Dynamic Graph Algorithms

Giuseppe F. Italiano

University of Rome "Tor Vergata"

giuseppe.italiano@uniroma2.it

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

Outline

Dynamic Graph Problems – Quick Intro

Lecture 1. (Undirected Graphs)
Dynamic Connectivity

Lecture 2. (Undirected/Directed Graphs)
Dynamic Shortest Paths

Lecture 3. (Non-dynamic?)
2-Connectivity in Directed Graphs

Outline

Dynamic Graph Problems – Quick Intro

Lecture 1. (Undirected Graphs)
Dynamic Connectivity

Lecture 2. (Undirected/Directed Graphs)
Dynamic Shortest Paths

Lecture 3. (Non-dynamic?)
2-Connectivity in Directed Graphs

Several Variants

- *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*

Several Variants

- ***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*

Miscellanea

- Without loss of generality, directed graphs
- W.l.o.g., update operations restricted to edge cost changes: cost decreases can simulate insertions; cost increases can simulate deletions. (If edge not there, cost of $+\infty$)
- Subpath Optimality (Optimal Substructure): any subpath of a shortest path is a shortest path

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( )]

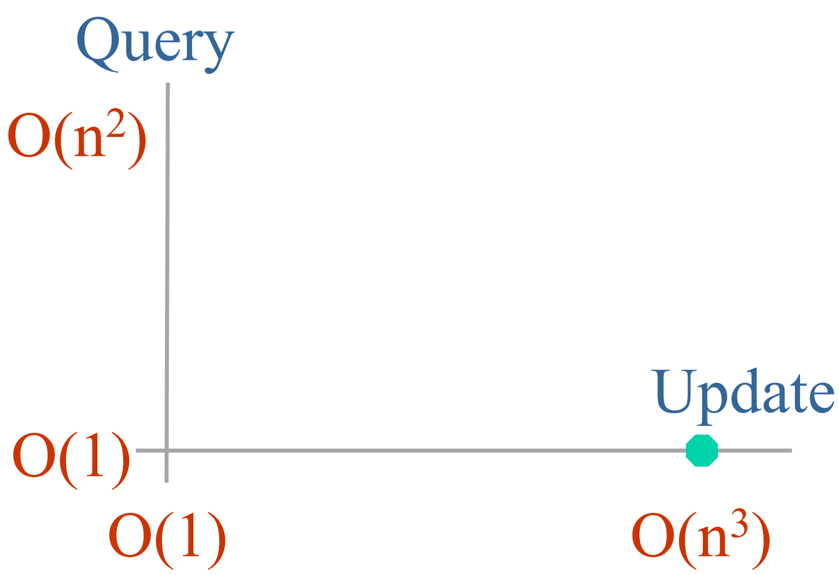
Query(x,y): return distance from x to y (or shortest path from x to y)

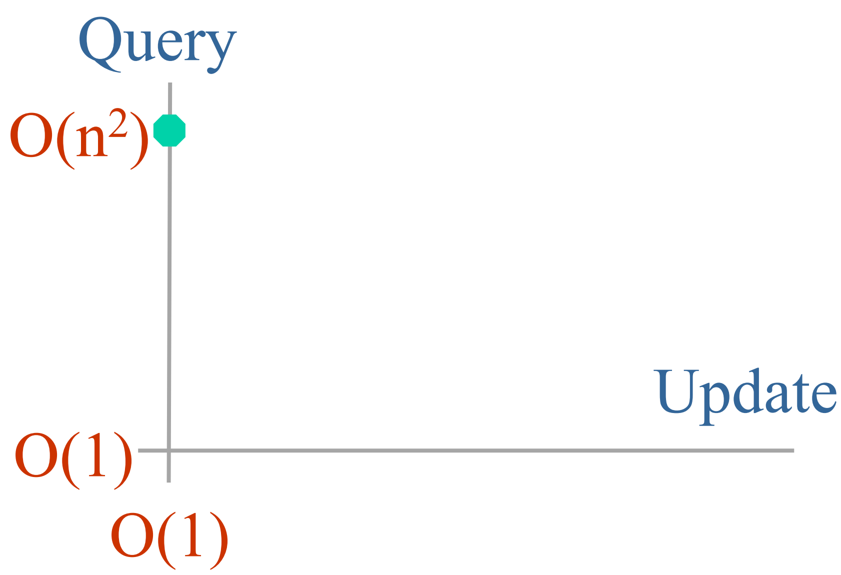
Simple-minded Approaches

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

Simple-minded Approaches

Fast query approach	Fast update approach
Rebuild the distance matrix from scratch after each update.	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

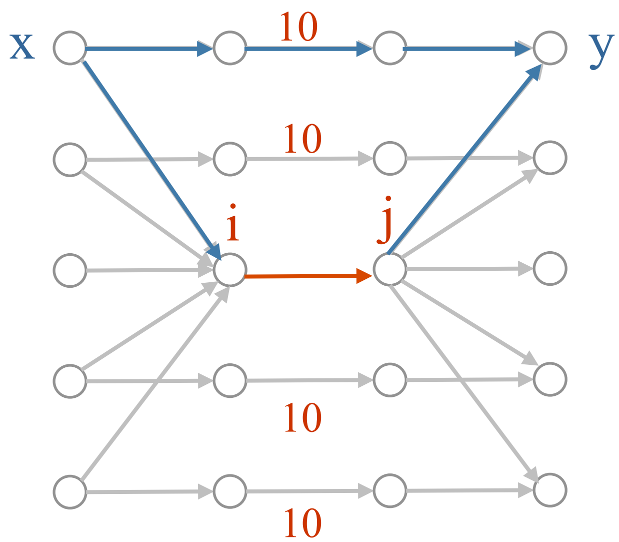
Until 1999, none of them was better in the worst case than recomputing APSP from scratch (~ cubic time!)

	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, TR 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

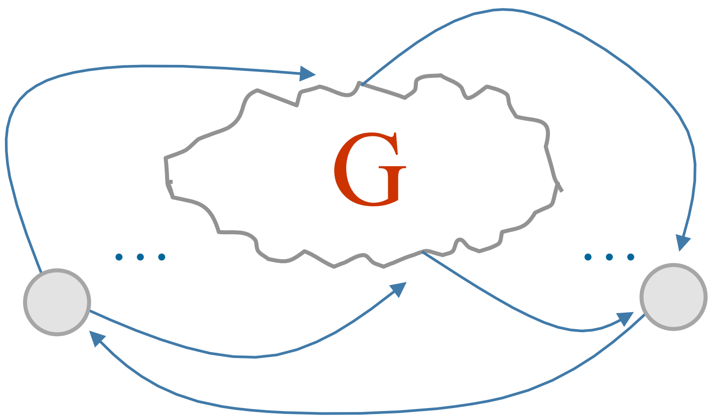
$$D(x,i) + w(i,j) + D(j,y) < D(x,y)$$

Quite easy: $O(n^2)$

Fully Dynamic APSP

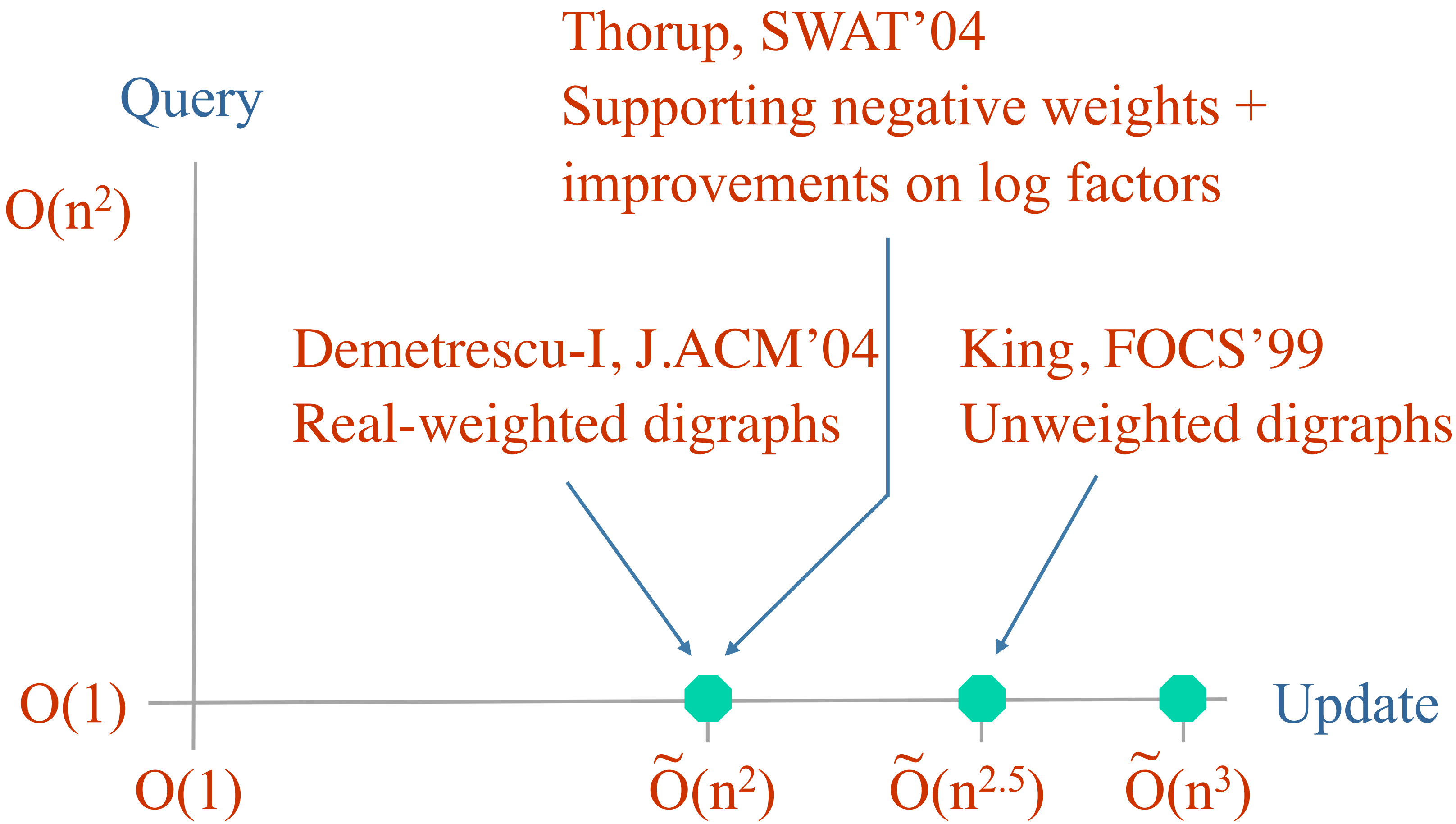
- Edge deletions (edge cost increases)

  Seem the hard operations. Intuition:



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

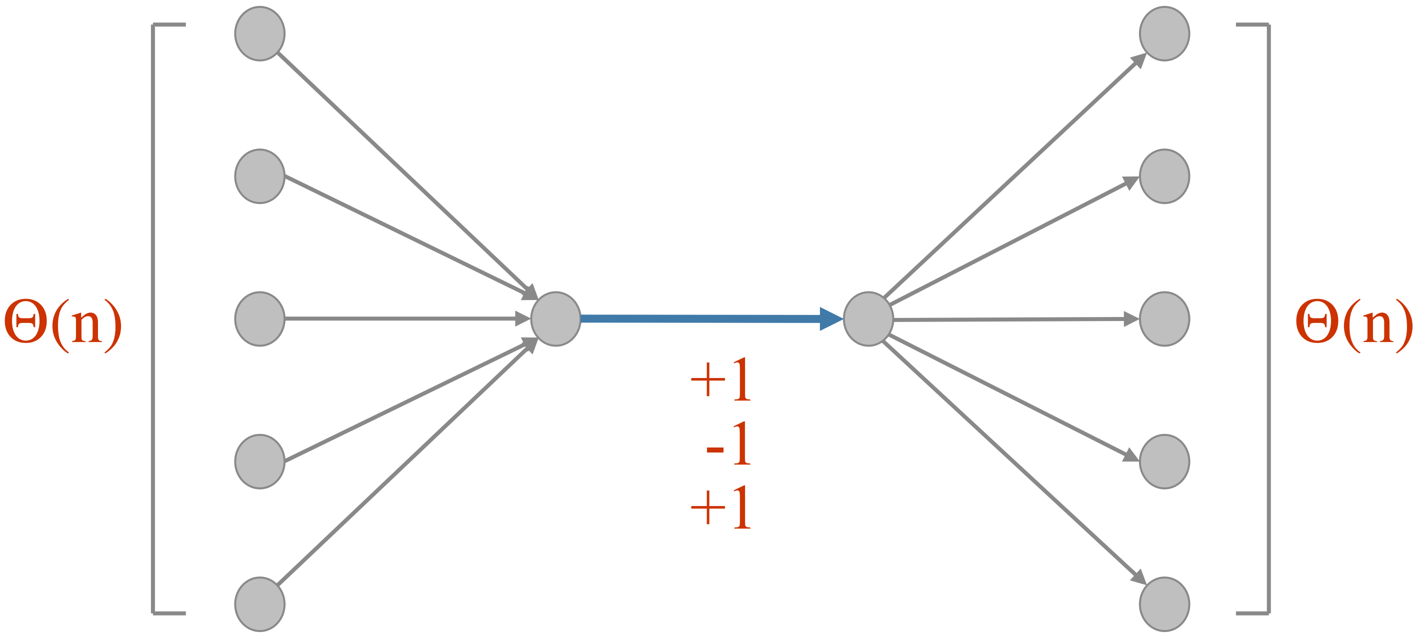
Dynamic APSP



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

Dynamic Shortest Paths

Many interesting ideas and techniques introduced

- Algebraic graph methods
- Decremental BFS [Even & Shiloach 1981]
- Locally shortest paths
- Long paths property
- Path decompositions
- …

