\tilde{O}(n^{2.5}\sqrt{C})$ update time	$O(1)$ query time	$\tilde{O}(n^{2.5}\sqrt{C})$ space



More details in

Long paths decomposition:
[Ullman-Yannakakis'91]
J.D. Ullman and M. Yannakakis.
High-probability parallel transitive-closure algorithms.
SIAM Journal on Computing, 20(1), February 1991

King's algorithm:
[King'99]
Valerie King
Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs.
FOCS 1999: 81-91

Main Ingredients

Long paths property

Output bounded

Decremental BFS

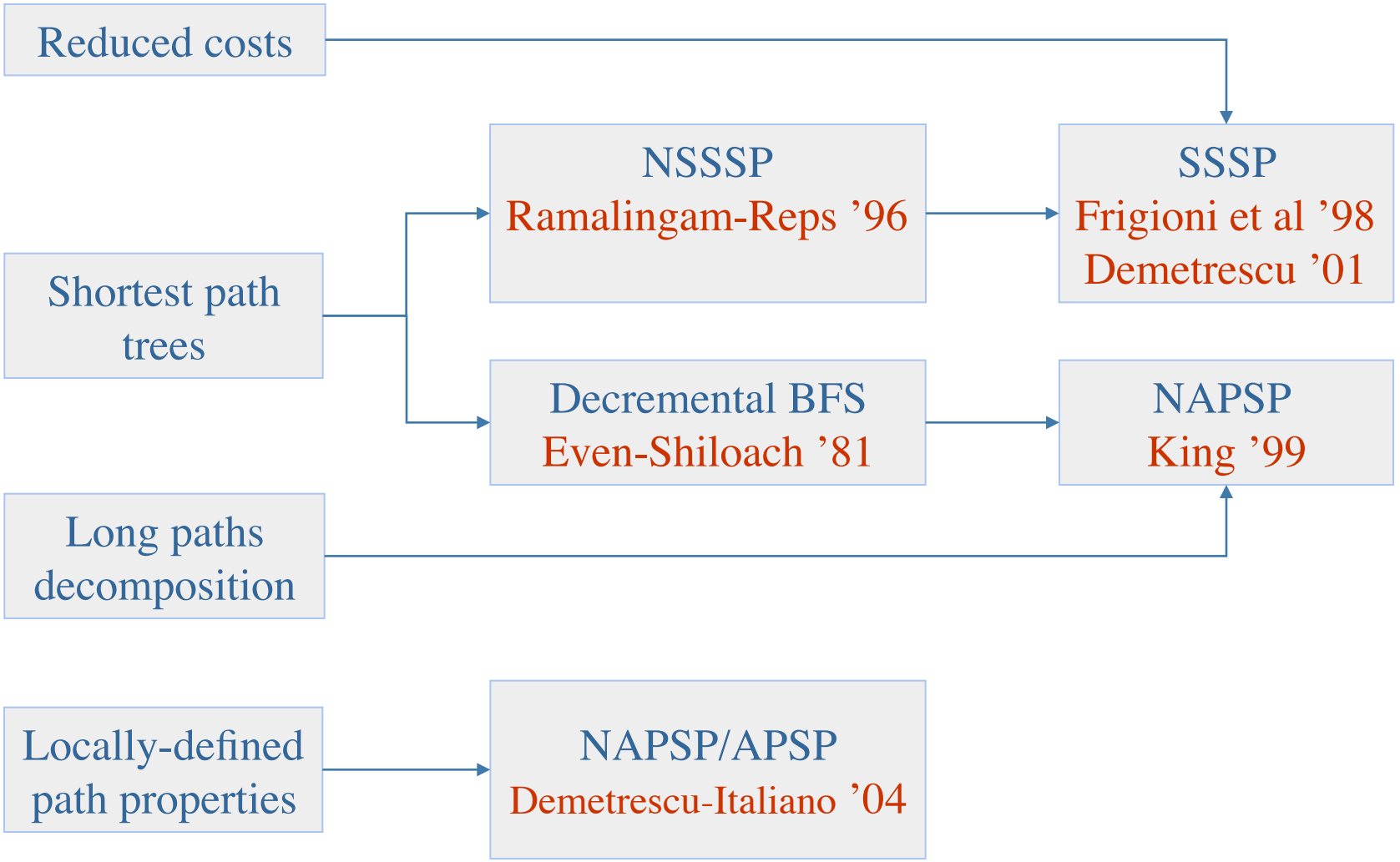
Path decompositions

Locally-defined path properties

Counting

Algebraic techniques

Dynamic shortest paths: roadmap



Heuristic to speed up Dijkstra (NAPSP)

Dijkstra's algorithm for NAPSP

Run Dijkstra from all vertices "in parallel"

Edge scanning bottleneck for dense graphs [Goldberg]

1. Extract shortest pair (x,y) from heap:

2. Scan all neighbors y' of y

3. Possibly insert (x,y') into heap or decrease its priority

Can we do better?

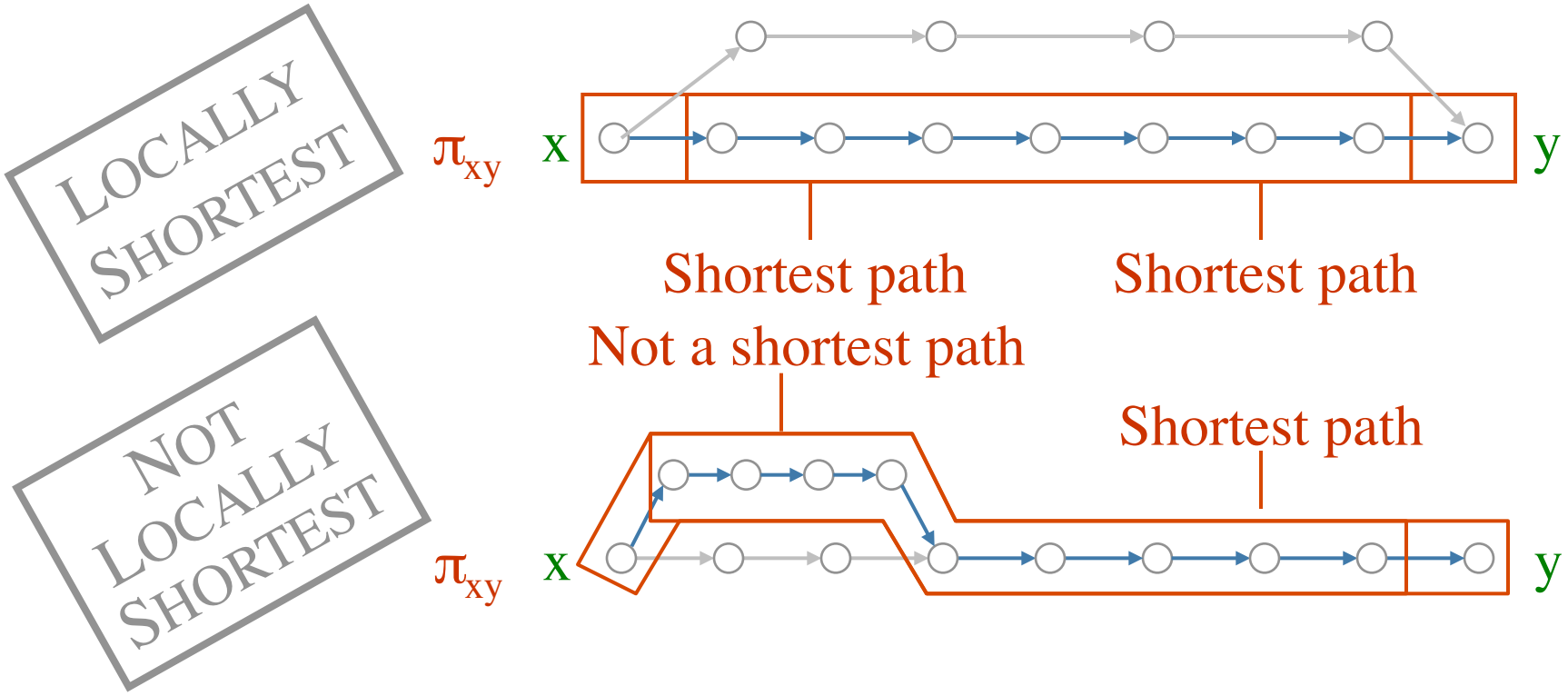
1. Extract shortest pair (x,y) from heap:

2. Scan only y' for which (a,y') shortest (subpath opt.)

3. Possibly insert (x,y') into heap or decrease priority

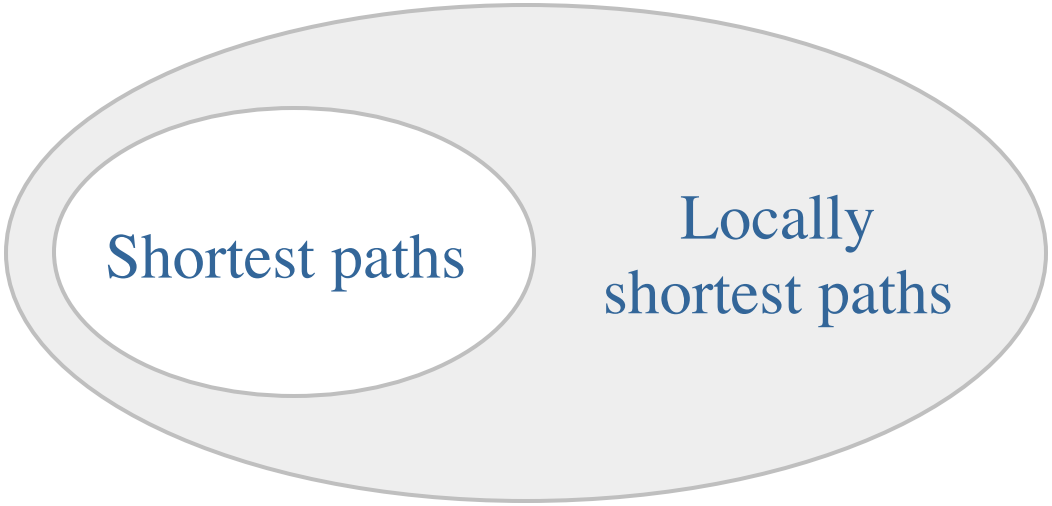
Locally Shortest Paths [Demetrescu-I., J.ACM04]

A path is *locally shortest* if all of its **proper** subpaths are shortest paths



Locally shortest paths

By optimal-substructure property of shortest paths:



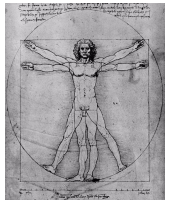
How much do we gain?

Running time on directed graphs
with real non-negative edge weights

O(**#LS-paths** + $n^2 \log n$) time | $O(n^2)$ space

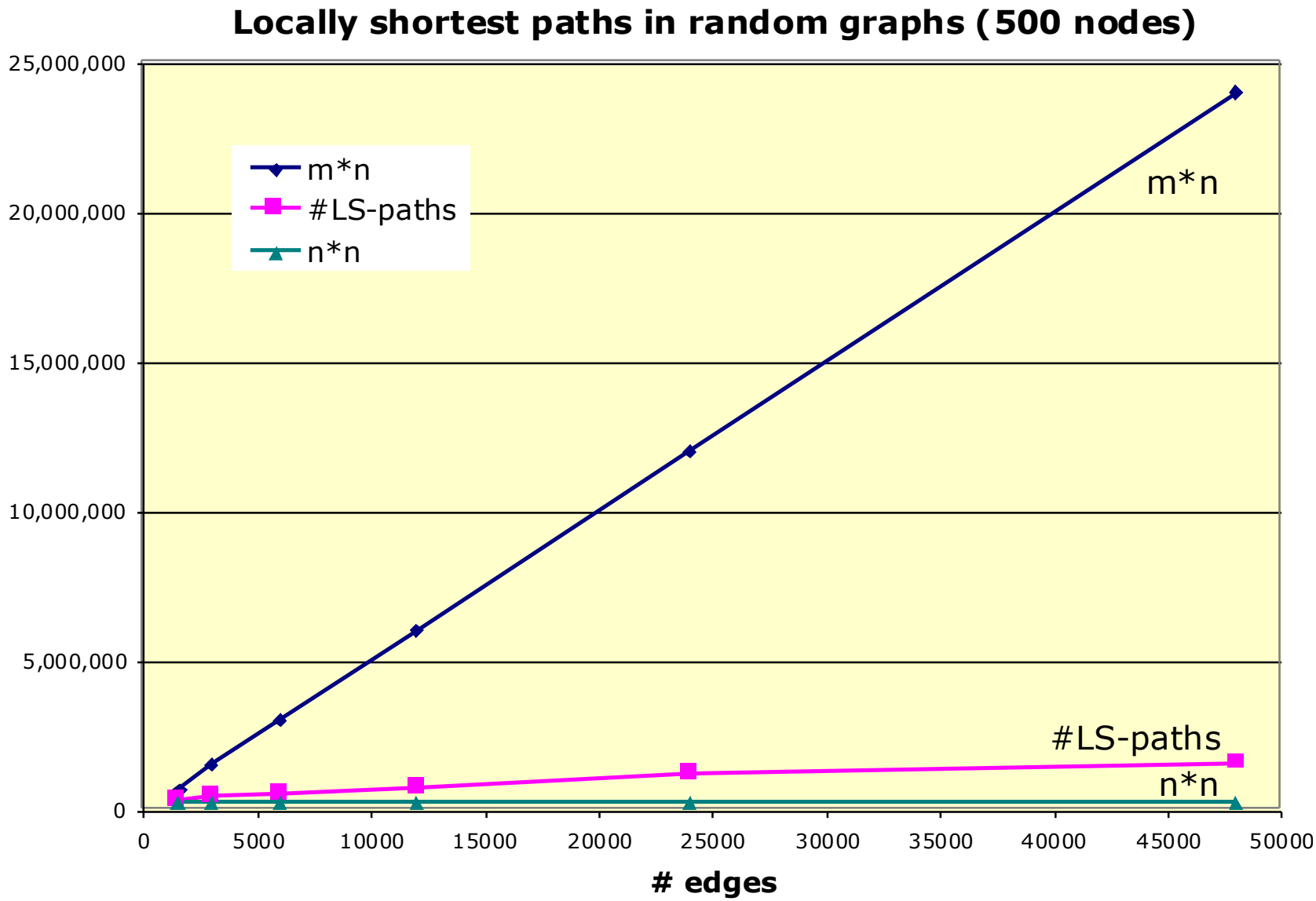
Q.: How many locally shortest paths ?

