Dynamic Single Source Shortest Path: Theoretical and Experimental Results

Alberto Marchetti-Spaccamela

Dip. Informatica e Sistemistica
Università di Roma "La Sapienza"

AMORE School: Konstanz, October 17, 2000

Summary

PART 1

- Dynamic graph algorithms
- Cost analysis and Output complexity
- The fully dynamic single source shortest path problem
  1. Positive real edge weights:
     weight decrease and weight increase
  2. Experimental analysis

PART 2

- The dynamic single source shortest path problem with real weights
- Experimental analysis
- Reachability

Dynamic Graph Algorithms

Input

- A (weighted) graph $G = (V, E)$;
- A property $\mathcal{P}$ on G (e.g., *connectivity*);
- A sequence σ of updates (insertions, deletions, weight updates of edges) on G and queries on $\mathcal{P}$;

Goal

Maintain $\mathcal{P}$ without recomputing $\mathcal{P}$ from scratch after each update allow fast queries.

Semi dynamic:

- *Incremental:* only edge insertions can be handled efficiently;
- *Decremental:* only edge deletions can be handled efficiently;

Fully dynamic: both insertions and deletions are supported.

Fully dynamic harder than semi dynamic

Dynamic Graph Algorithms: Applications

- Networks management
- Enumerative algorithms for combinatorial analysis
- Incremental compilation and data flow analysis
- Graphical systems
- VLSI design

Example

Planning a transportation network robust to a certain number of interruptions (road works, traffic jam, ...) or choosing improvements from a set of alternatives.

Many istances of the original problem on networks that are *slightly* different.

Dynamic single source shortest path problem

Given a graph $G = (V, E)$ ($|V| = n$, $|E| = m$), a source vertex s and a weight function $w : E \to \mathcal{R}$.

- $d(x)$ denotes the distance of vertex x from s;
- $T(s)$ is a shortest path tree rooted in s for G.

Goal: Maintaining the information on d and $T(s)$ while performing an arbitrary (fully dynamic) sequence of the following operations:

- `Distance`(x)
- `Min_Path`(x)
- `Insert`(x, y, w)
- `Weight_Decrease`(x, y, ϵ)
- `Delete`(x, y)
- `Weight_Increase`(x, y, ϵ)

Cost analysis

- I = the set of all possible input changes;
- S = the set of all possible sequences of input changes;
- $\sigma = \langle op_1, op_2, \ldots, op_k \rangle \in S, \quad op_i \in I.$

1. *worst case cost:* $A_{wc}(op_i) = \max_{op \in I}\{\mathrm{cost}_A(op)\}$
2. *amortized cost:* (Tarjan 85) $A_a(op_i) = \max_{\sigma \in \mathrm{S}}\{\frac{\mathrm{cost}_A(\sigma)}{|\sigma|}\}$

Problem:

No efficient fully dynamic solution is known in these models for several fundamental problems on directed graphs (e.g., *transitive closure*, *shortest paths*) (i.e. better than recomputing from scratch) even if only positive weights are allowed!

State of the art (usual cost model)