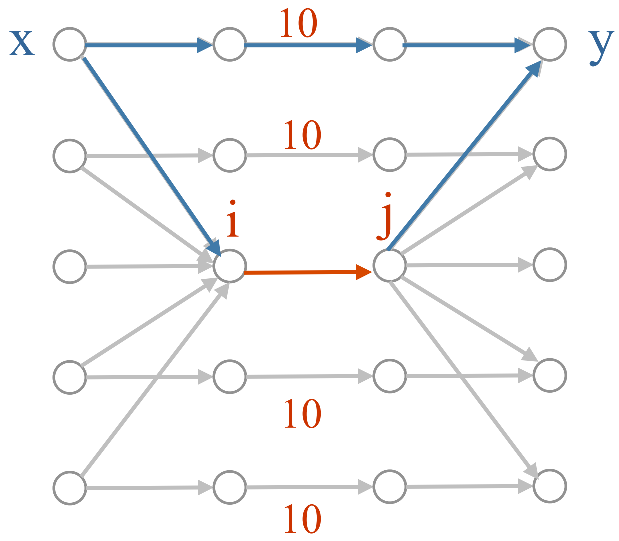
Dynamic Shortest Paths

Many interesting ideas and techniques introduced

- Algebraic graph methods
- Decremental BFS [Even & Shiloach 1981]
- Locally shortest paths
- Long paths property
- Path decompositions
- …

Fully Dynamic APSP (Recall)

Edge insertions (edge cost decreases)



For each pair x,y check whether

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

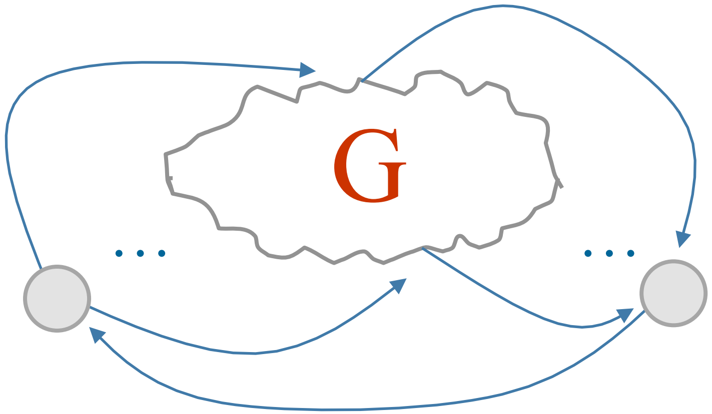
Quite easy: $O(n^2)$ $O(mn^2) = O(n^4)$ over a sequence

Question 1 : Can we do better?

Fully Dynamic APSP (Recall)

- Edge deletions (edge cost increases)

  Seem the hard operations. Intuition:



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

Question 2 : Can we keep this info?

Incremental Shortest Path

Edge insertions only

Show how to improve the $O(n^4)$ bound over $O(n^2)$ edge insertions ($O(n^2)$ worst-case per insertion)

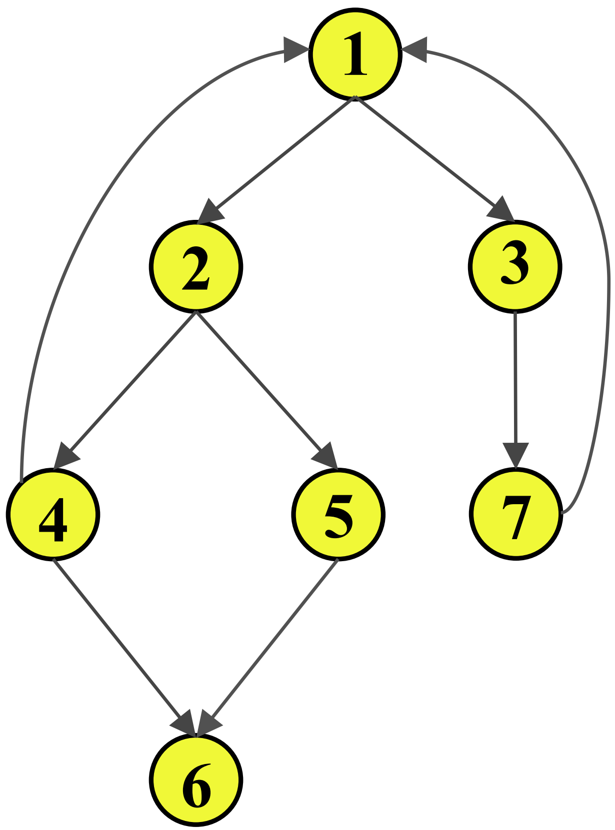
Unweighted (directed) graphs: $O(n^3 \log n)$ over $O(n^2)$ edge insertions ($O(n \log n)$ amortized per insertion)

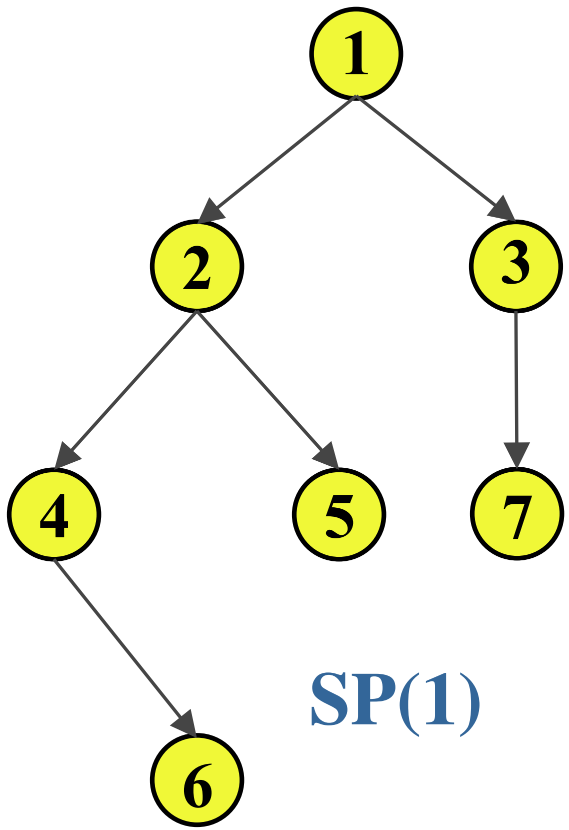
[Ausiello, I. , Marchetti-Spaccamela, Nanni J. Algs 1991]

Terminology

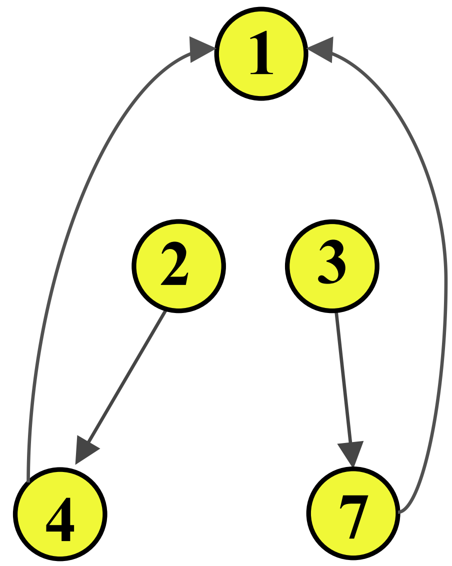
SP(v) : Shortest path tree rooted at vertex v

$SP^R(v)$: Shortest path tree rooted at v in reverse graph





SP(1)



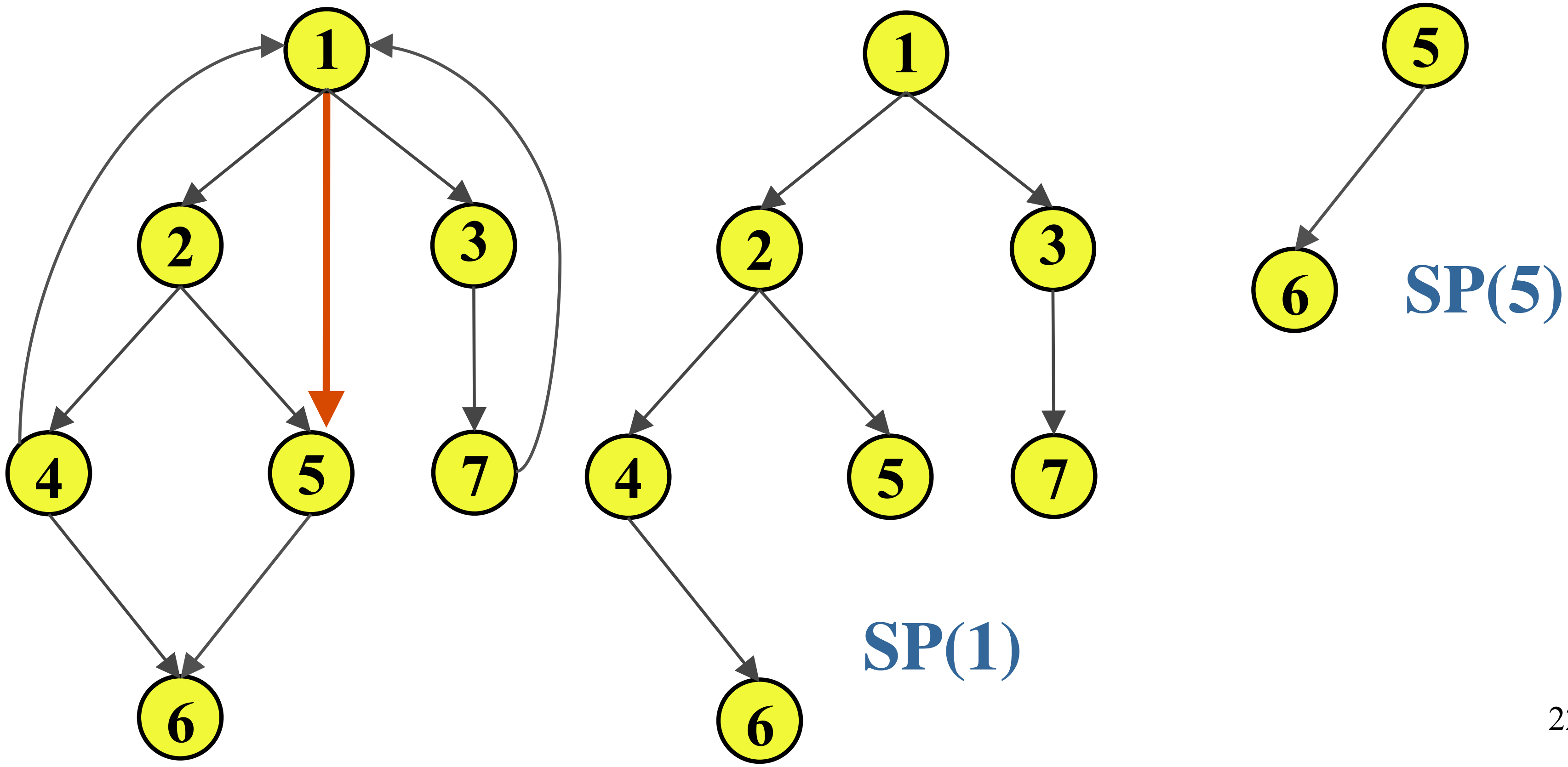
$SP^R(1)$

O(n^2) Update

When edge (i,j) is inserted do the following:

for each v in V, update SP(v) by considering SP(j)

(basic update)

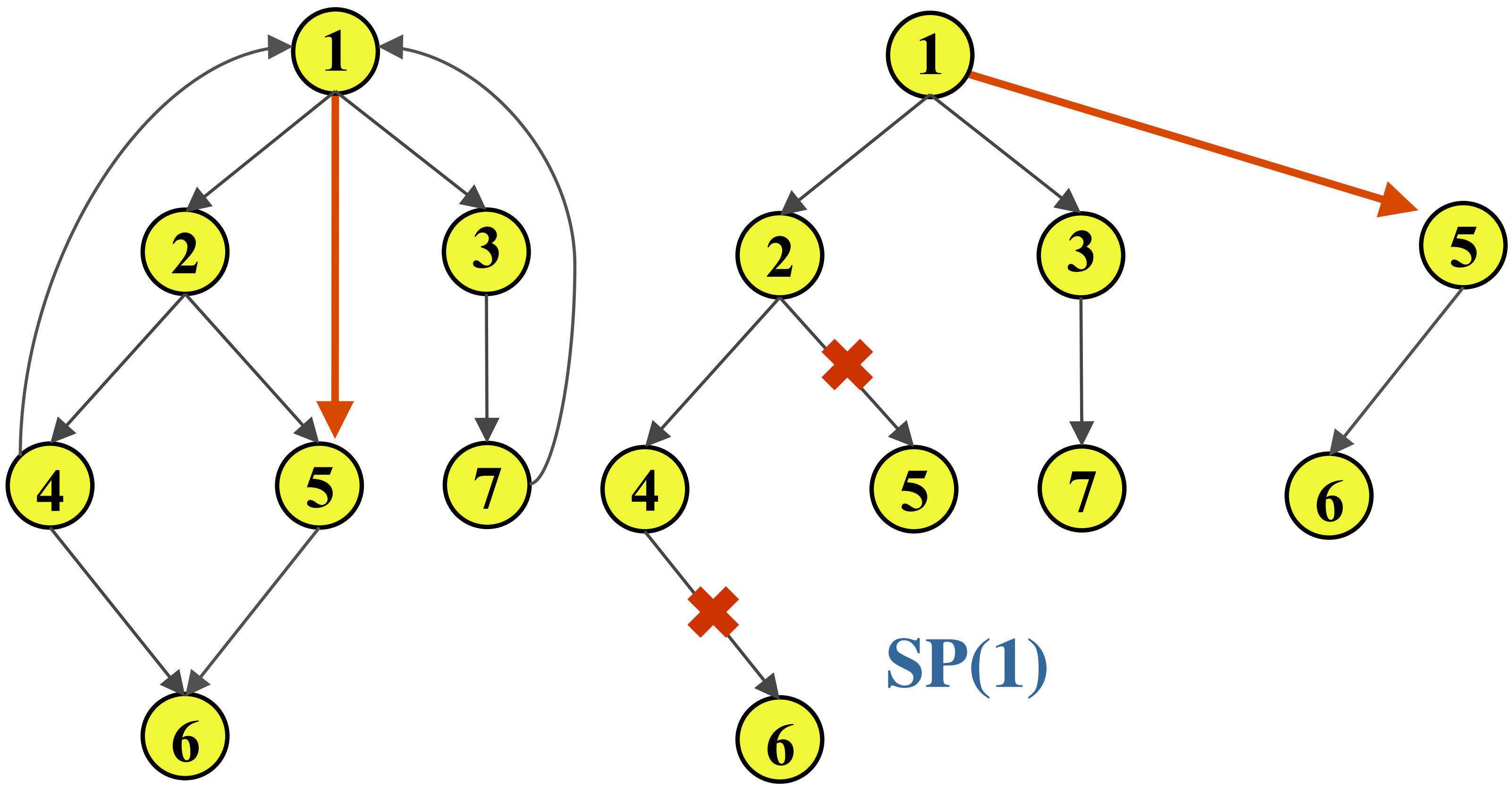


$O(n^2)$ Update

When edge (i,j) is inserted do the following:

