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 have been used for centuries to model relationships...
Graphs...
Graphs...
Dynamic Graphs

Sometimes, life looks a bit more dynamic …
Dynamic Graphs

Graphs subject to **update** operations

Typical updates:

- $\text{Insert}(u, v)$
- $\text{Delete}(u, v)$
- $\text{SetWeight}(u, v, w)$
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**
  \[ \text{MinCut}() : \text{Cut}(x,y) : \]
  Min cut? Are \( x \) and \( y \) on the same side of a min cut of \( G \)?

- **Dynamic Planarity Testing**
  \[ \text{planar}() : \]
  Is \( G \) planar?

- **Dynamic \( k \)-connectivity**
  \[ \text{k-connected}() : \text{k-connected}(x,y) : \]
  Is \( G \) \( k \)-connected? 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:
- Total worst-case time over sequence of ops
  
  \[
  \frac{\text{Total worst-case time}}{\# \text{ operations}}
  \]
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\ldots111 \quad \rightarrow \quad 100000\ldots000

Total cost to count to n: \( n \log n \)
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

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 $\leq 2$
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

<table>
<thead>
<tr>
<th>Theory</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number types: (\mathbb{N}, \mathbb{R})</td>
<td><code>int, float, double</code></td>
</tr>
<tr>
<td>Only asymptotics matter</td>
<td>Seconds do matter</td>
</tr>
<tr>
<td>Abstract algorithm description</td>
<td>Non-trivial implementation decisions, error-prone</td>
</tr>
<tr>
<td>Unbounded memory, unit access cost</td>
<td>Memory hierarchy / bandwidth</td>
</tr>
<tr>
<td>Elementary operations take constant time</td>
<td>Multicore CPUs, Instruction pipelining, …</td>
</tr>
</tbody>
</table>
Bridging the Gap between Theory & Practice

Wish to combine theory and practice...

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

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

...i.e., nothing works and we don't know why.
The Algorithm Engineering cycle

Algorithm design

Theoretical analysis

Algorithm implementation

Experimental analysis

Deeper insights

Bottlenecks, Heuristics

More realistic models

Hints to refine analysis
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)


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.

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:

$\text{Update}(v,w):$ update edges incident to $v$ $\lbrack w(\ ) \rbrack$

$\text{Query}(x,y):$ return distance from $x$ to $y$ (or shortest path from $x$ to $y$)
# Simple-minded Approaches

<table>
<thead>
<tr>
<th>Fast query approach</th>
<th>Fast update approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep the solution up to date.</td>
<td>Do nothing on graph.</td>
</tr>
<tr>
<td>Rebuild it from scratch at each update.</td>
<td>Visit graph to answer queries.</td>
</tr>
</tbody>
</table>
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!)

- 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Graph Type</th>
<th>Weight Domain</th>
<th>Update Complexity</th>
<th>Query Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramalin. &amp; Rep 96</td>
<td>general real</td>
<td>$O(n^3)$</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>King 99</td>
<td>general [0,C]</td>
<td>$O(n^{2.5} (C \log n)^{0.5})$</td>
<td>$O(1)$</td>
<td></td>
</tr>
</tbody>
</table>
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

Query

O(1)  

O(1)  

O(n^2)

Thorup, SWAT’04
Supporting negative weights + improvements on log factors

Demetrescu-I, J.ACM’04
Real-weighted digraphs

King, FOCS’99
Unweighted digraphs

Update

O(n^2)

O(n^{2.5})

O(n^3)

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 \( G \))**

<table>
<thead>
<tr>
<th>update</th>
<th>query</th>
<th>authors</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(n^2 \log n) )</td>
<td>( O(1) )</td>
<td>King, FOCS’ 99</td>
<td></td>
</tr>
<tr>
<td>( O(n^2) )</td>
<td>( O(1) )</td>
<td>King-Sagert, JCSS ‘02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Demetrescu-I., Algorithmica’ 08</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sankowski, FOCS’ 04</td>
<td>DAGs</td>
</tr>
<tr>
<td>( O(n^{1.575}) )</td>
<td>( O(n^{0.575}) )</td>
<td>Demetrescu-I., J.ACM’ 05</td>
<td>DAGs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sankowski, FOCS’ 04</td>
<td></td>
</tr>
<tr>
<td>( O(m n^{1/2}) )</td>
<td>( O(n^{1/2}) )</td>
<td>Roditty, Zwick, SIAM J. Comp.’ 08</td>
<td></td>
</tr>
<tr>
<td>( O(m+n \log n) )</td>
<td>( O(n) )</td>
<td>Roditty, Zwick, FOCS’ 04</td>
<td></td>
</tr>
</tbody>
</table>

