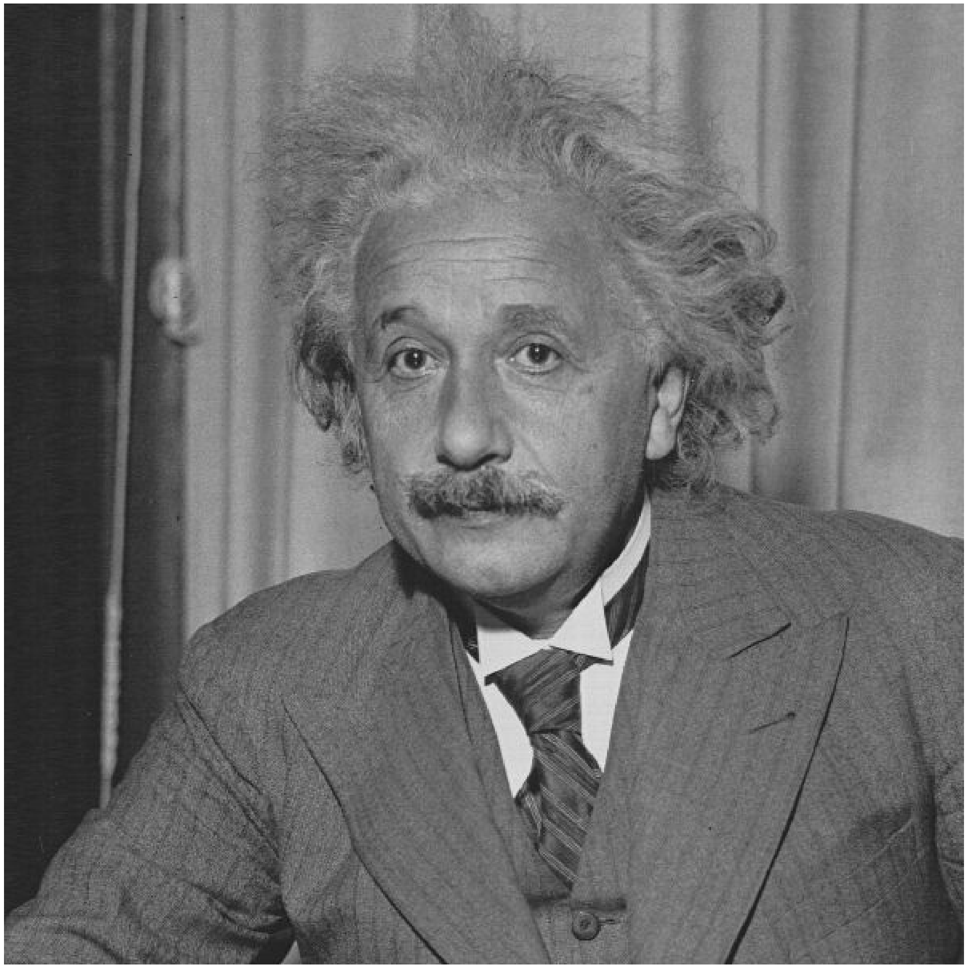
Point-to-Point Shortest Path Algorithms with Preprocessing

Andrew V. Goldberg
Microsoft Research – Silicon Valley
www.research.microsoft.com/∼goldberg/

Joint work with
Chris Harrelson, Haim Kaplan, and Renato Werneck

Einstein Quote



Everything should be made as simple as possible, but not simpler

Shortest Path Problem

Variants

- Non-negative and arbitrary arc lengths.
- Point to point, single source, all pairs.
- Directed and undirected.

Here we study

- Point to point, non-negative length, directed problem.
- Allow preprocessing with limited (linear) space.

Many applications, both directly and as a subroutine.

Shortest Path Problem

Input: Directed graph $G = (V, A)$, non-negative length function $\ell : A \to \mathbf{R}^+$, source $s \in V$, terminal $t \in V$.

Preprocessing: Limited space to store results.

Query: Find a shortest path from s to t.

Interested in exact algorithms that search a subgraph.

Related work: reach-based routing [Gutman 04], hierarchical decomposition [Schultz, Wagner & Weihe 02], [Sanders & Schultes 05, 06], geometric pruning [Wagner & Willhalm 03], arc flags [Lauther 04], [Köhler, Möhring & Schilling 05], [Möhring et al. 06], DIMACS Implementation Challenge 2006.

Motivating Application

Driving directions

- Run on servers and small devices.
- Implementations until recently:
  - Use base graph based on road categories and manually augmented.
  - Runs (bidirectional) Dijkstra or A* with Euclidean bounds on "patched" graph.
  - Non-exact, not very efficient.
- Interested in exact and very efficient algorithms.
- Big graphs: Western Europe, USA, North America: 18 to 30 million vertices.

Outline

- Scanning method and Dijkstra's algorithm.
- Bidirectional Dijkstra's algorithm.
- A^* search.
- ALT Algorithm
- Definition of reach
- Reach-based algorithm
- Combining reach and A^*

Scanning Method

- For each vertex v maintain its distance label $d_s(v)$ and status $S(v) \in \{\mathtt{unreached}, \mathtt{labeled}, \mathtt{scanned}\}$.
- Unreached vertices have $d_s(v) = \infty$.
- If $d_s(v)$ decreases, v becomes labeled.
- To scan a labeled vertex v, for each arc (v, w), if $d_s(w) > d_s(v) + \ell(v, w)$ set $d_s(w) = d_s(v) + \ell(v, w)$.
- Initially for all vertices are unreached.
- Start by decreasing $d_s(s)$ to 0.
- While there are labeled vertices, pick one and scan it.
- Different selection rules lead to different algorithms.

Dijkstra's Algorithm

[Dijkstra 1959], [Dantzig 1963].

- At each step scan a labeled vertex with the minimum label.
- Stop when t is selected for scanning.

Work almost linear in the visited subgraph size.

Bidirectional Algorithm

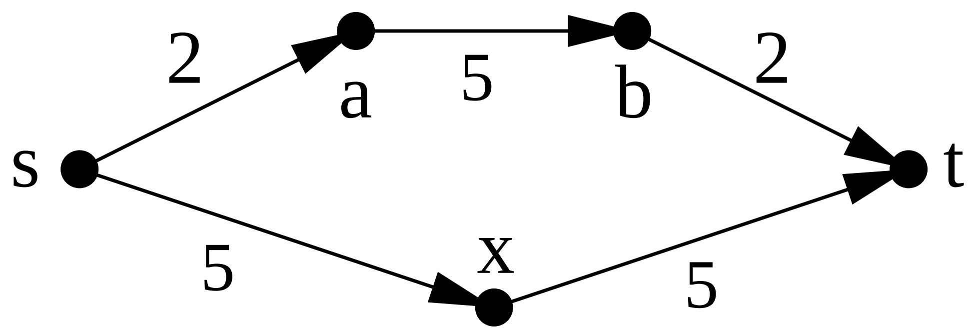
Reverse Algorithm: Run algorithm from t in the graph with all arcs reversed, stop when t is selected for scanning.