A.: #LS-paths ≤ mn. No gain in asymptopia…

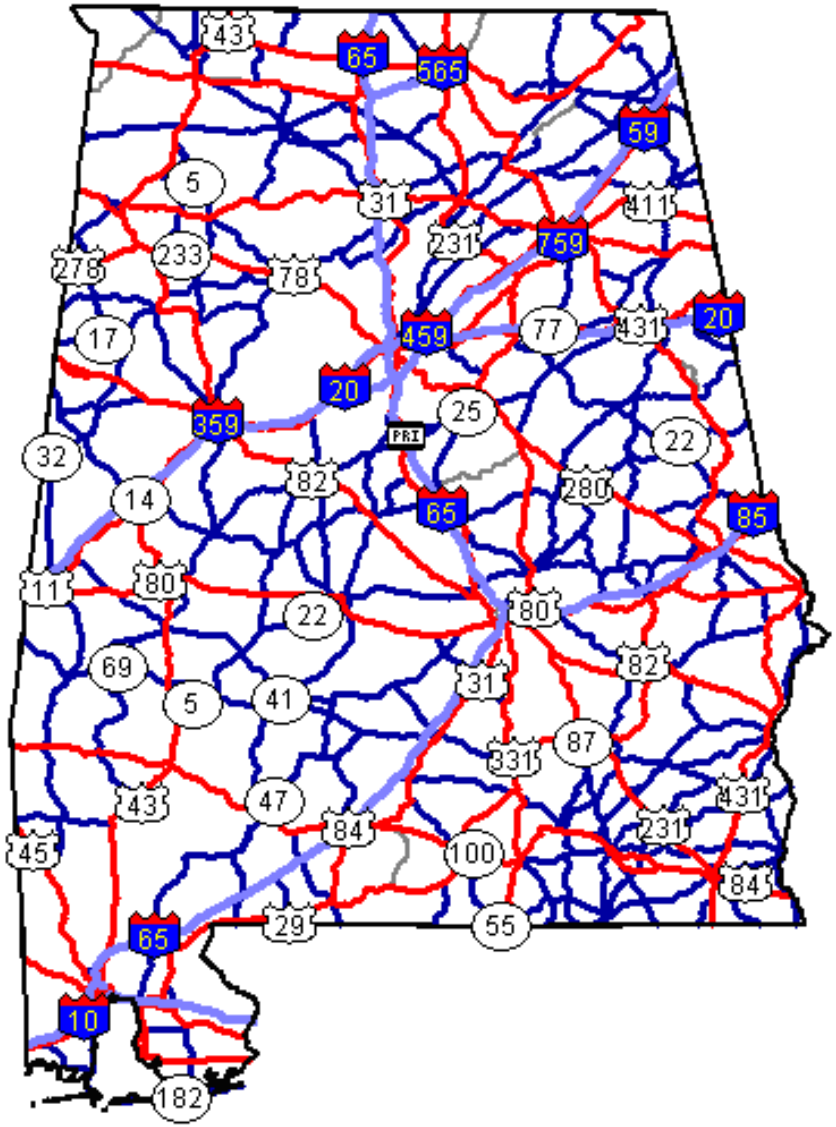


Q.: How much can we gain in practice?

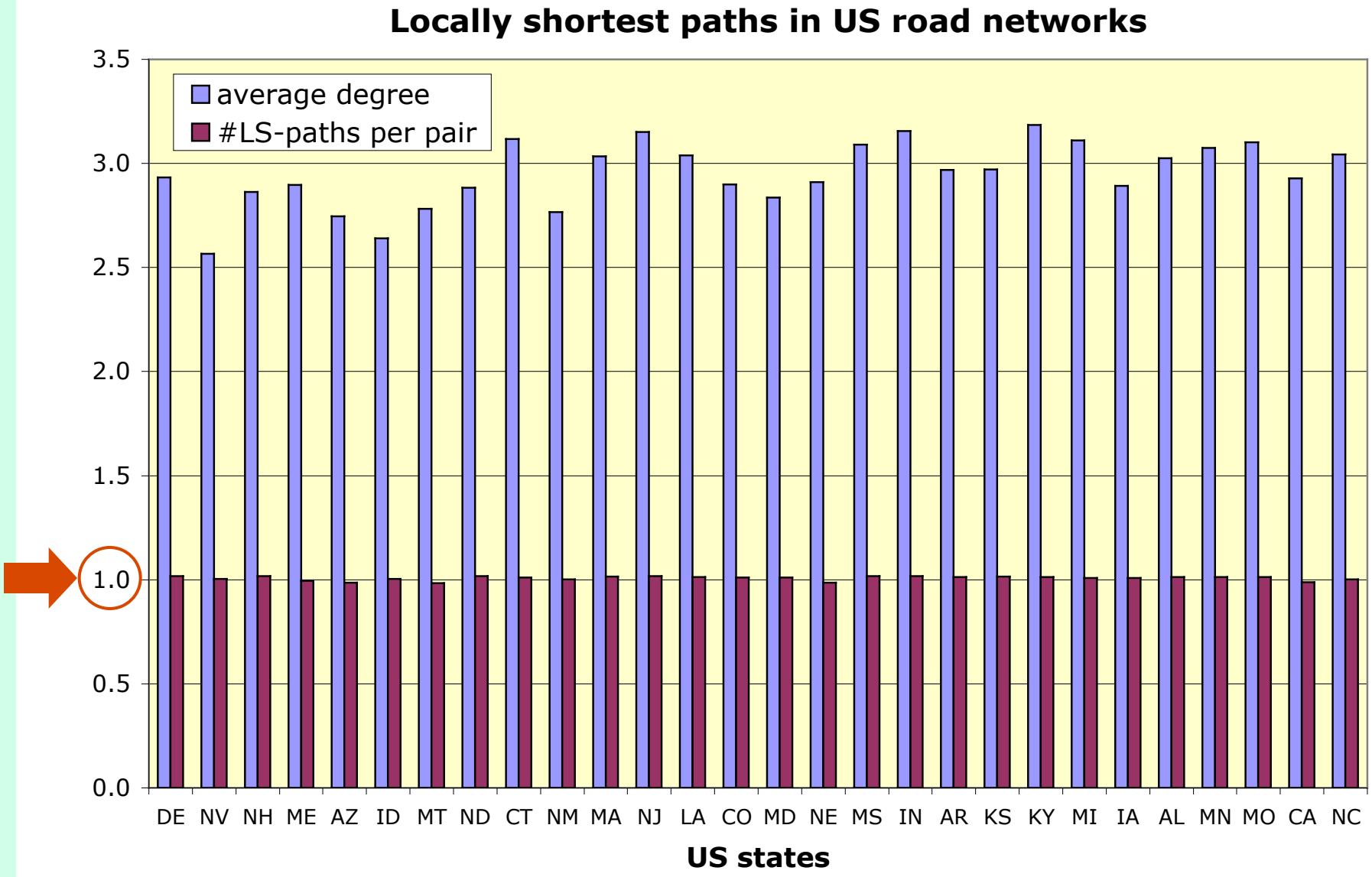
How many LSPs in a graph?



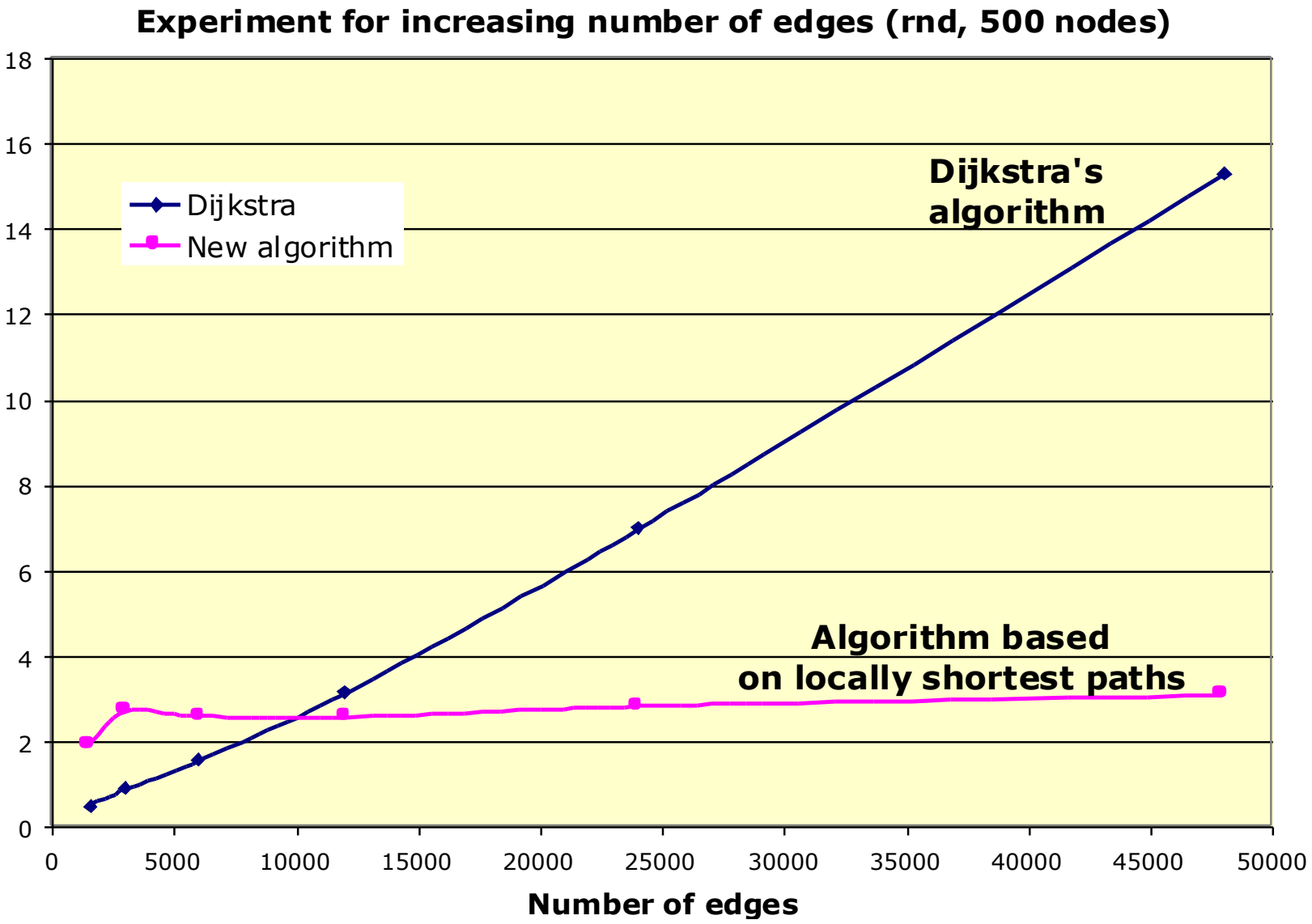
Real-world Graphs?