**Decremental bounds:** Baswana, Hariharan, Sen, J.Algs.’ 07
## Other Problems

*Dynamic Connectivity (undirected graph $G$)*

<table>
<thead>
<tr>
<th>update</th>
<th>query</th>
<th>authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(\log^3 n)$</td>
<td>$O(\log n / \log \log n)$</td>
<td>Henzinger, King, J. ACM ‘99 (randomized)</td>
</tr>
<tr>
<td>$O(n^{1/3} \log n)$</td>
<td>$O(1)$</td>
<td>Henzinger, King, SIAM J. Comp.’01</td>
</tr>
<tr>
<td>$O(\log^2 n)$</td>
<td>$O(\log n / \log \log n)$</td>
<td>Holm, de Lichtenberg, Thorup, J.ACM’ 01</td>
</tr>
</tbody>
</table>

Lower bounds:

- $\Omega(\log n)$ update: Patrascu & Demaine, SIAM J.Comp’ 06
- $\Omega((\log n / \log \log n)^2)$ update: Patrascu & Tarnita, TCS’ 07
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

Reduced costs

Shortest path trees

Long paths decomposition

Locally-defined path properties

NSSSSP
Ramalingam-Reps ’96

Decremental BFS
Even-Shiloach ’81

NAPSP
King ’99

SSSP
Frigioni et al ’98
Demetrescu ‘01

NAPSP/APSP
Demetrescu-Italiano ’04
Dynamic shortest paths: roadmap

Shortest path trees

NSSSP
Ramalingam-Reps ’96
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** \((u,v,\varepsilon)\): Increase weight \( w(u,v) \) by \( \varepsilon \)
- **Decrease** \((u,v,\varepsilon)\): Decrease weight \( w(u,v) \) by \( \varepsilon \)
- **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($u, v, \varepsilon$)

Shortest paths tree before the update
Increase\((u,v,\varepsilon)\)
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

\[ C[u,v] \leftarrow C[u,v] + C[u,x] \cdot C[y,v] \quad \text{O}(n^2) \]

\[ C[u,v] \leftarrow C[u,v] - C[u,x] \cdot C[y,v] \quad \text{O}(n^2) \]

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)
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
Arithmetic mod primes

Reduce wordsize to $2c \log n$: pick random prime $p$ between $n^c$ and $n^{c+1}$ and perform arithmetic mod $p$ in $O(1)$ 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**$(J,I)$: $M \leftarrow M + J \cdot I$ \hspace{1cm} $O(n^2)$

**Query**$(i,j)$: return $M[i,j]$ \hspace{1cm} $O(1)$
Maintaining dynamic integer matrices

How can we trade off operations?

Lazy approach: buffer at most $n^\varepsilon$ updates
Global rebuilding every $n^\varepsilon$ updates
Rebuilding done via matrix multiplication
Maintaining dynamic integer matrices

\[ m + j_1 \cdot i_1 + j_2 \cdot i_2 + j_3 \cdot i_3 \]
Maintaining dynamic integer matrices

Global rebuilding every $n^\epsilon$ updates

$m + j_1 \cdot i_1 + j_2 \cdot i_2 + j_3 \cdot i_3$

$O(n^{\omega(1, \epsilon, 1)})$
\[
\forall u,v: \quad C[u,v] \leftarrow C[u,v] + C[u,x] \cdot C[y,v]
\]
Query Time

\[ m + j_1 \cdot i_1 + j_2 \cdot i_2 + j_3 \cdot i_3 \]

Total Query time \( O(n^\varepsilon) \)
Update Time

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

$$\forall \ u,v:\ C[u,v] \leftarrow C[u,v] + C[u,x] \cdot C[y,v]$$

Carried out via $O(n)$ queries

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

2. Global rebuild every $n^\varepsilon$ updates

Carried out via (rectangular) matrix multipl.