Bidirectional Algorithm

- Run forward Dijkstra from s and backward from t.
- Maintain μ, the length of the shortest path seen: when scanning an arc (v, w) such that w has been scanned in the other direction, check if the corresponding s-t path improves μ.
- Stop when about to scan a vertex x scanned in the other direction.
- Output μ and the corresponding path.

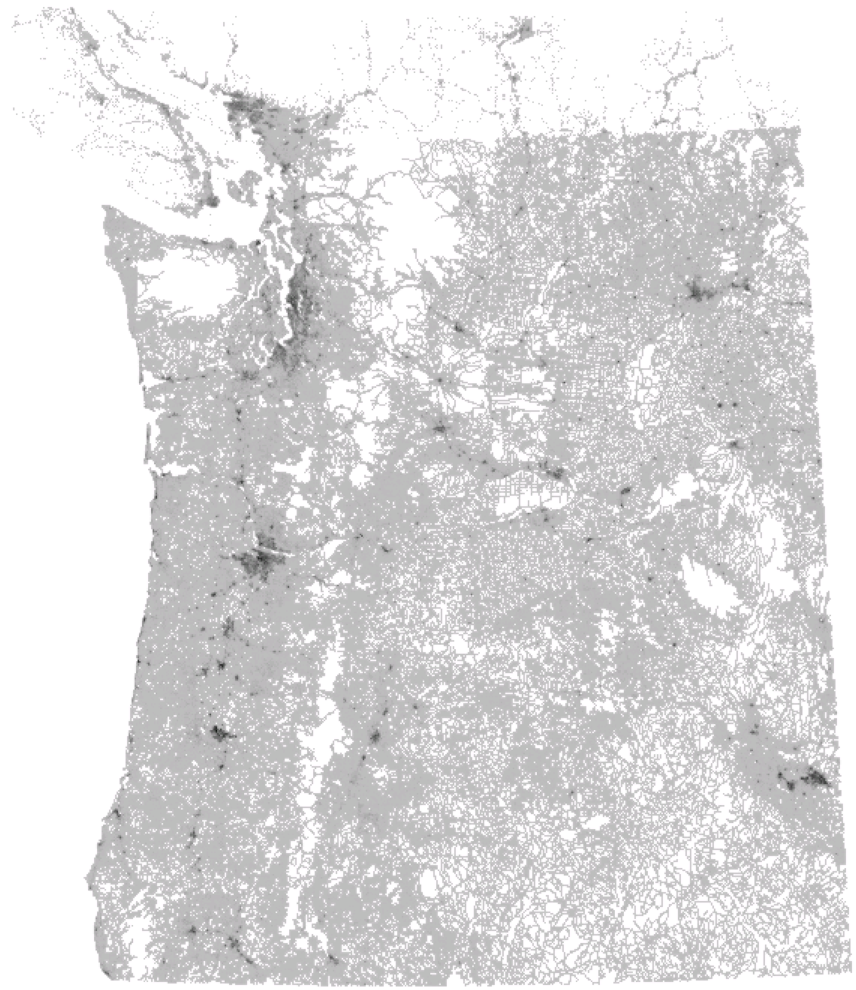
Bidirectional Algorithm: Pitfalls

The algorithm is not as simple as it looks.



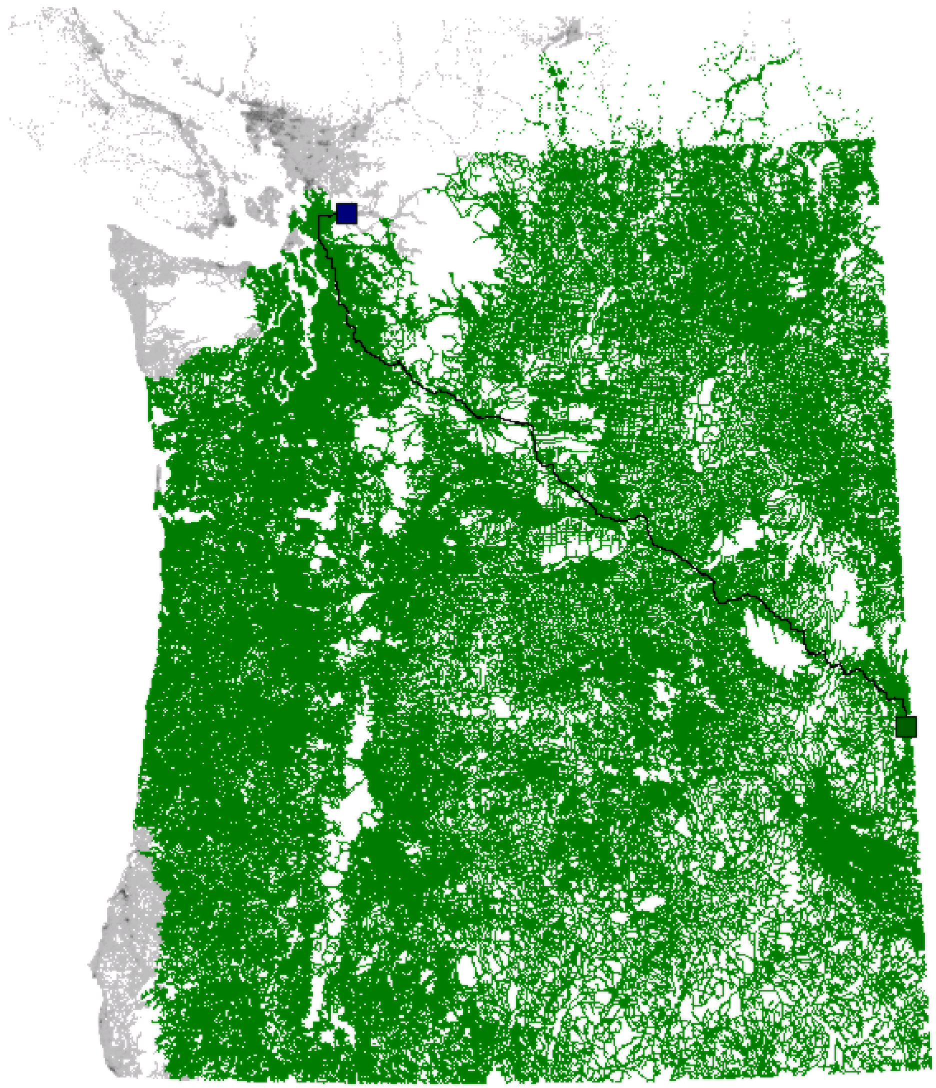
The searches meat at x, but x is not on the shortest path.