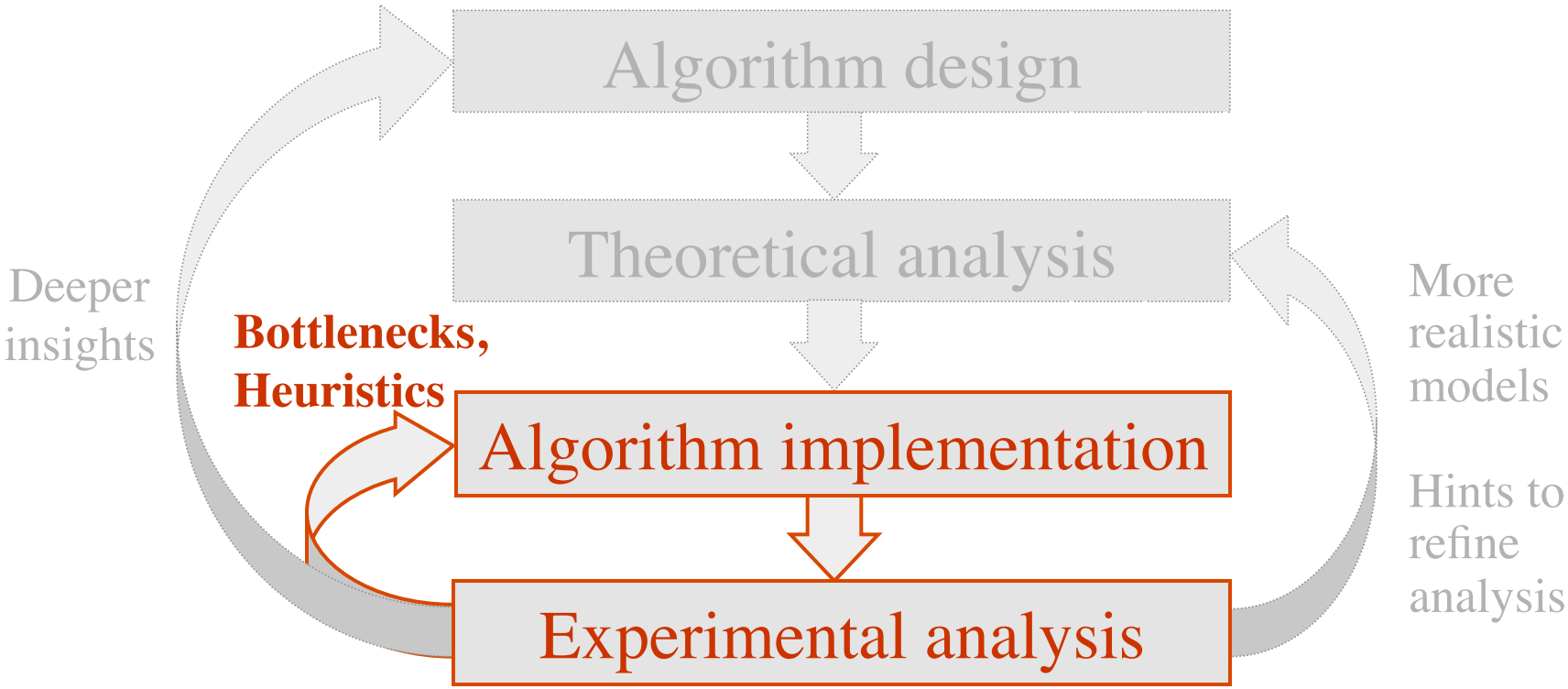
US road networks



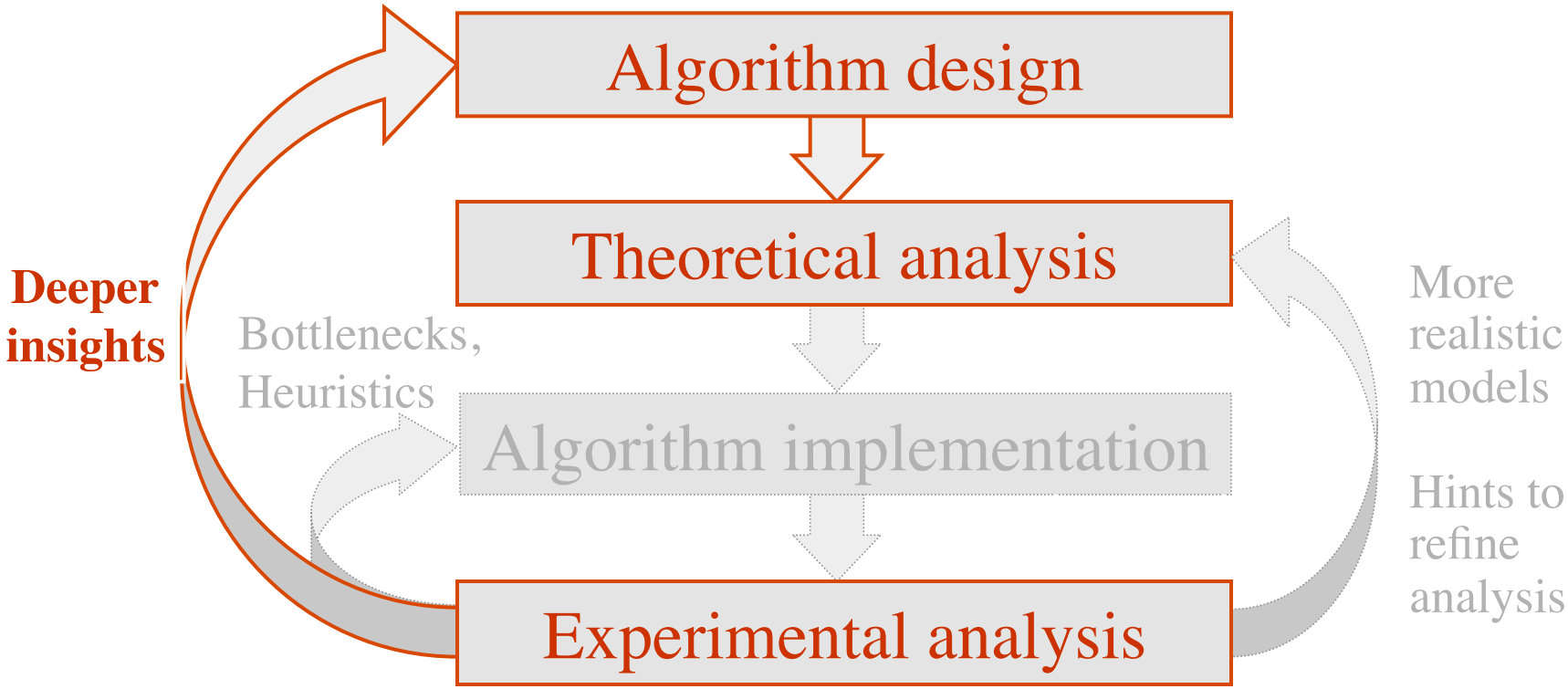
Can we exploit this in practice?



What we have seen so far:



Return Trip to Theory:



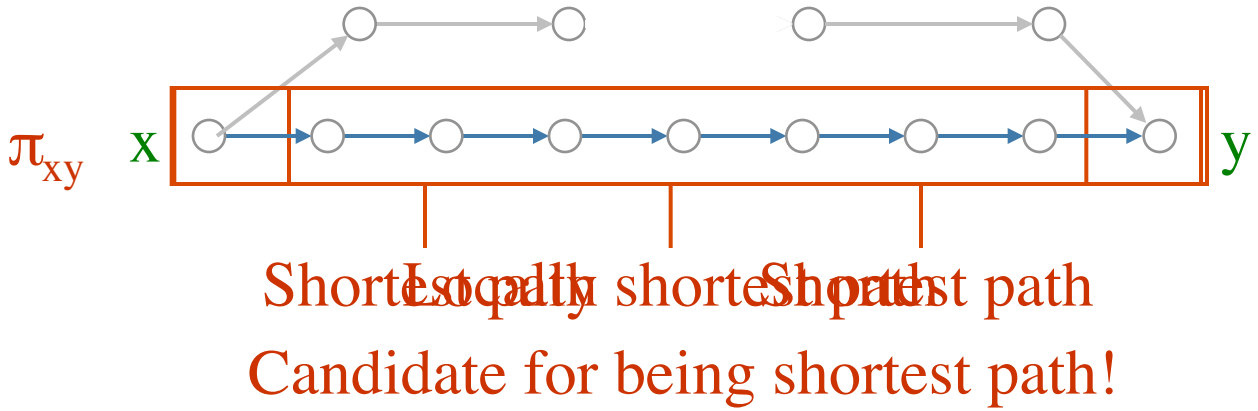
Back to Fully Dynamic APSP

Given a weighted directed graph G=(V,E,w), perform any intermixed sequence of the following operations:

Update(u,v,w):	update cost of edge (u,v) to w
Query(x,y):	return distance from x to y (or shortest path from x to y)

Recall Fully Dynamic APSP

- Hard operations seem edge deletions (edge cost increases)
- When edge (shortest path) deleted: need info about second shortest path? (3rd, 4th, …)
- Hey… what about locally shortest paths?



Locally Shortest Paths for Dynamic APSP

Idea:

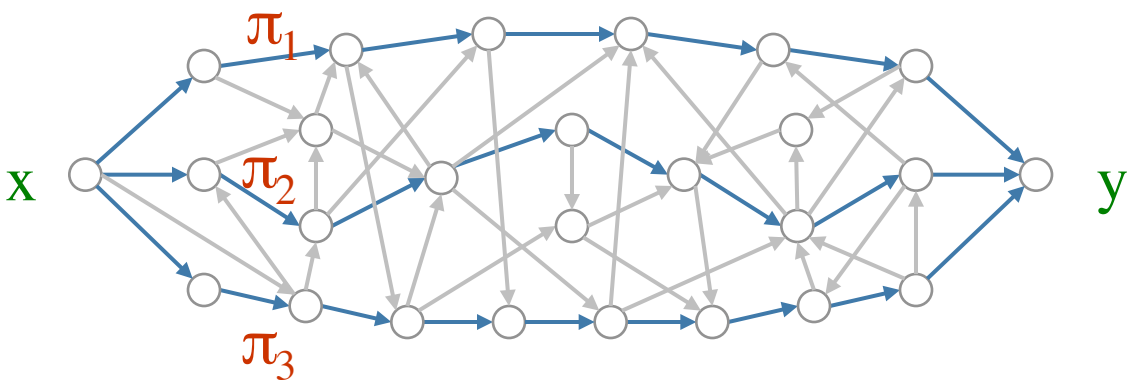
Maintain all the locally shortest paths of the graph

How do locally shortest paths change in a dynamic graph?

Assumptions behind the analysis

Property 1

Locally shortest paths π_{xy} are internally vertex-disjoint

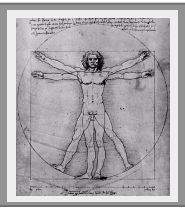


This holds under the assumption that there is a unique shortest path between each pair of vertices in the graph

(Ties can be broken by adding a small perturbation to the weight of each edge)

Tie Breaking

Assumptions



Shortest paths are unique

In theory, tie breaking is not a problem

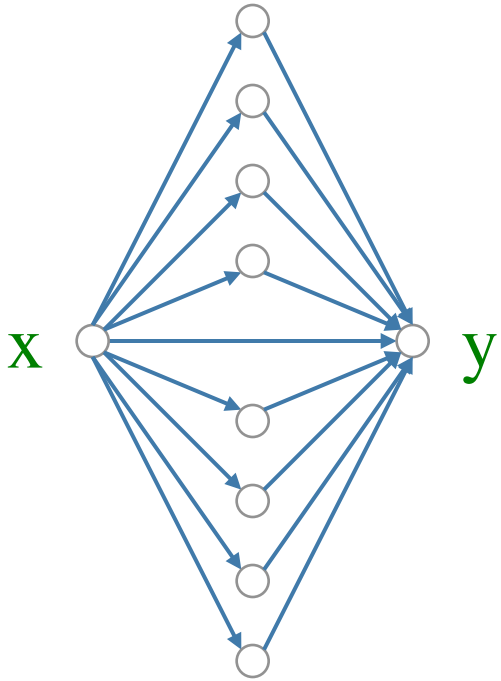
Practice

In practice, tie breaking can be subtle

Properties of locally shortest paths

Property 2

There can be at most n-1 LS paths connecting x,y

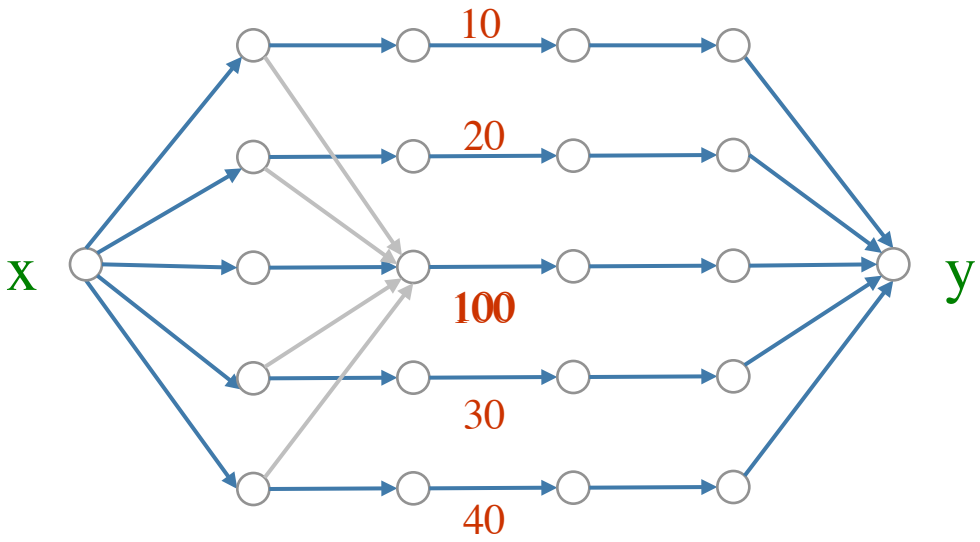


This is a consequence of vertex-disjointess…

Appearing locally shortest paths

Fact 1

At most mn (n^3) paths can **start** being locally shortest after an edge weight increase



Disappearing locally shortest paths

Fact 2

At most n^2 paths can **stop** being locally shortest after an edge weight increase

π stops being locally shortest after increase of ***e***

subpath of π (was shortest path) must contain ***e***

shortest paths are unique: at most n^2 contain ***e***

Maintaining locally shortest paths

Locally shortest paths appearing after increase: $< n^3$

Locally shortest paths disappearing after increase: $< n^2$

The amortized number of changes in the set of locally shortest paths at each update in an increase-only sequence is $O(n^2)$

An increase-only update algorithm

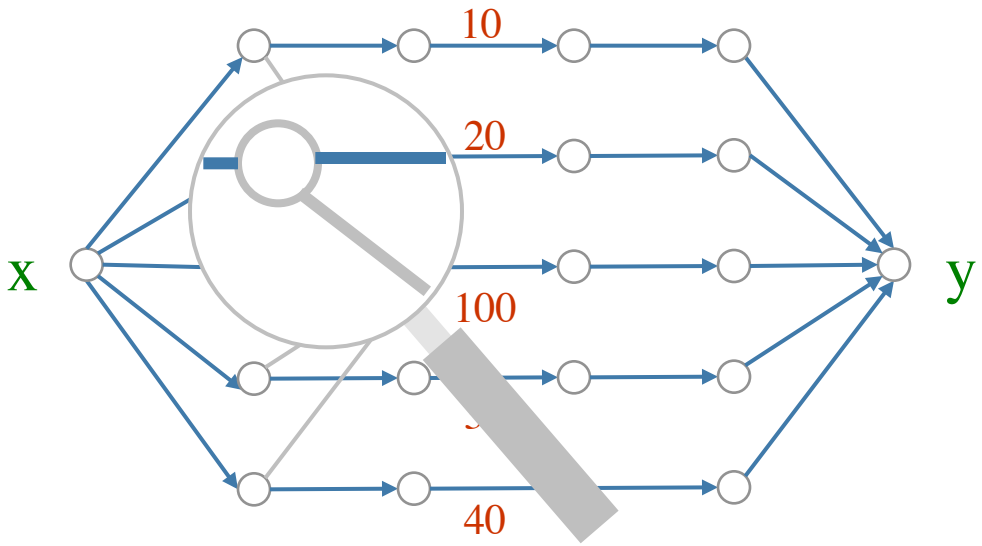
This gives (almost) immediately:

$O(n^2 \log n)$ amortized time per increase

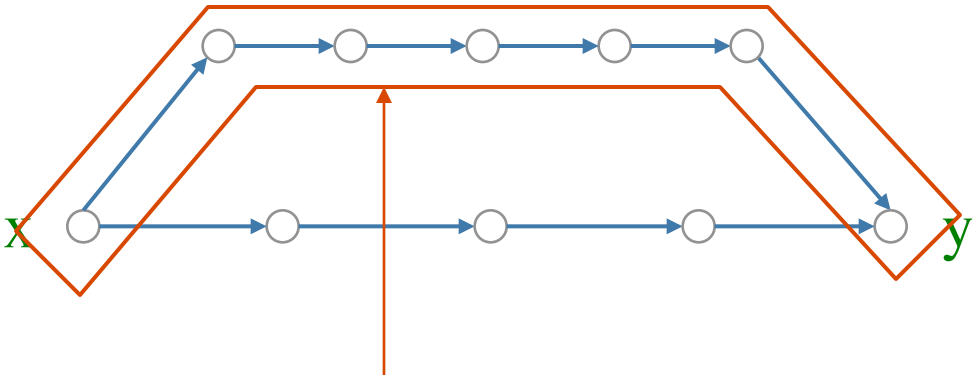
$O(mn)$ space

Maintaining locally shortest paths

What about fully dynamic sequences?