Amortized time: $\mathcal{O}(n^{\omega(1, \varepsilon, 1)}/n^{\varepsilon})$
Dynamic Transitive Closure [Demetrescu-I., J.ACM05]

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

for any $0 < \epsilon < 1$

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

Best bound for rectangular matrix multiplication
[Huang/Pan98]

$\epsilon < 0.575$

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

Reduced costs

Shortest path trees

NSSSP

Ramalingam-Reps ’96

Decremental BFS

Even-Shiloach ’81

SSSP

Frigioni et al ’98

Demetrescu ’01
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 $\Omega(m)$ deletions)
Can we do better than $O(mn)$?

Roditty and Zwick [2011] have shown two reductions:

- Boolean matrix multiplication
- Weighted (static) undirected APSP

$\rightarrow$

- (off-line) decremental undirected BFS
- (off-line) decremental undirected SSSP
Matrix mult. \rightarrow Decremental BFS

A and B Boolean matrices

Wish to compute \( C = A \cdot B \)

\( C[x,y] = 1 \) 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
First row: $C[1,x] = 1$ iff dist($s,x$) = 3

Second row: $C[2,x] = 1$ iff dist($s,x$) = 4

Third row: $C[3,x] = 1$ iff dist($s,x$) = 5

…

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

$n$ deletions and $n^2$ queries
More details in

*Decremental BFS:*
[Even-Shiloach’81]
S. Even and Y. Shiloach,
An On-line Edge Deletion Problem,

*Reductions to decremental BFS:*
[Roditty-Zwick’11]
Liam Roditty, Uri Zwick,
On dynamic shortest paths problems
Main Ingredients

- Long paths property
- Output bounded
- Decremental BFS
- Path decompositions
- Locally-defined path properties
- Counting
- Algebraic techniques
Dynamic shortest paths: roadmap

Reduced costs

Shortest path trees

Long paths decomposition

NSSSP
Ramalingam-Reps ’96

Decremental BFS
Even-Shiloach ’81

SSSP
Frigioni et al ’98
Demetrescu ’01

NAPSP
King ’99
Make decremental fully dynamic

For each vertex $v$:

- $\text{IN}(v)$ maintained as a decremental BFS tree
- $\text{OUT}(v)$ maintained as a decremental BFS tree

Building block:
- pair of IN/OUT trees
- keeps track of all paths of length $\leq d$ passing through $v$
Make decremental fully dynamic

Rebuild IN(v), OUT(v)

Rebuild IN(v), OUT(v)

deletions only for IN(v), OUT(v)

sequence of ops

Total cost for rebuilding IN, OUT trees + deleting edges in between: $O(md)$
This is charged to $\text{insert}(v)$
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 \( \leq 2 \) 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

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

$X = \text{adjacency matrix} + I \quad X^{n-1} = \text{transitive closure}$

paths with $\leq 2$ edges
paths with $\leq 4$ edges
paths with $\leq 8$ edges
Dynamic Transitive Closure  [King, FOCS’ 99]

\[ G_0 = G \]

\[ G_1 \]

\[ G_2 \]

\[ G_3 \]

\[ \ldots \]

\[ G_{\log n} \]

Reachability queries in \( G_{[\log n]} \)

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

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

If there is a path from x to y in G of length ≤ k, then there is an edge (x,y) in \( G_{[\log k]} \)

Invariant:
\( G_{[\log k]} (x,y) \in G_{[\log k]} \) if and only if x ∈ IN(v) and y ∈ OUT(v) for some v in \( G_{[\log k-1]} \)

IN/OUT trees in \( G_{[\log k]} \) for each vertex
Dynamic Transitive Closure  [King, FOCS’ 99]

$G_0 = G$

Deletion of any subset of the edges of $G$

$G_1$

Edge deletions (amortized against the creation of trees)
Dynamic Transitive Closure  [King, FOCS’ 99]

\[ G_0 = G \]

**Insertion of edges incident to a vertex** \( v \)

- \( G_1 \)
- \( G_2 \)
- \( G_3 \)
- \( \ldots \)
- \( G_{\log n} \)

**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 \( v \)

\[ G_1 \]

\[ G_2 \]

\[ G_3 \]

\[ \ldots \]

\[ G_{\log n} \]

Correctness?