Example Graph



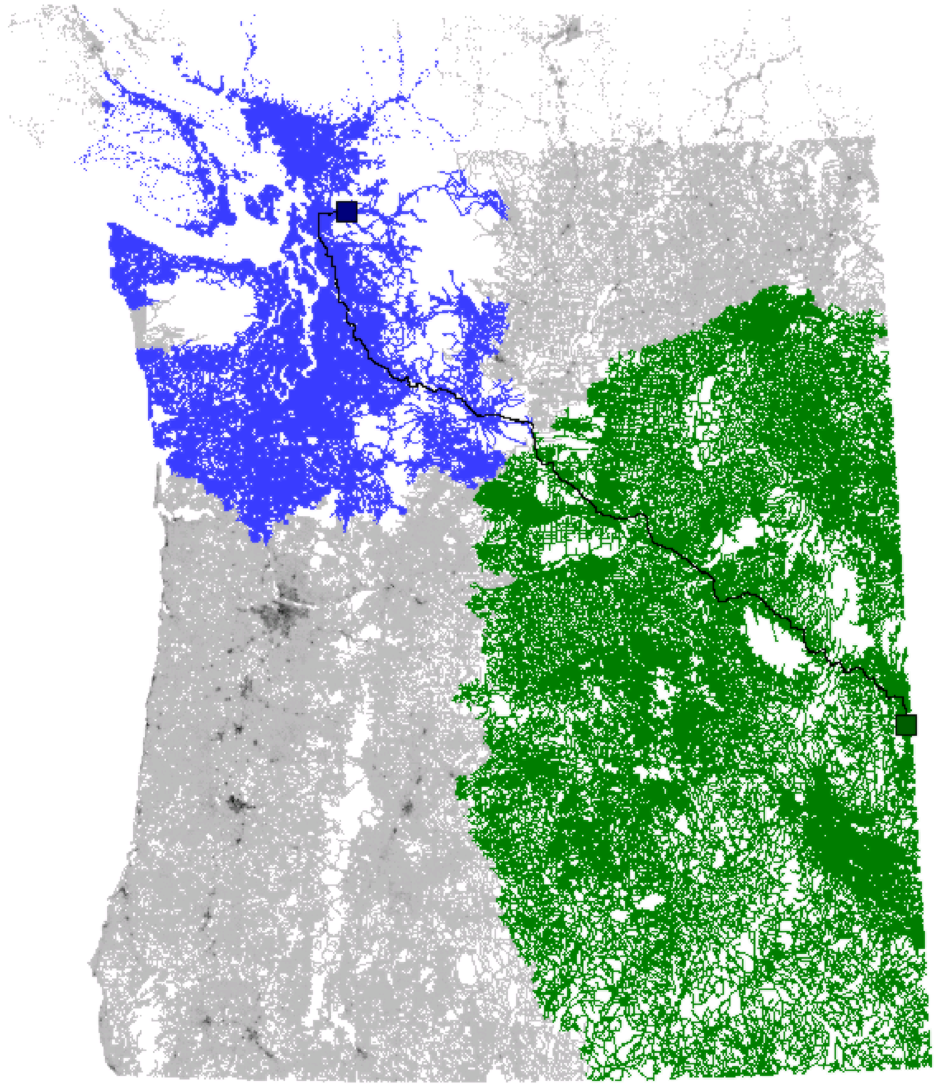
1.6M vertices, 3.8M arcs, travel time metric.

Dijkstra's Algorithm



Searched area

Bidirectional Algorithm



forward search/ reverse search

A* Search

[Doran 67], [Hart, Nilsson & Raphael 68]

Similar to Dijkstra's algorithm but:

- Domain-specific estimates $\pi_t(v)$ on dist(v, t) (potentials).
- At each step pick a labeled vertex with the minimum $k(v) = d_s(v) + \pi_t(v)$.
  Best estimate of path length through v.
- In general, optimality is not guaranteed.

Feasibility and Optimality

Potential transformation: Replace $\ell(v, w)$ by $\ell_{\pi_t}(v, w) = \ell(v, w) - \pi_t(v) + \pi_t(w)$ (reduced costs).

Fact: Problems defined by ℓ and ℓ_{π_t} are equivalent.

Definition: π_t is *feasible* if $\forall (v, w) \in A$, the reduced costs are nonnegative.
Estimates are "locally consistent:" $\pi_t(w) + \ell(v, w) \geq \pi_t(v)$.

Optimality: If π_t is feasible, the A* search is equivalent to Dijkstra's algorithm on transformed network, which has nonnegative arc lengths. A* search finds an optimal path.

Different order of vertex scans, different subgraph searched.

Fact: If π_t is feasible and $\pi_t(t) = 0$, then π_t gives lower bounds on distances to t.

Computing Lower Bounds

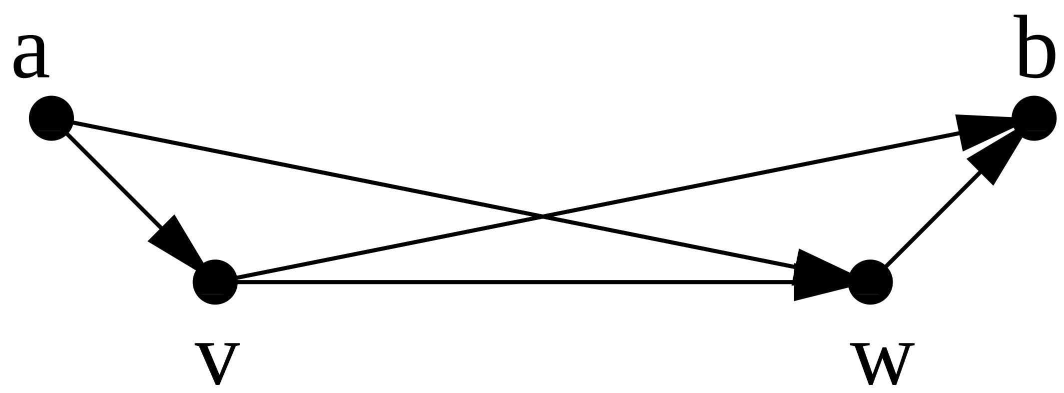
Euclidean bounds:

[folklore], [Pohl 71], [Sedgewick & Vitter 86].