Semi Dynamic

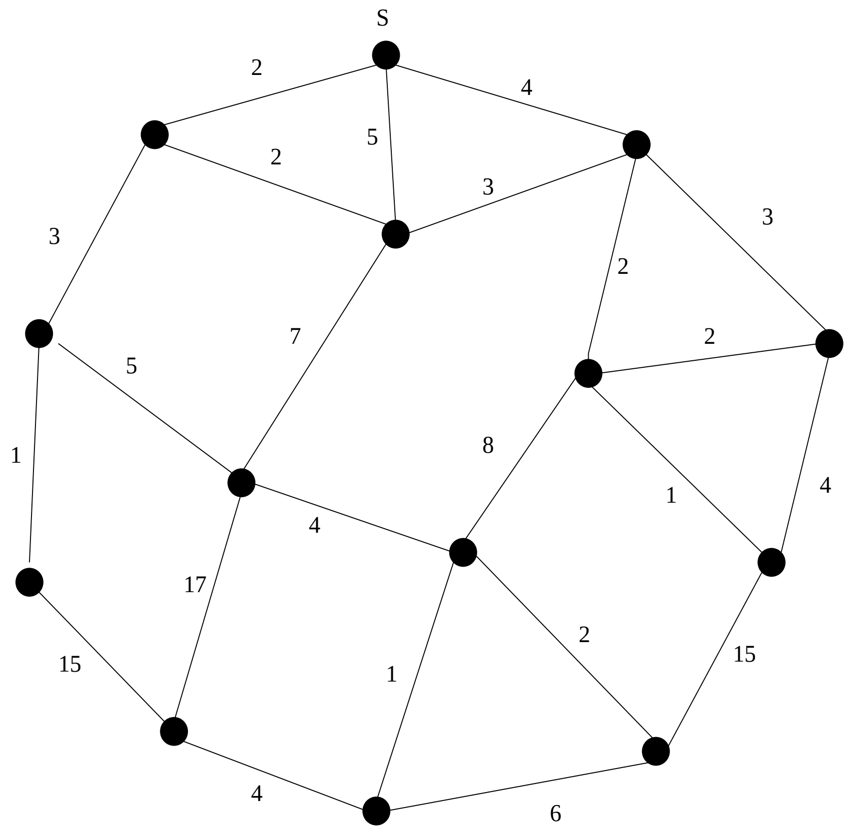
- (integer edge weights in $[1..C]$):
  - Incremental solution
    $O(1)$ w.c. time per `Distance` query
    $O(nC)$ am. time per `Insert`
  - Decremental solution
    $O(1)$ w.c. time per `Distance` query
    $O(nC)$ am. time per `Delete`
  - Ausiello, et al. J.Alg 91, Franciosa et al. – STACS'97

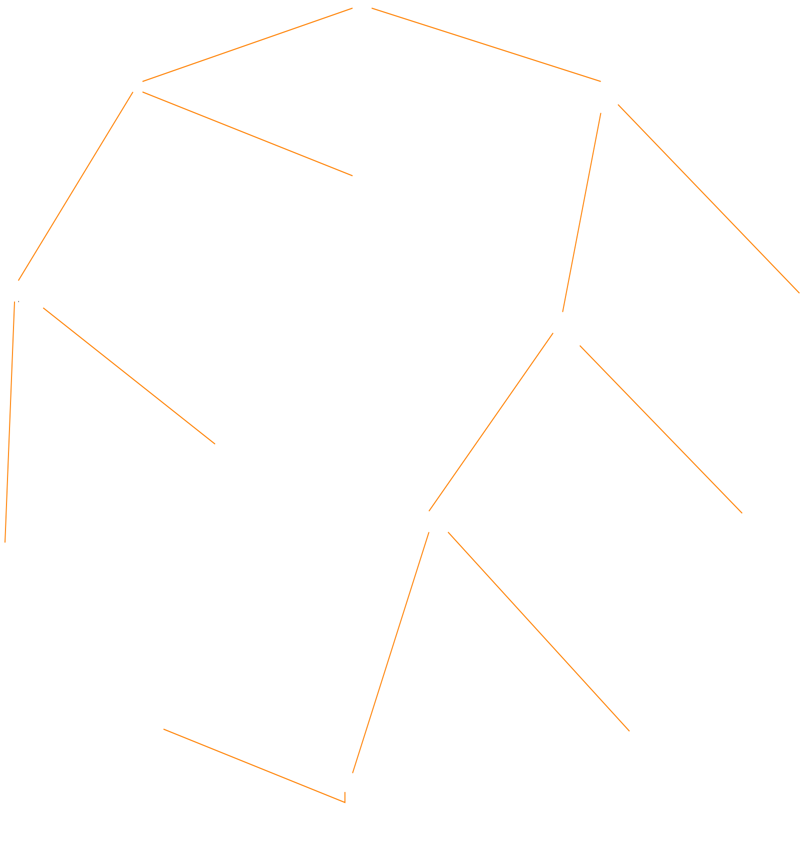
- Other results are known for particular classes of graphs (planar, bounded treewidth, ...) [CZ95, FM93, KS93, KRRS93]
- Probabilistic algorithms that do not guarantee correctness