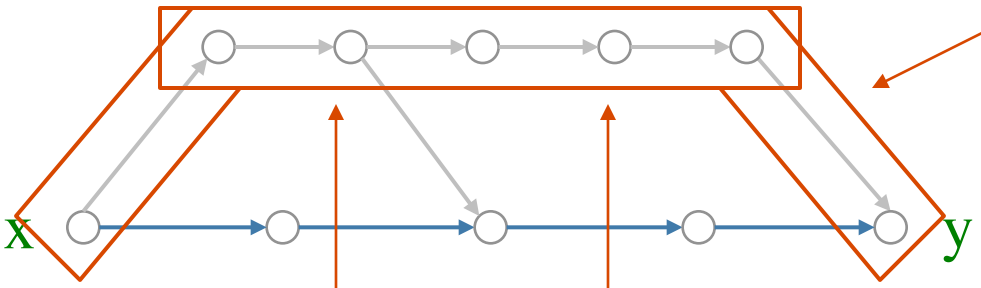
How to pay only once?



This path remains the same while flipping between being LS and non-LS:

Would like to have update algorithm that pays only once for it until it is further updated...

Looking at the substructure



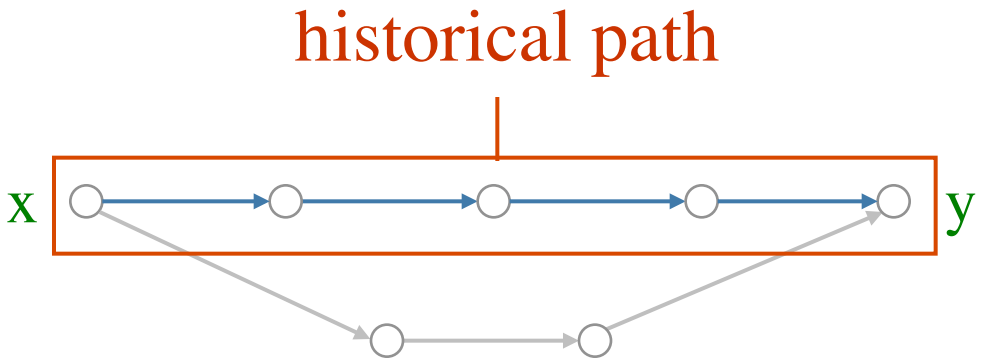
It is not dead!

This path is no longer a shortest path after the insertion…

…but if we removed the same edge it would be a shortest path again!

Historical paths

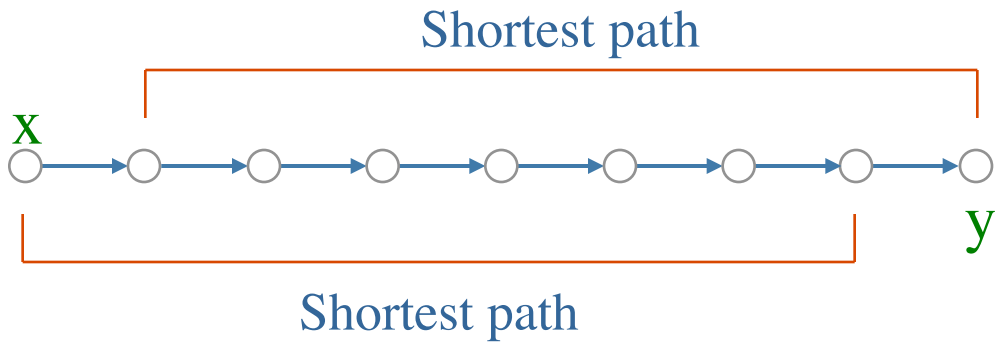
A path is **historical** if it was shortest
at some time since it was last updated



Locally historical paths

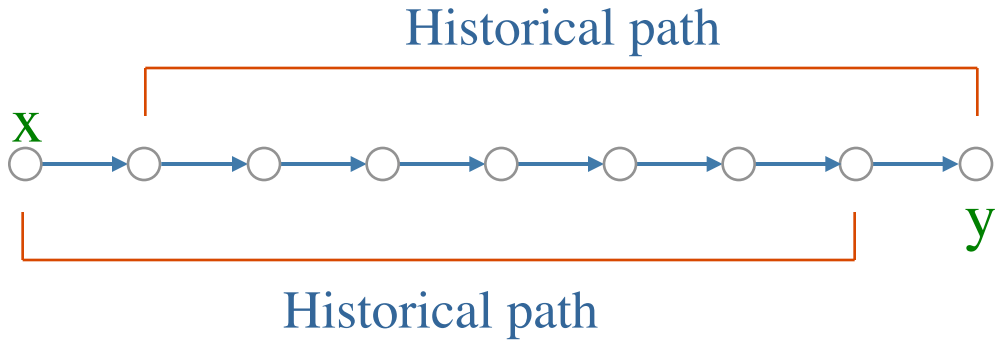
Locally shortest path

π_{xy}

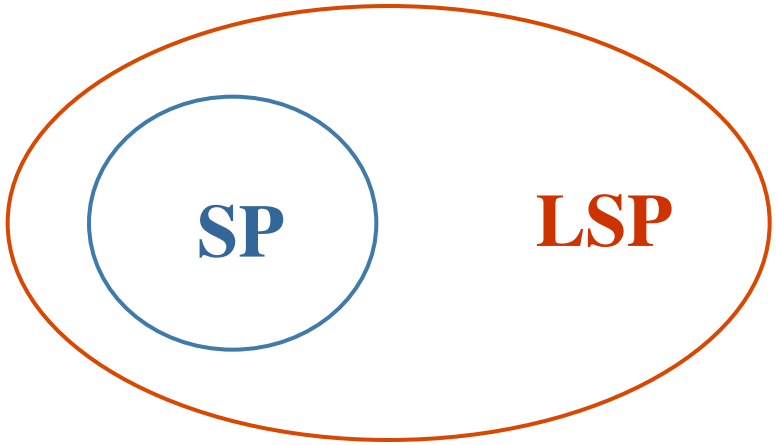


Locally historical path

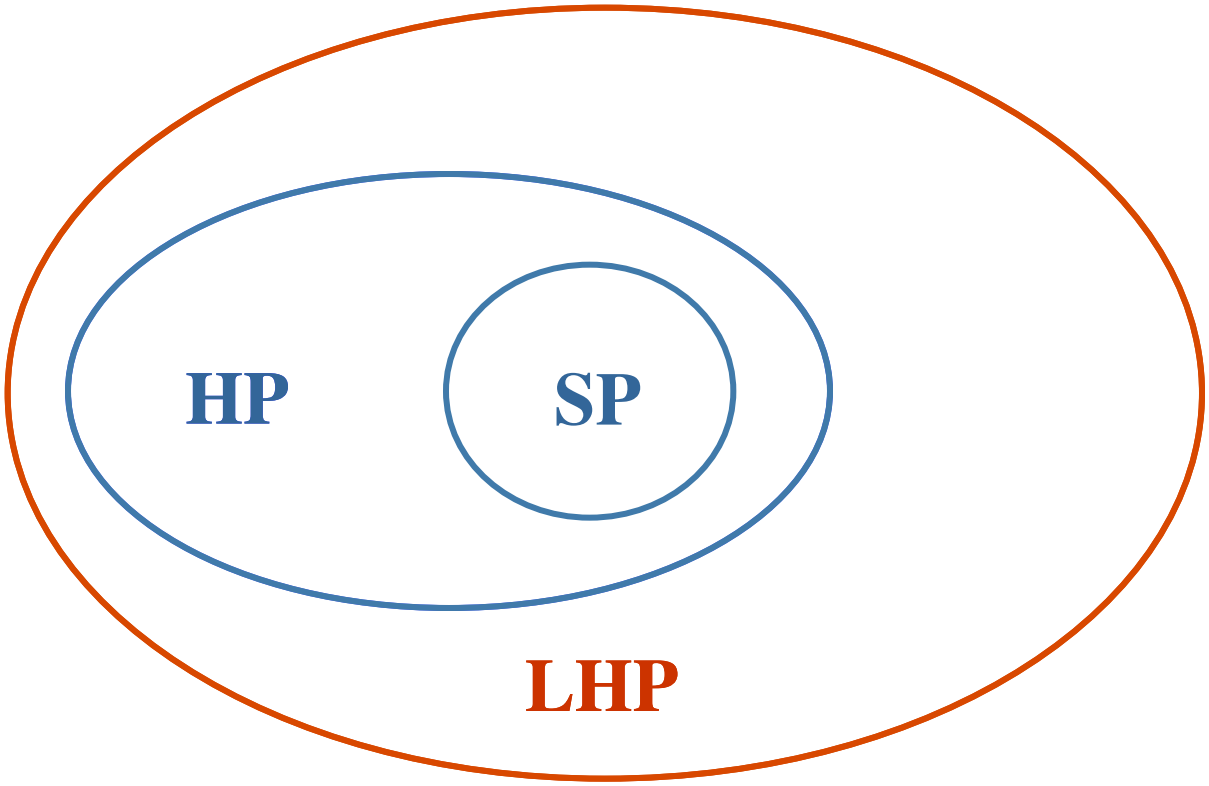
π_{xy}



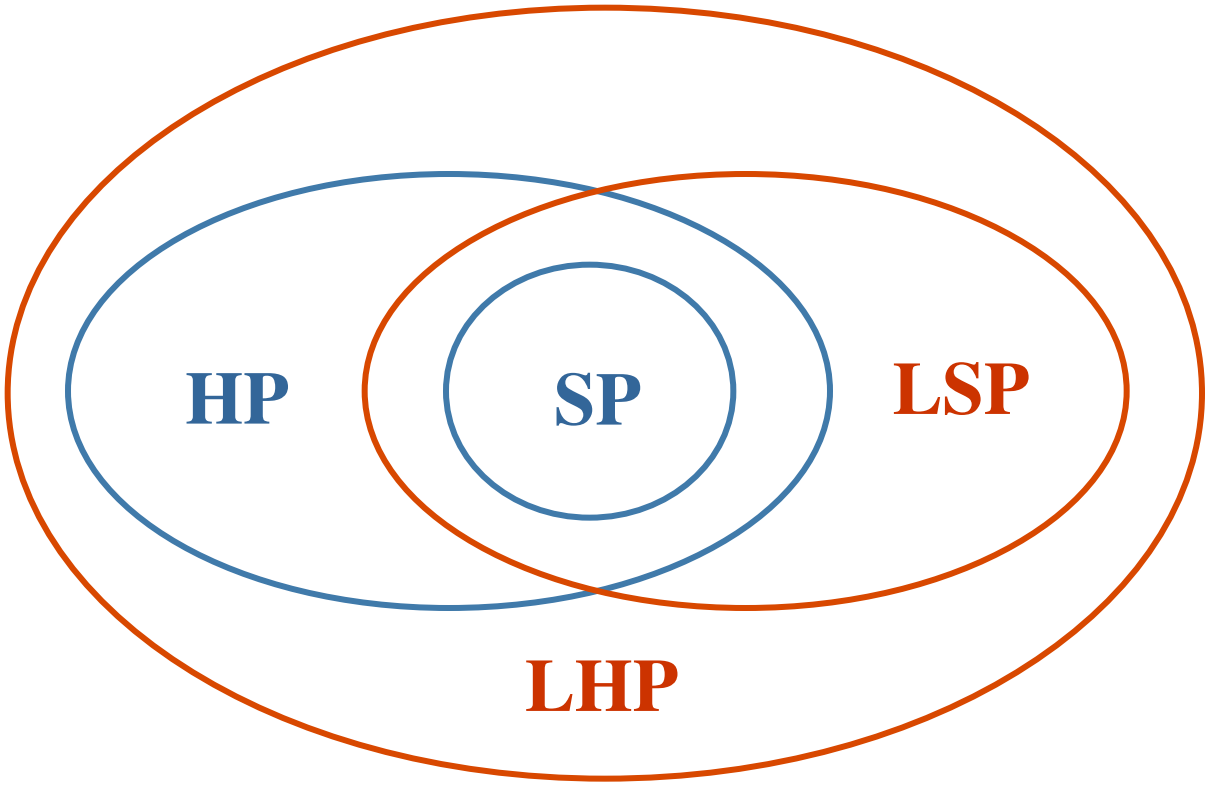
Key idea for partially dynamic



Key idea for fully dynamic



Putting things into perspective…



The fully dynamic update algorithm

Idea: Maintain all the locally historical paths of the graph

Fully dynamic update algorithm very similar to partially dynamic, but maintains locally historical paths instead of locally shortest paths (+ performs some other operations)

$O(n^2 \log^3 n)$ amortized time per update

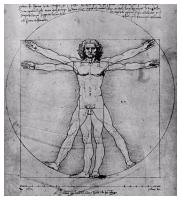
$O(mn \log n)$ space

More details in

Locally shortest paths:
[Demetrescu-Italiano'04]
C. Demetrescu and G.F. Italiano
A New Approach to Dynamic All Pairs Shortest Paths
Journal of the Association for Computing Machinery (JACM), 51(6), pp. 968-992, November 2004

Dijkstra's variant based on locally shortest paths:
[Demetrescu-Italiano'06]
Camil Demetrescu, Giuseppe F. Italiano: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Transactions on Algorithms 2 (4): 578-601 (2006).