For graph embedded in a metric space, use Euclidean distance.

Limited applicability, not very good for driving directions.

We use triangle inequality

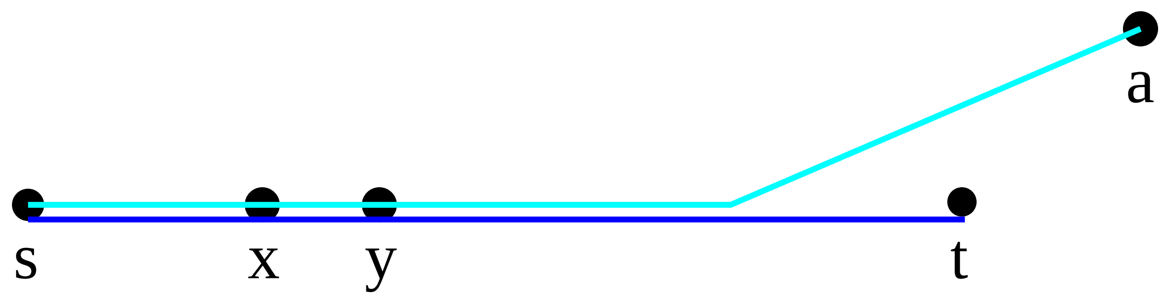


dist$(v, w) \geq$ dist$(v, b)-$dist(w, b); dist$(v, w) \geq$ dist$(a, w)-$dist(a, v).

Lower Bounds (cont.)

- Maximum of feasible potentials is feasible.
- Select landmarks (a small number).
- For all vertices, precompute distances to and from each landmark.
- For each s, t, use max of the corresponding lower bounds for $\pi_t(v)$.

Why this works well (when it does)



$$\ell_{\pi_t}(x, y) = 0$$

Bidirectional Lower-bounding

Forward reduced costs: $\ell_{\pi_t}(v,w) = \ell(v,w) - \pi_t(v) + \pi_t(w)$.

Reverse reduced costs: $\ell_{\pi_s}(v,w) = \ell(v,w) + \pi_s(v) - \pi_s(w)$.

What's the problem?

Bidirectional Lower-bounding

Forward reduced costs: $\ell_{\pi_t}(v,w) = \ell(v,w) - \pi_t(v) + \pi_t(w)$.

Reverse reduced costs: $\ell_{\pi_s}(v,w) = \ell(v,w) + \pi_s(v) - \pi_s(w)$.

Fact: π_t and π_s give the same reduced costs iff $\pi_s + \pi_t = \text{const}$.

[Ikeda et at. 94]: use $p_s(v) = \frac{\pi_s(v) - \pi_t(v)}{2}$ and $p_t(v) = -p_s(v)$.

Other solutions possible. Easy to lose correctness.

ALT algorithms use A^* search and landmark-based lower bounds.

Landmark Selection

Preprocessing

- Random selection is fast.
- Many heuristics find better landmarks.
- Local search can find a good subset of candidate landmarks.
- We use a heuristic with local search.

Preprocessing/query trade-off.

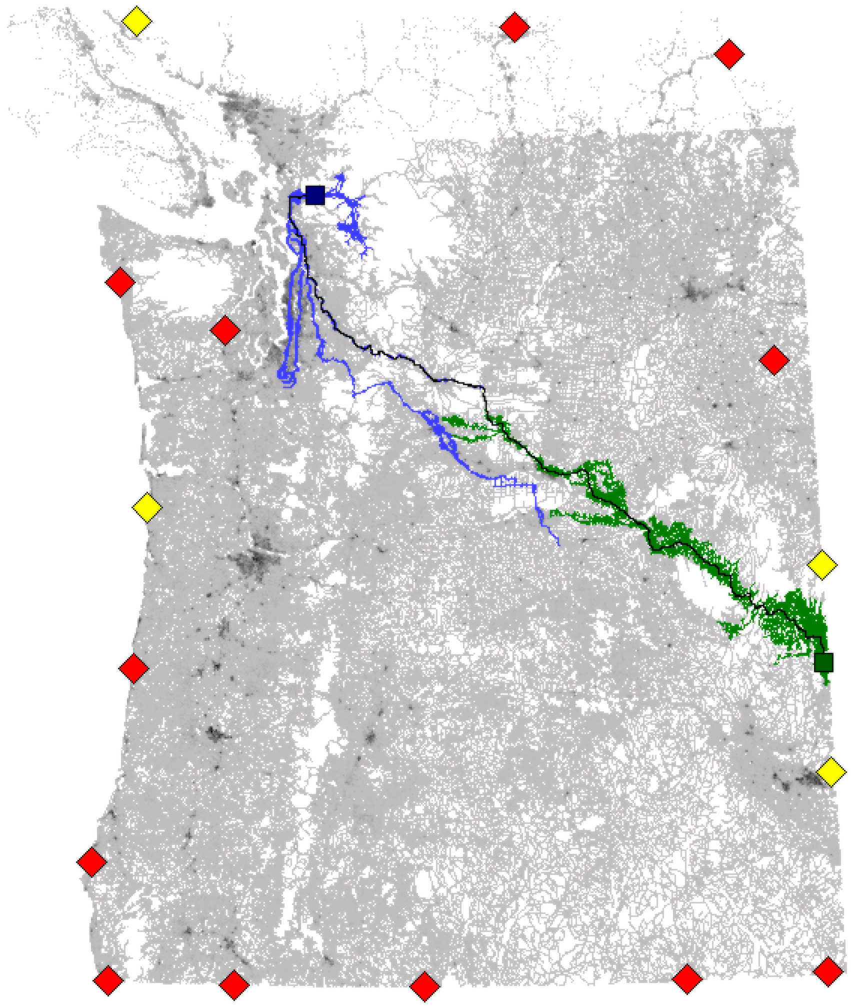
Query

- For a specific s, t pair, only some landmarks are useful.
- Use only active landmarks that give best bounds on dist(s, t).
- If needed, dynamically add active landmarks (good for the search frontier).
- Only three active landmarks on the average.

