Engineering (dynamic) shortest path algorithms (a round trip between theory and experiments)

Camil Demetrescu

University of Rome “La Sapienza”

PATH’05
DIKU Summer School on Shortest Paths
Why engineering algorithms?

In theory, theory and practice are the same.

In practice, theory and practice may be quite 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>Instruction pipelining, …</td>
</tr>
</tbody>
</table>
The algorithm engineering cycle

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

Deeper insights

Bottlenecks, Heuristics

More realistic models

Hints to refine analysis
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.

*Available at this page:*
http://www-users.cs.york.ac.uk/~tw/empirical.html
Dynamic shortest paths: roadmap

Reduced costs

NSSSSP
Ramalingam-Reps ’96

SSSP
Frigioni et al ’98
Demetrescu ’01

Shortest path trees

Decremental BFS
Even-Shiloach ’81

Long paths decomposition

NAPSP
King ’99

Locally-defined path properties

NAPSP/APSP
Demetrescu-Italiano ‘03
Thorup ‘05
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:

- \( \text{Increase}(u,v,\varepsilon) : \) Increase weight \( w(u,v) \) by \( \varepsilon \)
- \( \text{Decrease}(u,v,\varepsilon) : \) Decrease weight \( w(u,v) \) by \( \varepsilon \)
- \( \text{Query}(v) : \) 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 as long as a static SSSP computation in the worst case!

Very efficient in practice
Increase\((u,v,\varepsilon)\)
Increase\((u,v,\varepsilon)\)

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)
Exercise 1

Let $G = (V,E,w)$ be a weighted directed graph, let $s$ be a source vertex, and let $T$ be a shortest path tree of $G$ rooted at $s$.

Let $A$ be the set of vertices in the subtree of $T$ rooted at $v$. Prove that no edge from $A$ to $V-A$ can become part of $T$ as a result of an increase $(u,v,\varepsilon)$ operation that increases the weight of edge $(u,v)$ by positive amount $\varepsilon$. 
Non-negative vs. negative edge weights

**Non-negative edge weights:**
No dynamic algorithm better than rebuilding from scratch (in the worst case) → **Main open problem!**

**Arbitrary edge weights:**
Best static algorithm as high as $O(m\sqrt{n})$, $O(mn)$ in general (Andrew Goldberg’s lecture)
Dynamic algorithms instead much faster than rebuilding from scratch:
   - update operations in the same bounds as static computations with non-negative weights (e.g., $O(m+n \log n)$ using Dijkstra)
Dynamic shortest paths: roadmap

- Reduced costs
- Shortest path trees

NSSSP
Ramalingam-Reps ’96

SSSP
Frigioni et al ’98
Demetrescu ’01
Graph reweighting using reduced weights

\[ G = (V, E, w) \quad \text{w : } E \rightarrow \mathbb{R} \]

Reweighting

\[ G_h = (V, E, w_h) \]

\[ \begin{align*}
    h : V &\rightarrow \mathbb{R} \quad \text{(potential func.)} \\
    w_h(u, v) &= w(u, v) + h(u) - h(v)
\end{align*} \]

Nice fact:

\[ P \text{ is a shortest path in } G \iff P \text{ is a shortest path in } G_h \]
Getting non-negative reduced weights

If we choose:
\[ h(v) := d(v) = \text{distance from } s \text{ to } v \text{ in } G \]

Claim:
\[ w_d(u,v) = w(u,v) + d(u) - d(v) \geq 0 \]

Proof:
\[ d(v) \leq w(u,v) + d(u) \quad [\text{Bellman cond.}] \]
\[ 0 \leq w(u,v) + d(u) - d(v) = w_d(u,v) \]
A cute property of $G_d$

Claim:
For any $v$, the distance $d_d(v)$ from $s$ to $v$ in $G_d$ is zero:

Proof:

$P = <s, v_1, v_2, \ldots, v_k, v> = \text{shortest path from } s \text{ to } v$

$d_d(v) = w_d(P) =

w_d(s,v_1) + \ldots + w_d(v_k,v) =

w(s,v_1) + \ldots + w(v_k,v) + d(s) - d(v_1) + d(v_1) - d(v_2) + \ldots =

w(P) + d(s) - d(v) = w(P) + 0 - w(P) = 0 \qed
An increase algorithm

Maintain $G$ and $d$ subject to the operation:

\[ \text{increase}(\Delta : E \to \mathbb{R}) \quad \Delta = \text{any non-neg. function} \]