Further Improvements [Thorup, SWAT'04]

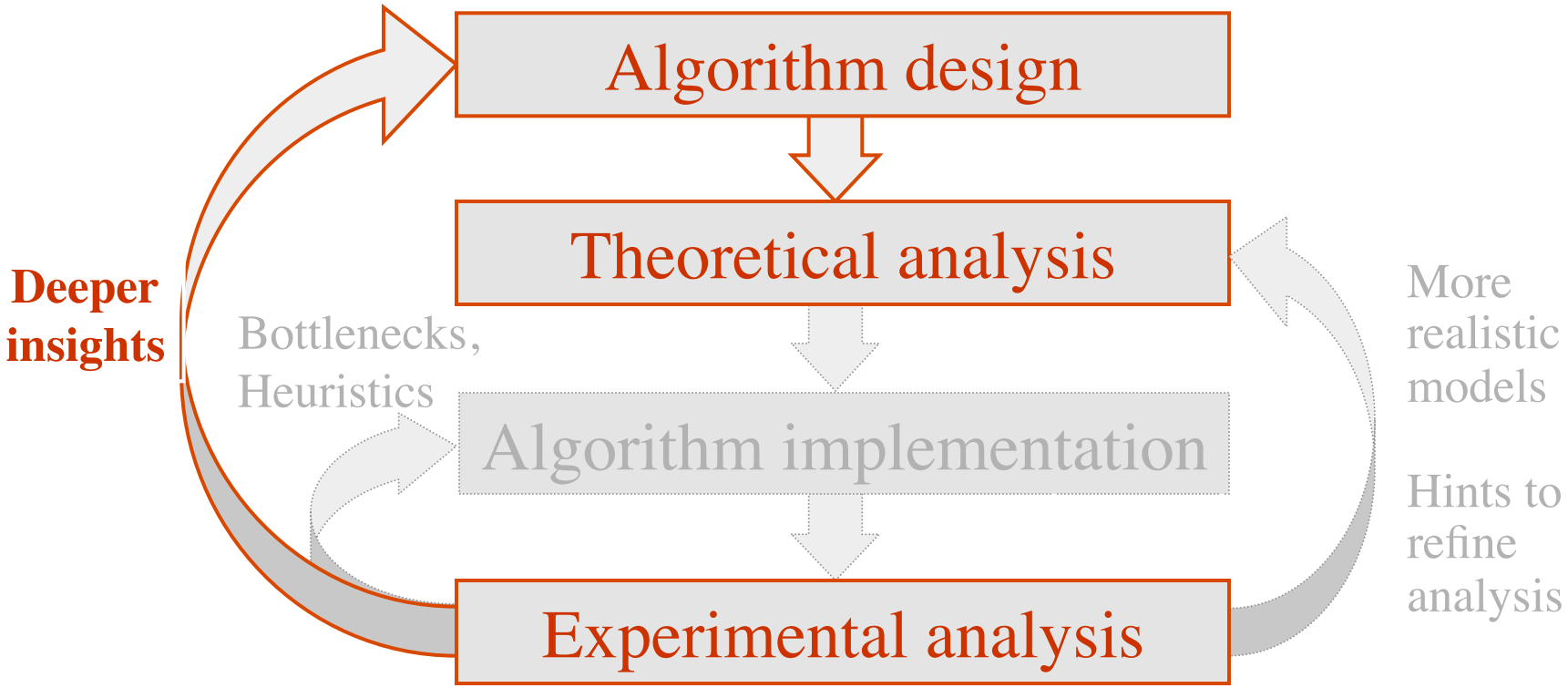


Using locally historical paths, Thorup has shown:

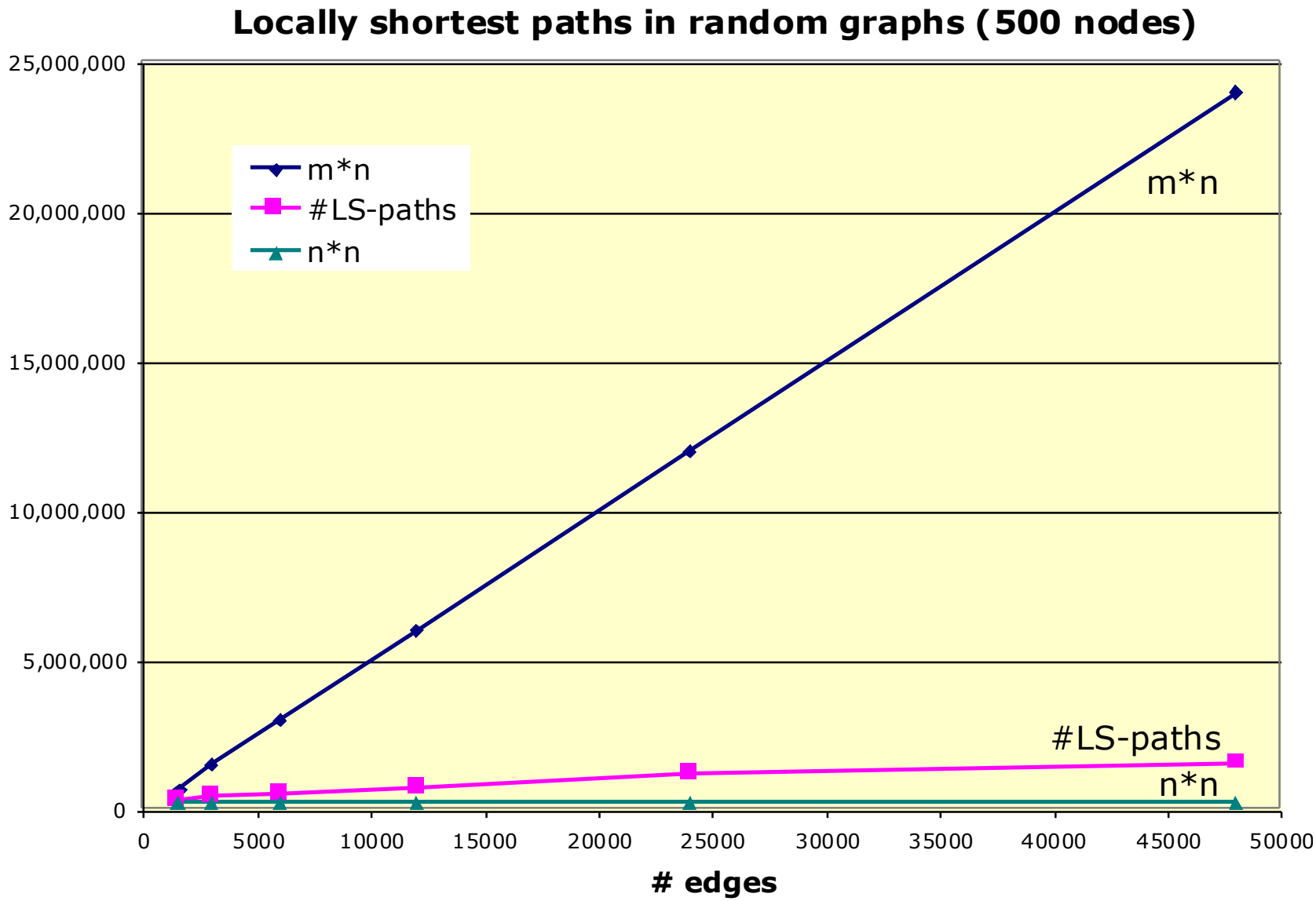
$O(n^2 (\log n + \log^2 (m/n)))$ amortized time per update

$O(mn)$ space

Another Return Trip to Theory:



How many LSPs in a graph?



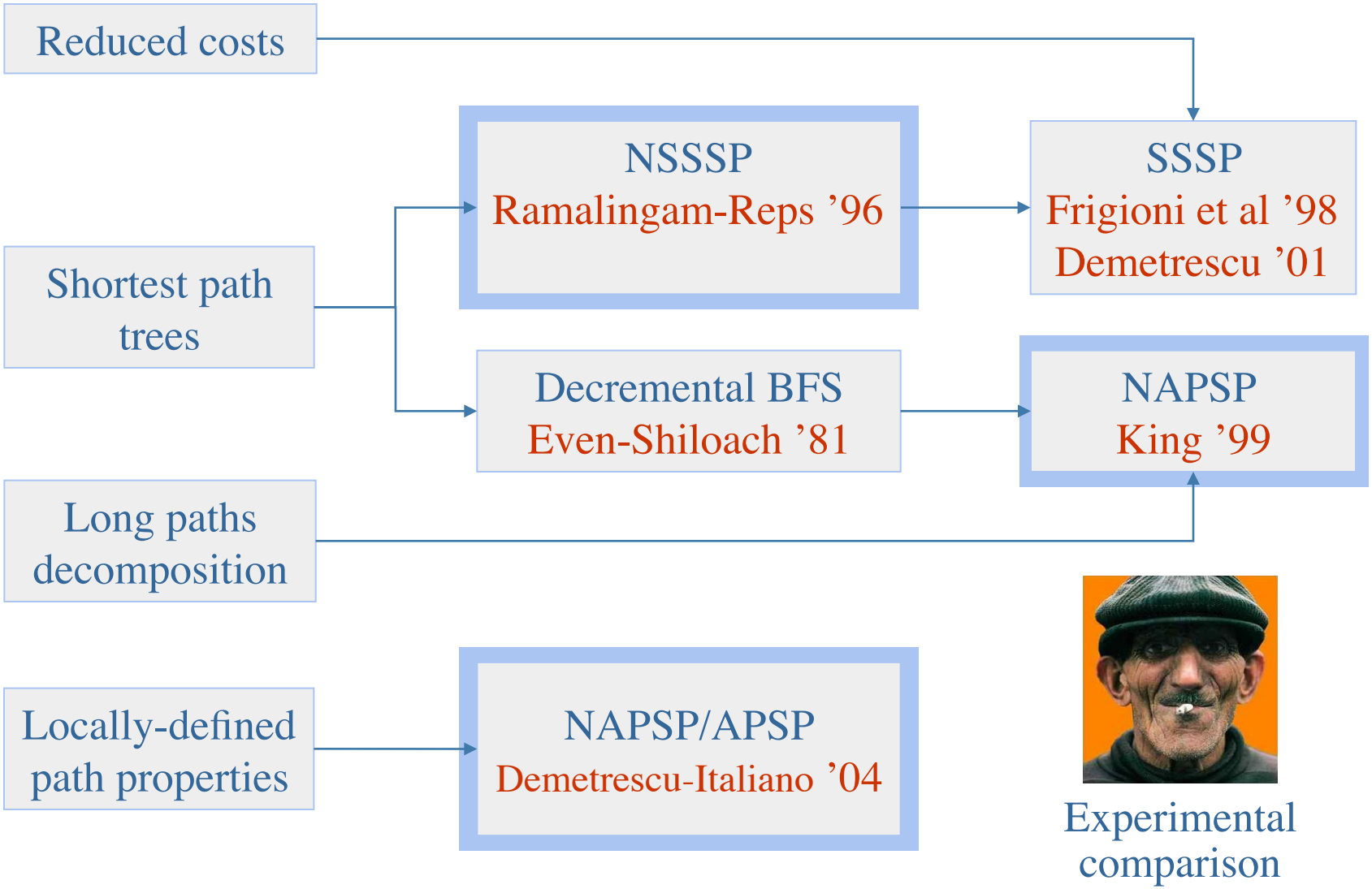
LSP's in Random Graphs

Peres, Sotnikov, Sudakov & Zwick [FOCS 10]
Complete directed graph on n vertices with edge weights chosen independently and uniformly at random from [0;1]:

Number of locally shortest paths is $O(n^2)$, in expectation and with high probability.

This yields immediately that APSP can be computed in time $O(n^2)$, in expectation and with high probability.