Allows using many landmarks with small time overhead.

Bidirectional ALT Example

ALT algorithm: A^* search with landmark bounds.

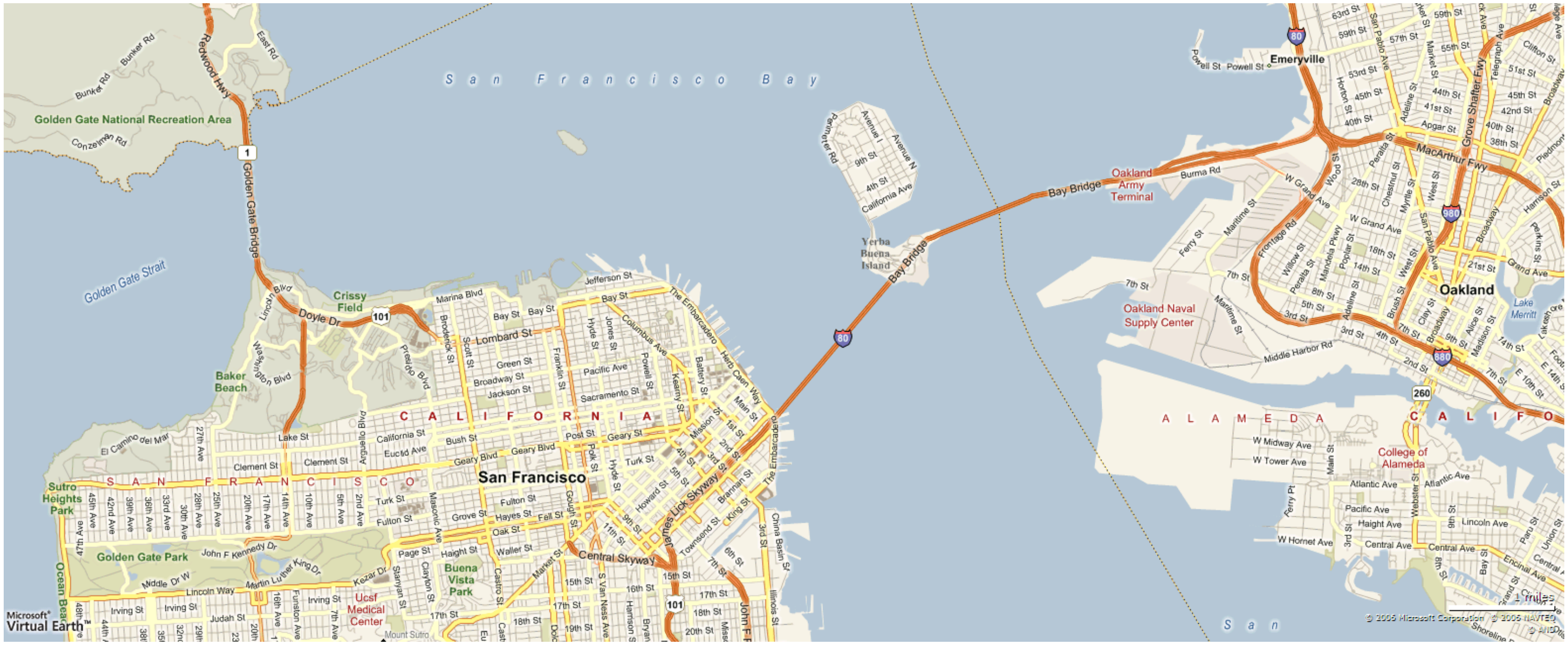


Experimental Results

Northwest (1.6M vertices), random queries, 16 landmarks.

method	preprocessing		query		
	minutes	MB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	28	518 723	1 197 607	340.74
ALT	4	132	16 276	150 389	12.05

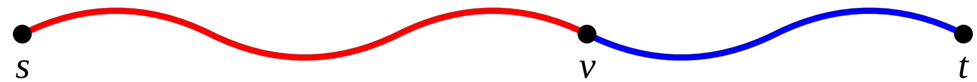
Reach Intuition



Identify local intersections and prune them when searching far from s and t.

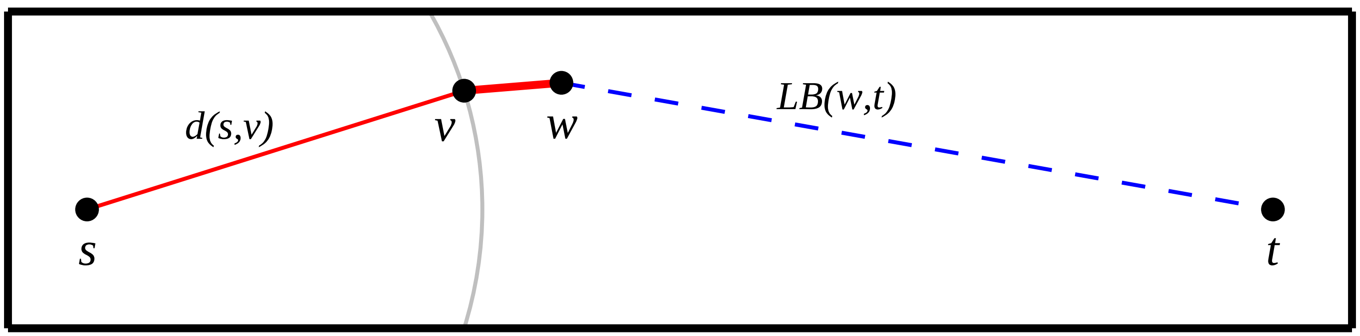
Reaches

[Gutman 04]



- Consider a vertex v that splits a path P into P_1 and P_2. $r_P(v) = \min(\ell(P_1), \ell(P_2))$.
- $r(v) = \max_P(r_P(v))$ over all shortest paths P through v.