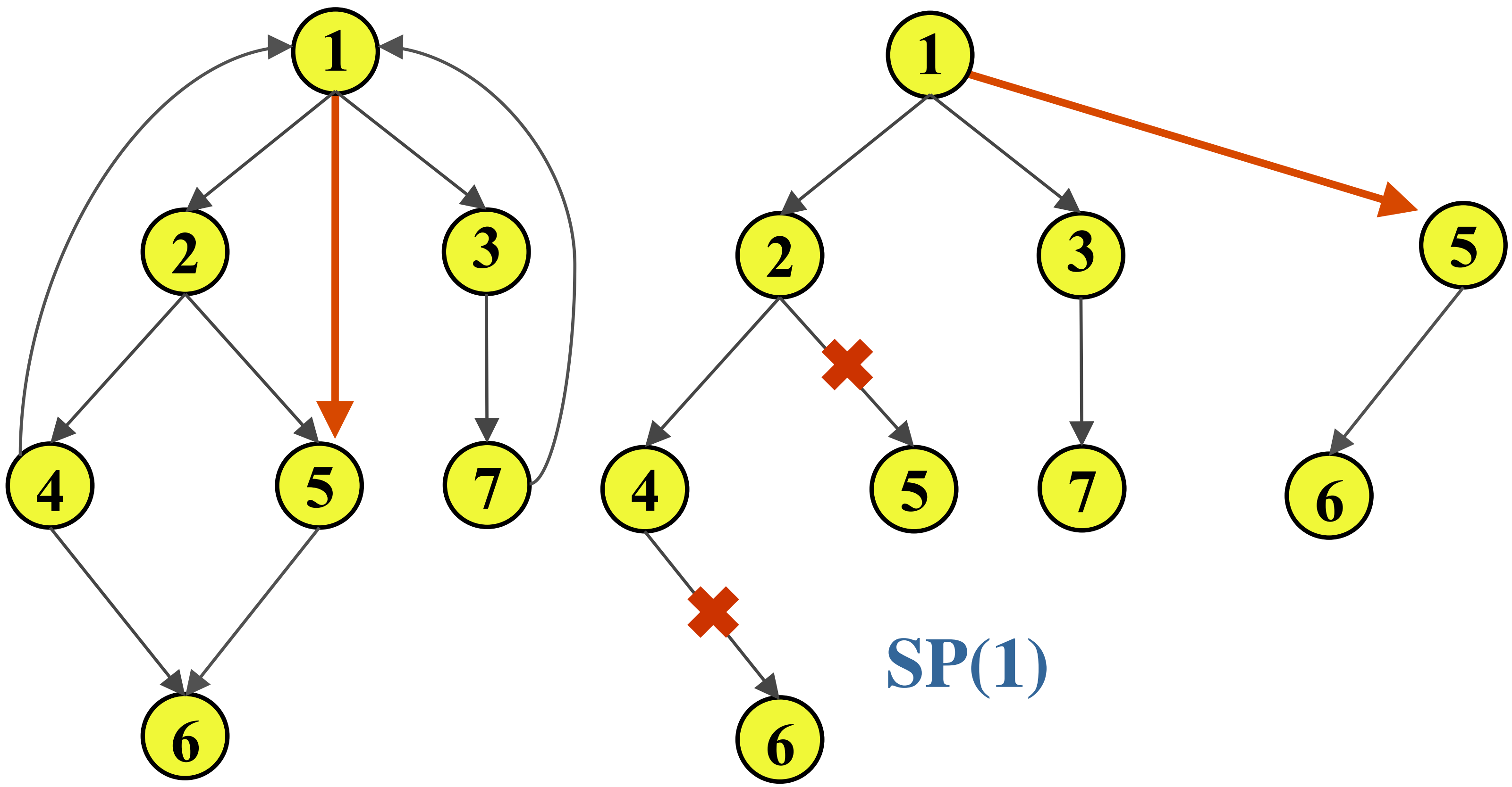
for each v in V, update SP(v) by considering SP(j)

(basic update)



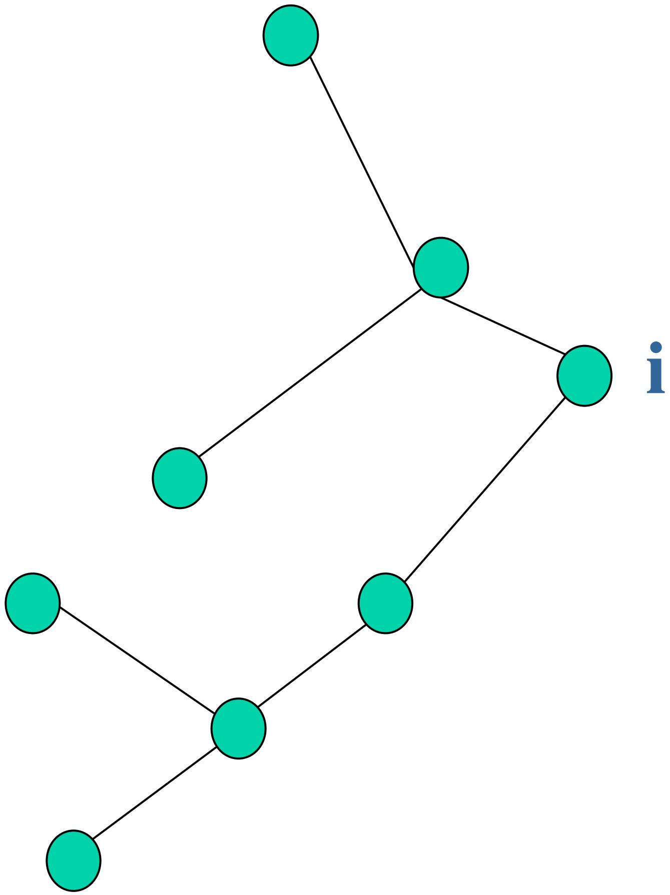
First Idea

When edge (i,j) is inserted do the following:
for each v in **$SP^R(i)$**, update SP(v) by considering SP(j)
(basic update)

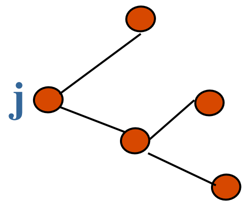


SP(1)

First Idea

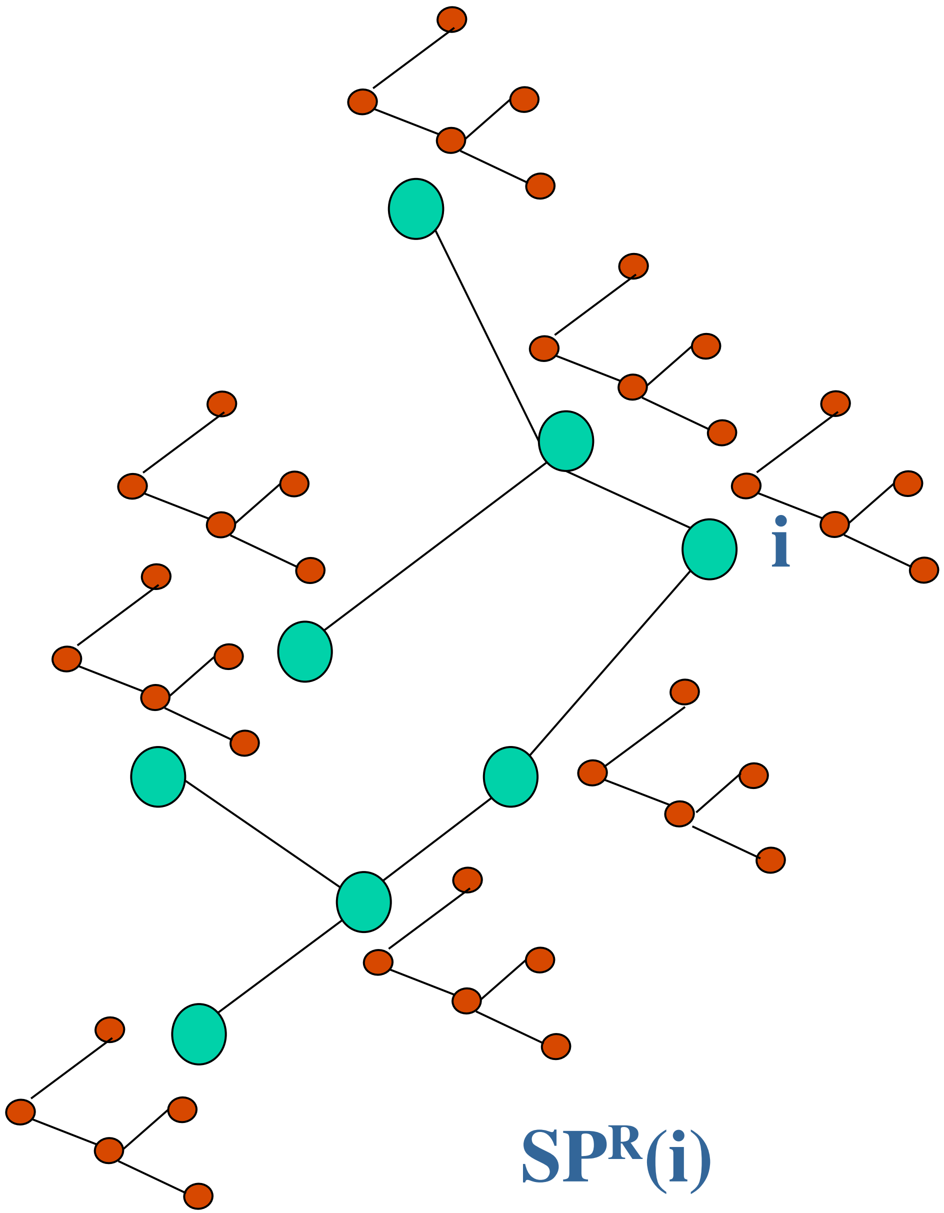


$SP^R(i)$

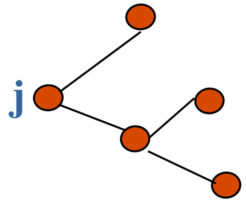


SP(j)

First Idea



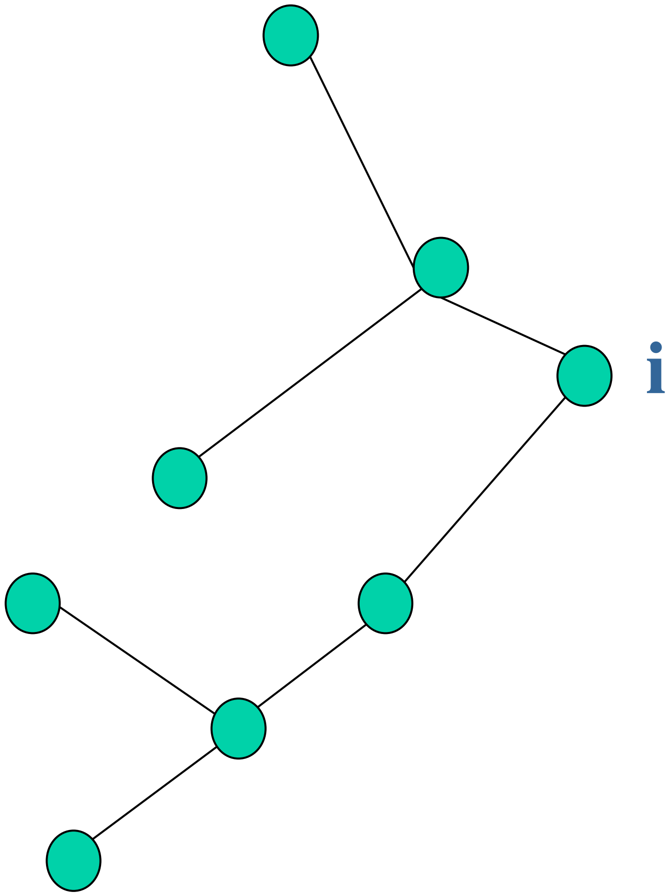
$SP^R(i)$



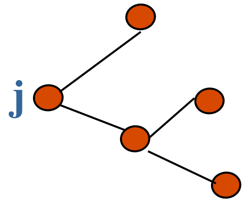
SP(j)