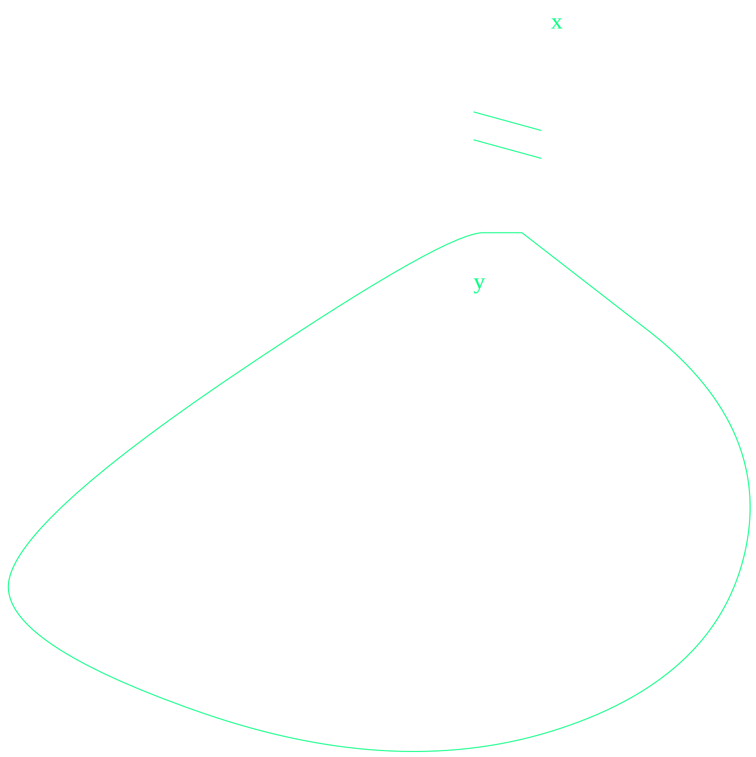
Output complexity

Ramalingam and Reps: evaluate the cost of dynamic algorithms as a function of the number of modifications determined by each input change.

Example.






x
y

1 input change determines 3 output changes !!!

Output complexity models

Ramalingam and Reps (91,96)

RR propose algo. whose cost depends on

$|V_\delta| + |E_\delta|, \quad \delta \in I$

V_δ = set of vertices *affected* by δ

E_δ = set of edges with *at least* one affected endpoint

Frigioni, Marchetti-Spaccamela and Nanni (96,98)

$\Delta(\cdot)$ = *intrinsic cost* of the algorithm (explicit updates)

$\Delta(\sigma)$ = number of output updates determined by σ

1. *worst case:* $A_{wc} = \max_{\delta \in I}\{\frac{\text{cost}_A(\delta)}{\Delta(\delta)}\}$

   $\Delta(\delta)$ = number of output updates determined by δ

2. *amortized:* $A_a = \max_{\sigma \in \mathrm{S}}\{\frac{\text{cost}_A(\sigma)}{\Delta(\sigma)}\}$

Note that: $\Delta(\delta) \leq ||\delta|| \leq n|V_\delta|$

In the sequel we assume positive weights

Decrease the weight of edge (x, y) by ε

Distances of vertices from s can only decrease.

Affected vertex: every vertex changing its distance from s as a consequence of an input change δ.

Vertices in $T(y)$ decrease their distance by ε.

Vertices not in $T(y)$ might decrease their distance as well.

$d'(z)$ the new distance of z after the update

Decrease the weight of edge (x, y) by ε

Ramalingam, Reps (TCS 96)

Apply Dijkstra's algorithm to the subgraph induced by edges with *at least* one affected endpoint:

1. y is affected
2. For each *affected* vertex traverse the edges (z, q) outgoing z in order to discover whether q is affected or not.