1. Update $G$ by letting: $w \leftarrow w + \Delta$ \quad $O(m)$

2. Build $G_d$ ( $w_d$ is obviously non-negative ) \quad $O(m)$

3. Compute for each $v$ its distance $d_d(v)$ from $s$ in $G_d$ \quad e.g., $O(m + n \log n)$

4. For each $v$, update $d(v) \leftarrow d(v) + d_d(v)$ \quad $O(n)$

Exercise 2: prove that $d(v)$’s are correctly updated
A decrease algorithm

decrease(u, v, \varepsilon)

1. Update G by letting: \( w(u,v) \leftarrow w(u,v) - \varepsilon \) \( \text{O}(1) \)

2. Build \( G_d \), then remove \((u,v)\) from it and add \((s,v)\) with \( w_d(s,v) \leftarrow w_d(u,v) \) \( \text{O}(m) \)

3. Compute for each \( v \) its distance \( d_d(v) \) from \( s \) in \( G_d \) \( \text{O}(?) \)

4. For each \( v \), update \( d(v) \leftarrow d(v) + d_d(v) \) \( \text{O}(n) \)
Exercises

**Exercise 3:** how fast can step 3 be implemented?

**Exercise 4:** how can we detect negative cycles?

**Exercise 5:** prove that $d(v)$’s are correctly updated

**Exercise 6:** can we extend this to decrease $\omega(1)$ edges at the same time within the same time bounds? Would that be a breakthrough result?
Theory and practice

In theory, for arbitrary edge weights, we can do much better than rebuilding from scratch

\[ O(m \cdot n) \quad \rightarrow \quad O(m + n \cdot \log n) \]

In practice, can we get fast codes?

Two tricks:

• Only work for vertices affected by the update (Ramalingam-Reps’ approach)

• Avoid to build \( G_d \) explicitly
Exercise 7: write decrease \((u,v,\varepsilon)\)

\[
\begin{align*}
\text{increase}(u,v,\varepsilon) & \quad \text{Exercise 7: write decrease } (u,v,\varepsilon) \\
w(u,v) & \leftarrow w(u,v) + \varepsilon \\
\text{if } (u,v) \notin T(v) \text{ then return} \\
\text{let } H \text{ be a priority queue} \\
\text{add } x \in T(v) \text{ to } H \text{ with priority:} \\
p(x) & = \min_{(z,x): z \notin T(v)} d(z) + w(z,x) - d(x) \\
\text{while } (H \neq \emptyset) \\
x & \leftarrow \text{min priority vertex in } H \\
d(x) & \leftarrow d(x) + p(x) \\
\text{for each } (x,y) \\
\text{if } d(x) + w(x,y) - d(y) < p(y) \text{ then} \\
p(y) & \leftarrow d(x) + w(x,y) - d(y)
\end{align*}
\]
Experimental setup

Experimental platform:
- C++ using LEDA, g++ compiler
- UNIX Solaris on SPARC Ultra 10 at 300 Mhz

Test sets:
- Random graphs & random update sequences
  (we used potentials technique to avoid negative cycles)

Performance indicators:
- Running time (msec)
- Number of updated vertices per operation
Static vs. dynamic

\[ m = 0.5n^2, \text{Edge Weights in } [-1000,1000] \]
Can we do any better?

If shortest paths are not unique, not all the vertices in $T(v)$ may actually change distance.

**Output-bounded cost model** (Ramalingam-Reps): an optimal algorithm should spend time proportional to actual change in output solution due to update operation (e.g., changes in the shortest paths tree).

Ramalingam & Reps (and later Frigioni et al.) have devised algorithms in this model for dynamic SSSP.
Static vs. dynamic

For the given data points, the graph shows the average running time per operation (msec) as a function of the number of vertices. The graph compares three methods: BFM, SIMPLE, and RR. The horizontal axis represents the number of vertices, and the vertical axis represents the average running time per operation.

The data points are as follows:

- **BFM**:
  - Number of vertices: 0, Running time: 0.807 msec
  - Number of vertices: 100, Running time: 17.984 msec
  - Number of vertices: 500, Running time: 625.325 msec

- **SIMPLE**:
  - Number of vertices: 0, Running time: 1.137 msec
  - Number of vertices: 100, Running time: 6.108 msec
  - Number of vertices: 500, Running time: 353.704 msec

- **RR**:
  - Number of vertices: 0, Running time: 2.191 msec
  - Number of vertices: 100, Running time: 5.79 msec
  - Number of vertices: 500, Running time: 6.108 msec

The formula used for the data generation is $m=0.5n^2$, with edge weights in the range [-1000, 1000].
Number of updated vertices