Path \( a, b, c \) in \( G_{i-1} \)

\[ \Rightarrow (a, c) \text{ in } G_i ? \]
Main Ingredients

- Long paths property
- Output bounded
- Decremental BFS
- Path decompositions
- Locally-defined path properties
- Counting
- Algebraic techniques
A real-life problem

Highway

Roads
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 $\left(\frac{c \cdot n \ln n}{k}\right)$.

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

Probability $\geq 1 - \left(\frac{1}{n^c}\right)$ (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

$$(1-p)^k = \left(1 - \frac{c \ln n}{k}\right)^k < n^{-c}$$
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 $\geq 1 - \left(1 / n^{\alpha c}\right)$ for some $\alpha > 0$. 
Exploit Long Paths Property

Randomly pick a set $S$ of vertices in the graph

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

Then on any path in the graph

every $k$ vertices there is a vertex in $S$, with probability $\geq 1 - \left( \frac{1}{n^\alpha} \right)$
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)

Highway

<k <k

Road Road

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

Diagram:

```
|<k|<k|<k|<k|<k|<k|<k|<k|
```

Rome --- Brno --- Palmse
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 $\quad$ $O(1)$ query time $\quad$ $\tilde{O}(n^{2.5}\sqrt{C})$ space
More details in

*Long paths decomposition:*
[Ullman-Yannakakis’91]
J.D. Ullman and M. Yannakakis.
High-probability parallel transitive-closure algorithms.

*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

Reduced costs

NSSSP
Ramalingam-Reps ’96

Decremental BFS
Even-Shiloach ’81

NAPSP
King ’99

SSSP
Frigioni et al ’98
Demetrescu ’01

Shortest path trees

NSSSP
Ramalingam-Reps ’96

SSSP
Frigioni et al ’98
Demetrescu ’01

Long paths decomposition

Decremental BFS
Even-Shiloach ’81

NAPSP
King ’99

Locally-defined path properties

NAPSP/APSP
Demetrescu-Italiano ’04
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:
   \[
   \begin{array}{c}
   x \\
   \end{array}
   \begin{array}{c}
   \rightarrow
   \\
   \end{array}
   \begin{array}{c}
   y
   \end{array}
   \]
   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:
   \[
   \begin{array}{c}
   x \\
   \end{array}
   \begin{array}{c}
   \rightarrow
   \\
   \end{array}
   \begin{array}{c}
   a
   \\
   \rightarrow
   \\
   \end{array}
   \begin{array}{c}
   y
   \end{array}
   \]
   2. Scan only \(y'\) for which \((a,y')\) shortest (subpath opt.)
   3. Possibly insert \((x,y')\) into heap or decrease priority
A path is *locally shortest* if all of its *proper* subpaths are shortest paths.
Locally shortest paths

By optimal-substructure property of shortest paths:

Shortest paths

Locally shortest paths
How much do we gain?

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

\[ O(\ #LS\text{-paths} + n^2 \log n) \text{ time} \quad O(n^2) \text{ space} \]

Q.: How many locally shortest paths?

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

Q.: How much can we gain in practice?
How many LSPs in a graph?

Locally shortest paths in random graphs (500 nodes)

- $m \times n$
- $n \times n$
- $\#LS\text{-paths}$

Graph showing the relationship between the number of edges and the number of locally shortest paths for graphs with 500 nodes.
Real-world Graphs?
US road networks

Locally shortest paths in US road networks

average degree
#LS-paths per pair
Can we exploit this in practice?

Experiment for increasing number of edges (rnd, 500 nodes)

- Dijkstra's algorithm
- Algorithm based on locally shortest paths

Number of edges
What we have seen so far:

- Algorithm design
- Theoretical analysis
- Algorithm implementation
- Experimental analysis

Deeper insights

Bottlenecks, Heuristics

More realistic models
Hints to refine analysis
Return Trip to Theory:

- Algorithm design
- Theoretical analysis
- Algorithm implementation
- Experimental analysis

Deeper insights:
- Bottlenecks, Heuristics

More realistic models:
- Hints to refine analysis
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 $\pi_{xy}$ 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-disjointness…
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.