Steps a1 and 2 can be performed concurrently

The algorithm maintains the single source shortest paths tree; hence optimal query time

Decrease the weight of edge (x, y) by ε

Ramalingam, Reps (TCS 96):

Q is a priority queue

1. $d'(y) = d(y) - \varepsilon$
2. initialize Q with y and with priority $d'(y)$
3. **while** Q is not empty **do begin**
   (a) $z =$ Pop (Q)
   (b) $d'(z) =$ priority of z in Q
   (c) For each edge (z, q) (eventually) update priority of q
       end

   end

Apply Dijkstra's algorithm to the subgraph induced by edges with *at least* one affected endpoint:

1. y is affected
2. For each *affected* vertex traverse the edges (z, q) outgoing z in order to discover whether q is affected or not.

k-bounded ownership

An *ownership function* for $G = (V, E)$ is any function $A : E \to V$ s. t. $\forall\ (x, y) \in E$, $A(x, y)$ (the *owner* of (x, y)) is either x or y.

Also defined as orientation of edges in an undirected graph (Eppstein)

A is *k-bounded* if $\forall\ x \in V$, the set $A^{-1}(x) = \{(x, y) \mid A(x, y) = x\}$ of the edges owned by x has cardinality at most k.

If G has a k-bounded ownership function then the algorithms require $O(k \log n)$ time per output update.

Optimal query time and space.

k-bounded ownership

k	Graph
$O(\sqrt{m})$	general graphs with m edges
$O(1+\sqrt{\gamma})$	graphs with genus γ
$k \leq 3$	planar graphs
$k \leq a$	graphs with arboricity a
$k \leq d$	graphs with maximum degree d
$k \leq t$	graphs with treewidth t
$k \leq p$	graphs with pagenumber p

A 2-apx greedy algorithm:
repeat

1. sort vertices accoording to their degree
2. let x be the vertex with min degree
3. assign to $own(x)$ all edges of x
4. delete x (and all adjacent edges) from G

until finished

Decreasing the weight of edge (x, y)

Goal:
find an algorithm whose cost depends the number n_A of affected vertices.

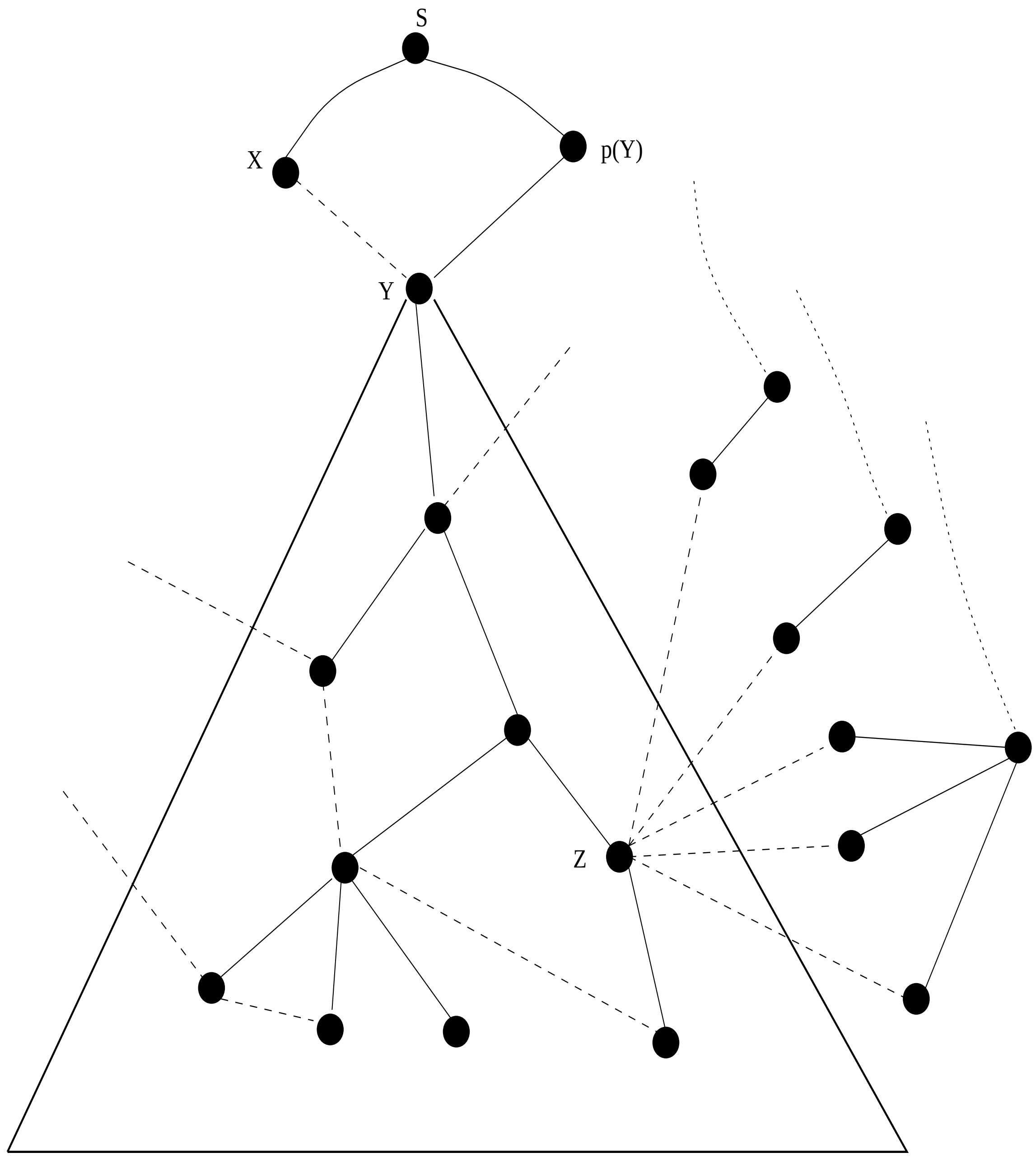
Problem:
For each affected vertex z the algorithm should traverse only edges (z, q) outgoing z, such that q is affected as well.