Average processed vertices per operation

Edge weight interval $[-2^k, 2^k]$

$n=300$, $m=0.5n^2=45000$
Further readings

**Ramalingam & Reps’ approach + RR algorithm:**
[Ramalingam-Reps’96]

**SIMPLE implementation of RR approach + experiments:**
[Demetrescu’01]

Other computational study (not covered in this lecture):
Luciana S. Buriol, Mauricio G. C. Resende and Mikkel Thorup, Speeding up dynamic shortest paths
http://citeseer.ist.psu.edu/689842.html
Dynamic shortest paths: roadmap

Reduced costs

NSSSP
Ramalingam-Reps ’96

Decremental BFS
Even-Shiloach ’81

SSSP
Frigioni et al ’98
Demetrescu ’01
Decremental BFS  [Even-Shiloach’81]

delete(S): delete all edges in S from graph G
query(v): return distance from s to v in graph G (if it is at most d)

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

O(md) total time over any deletion sequence
O(d) time per deletion (amortized over Ω(m) deletions)
Can we do any better than $O(mn)$?

Roditty and Zwick in ESA 2004 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

We 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

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

First row:  \( C[1,x] = 1 \) iff \( \text{dist}(s,x) = 3 \)
Second row:  \( C[2,x] = 1 \) iff \( \text{dist}(s,x) = 4 \)
Third row:  \( C[3,x] = 1 \) iff \( \text{dist}(s,x) = 5 \)

…  

…  

\[
\begin{array}{cccccc}
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]
n deletions and $n^2$ queries

Decremental BFS in $o(mn)$ total time would yield boolean matrix multiplication in $o(mn)$
Undirected APSP $\rightarrow$ decremental SSSP

$W = \text{largest edge weight}$

Exercise 8: does this reduction work in directed graphs?

$n$ deletions and $n^2$ queries

Decremental SSSP in $o(mn)$ total time would yield undirected APSP in $o(mn)$
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’04]
Liam Roditty, Uri Zwick,
On dynamic shortest paths problems
Dynamic shortest paths: roadmap

- Reduced costs
- Long paths decomposition
- Shortest path trees
  - Decremental BFS
    - Even-Shiloach ’81
  - NSSSP
    - Ramalingam-Reps ’96
  - SSSP
    - Frigioni et al ’98
    - Demetrescu ’01
  - NAPSP
    - King ’99
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]

\[ \tilde{O}(n^{2.5} \sqrt{C}) \text{ update time} \quad O(1) \text{ query time} \quad \tilde{O}(n^{2.5} \sqrt{C}) \text{ space} \]

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

---

A graphical representation shows a path from Nyhavn to Universitetsparken, where each segment is less than \( k \).
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
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

NAPSP
King ’99

Long paths decomposition

Locally-defined path properties

NAPSP/APSP
Demetrescu-Italiano ‘03 Thorup ‘05
A path is *locally shortest* if it is shortest without either endpoints.
Locally shortest paths (LSP’s)

By optimal-substructure property of shortest paths:

Exercise 9: let S be the set of locally shortest paths of a graph (it holds $|S| < m \cdot n$, if shortest paths are unique). How much space do we need to represent all paths in S?
Using LSP’s 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:

```
\[
\begin{array}{c}
\text{x} \\
\end{array}
\begin{array}{c}
\longrightarrow \\
\text{a} \\
\end{array}
\begin{array}{c}
\longrightarrow \\
\text{y} \\
\end{array}
\]
```

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:

```
\[
\begin{array}{c}
\text{x} \\
\end{array}
\begin{array}{c}
\longrightarrow \\
\text{a} \\
\end{array}
\begin{array}{c}
\longrightarrow \\
\text{y} \\
\end{array}
\]
```

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

3. Possibly insert \((x, y')\) into heap or decrease priority
How much do we gain?

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

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

Q.: How many locally shortest paths?

A.: $\#\text{LS-paths} \leq mn$. No gain in “asymptopia”…

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

Locally shortest paths in random graphs (500 nodes)

- $m \times n$
- $n \times n$

# edges

$\# \text{LS-paths}$

$m \times n$

$n \times n$
Locally shortest paths in US road networks
Can we exploit this in practice?

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

- Dijkstra's algorithm
- New algorithm

Algorithm based on locally shortest paths
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 NAPSP variant based on locally shortest paths:
[Demetrescu-Emiliozzi-Italiano’04]
Camil Demetrescu, Stefano Emiliozzi, Giuseppe F. Italiano: Experimental analysis of dynamic all pairs shortest path algorithms. SODA 2004: 369-378
Dynamic shortest paths: roadmap

- Reduced costs
  - Shortest path trees
    - Locally-defined path properties
      - Long paths decomposition

- NSSSSP
  - Ramalingam-Reps ’96
    - Decremental BFS
      - Even-Shiloach ’81
        - NAPSP/APSP
          - Demetrescu-Italiano ‘03
            - Thorup ‘05