Still $O(n^2)$ update

Second Idea

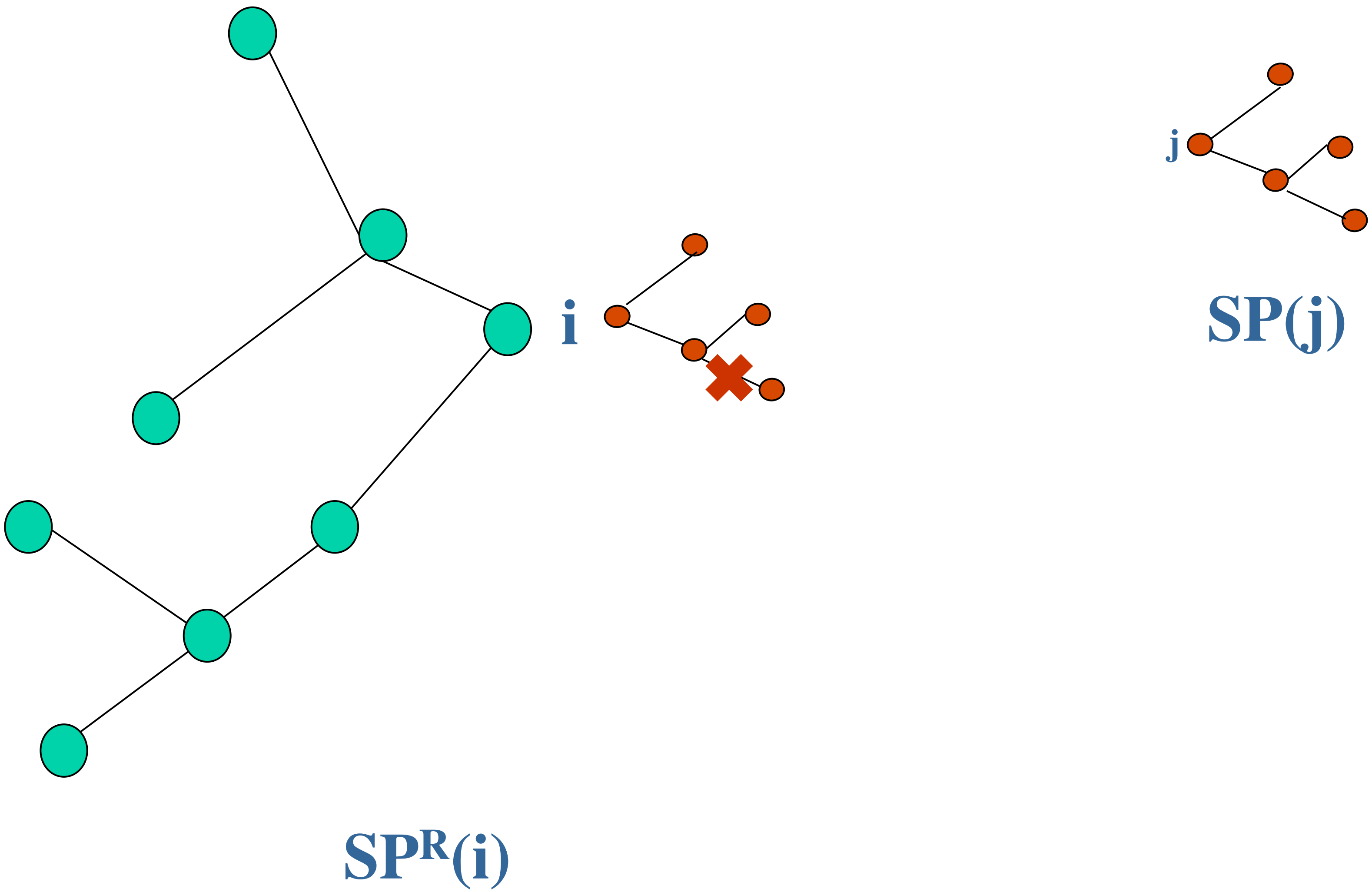


$SP^R(i)$

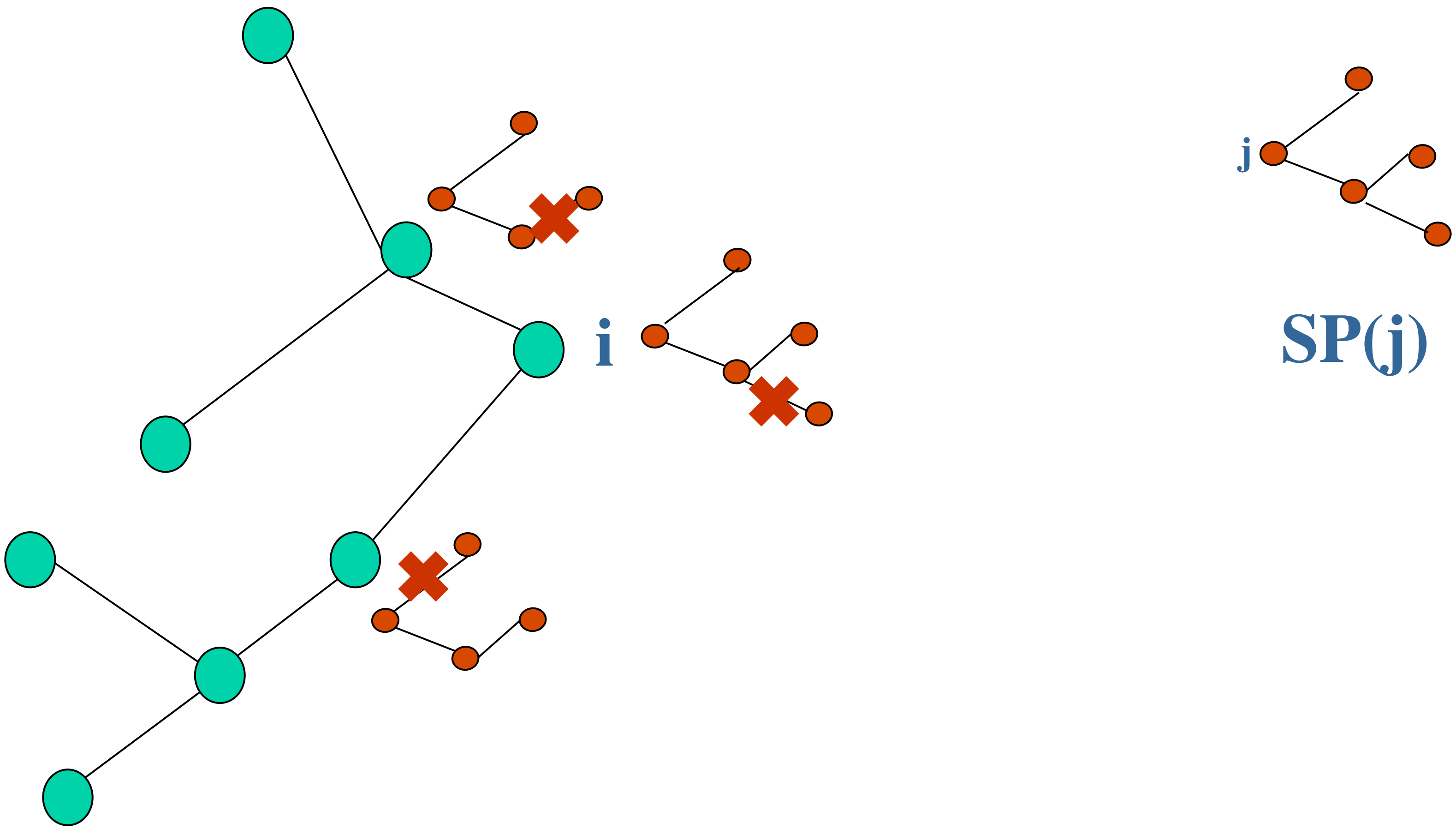


SP(j)

Second Idea

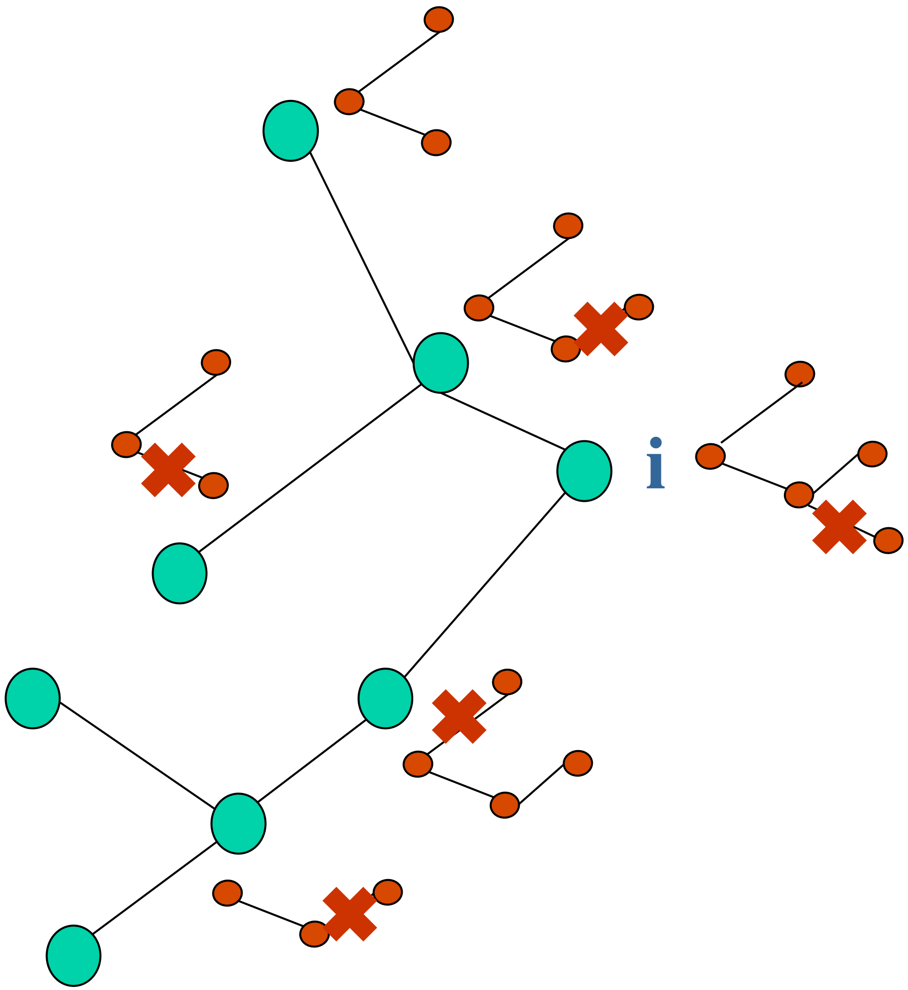


Second Idea

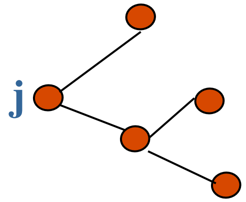


$SP^R(i)$

Second Idea

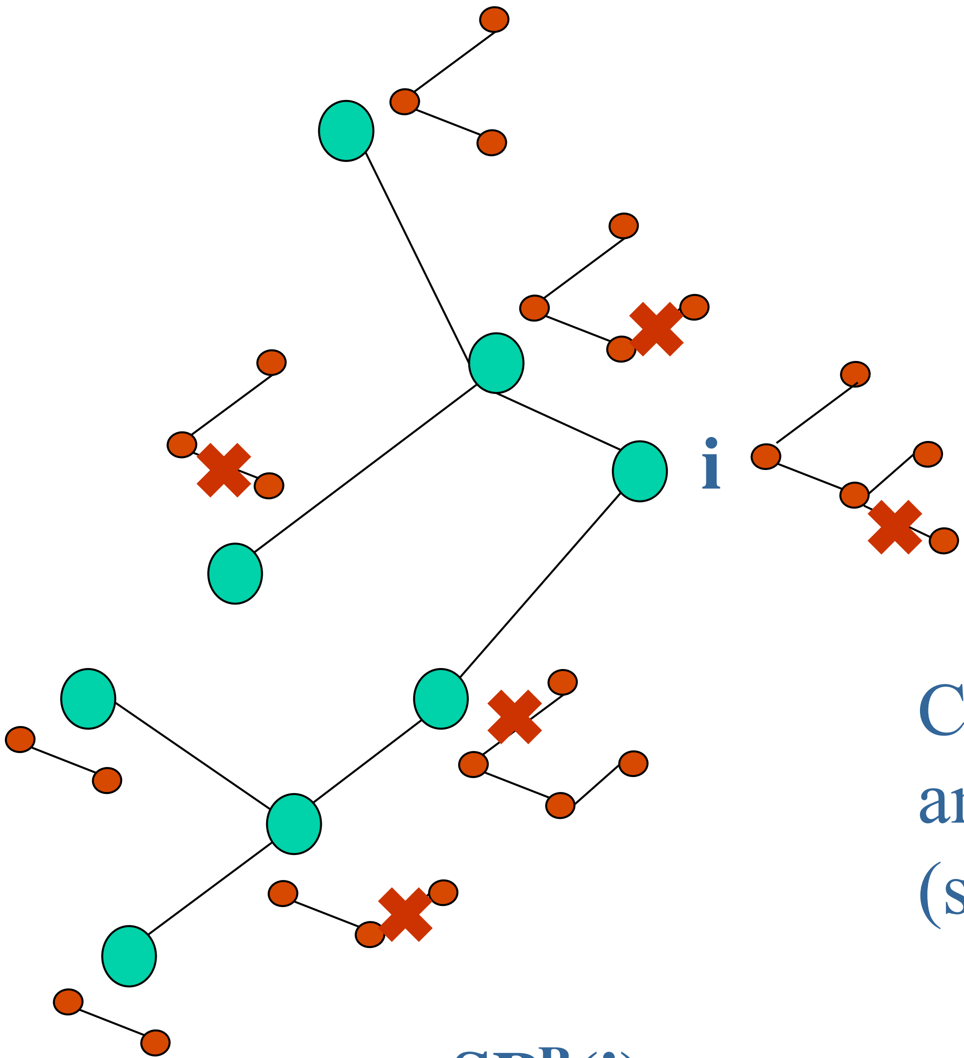


$SP^R(i)$

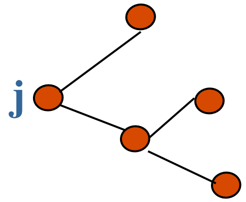


SP(j)

Second Idea



$SP^R(i)$



SP(j)

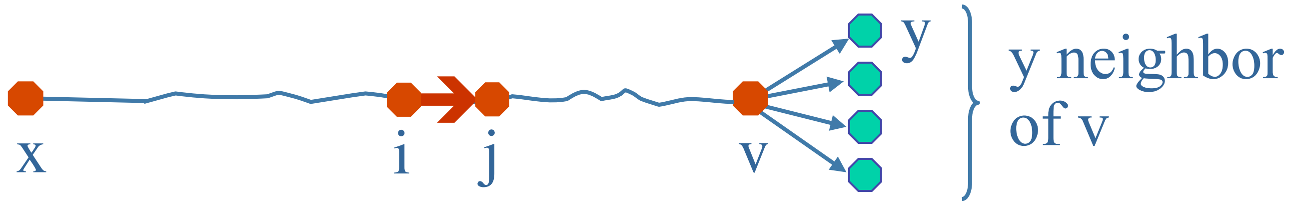
Can show O(n log n)
amortized update
(see paper for details)

What are we doing exactly?