Frigioni, MS, Nanni 98:
For each affected vertex z, the algorithm traverses edges (z, q) such that:

- either q is affected as well
- or (z, q) is owned by z

$O(kn_A)$ edges are traversed, if G has a k-bounded acct. fct.
Eacg edge traversal costs $O(\log n)$.

An example



Which are the neighbors of z that improve their distance from s choosing z as new parent in the tree?

Green edges = edges owned by z (at most k)

Blue edges = edges not owned by z

The Algorithm

- $d'(z)$ = new distance of z from s;
- Any vertex z keeps a max-based heap $B(z)$ of all adjacent vertices (edges) not owned by z. Every $q \in B(z)$ has priority:

$$b_z(q) = d(q) - w_{z,q}$$

Key observation: For each *affected* vertex z, $(d'(z) < d(z))$ if $d'(z) < b_z(q)$ then q is affected!

$$d'(z) < b_z(q) = d(q) - w_{z,q}$$

$d'(z) + w_{z,q} < d(q) \quad \longrightarrow \quad q \quad$ is affected!

The Algorithm

1. $d'(y) = d(y) - \varepsilon$
2. initialize Q with y and with priority $d'(y)$
3. **while** Q is not empty **do begin**
   (a) $z =$ Pop (Q)
   (b) $d'(z) =$ priority of z in Q
   (c) (eventually) update priority of q for each edge (z, q) owned by z
   (d) update priority of q for each edge (z, q) not owned by z s.t. (z, q) $d'(z) < b_z(q)$
   (e) **end**

   end

Q is a priority queue

Maintaining for each v and w $b_w(v)$ can be done without extra costs

Increase the weight of edge (x, y) by ε

Definition:

Given $G = (V, E)$ and $T(s)$, a vertex z is affected if the weight of edge (x, y) is increased if

- either z increases its distance from source s,
- or z must change its parent while maintaining the same distance from the root.

The set of affected vertices is a subset of $T(y)$

y is affected

$d'(z)$ the new distance of z after the update

V_δ is the no. of affected vert.

E_δ is the no. of edges incident at least one affected vert.