Dynamic shortest paths: roadmap



A comparative experimental analysis

(Dynamic) NAPSP algorithms under investigation

	Name	Weight	Update	Query
Dijkstra 59 (FT 87)	S-DIJ	real	$O(mn + n^2 \log n)$	$O(1)$
Demetrescu/I. 06	S-LSP	real	$O(\#LSP + n^2 \log n)$	$O(1)$
Ramaling./Reps 96 (SIMPLE)	D-RRL	real	$O(mn + n^2 \log n)$	$O(1)$
King 99	D-KIN	[0,C]	$O(n^{2.5} (C \log n)^{0.5})$	$O(1)$
Demetrescu/I. 04	D-LHP	real	$\tilde{O}(n^2)$	$O(1)$

Experimental setup

- Test sets
  - Random (strongly connected)
  - US road maps (n = hundreds to thousands)
  - AS Internet subgraphs (thousands of nodes)
  - Pathological instances
  - Random updates / pathological updates
- Hardware (few years ago)
  - Athlon 1.8 GHz - 256KB cache L2 - 512MB RAM
  - Pentium IV 2.2GHz - 512KB cache L2 - 2GB RAM
  - PowerPC G4 500MHz - 1MB cache L2 - 384MB RAM
  - IBM Power 4 - 32MB cache L3 - 64GB RAM

Experimental setup

Operating systems

- Linux
- Solaris
- Windows 2000/XP
- Mac OS X

Compilers & Analysis Tools

- gcc (GNU)
- xlc (Intel compiler)
- Microsoft Visual Studio
- Metrowerks CodeWarrior

- Valgrind (monitor memory usage)
- Cachegrind (cache misses)

Implementation issues

For very sparse graphs, heap operations are crucial, so good data structures (buckets, smart queues, etc.) make a difference…

In our experiments, we were mainly interested in edge scans for different graph densities

Not the best possible implementations (some library overhead): we look for big (> 2x) relative performance ratios

A lot of tuning: we tried to find a good setup of relevant parameters for each implementation

Algorithm D-RRL [Demetr.'01]

Directed graphs with real edge weights

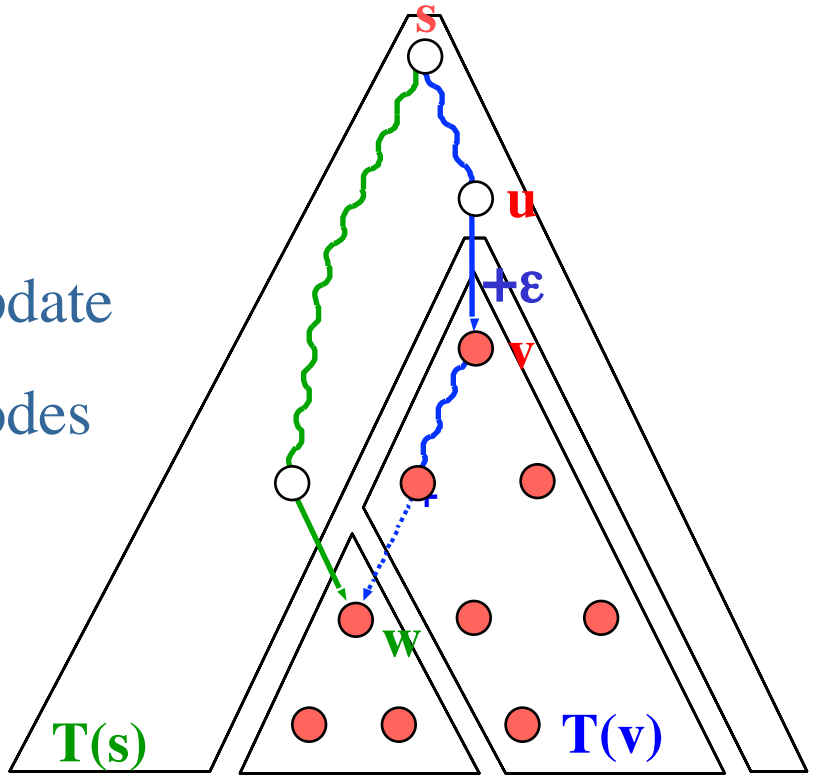
$O(mn+n^2 \log n)$ update time	$O(1)$ query time	$O(n^2)$ space

Approach:

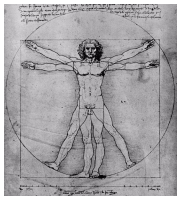
Maintain n shortest path trees

Work on each tree after each update

Run Dijkstra variant only on nodes of the affected subtree (SIMPLE algorithm described earlier)



Algorithm D-KIN [King' 99]

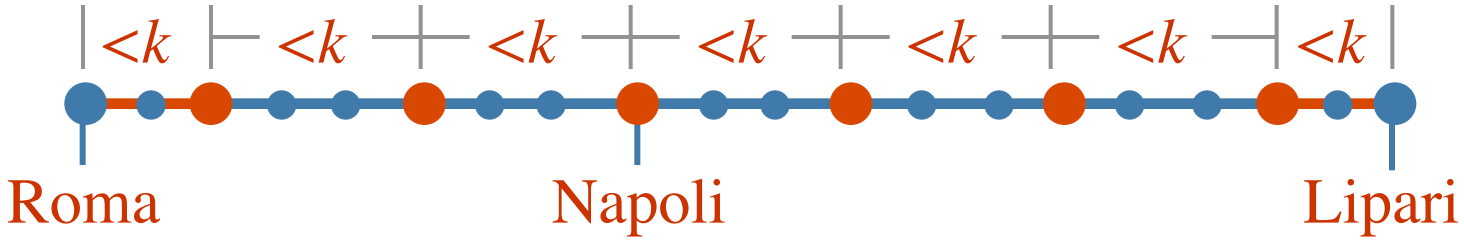


Directed graphs with integer edge weights in [0,C]

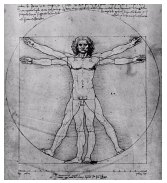
$\tilde{O}(n^{2.5}\sqrt{C})$ update time	$O(1)$ query time	$\tilde{O}(n^{2.5}\sqrt{C})$ space

Approach:

1. Maintain dynamically shortest paths up to length $k=(nC)^{0.5}$ using variant of decremental data structure by Even-Shiloach
2. Stitch together short paths from scratch to form long paths exploiting long paths decomposition



Algorithm D-LHP [Dem.-Italiano' 03]

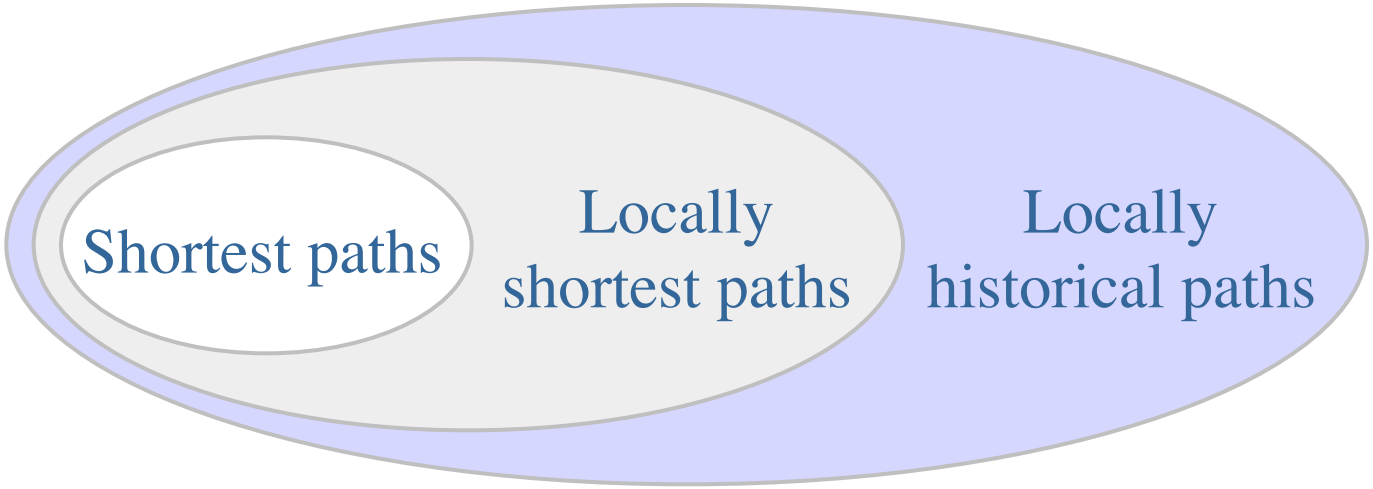


Directed graphs with real edge weights

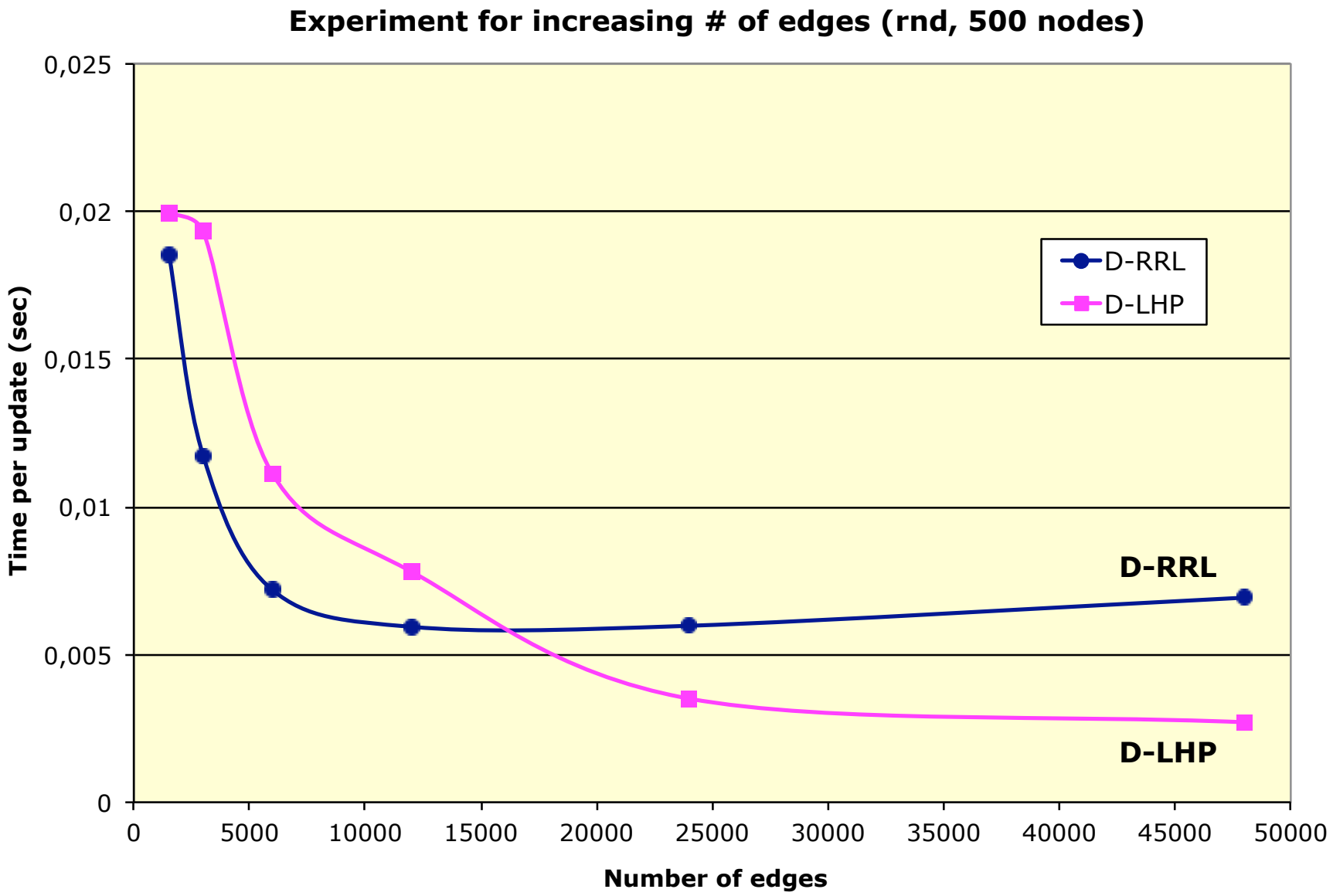
$\tilde{O}(n^2)$ time per update	$O(1)$ time per query	$\tilde{O}(mn)$ space

Approach:

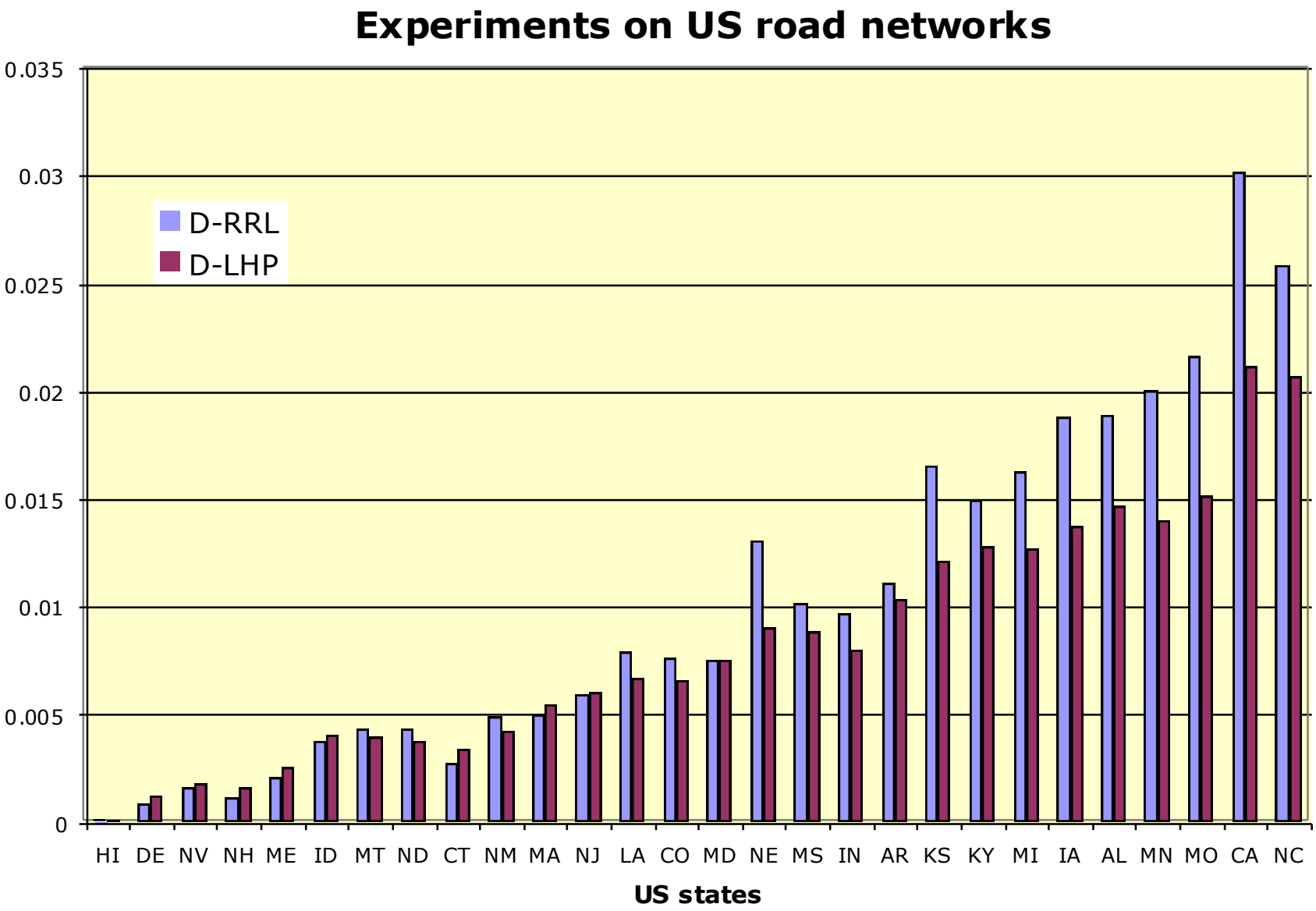
Maintain locally historical paths (LHP):
paths whose proper subpaths have been shortest paths…