Using reaches to prune Dijkstra:

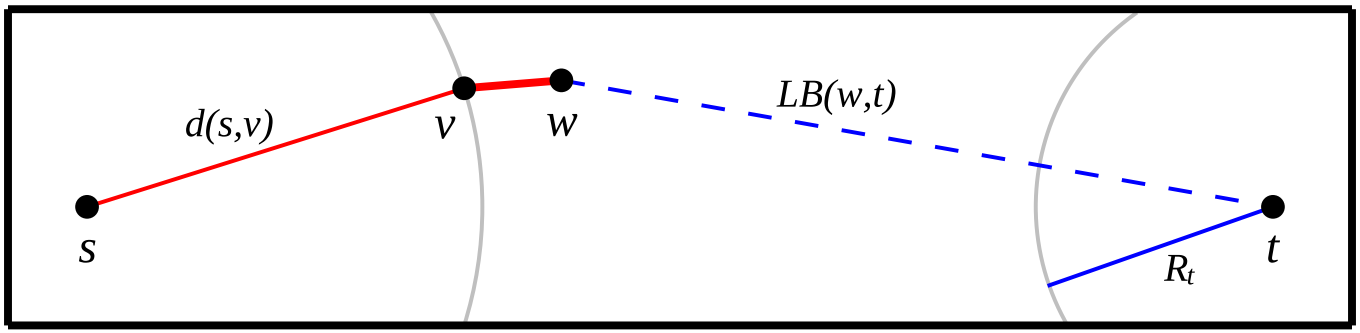


If $r(w) < \min(d(v) + \ell(v,w), LB(w,t))$ then prune w.

- Can efficiently compute and use reach upper bounds.
- Using shortcuts ("virtual express lanes") [Sanders & Schultes 06] is crucial.

Can use landmark lower bounds if available.

Bidirectional search gives implicit bounds (R_t below).



Reach-based query algorithm is Dijkstra's algorithm with pruning based on reaches. Given a lower-bound subroutine, a small change to Dijkstra's algorithm.

Computing Reaches

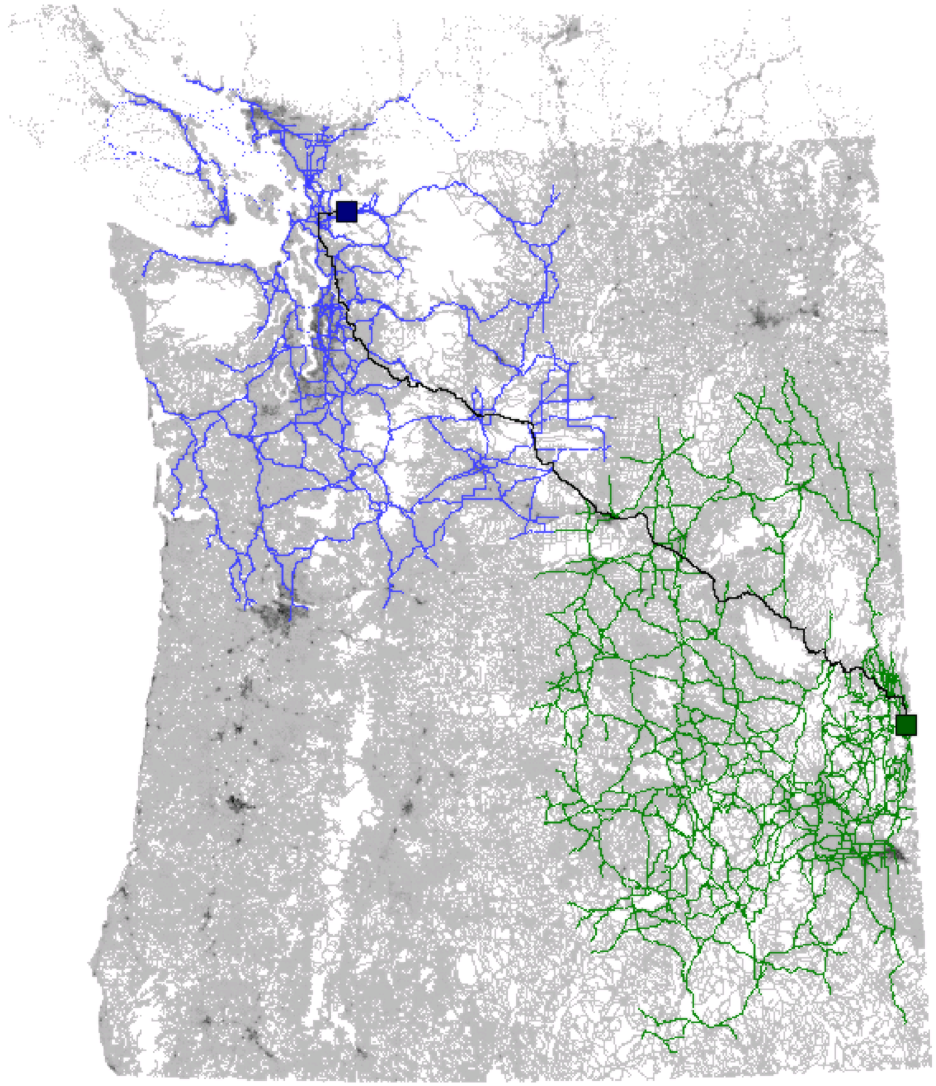
- A natural exact computation uses all-pairs shortest paths.
- Overnight for 0.3M vertex graph, years for 30M vertex graph.
- Have a heuristic improvement, but it is not fast enough.
- Can use reach upper bounds for query search pruning.

Iterative Approximation Algorithm: [Gutman 04]

- Use partial shortest path trees of depth $O(\epsilon)$ to bound reaches of vertices v with $r(v) < \epsilon$.
- Delete vertices with bounded reaches, add penalties.
- Increase ϵ and repeat.

Query time does not increase much; preprocessing faster but still not fast enough.

Reach Algorithm



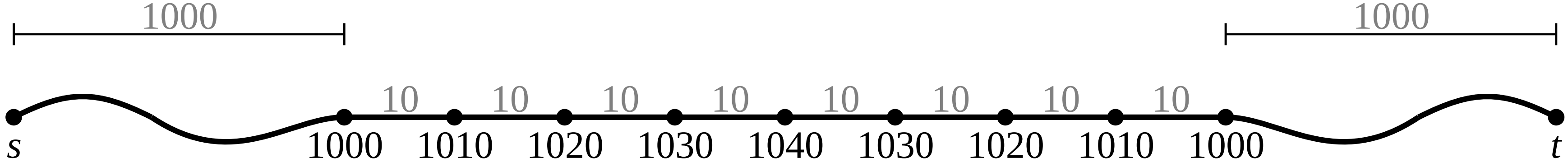
Experimental Results

Northwest (1.6M vertices), random queries, 16 landmarks.

method	preprocessing		query		
	minutes	MB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	28	518 723	1 197 607	340.74
ALT	4	132	16 276	150 389	12.05
Reach	1 100	34	53 888	106 288	30.61