Increase the weight of edge (x, y) by ε

Ramalingam, Reps (TCS 96):

1. Find the set of affected vertices

2. Apply a Dijkstra's like procedure to the subgraph induced by affected vertices:

1. Find the set of affected vertices:

Search $T(y)$ starting from y

For each considered vert. z consider all edges (z, q) s.t. $q \notin T(y)$ searching an alternative path

Cost of 1. : $O(V_\delta)$

Increase the weight of edge (x, y) by ε

Ramalingam, Reps (TCS 96) cont.:

2. Apply a Dijkstra's like procedure to the subgraph induced by affected vertices:

Q is a priority queue

1. initial priority of each vertex =
   lenght of short. path not passing thru affected vert. (or ∞)
2. at each iteration extract vert. z with min priority
3. when vertex z is deleted from Q traverse all edges (z, q) to (event.) update priority of q.

Cost of 2. : $O(E_\delta \log V_\delta)$

Increase the weight of edge (x, y) by ε

Frigioni MS Nanni (J. Alg. 00):

- use of ownership fct. and
- each vertex z stores info of not owned edges (z, q)
  that allows to detect whether q is affected

2. Apply a Dijkstra's like procedure to the subgraph induced by affected vertices:

1. initial priority of each vertex =
   lenght of short. path not passing thru affected vert. (or ∞)
2. loop
   (a) extract vert. z with min priority
   (b) traverse edges (z, q) s.t.
       either (z, q) is owned by z
       or q is affected

Cost of 2. : Same time and space bounds as the weight decrease

Experimental results
(*Frigioni et al. 1997*)

Platform: `UNIX`, `C++`, `LEDA`

Algorithms

- Dijkstra (`LEDA` Implementation)
- Ramalingam-Reps (TCS 96)
- Frigioni-Marchetti-Nanni (SODA'96)

Experiments

- Random graphs and real world graphs
- Positive integer edge weights
- Random sequences of updates

Some graphics

TOTAL CPU TIME
random sequences of 10000 edge operations
random graphs (edge density: 50%)
(average on 5 experiments)

#nodes	#edges	Dij	RR	FMN
50	612	1,28603	0,45033	0,76038
100	2475	1,89683	0,45686	0,83812
150	5587	2,50931	0,46737	0,94515
200	9950	3,32532	0,46861	0,99000
250	15562	4,56150	0,47243	0,96933
300	22425	5,88450	0,48552	1,03554
350	30537	6,73867	0,47168	0,95846
400	39900	7,87904	0,51132	1,18642
450	50512	8,72722	0,48276	1,10121
500	62375	10,38706	0,53324	1,30023

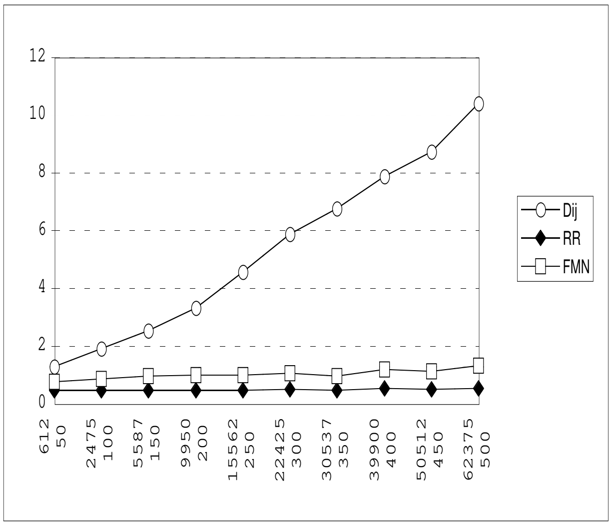


Figure 1: CPU time (in seconds) to process random sequences of 10000 operations on graphs with constant edge density (50%) and number of vertices ranging from 50 to 500.