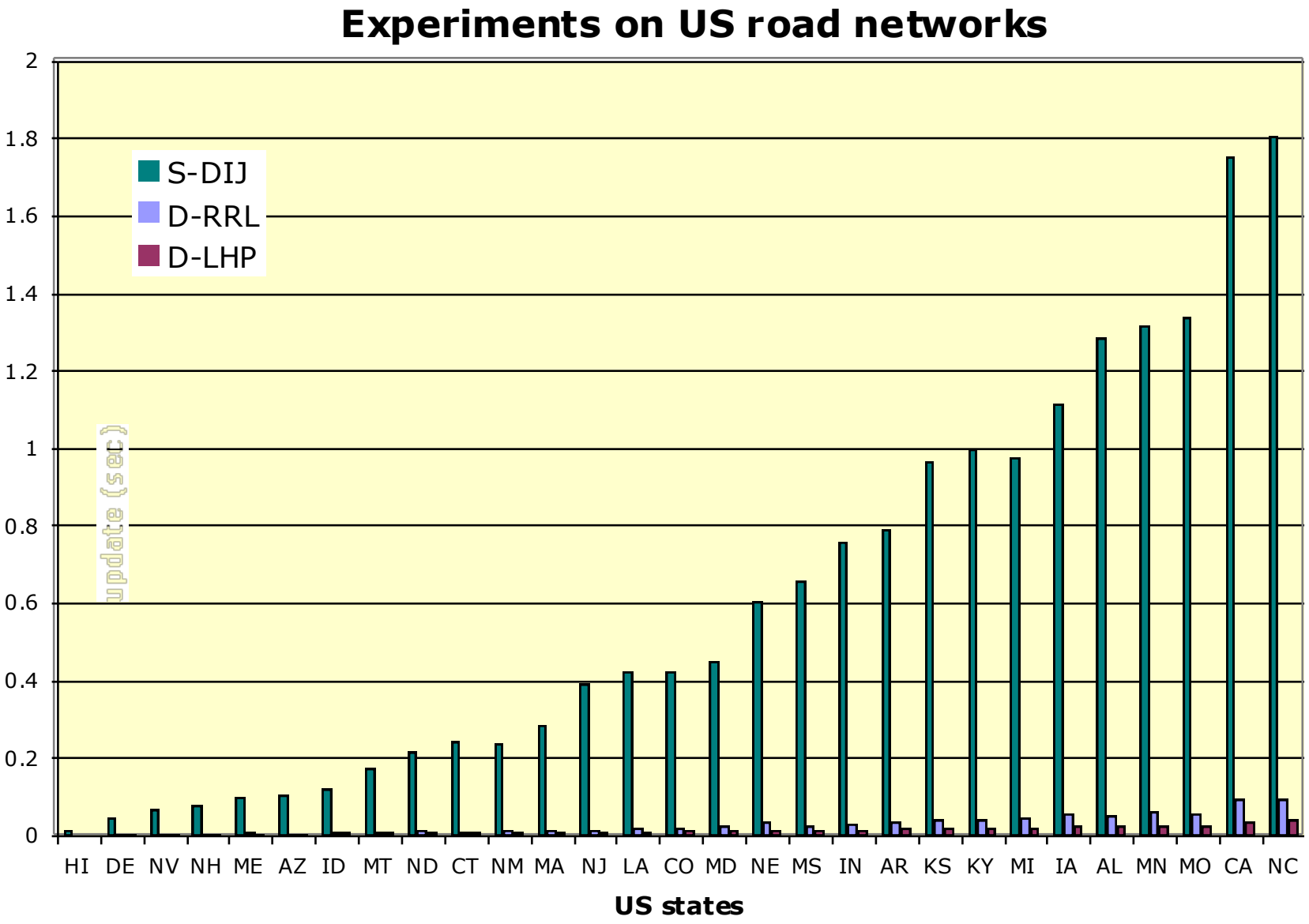
Are LHPs useful in practice? (update time)



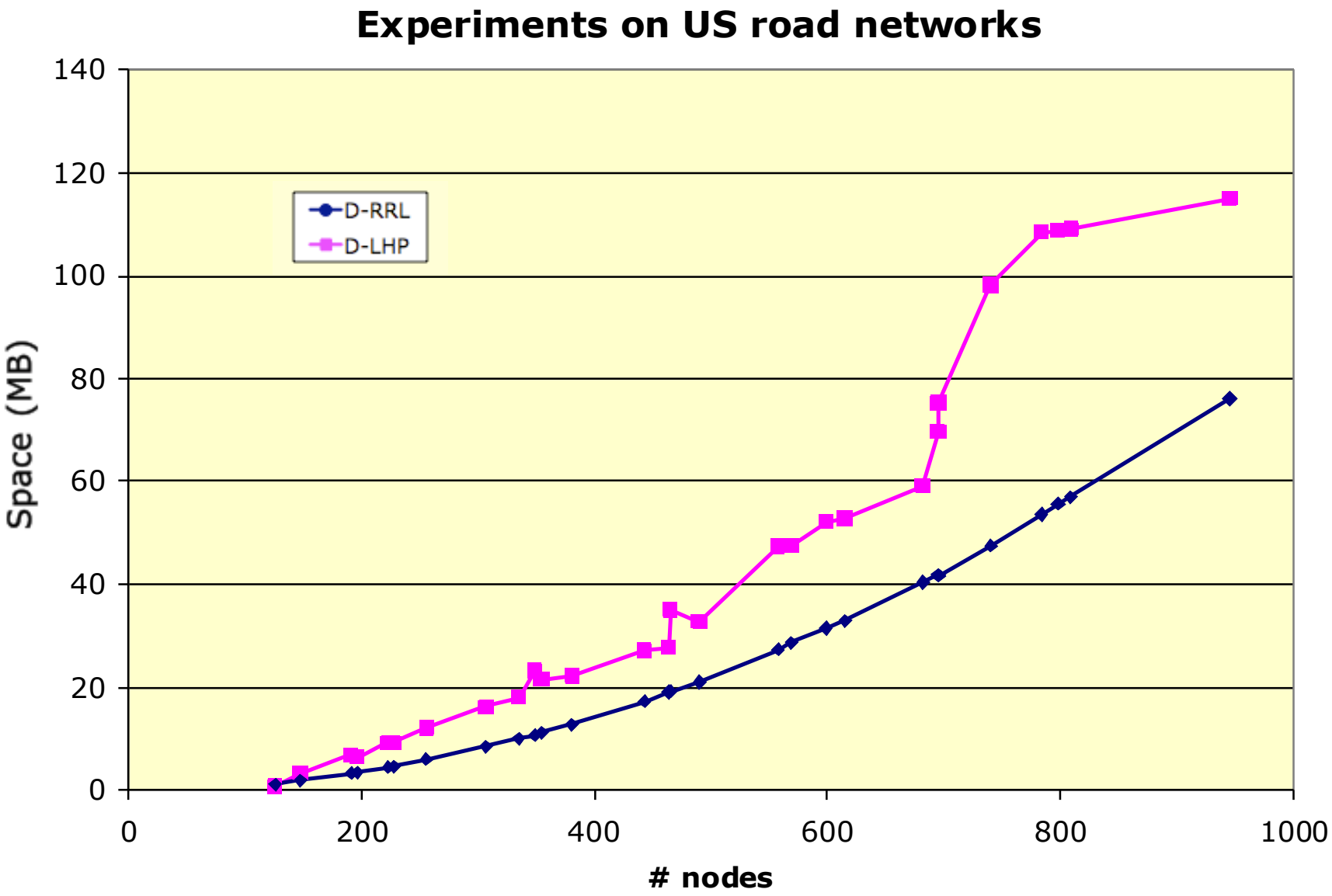
What about real graphs? (update time)



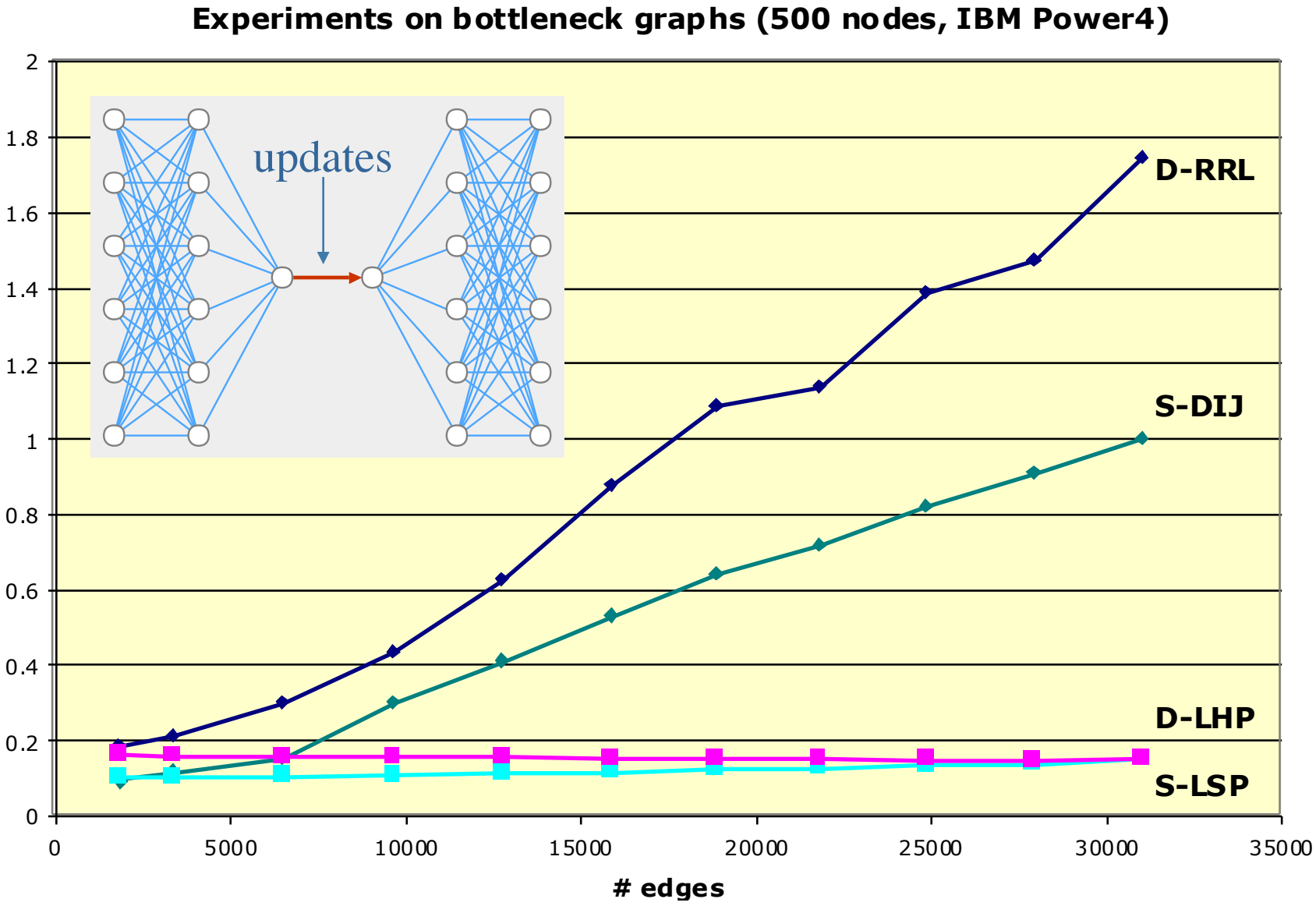
Zoom out to static (update time)