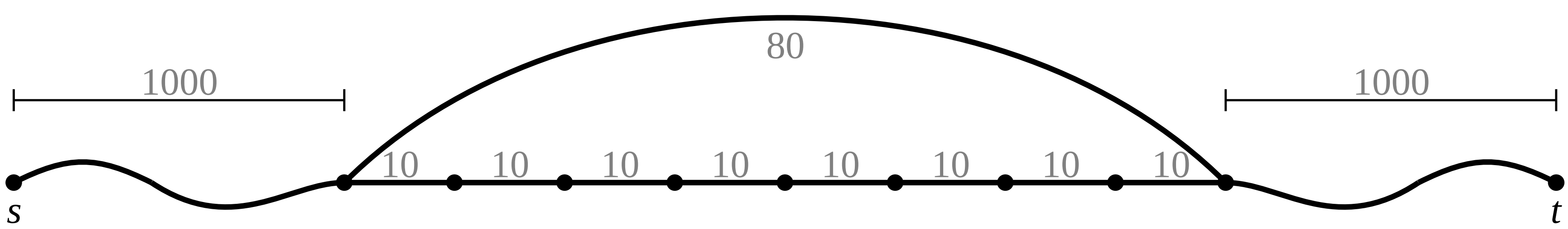
Shortcuts

- Consider the graph below.
- Many vertices have large reach.



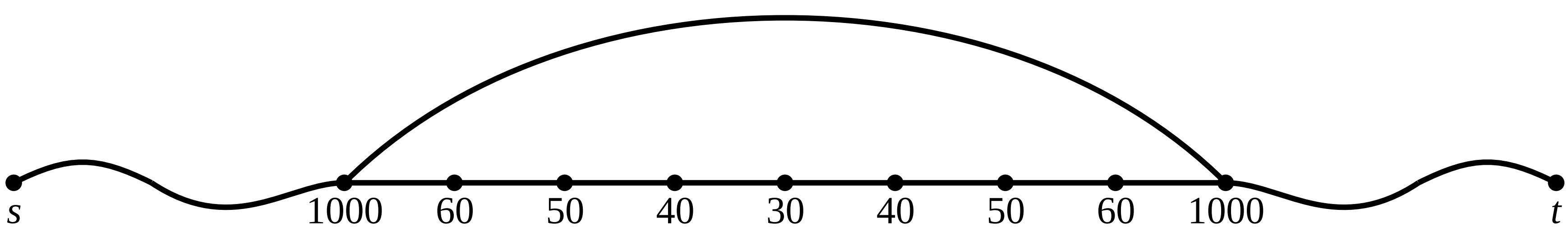
Shortcuts

- Consider the graph below.
- Many vertices have large reach.
- Add a shortcut arc, break ties by the number of hops.



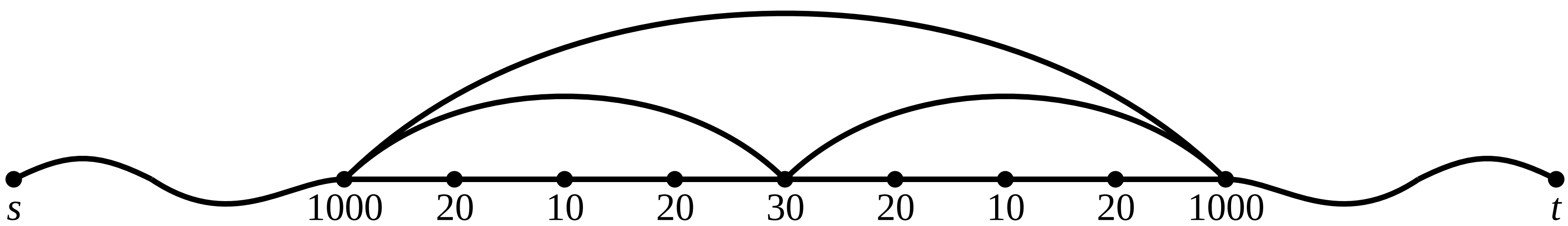
Shortcuts

- Consider the graph below.
- Many vertices have large reach.
- Add a shortcut arc, break ties by the number of hops.
- Reaches decrease.



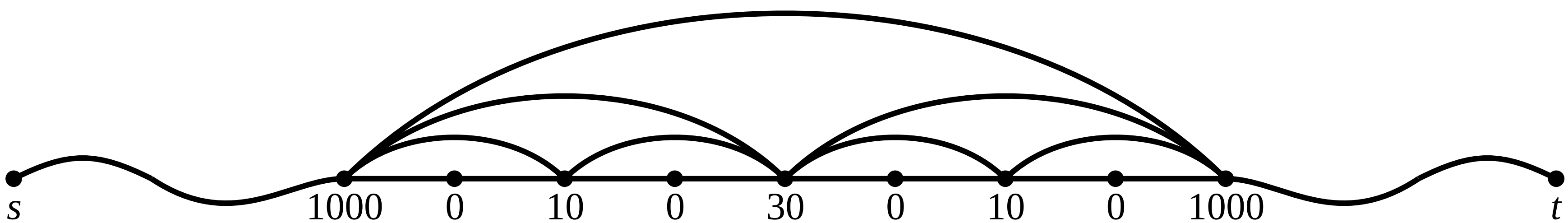
Shortcuts

- Consider the graph below.
- Many vertices have large reach.
- Add a shortcut arc, break ties by the number of hops.
- Reaches decrease.
- Repeat.



Shortcuts

- Consider the graph below.
- Many vertices have large reach.
- Add a shortcut arc, break ties by the number of hops.
- Reaches decrease.
- Repeat.
- A small number of shortcuts can greatly decrease many reaches.