NUMBER OF SCANNED EDGES
modifying sequences of 1000 operations
random graphs
(average on 3 experiments)

#nodes	#edges	RR	FMN
200	1990	55140	39298
200	3980	97068	66869
200	5970	127814	85901
200	7960	174779	120717
200	9950	212046	143454
200	11940	233291	156836
200	13930	259088	174416
200	15920	289771	192737
200	17910	328451	217438
200	19900	355551	233171

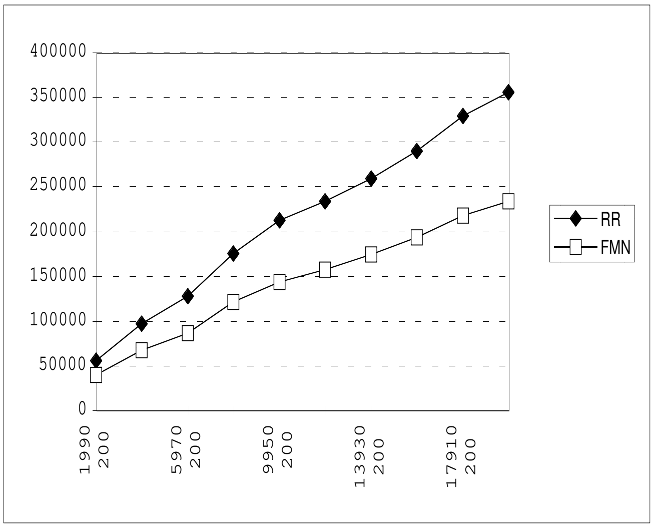


Figure 2: number of edges scanned while processing random sequences of 1000 modifying operations on graphs with 200 vertices and variable edge density (ranging from 10% to 90%).

TOTAL CPU TIME
ad hoc sequences of 1000 operations
special graphs
(single experiment)

#nodes	#edges	RR	FMN
202	400	0,70004	0,06570
402	800	1,33943	0,08198
602	1200	2,03070	0,11350
802	1600	2,75082	0,11100
1002	2000	3,46326	0,13705
1202	2400	5,01197	0,14609
1402	2800	5,98160	0,16497
1602	3200	6,64268	0,20075
1802	3600	7,42286	0,20708
2002	4000	8,13032	0,21597

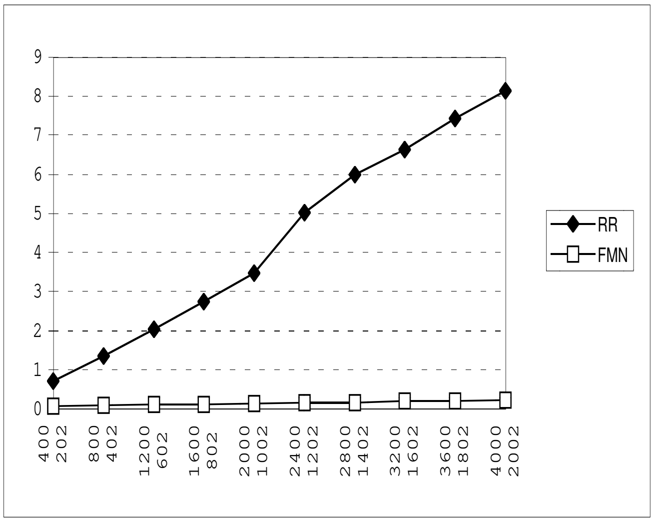


Figure 3: ad hoc sequences of updates on special graphs with variable size.

Total time (in seconds) Time ratio = total time / time(Dij)

source	Dij	RR	FMN	Dij	RR	FMN
AS1755	12.615026	0.653482	0.462232	100.00%	5.18%	3.66%
AS3561	12.563434	0.651382	0.470504	100.00%	5.18%	3.75%

Scanned edges Edge ratio = scanned edges / edges(Dij)

source	Dij	RR	FMN	Dij	RR	FMN
AS1755	6,725,010	20,466	15,401	100.00%	0.30%	0.23%
AS3561	6,706,948	21,073	15,829	100.00%	0.31%	0.24%

Figure 4: random sequences of 10000 edge insertions and deletions on the Internet graph.

Conclusions

- Efficient dynamic algorithms for the single source shortest path problem

  1. Simple data structures and algorithms ("practical" implementation) (RR 96)
  2. "almost" optimal output–bounded algorithms for large classes of graphs (FMSN 98 and 00)

- Open problems: all pairs shortest paths, better algortihms for general graphs