When edge (i,j) is inserted, avoid to look at all $O(n^2)$ pairs (x,y)

1. Look only at pairs (x,y) such that x that reaches i and y reachable from j

2. Inserting edge (i,j) does NOT improve shortest path from x to v

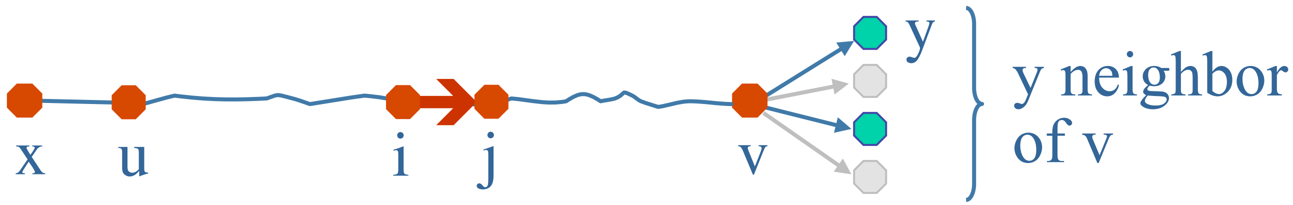


Do we need to look at pair (x,y)?

No, by subpath optimality

What are we doing exactly?

3. Inserting edge (i,j) DOES improve shortest path from x to v



Do we need to look at all pairs (x,y)?

Let u be the vertex immediately after x in the shortest path from x to v

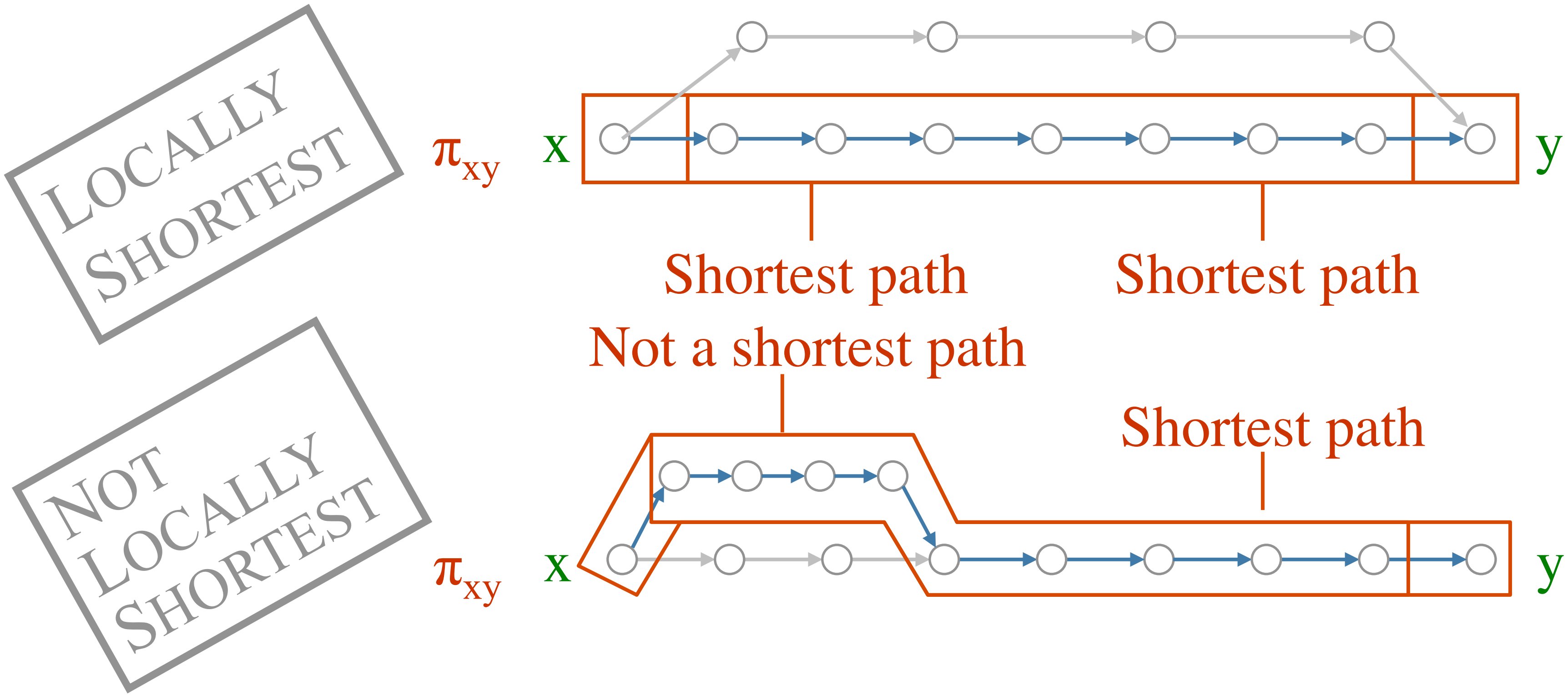
We need to look only at the pairs (x,y) such that shortest path from u to y was improved

Again by subpath optimality: if inserting (i,j) did not improve the shortest path from u to y, then it cannot improve the shortest path from x to y

Locally Shortest Paths

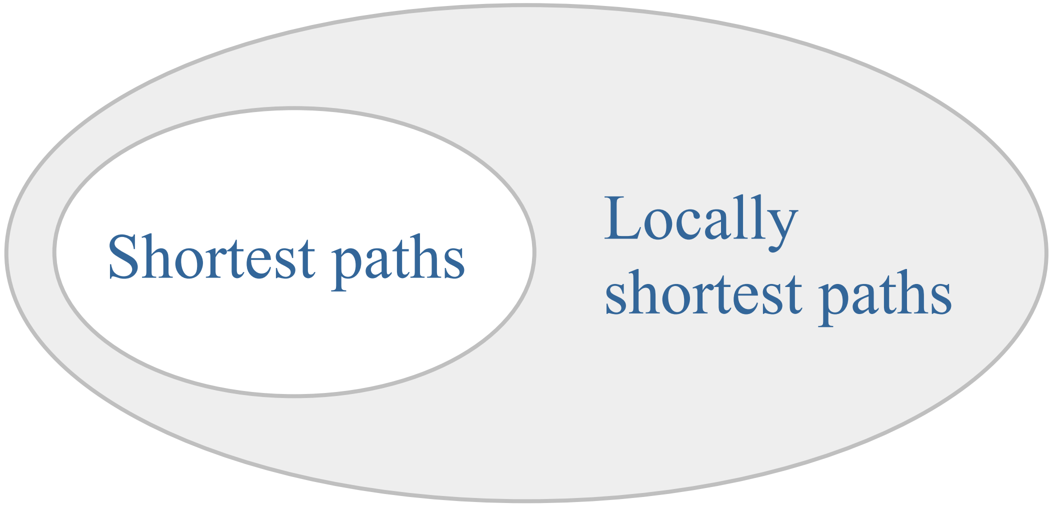
A path is *locally shortest* if all of its proper subpaths are shortest paths

[Demetrescu-I., J.ACM'04]



Locally shortest paths

By optimal-substructure property of shortest paths:



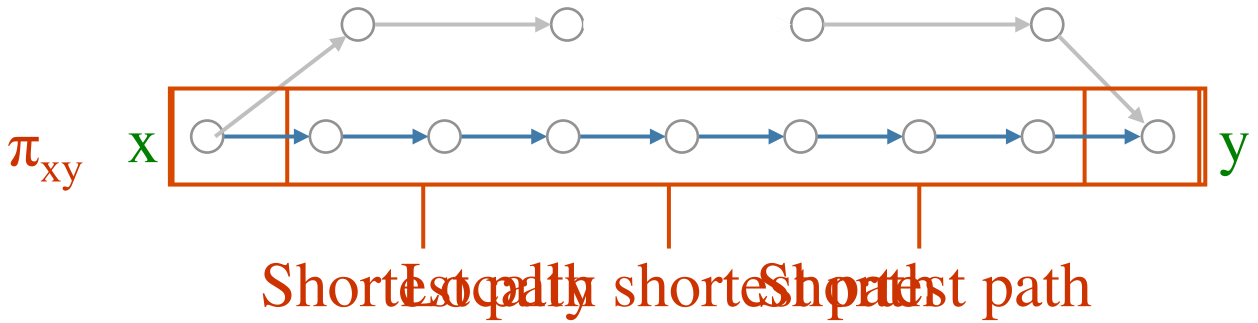
Back to Fully Dynamic APSP

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



Candidate for being shortest path!

Falls short of being a shortest path just because some other path (somewhere else) is better!

Locally Shortest Paths for Dynamic APSP

Idea: Maintain all the **locally shortest paths** of the graph

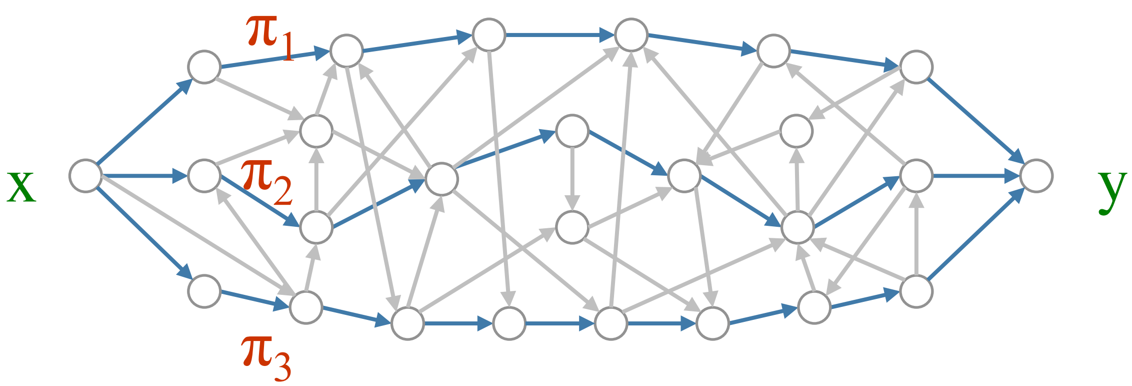
How do locally shortest paths change in a dynamic graph?

We know already what happens for insertions (cost decreases) only. What about deletions (cost increase) only?

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

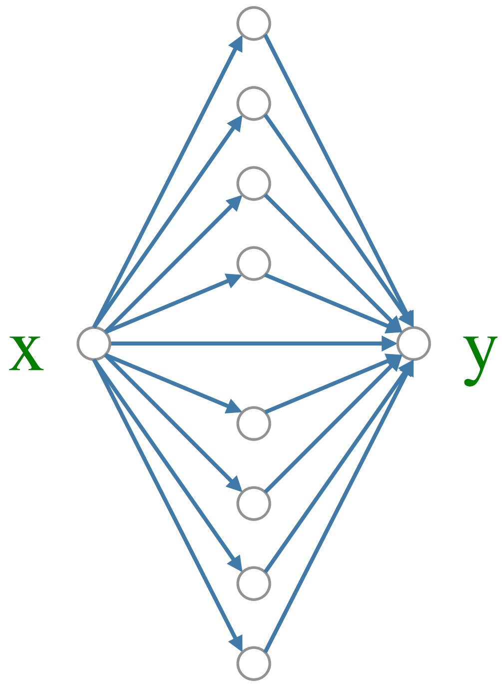
Practice

In practice, tie breaking can be subtle

Properties of locally shortest paths

Property 2

There can be at most (n-1) locally shortest paths connecting x,y

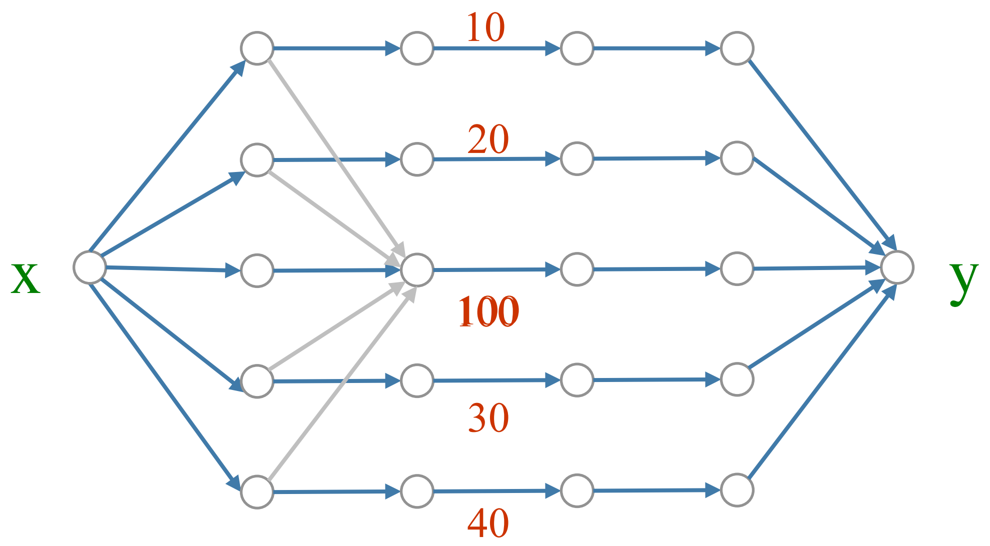


That's a consequence of vertex-disjointess…

Appearing locally shortest paths

Fact 1

At most n^3 (mn) 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 an increase: $\leq n^3$