$\pi$ stops being locally shortest after increase of $e$

subpath of $\pi$ (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

...but if we removed the same edge it would be a shortest path again!
A path is **historical** if it was shortest at some time since it was last updated.
Locally historical paths

Locally shortest path

$\pi_{xy}$

Locally historical path

$\pi_{xy}$

Shortest path

Historical path
Key idea for partially dynamic

SP

LSP
Key idea for fully dynamic
Putting things into perspective…
**The fully dynamic update algorithm**

<table>
<thead>
<tr>
<th>Idea:</th>
<th>Maintain all the <strong>locally historical paths</strong> of the graph</th>
</tr>
</thead>
</table>

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

- \( \text{O}(n^2 \log^3 n) \) amortized time per update
- \( \text{O}(mn \log n) \) 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.
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:

- Algorithm design
- Theoretical analysis
- Algorithm implementation
- Experimental analysis

Deeper insights

Bottlenecks, Heuristics

More realistic models
Hints to refine analysis
How many LSPs in a graph?

Locally shortest paths in random graphs (500 nodes)

- $m \times n$
- $#LS$-paths
- $n \times n$
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

- Reduced costs
  - NSSSP
    - Ramalingam-Reps ’96
  - SSSP
    - Frigioni et al ’98
    - Demetrescu ’01
- Shortest path trees
  - Decremental BFS
    - Even-Shiloach ’81
- Long paths decomposition
- Locally-defined path properties
  - NAPSP/APSP
    - Demetrescu-Italiano ’04

Experimental comparison
A comparative experimental analysis

(Dynamic) NAPSP algorithms under investigation

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
<th>Update</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra 59 (FT 87)</td>
<td></td>
<td>$O(mn + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Demetrescu/I. 06</td>
<td></td>
<td>$O(#LSP + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Ramaling./Reps 96 (SIMPLE)</td>
<td></td>
<td>$O(mn + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>King 99</td>
<td>$[0,C]$</td>
<td>$O(n^{2.5} (C \log n)^{0.5})$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Demetrescu/I. 04</td>
<td></td>
<td>$\tilde{O}(n^2)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
# 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 \hspace{1cm} \(O(1)\) query time \hspace{1cm} \(\tilde{O}(n^{2.5}\sqrt{C})\) 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

<table>
<thead>
<tr>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tilde{O}(n^2) ) time per update</td>
</tr>
<tr>
<td>( O(1) ) time per query</td>
</tr>
<tr>
<td>( \tilde{O}(mn) ) space</td>
</tr>
</tbody>
</table>

Approach:

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

[Diagram showing the relationship between shortest paths, locally shortest paths, and locally historical paths]
Are LHPs useful in practice? (update time)

Experiment for increasing # of edges (rnd, 500 nodes)
What about real graphs? (update time)

Experiments on US road networks

D-RRL
D-LHP

US states
Zoom out to static (update time)

Experiments on US road networks

US states

S-DIJ
D-RRL
D-LHP
Big issue: the space wall

Experiments on US road networks

Space (MB) vs. # nodes

- D-RRL
- D-LHP
Worst-case instances

Experiments on bottleneck graphs (500 nodes, IBM Power4)

# edges

D-RRL
S-DIJ
D-LHP
S-LSP
Different HW platforms

- **IBM Power 4** (32MB L3 cache)
- **Sun UltraSPARC IIi** (2MB L2 cache)
- **AMD Athlon** (256KB L2 cache)

Relative time performance D-RLL/D-LHP

Number of edges (x 100)

- D-LHP slower than D-RRL
- D-LHP faster than D-RRL
Cache effects

**Simulated cache miss ratio D-RRL/D-LHP**

**Performance ratio D-RRL/D-LHP on real architectures**

- **UltraSPARC IIi**
- **Xeon**
- **Athlon**
- **Power 4**

**Colorado road network**

- **Cache size**
  - 128KB
  - 256KB
  - 512KB
  - 1MB
  - 2MB
  - 4MB
  - 8MB
  - 16MB
  - 32MB

- **缓存效果**

- **缓存大小**
  - 128KB
  - 256KB
  - 512KB
  - 1MB
  - 2MB
  - 4MB
  - 8MB
  - 16MB
  - 32MB
The big picture (update time)

Experiment for increasing # of edges (rnd, 500 nodes, w=1..5)
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]
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, Algorithmica 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…
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.

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
  - $\text{Diameter}()$: 
    - what is the diameter of $G$?
  - Do we really need APSP for this?
Some Open Problems…

- Dynamic Strongly Connected Components (directed graph $G$)
  $\text{SCC}(x,y)$:
  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