- SSSP
  - Frigioni et al ’98
    - Demetrescu ’01

- NAPSP
  - King ’99

- 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>S-DIJ</td>
<td>$O(mn + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>D/Italiano 03</td>
<td>S-LSP</td>
<td>$O(#LSP + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Ramaling./Reps 96</td>
<td>D-RRL</td>
<td>$O(mn + n^2 \log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>King 99</td>
<td>D-KIN</td>
<td>$O(n^{2.5} (C \log n)^{0.5})$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>D/Italiano 03</td>
<td>D-LHP</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

- 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 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

- Update time: $O(mn + n^2 \log n)$
- Query time: $O(1)$
- Space: $O(n^2)$

**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-LHP [Dem.-Italiano’03]

Directed graphs with real edge weights

\[ \tilde{O}(n^2) \text{ time per update} \quad O(1) \text{ time per query} \quad \tilde{O}(mn) \text{ space} \]

Approach:

Maintain locally historical paths (LHP):
paths whose proper subpaths have been shortest paths…
Are LHPs useful in practice? *(update time)*

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

**Exercise 10:** how can the update time decrease as the number of edges increases?
What about real graphs? (update time)

Experiments on US road networks

<table>
<thead>
<tr>
<th>US states</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>NV</td>
</tr>
<tr>
<td>MT</td>
</tr>
<tr>
<td>ND</td>
</tr>
<tr>
<td>CT</td>
</tr>
<tr>
<td>NM</td>
</tr>
<tr>
<td>MN</td>
</tr>
<tr>
<td>IA</td>
</tr>
<tr>
<td>WI</td>
</tr>
<tr>
<td>IL</td>
</tr>
<tr>
<td>IN</td>
</tr>
<tr>
<td>KY</td>
</tr>
<tr>
<td>MO</td>
</tr>
<tr>
<td>AR</td>
</tr>
<tr>
<td>KS</td>
</tr>
<tr>
<td>MS</td>
</tr>
<tr>
<td>IN</td>
</tr>
<tr>
<td>AL</td>
</tr>
<tr>
<td>GA</td>
</tr>
<tr>
<td>FL</td>
</tr>
<tr>
<td>NC</td>
</tr>
<tr>
<td>SC</td>
</tr>
<tr>
<td>TN</td>
</tr>
<tr>
<td>KY</td>
</tr>
<tr>
<td>MS</td>
</tr>
<tr>
<td>IN</td>
</tr>
<tr>
<td>AL</td>
</tr>
<tr>
<td>GA</td>
</tr>
<tr>
<td>FL</td>
</tr>
<tr>
<td>NC</td>
</tr>
</tbody>
</table>

Legend:
- D - RRL
- D - LHP
Zoom out to static (update time)

Experiments on US road networks

- S-D IJ
- D-RR L
- D-LHP

US states
Big issue: hit the space wall
Experiments on US road networks

(on different platforms)

- IBM Power 4
- Sun UltraSPARC IIi
- AMD Athlon

Number of edges (x 100)

Relative time performance D-RLL/D-LHP

D-LHP faster than D-RRL

D-LHP slower than D-RRL

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

![Graph showing cache effects with simulated cache miss ratio D-RRL/D-LHP and performance ratio D-RRL/D-LHP on real architectures.](image)

- **Simulated cache miss ratio D-RRL/D-LHP**
- **Performance ratio D-RRL/D-LHP on real architectures**

**Colorado road network**

- Power 4: 1.59
- Athlon: 0.71
- Xeon: 0.87
- UltraSPARC IIi: 0.92

**Cache size**:
- 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 about pathological instances?

**Experiments on bottleneck graphs (500 nodes, IBM Power4)**

![Graph](image)

- D-RRL
- S-DIJ
- D-LHP
- S-LSP

What about pathological instances?
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):

<table>
<thead>
<tr>
<th>Very simple</th>
<th>Hard to beat!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(but quite bad on pathological instances)</td>
</tr>
</tbody>
</table>

D-LHP (Dem-Ita’03):

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

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

#locally shortest paths $\approx$ #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 3000 vertices, 80 bytes per vertex pair: quadratic space means 720 Mbytes
With RAMs order of GB, 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-Emiliozzi-Italiano’04]
Camil Demetrescu, Stefano Emiliozzi, Giuseppe F. Italiano: Experimental analysis of dynamic all pairs shortest path algorithms. SODA 2004: 369-378
“People who analyze algorithms have double happiness. First of all they experience the sheer beauty of mathematical patterns that surround elegant computational procedures. Then they receive a practical payoff when their theories make it possible to make other jobs done more quickly and more economically.”

D. E. Knuth