Locally shortest paths disappearing after an increase: $\leq n^2$

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

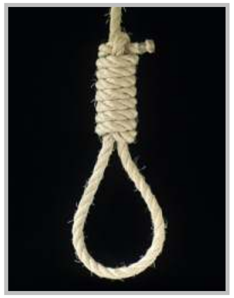
This gives (almost) immediately:

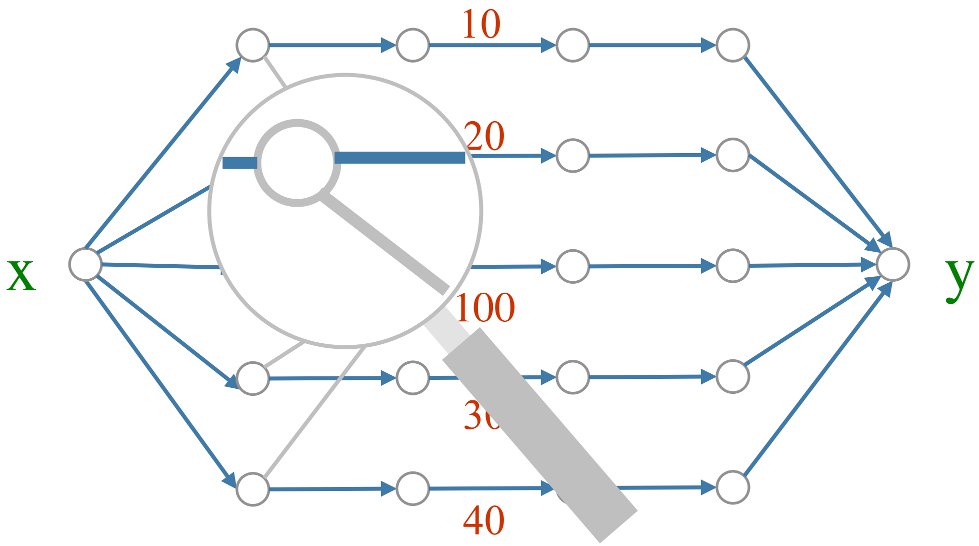
$O(n^2 \log n)$ amortized time per increase

$O(mn)$ space

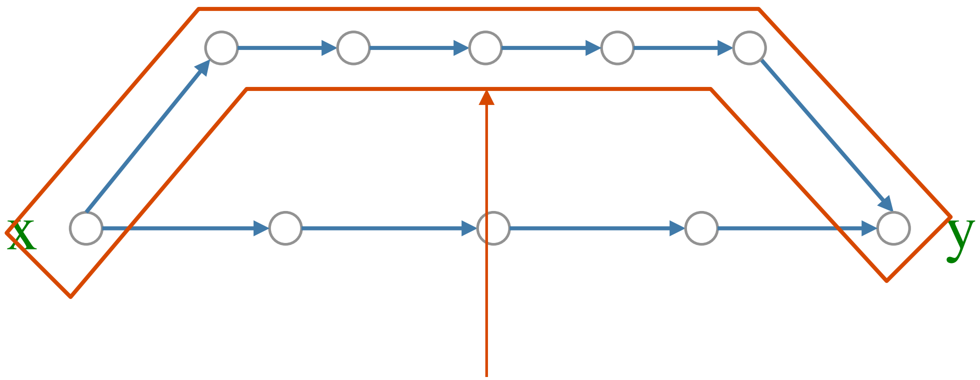
Maintaining locally shortest paths

What about fully dynamic sequences?





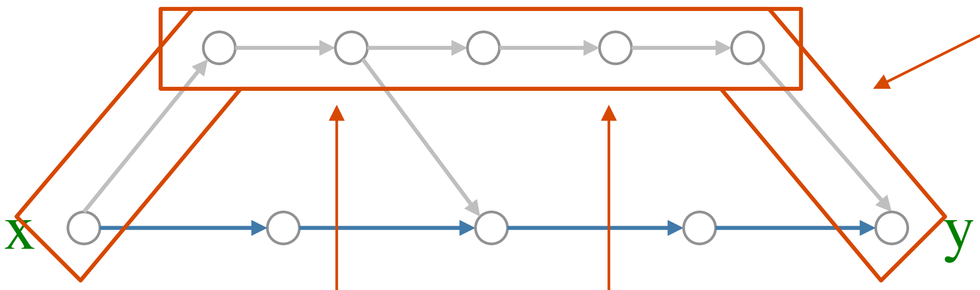
How to pay only once?



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's not dead!

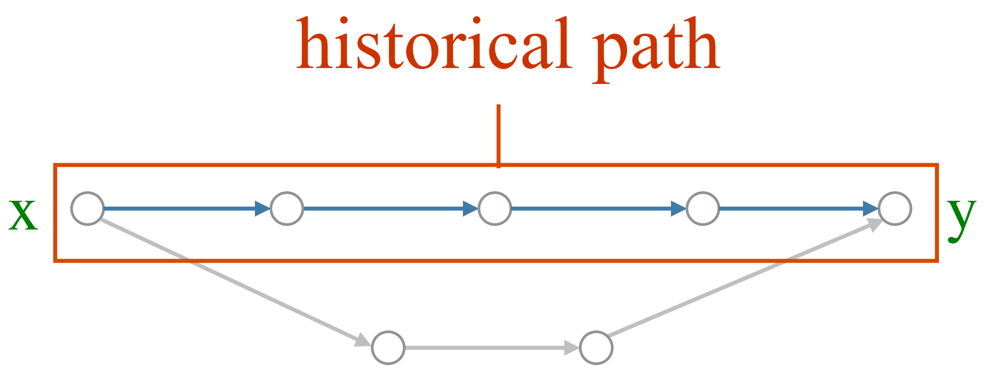
This path is no longer a shortest path after the insertion…