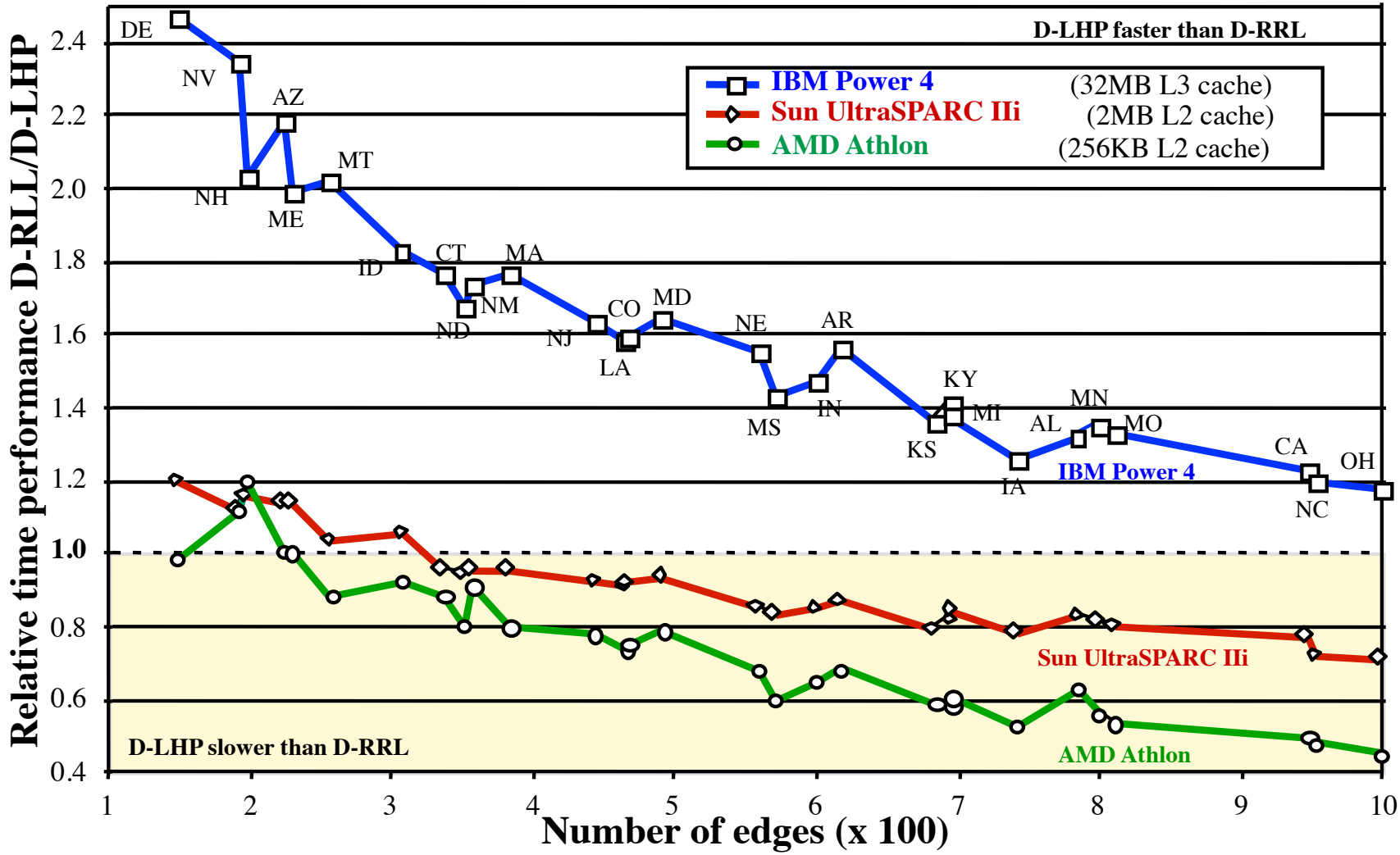
Big issue: the space wall



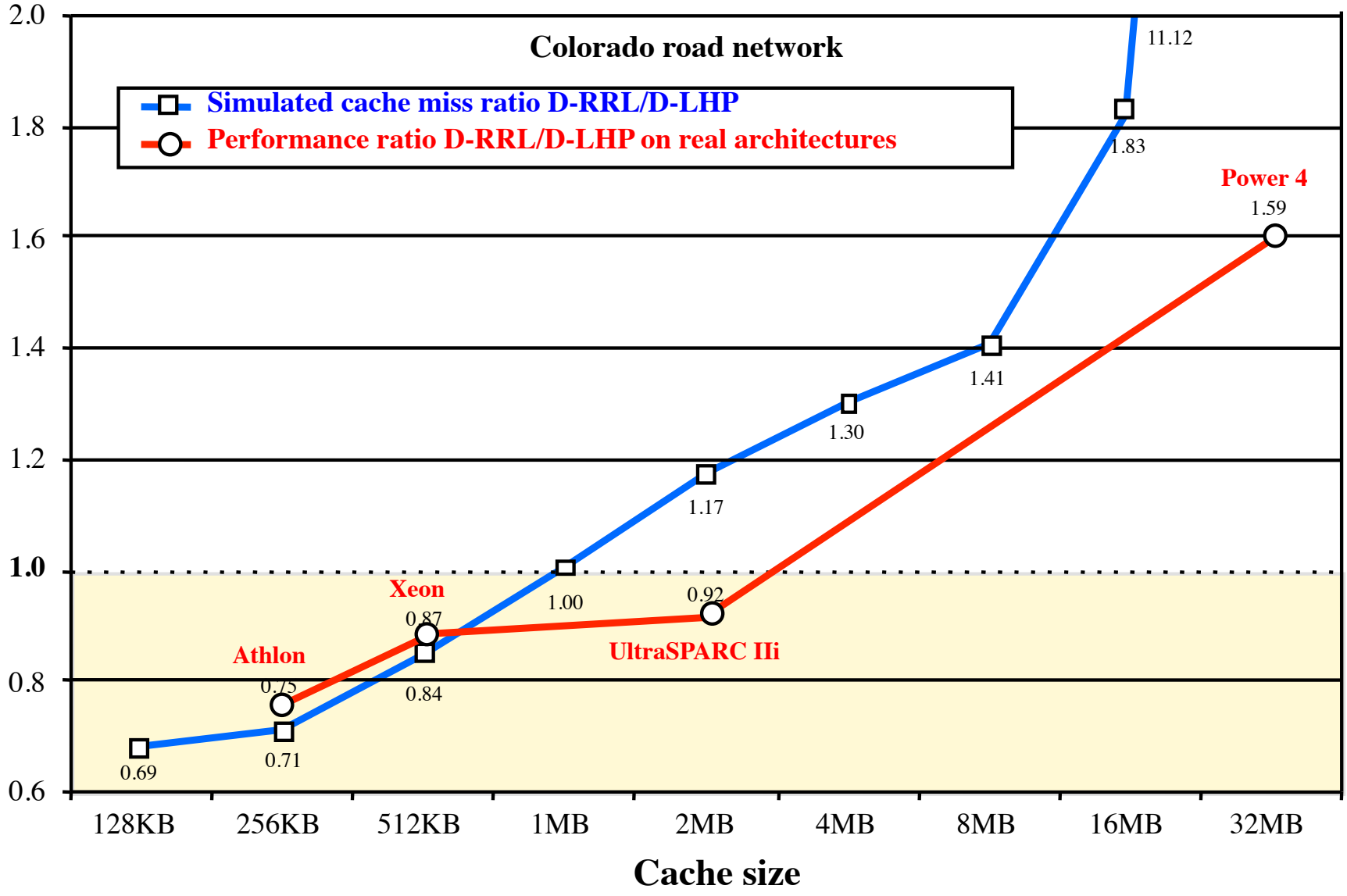
Worst-case instances



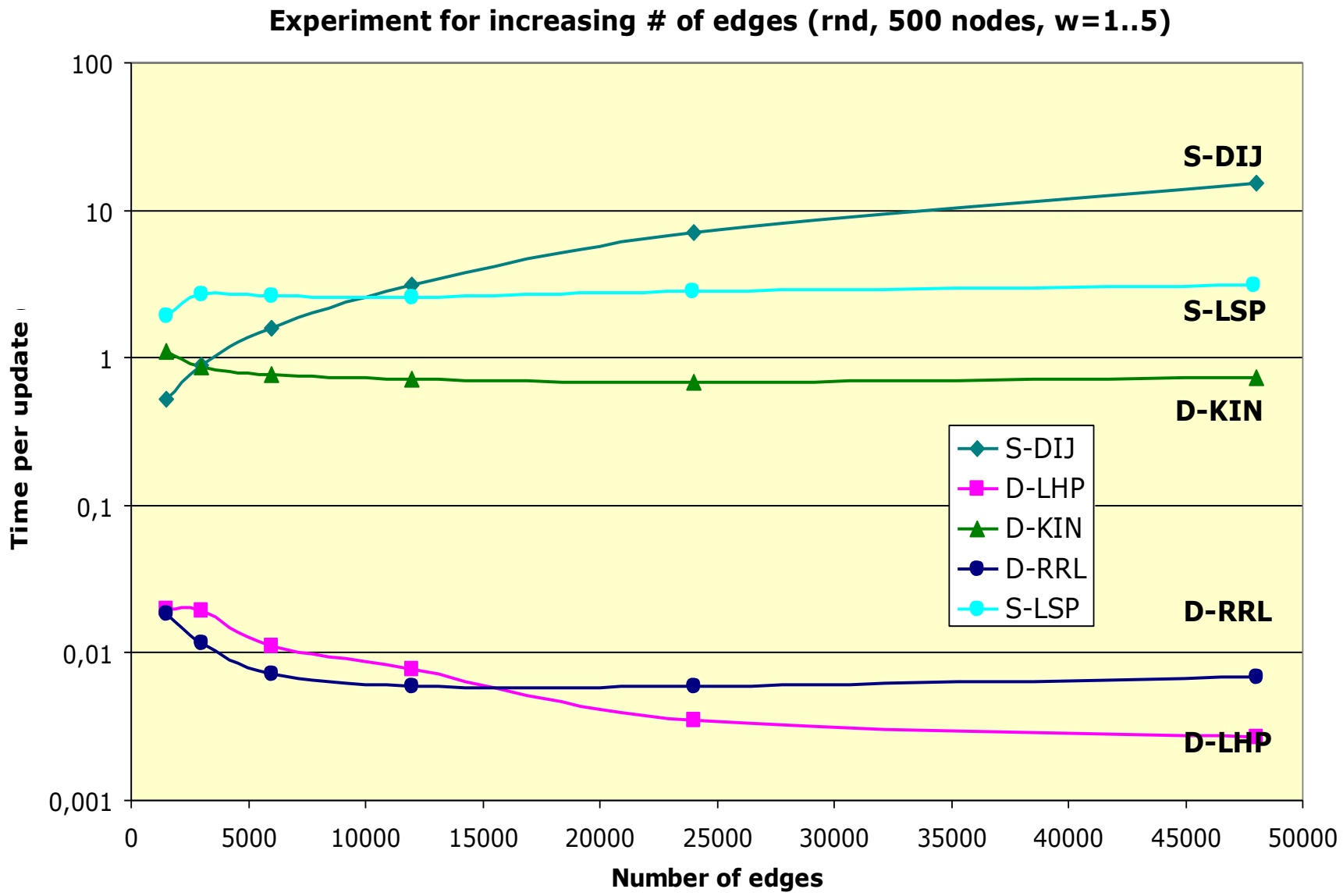
Different HW platforms



Cache effects



The big picture (update time)



What did we learn for sparse graphs?

Best that one could hope for (in practice):

Small data structure overhead

Work only on the affected shortest paths

D-RRL (Dem.'01 implem. Ram-Reps approach):

Very simple	Hard to beat! (but quite bad on pathological instances)

D-LHP (Dem-Ita'04):

Can be as efficient as D-RLL (best with good memory hierarchy: cache, memory bandwidth)

D-KIN (King'99):

Overhead in stitching and data structure operations

What did we learn for dense graphs?

Locally shortest/historical paths can be very useful

Dynamic algorithm D-LHP is the fastest in practice on all the test sets and platforms we considered

Even on static algorithm S-LSP can beat S-DIJ by a factor of 10x in practice on dense graphs

Concluding remarks

#locally shortest paths ≈ #shortest paths
in all the graphs we considered (real/synthetic)

Careful implementations might fully exploit this
(by keeping data structure overhead
as small as possible)

Space wall! Time kills you slowly, but space can kill you right away…
With 5000 vertices, 80 bytes per vertex pair:
quadratic space means 2GB
With current RAMs, that's about it!

More details in

Algorithm D-LHP:
[Demetrescu-Italiano'04]
C. Demetrescu and G.F. Italiano
A New Approach to Dynamic All Pairs Shortest Paths
Journal of the Association for Computing Machinery (JACM), 51(6), pp. 968-992, November 2004

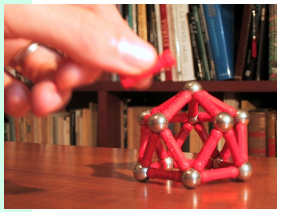
Computational study of dynamic NAPSP algorithms:
[Demetrescu-Italiano'06]
C. Demetrescu, G. F. Italiano: Experimental analysis of dynamic all pairs shortest path algorithms. ACM Transactions on Algorithms 2 (4): 578-601 (2006).

Outline

Dynamic Graph Problems

Methodology & State of the Art

Algorithmic Techniques



Conclusions

More Work to be done on Dynamic APSP

- Space is a **BIG** issue in practice
- More tradeoffs for dynamic shortest paths?

  Roditty-Zwick, Algoritmica 2011
  $\tilde{O}(mn^{1/2})$ update, $O(n^{3/4})$ query for unweighted
- Worst-case bounds?

  Thorup, STOC 05
  $\tilde{O}(n^{2.75})$ update

Some Open Problems…

- *Lower Bounds?*

Some Open Problems…

- *Fully Dynamic Single-Source Shortest Path?*

  Nothing better than simple-minded approaches

- *Fully Dynamic Single-Source Single-Sink Shortest Path?*

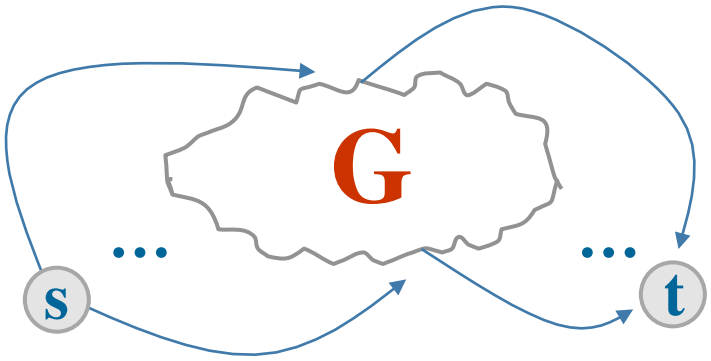
  Nothing better than simple-minded approaches

In static case, those problems are easier than APSP…

Fully Dynamic SSSP (SSSS) not easier?

Claim. *If Fully Dynamic SSSS can be solved in time O(f(n)) per update and query, then also Fully Dynamic APSP can be solved in time O(f(n)) per update and query.*

G'



Edges from s to G and from G to t have cost $+\infty$

All-Pairs $query_G(x,y)$ can be implemented in G' as follows:

$update_{G'}(s,x,0)$; $update_{G'}(y,t,0)$; $query_{G'}(s,t)$;

$update_{G'}(s,x,+\infty)$; $update_{G'}(y,t,+\infty)$

Some Open Problems…

- *Dynamic Maximum st-Flow*
  Dynamic algorithm only known for planar graphs
  $O(n^{2/3} \log^{8/3} n)$ time per operation
  I., Nussbaum, Sankowski & Wulf-Nilsen [STOC 2011]
  What about general graphs?

- *Dynamic Diameter*
  `Diameter()`:
  what is the diameter of G?
  Do we really need APSP for this?

Some Open Problems…

- *Dynamic Strongly Connected Components (directed graph G)*

  `SCC(x,y)`:

  Are vertices x and y in same SCC of G?

  In static case strong connectivity easier than transitive closure….

- *Static Problem: Find Biconnectivity Components of Directed Graphs*