This path remains a shortest path after the insertion

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

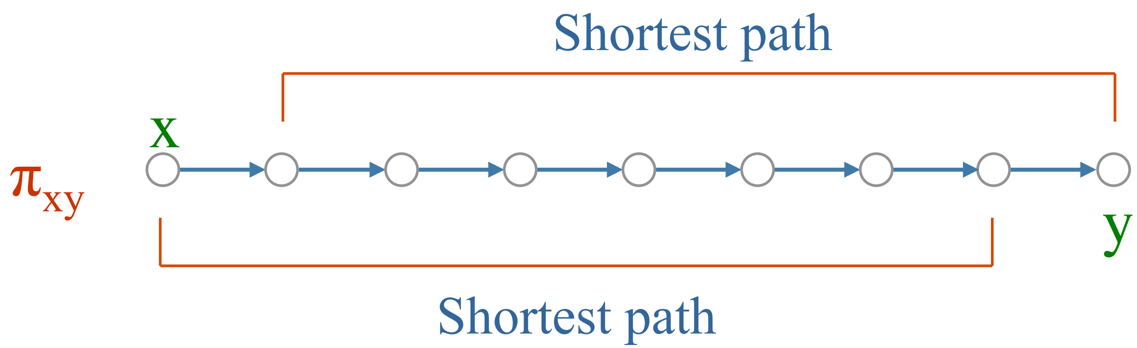
Historical paths

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

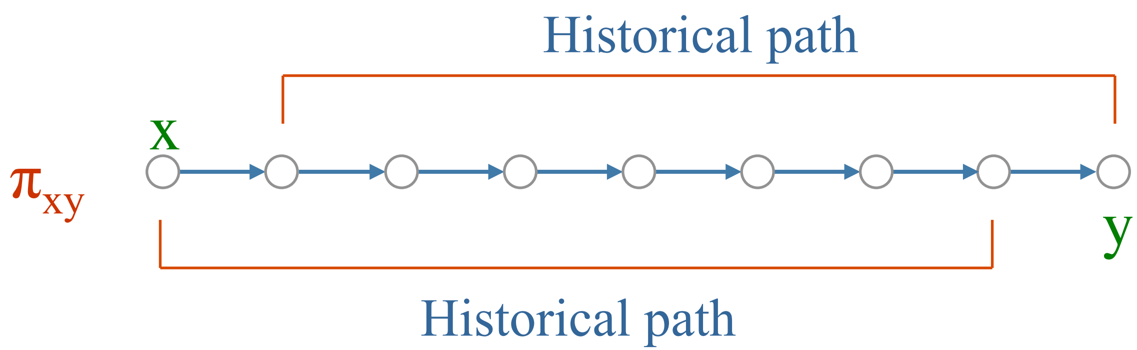


Locally historical paths

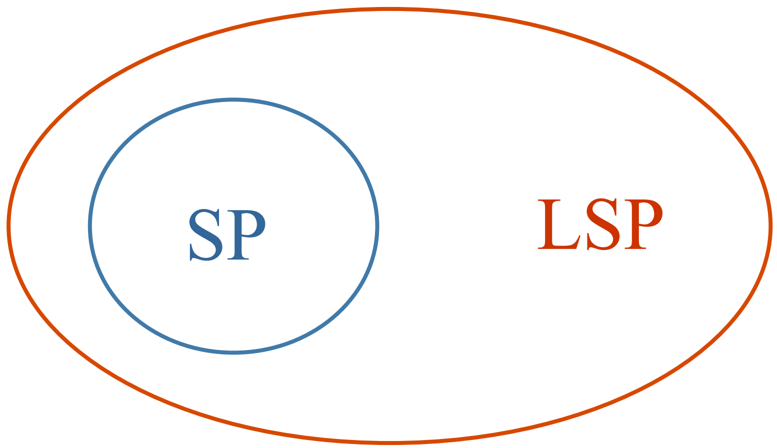
Locally shortest path



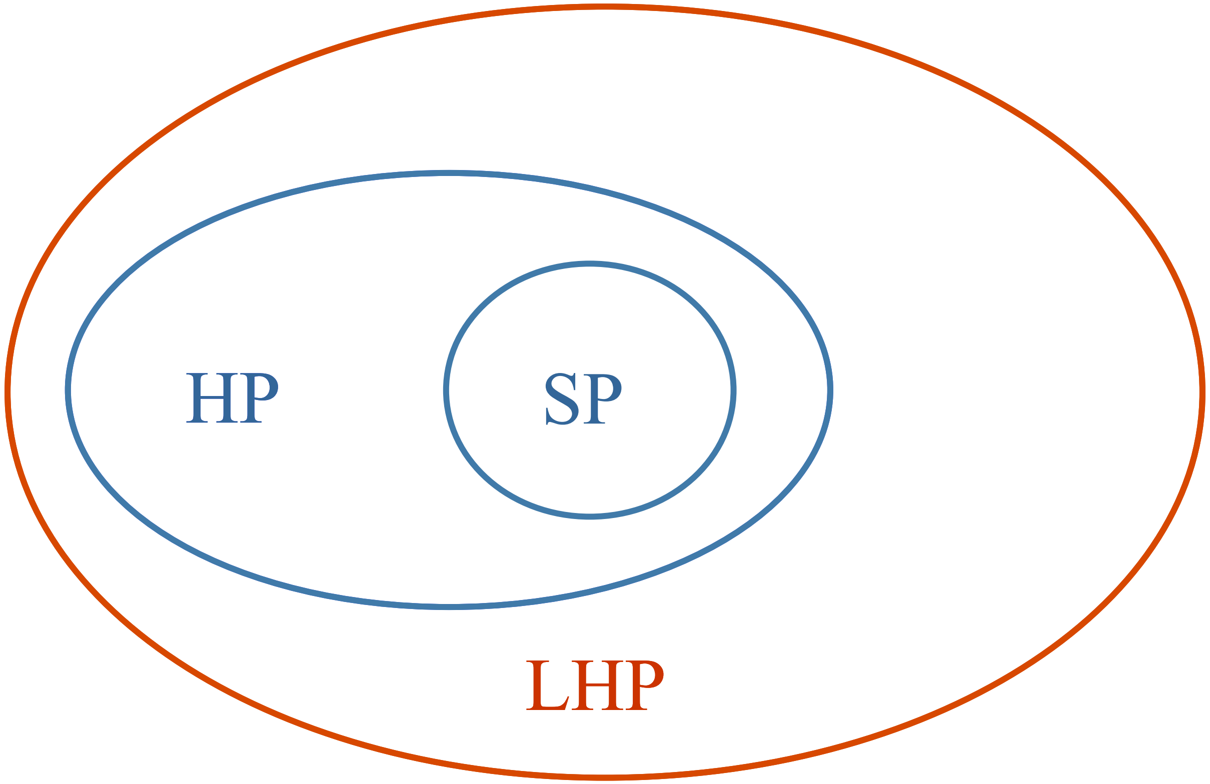
Locally historical path



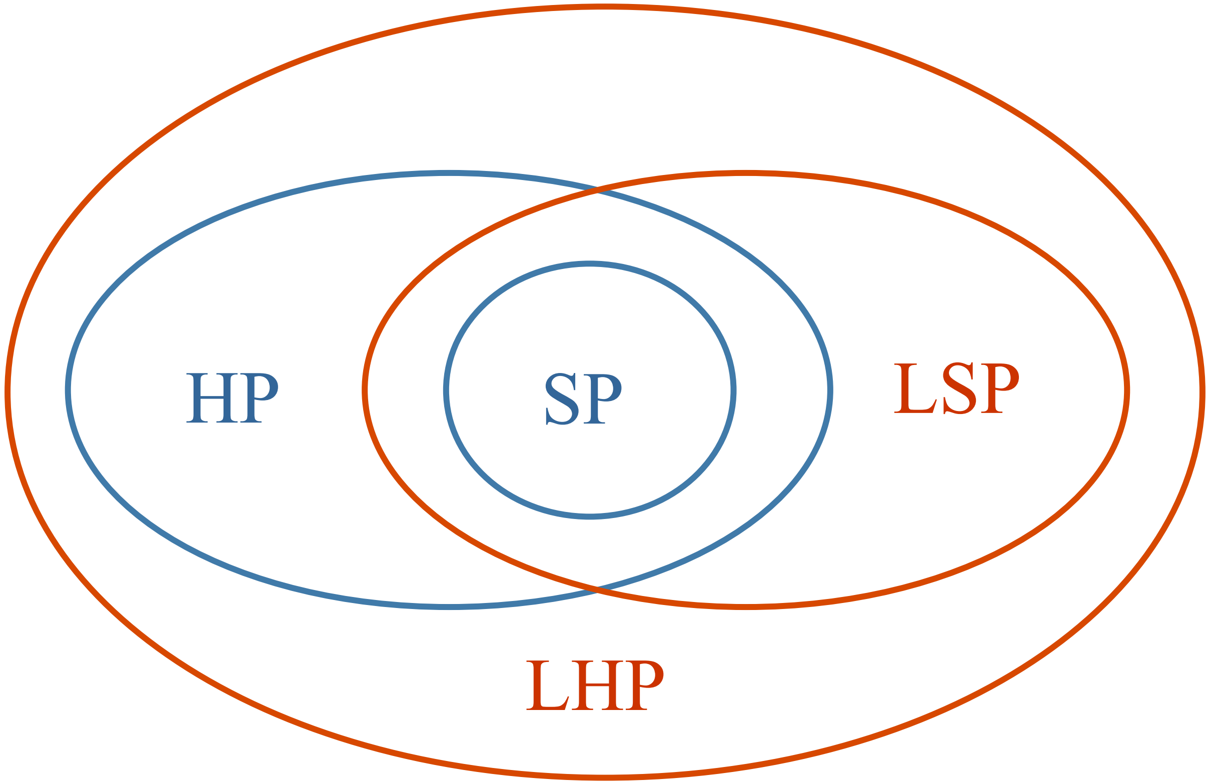
Key idea for partially dynamic



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

Full 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

Experimental study of dynamic NAPSP algorithms:
[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).

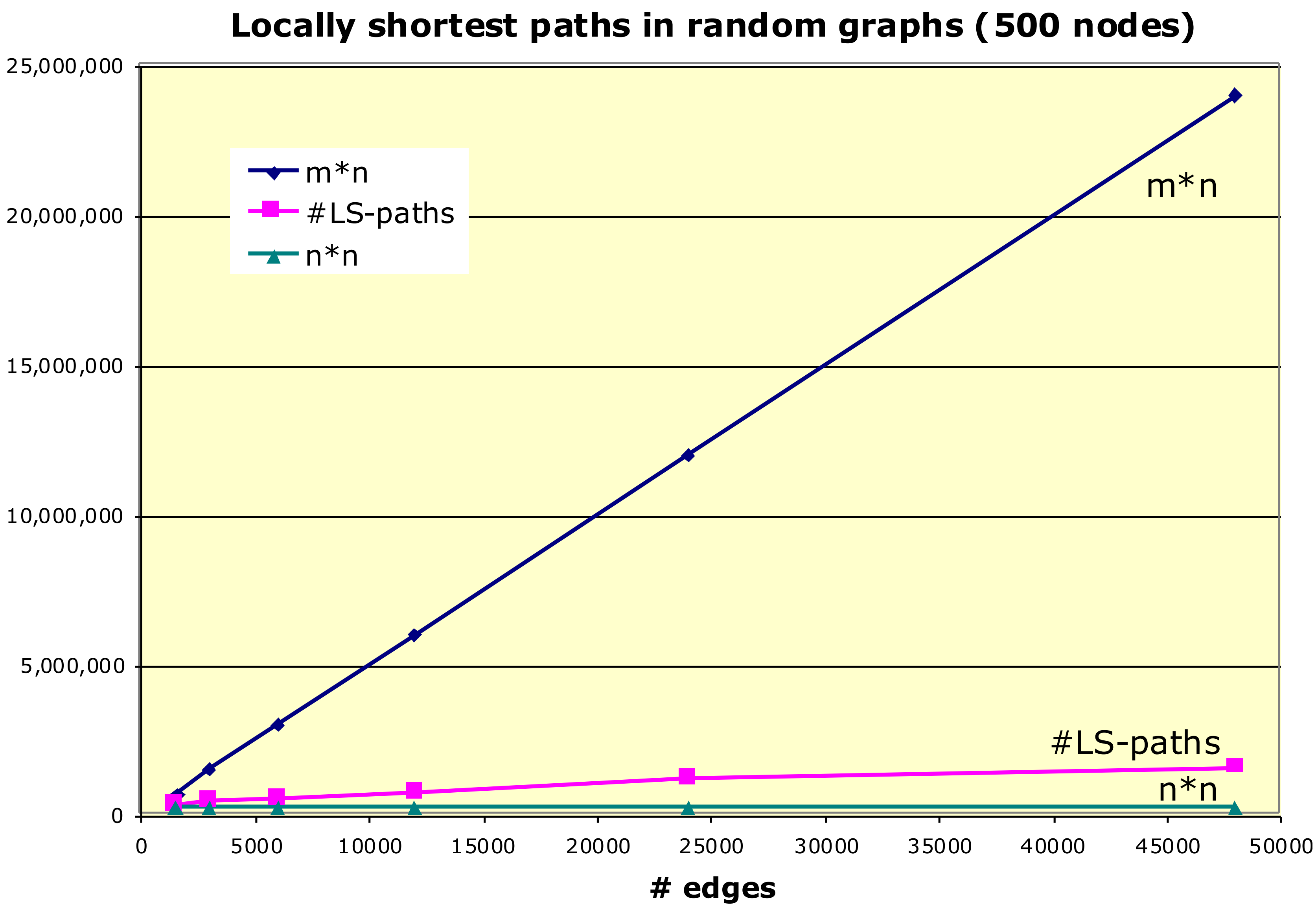
Further Improvements

Using locally historical paths, Thorup [SWAT'04] has shown:

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

$O(mn)$ space

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.

Lower Bounds

Polylog bounds for dynamic connectivity

But dynamic shortest paths seem stubbornly more difficult. Can we prove it?

Conditional lower bounds: basing hardness of dynamic problems on known conjectures (3SUM, All Pairs Shortest Paths, Triangle and Boolean Matrix Multiplication Conjectures and the Strong Exponential Time Hypothesis)

Lower Bounds

[Patrascu 2010]
For dynamic APSP either update or query time must be $\Omega(n^{\varepsilon})$

[Roditty and Zwick 2011]
Any decremental or incremental algorithm for SSSP with preprocessing time $O(n^{3-\varepsilon})$, and update time $O(n^{2-\varepsilon})$ and query time $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$ implies a truly subcubic time algorithm for APSP.

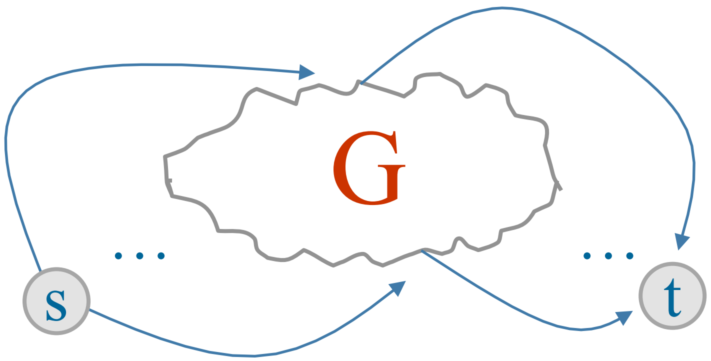
Note: Trivial algorithm recomputes shortest paths from a source in $O(m+n\log n) = O(n^2)$ time after each update!

[Abboud and Vassilevska Williams 2014]
Exclude the possibility of an algorithm that has both $O(n^{2-\varepsilon})$ time updates and $O(n^{2-\varepsilon})$ time queries, even for SSSS.

Dynamic SSSP (SSSS) not easier than APSP?

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 +∞

All-Pairs $\text{query}_G(x,y)$ can be implemented in G' as follows:

$\text{update}_{G'}(s,x,0)$; $\text{update}_{G'}(y,t,0)$; $\text{query}_{G'}(s,t)$;

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

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…

- *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?
  Do we really need transitive closure for this?
  In the static case strong connectivity easier than transitive closure….

References

A. Abboud, V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. FOCS 2014.

G. Ausiello, G.F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. Journal of Algorithms, 12(4):615-638, 1991.

A. Bernstein. Fully dynamic $(2 + \varepsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In FOCS, 693–702, 2009.

A. Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. In STOC, 725–734, 2013.

References

A. Bernstein and L. Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In SODA, 1355–1365, 2011.

C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. J. ACM 51(6):968–992, 2004. See also STOC 2003.

C. Demetrescu and G. F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. ACM Transactions on Algorithms 2(4): 578-601 (2006). See also SODA 2004.

C. Demetrescu and G.F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. Journal of Computer and System Sciences 72(5): 813-837 (2006). See also FOCS 2001.

References

S. Even and H. Gazit. Updating distances in dynamic graphs. Methods of Operations Research, 49:371–387, 1985.

S. Even and Y. Shiloach. An on-line edge-deletion problem. J. ACM, 28:1–4, 1981

D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Semi-dynamic algorithms for maintaining single source shortest paths trees. Algorithmica, 22(3):250–274, 1998.

D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms, 34:351-381, 2000.

References

M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the O(mn) barrier and derandomization. In FOCS, 538–547, 2013.

M. Henzinger, S. Krinninger, and D. Nanongkai. Sublinear-time maintenance of breadth-first spanning tree in partially dynamic networks. In ICALP, 607–619, 2013.

M. Henzinger, S. Krinninger, and D. Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In STOC, 674–683, 2014.

M. Henzinger, S. Krinninger, and D. Nanongkai. A subquadratic-time algorithm for dynamic single-source shortest paths. In SODA, 1053–1072, 2014.

References

V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In STOC, pages 81–91, 1999.

P. Loubal. A network evaluation procedure. Highway Research Record 205, pages 96–109, 1967.

J. Murchland. The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London, UK, 1967.

M. Patrascu. Towards polynomial lower bounds for dynamic problems. Proc. STOC, 603–610, 2010.

References

G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest path problem. Journal of Algorithms, 21:267–305, 1996.

V. Rodionov. The parametric problem of shortest distances. U.S.S.R. Computational Math. and Math. Phys., 8(5):336–343, 1968.

L. Roditty and U. Zwick. On dynamic shortest paths problems. Algorithmica, 61(2):389–401, 2011. See also ESA 2004.

L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. SIAM Journal on Computing, 41(3):670–683, 2012. See also FOCS 2004.

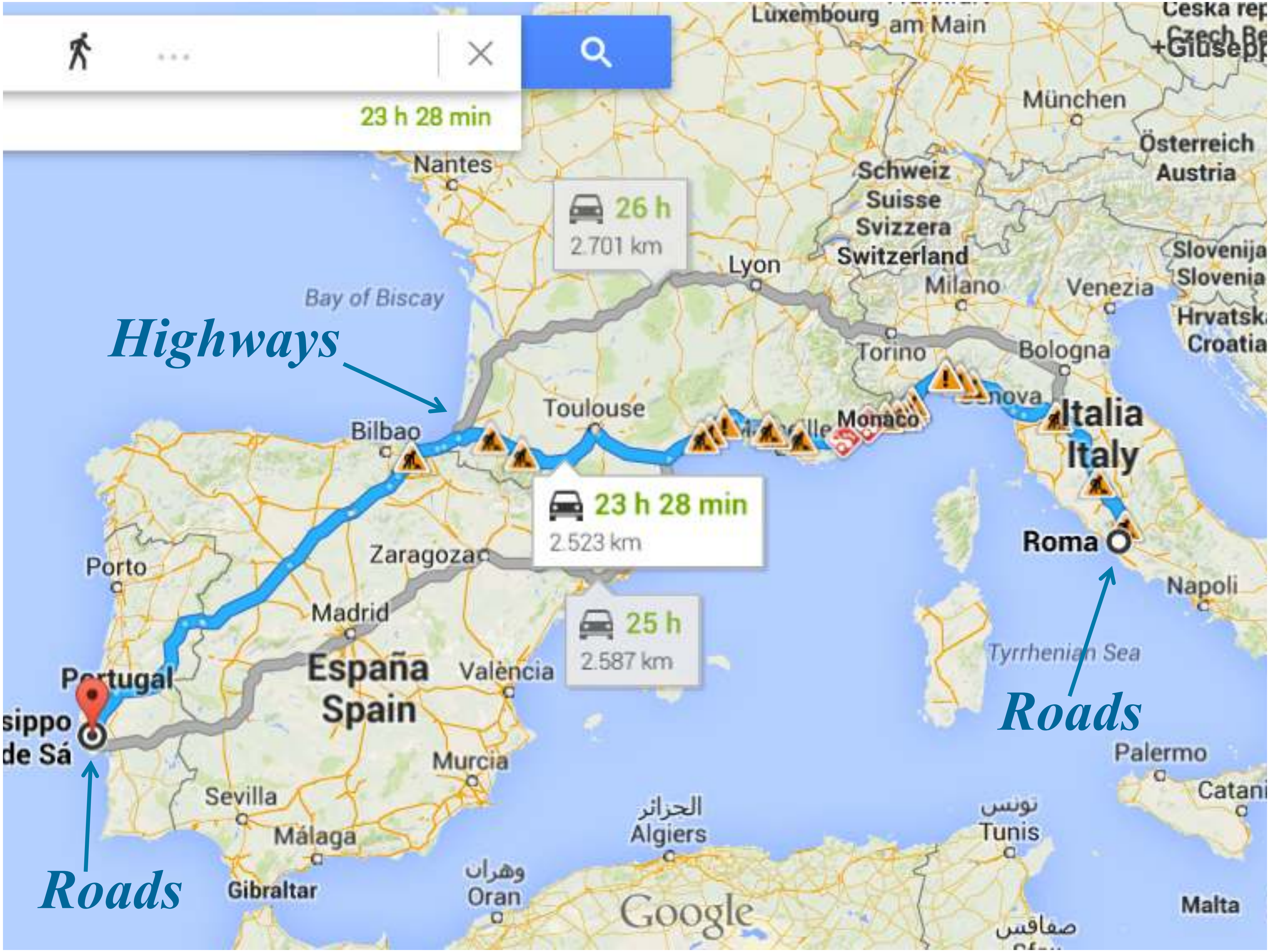
References

H. Rohnert. A dynamization of the all-pairs least cost problem. In Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science, (STACS'85), LNCS 182, 279–286, 1985.

M. Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04), 384–396, 2004.

M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005), 2005.

Long Paths Property



23 h 28 min
26 h
2.701 km
23 h 28 min
2.523 km
25 h
2.587 km
Highways
Roads
Roads
Bay of Biscay
Tyrrhenian Sea
Nantes
Lyon
Toulouse
Bilbao
Porto
Zaragoza
Madrid
España
Spain
Portugal
València
Murcia
Sevilla
Málaga
Gibraltar
Luxembourg
am Main
München
Österreich
Austria
Schweiz
Suisse
Svizzera
Switzerland
Milano
Venezia
Torino
Bologna
Monaco
Italia
Italy
Roma
Napoli
Palermo
Slovenija
Slovenia
Hrvatska
Croatia
الجزائر
Algiers
وهران
Oran
تونس
Tunis
Malta
صفاقس
Google

Are there roads and highways in graphs?

Long Paths Property
[Ullman-Yannakakis '91]

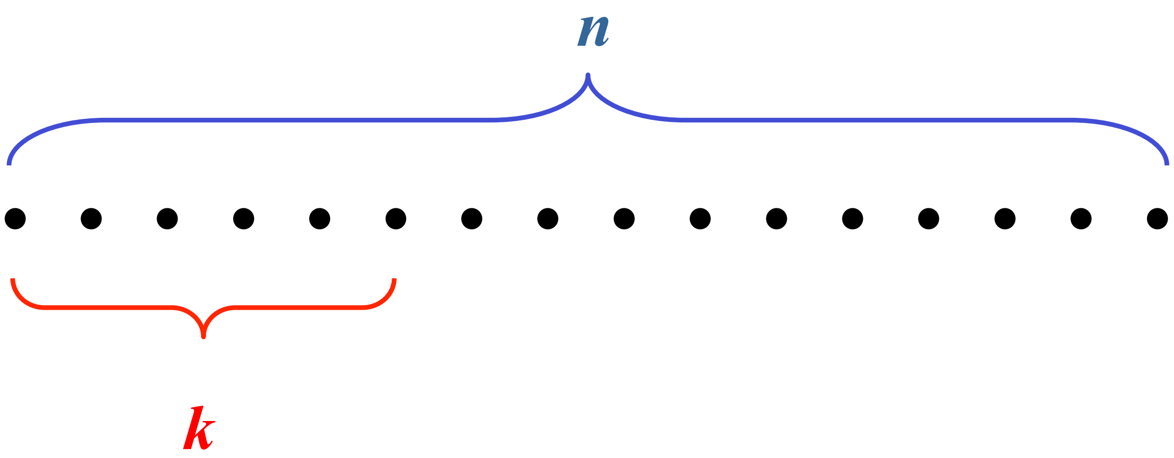
Let P be a path of length at least k.

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



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*** ($P \cap S \neq \emptyset$).

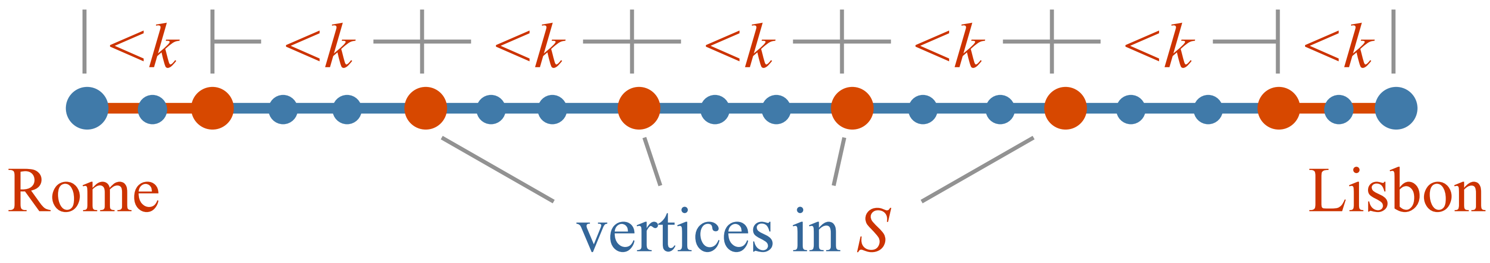
Probability $\geq 1 - (1 / n^{\alpha c})$ for some $\alpha > 0$.

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 $\geq 1 - (1 / n^{\alpha c})$

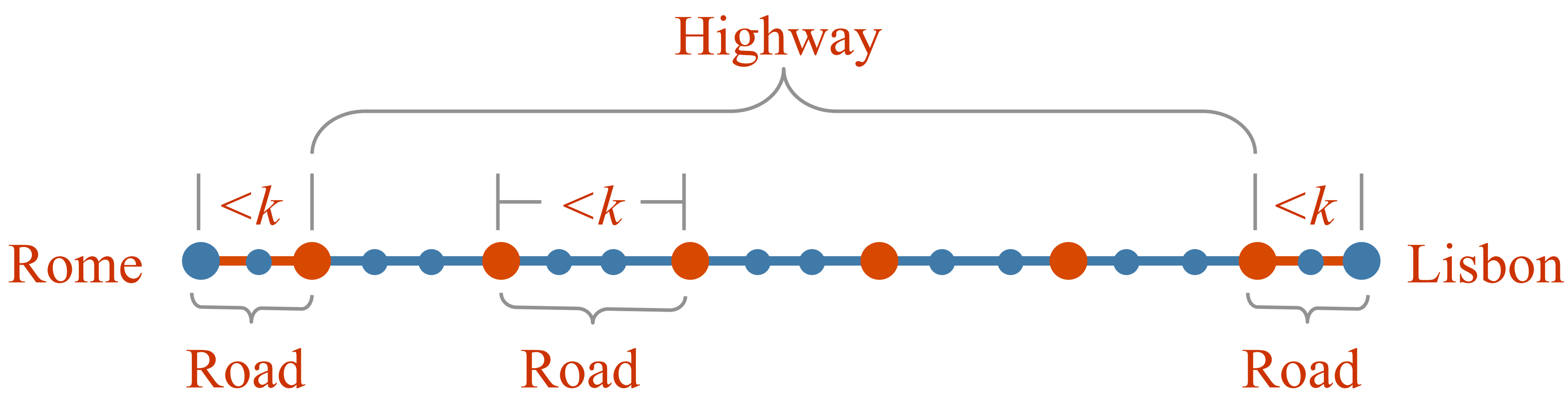


Roads and Highways in Graphs

Highway entry points = vertices in S

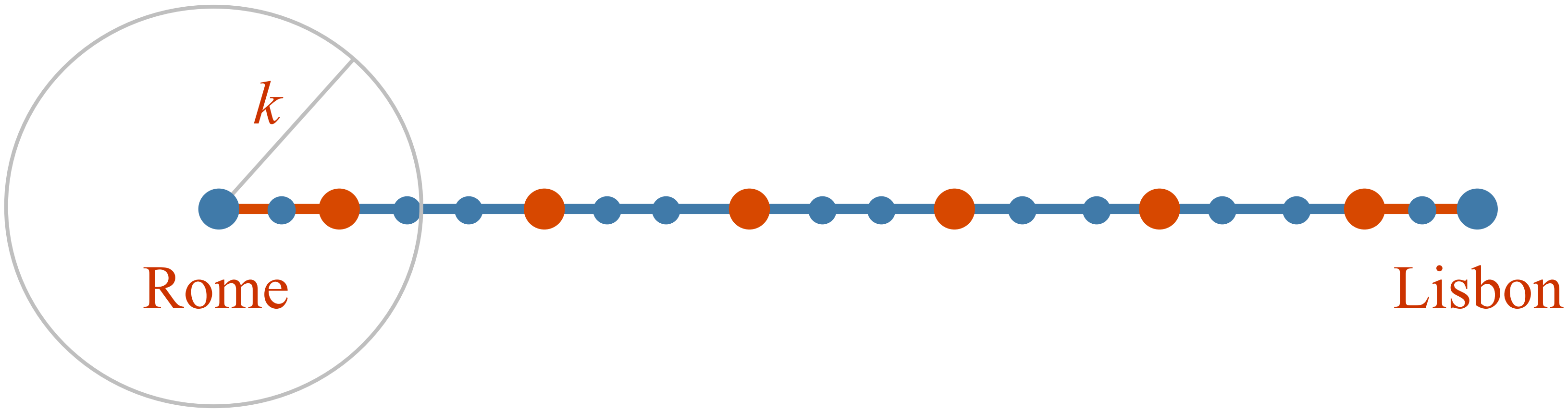
Road = shortest path using at most k edges

Highway = shortest path between two vertices in S



Computing Shortest Paths 1/3

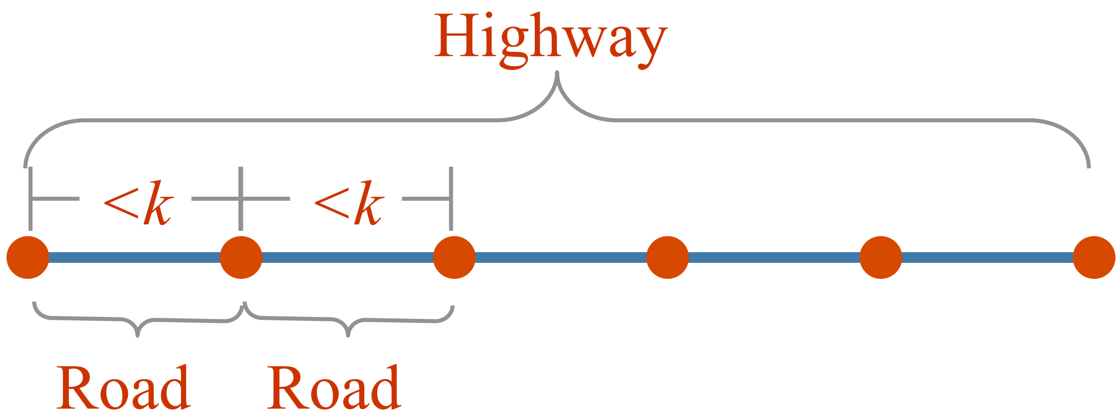
1 Compute roads
(shortest paths using at most *k* edges)



Even & Shiloach BFS trees may become handy…

Computing Shortest Paths 2/3

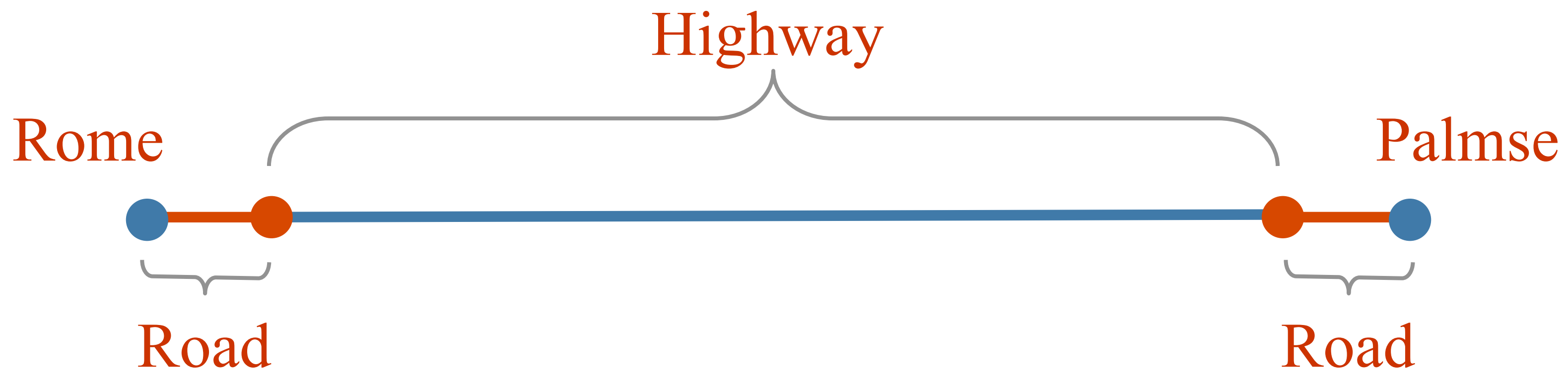
2 Compute highways
(by stitching together roads)



…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) in many papers, i.e., King [FOCS' 99], Demetrescu-I. [JCSS' 06], Roditty-Zwick [FOCS' 04], …