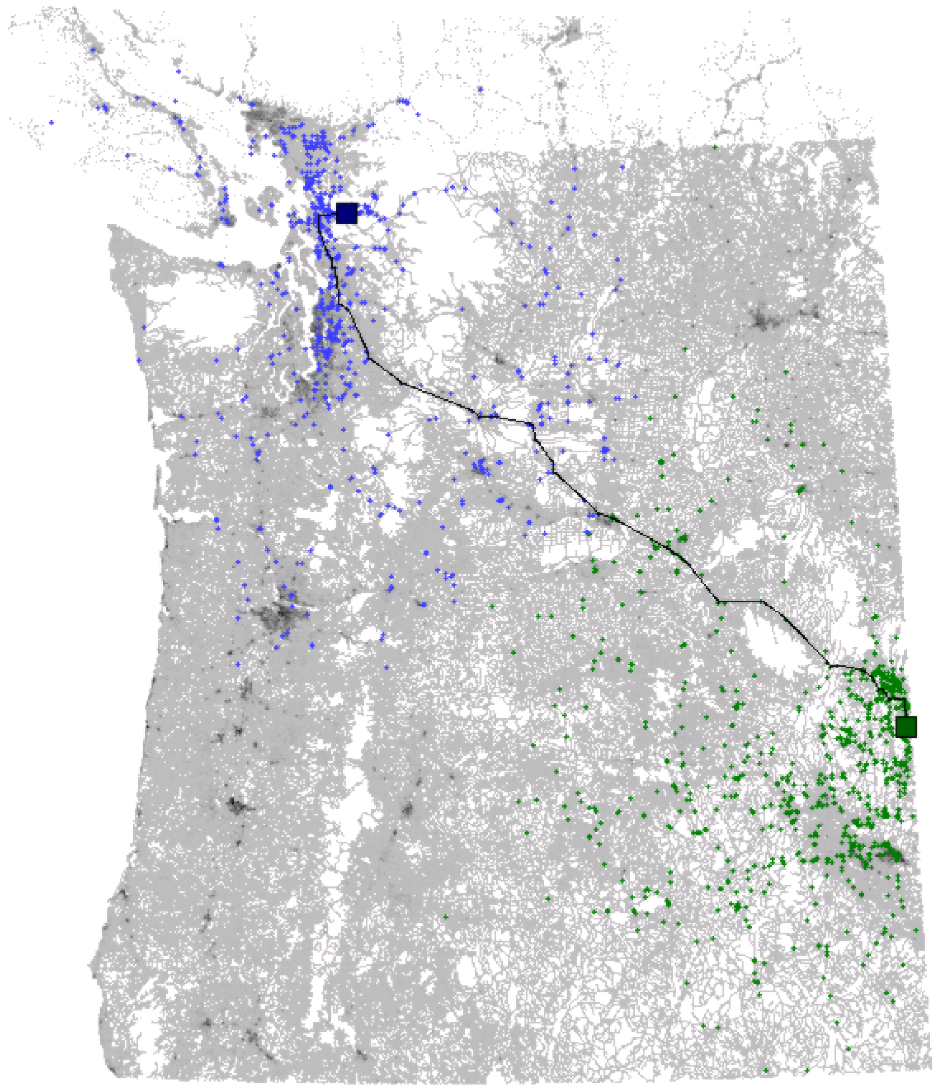


Shortcuts

[Sanders & Schultes 05, 06].

- During preprocessing we shortcut small (constant) degree vertices every time ϵ is updated.
- To shortcut, replace a vertex by a clique on its neighbors.
- The number of shortcut arcs is linear in n.
- Shortcuts greatly speed up preprocessing.
- Shortcuts speed up queries.
- Shortcuts require more space (extra arcs, auxiliary info.)

Reach with Shortcuts



Experimental Results

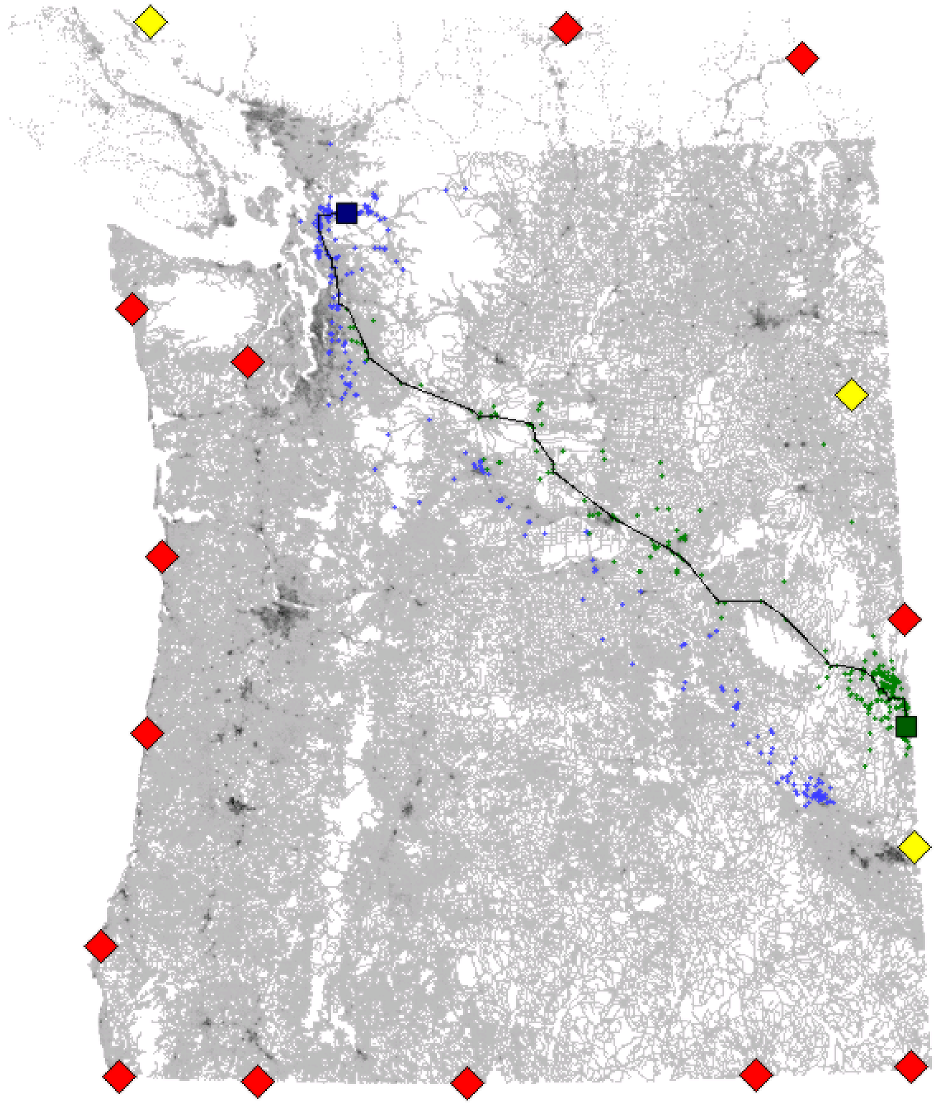
Northwest (1.6M vertices), random queries, 16 landmarks.

method	preprocessing		query		
	minutes	MB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	28	518 723	1 197 607	340.74
ALT	4	132	16 276	150 389	12.05
Reach	1 100	34	53 888	106 288	30.61
Reach+Short (RE)	17	100	2 804	5 877	2.39

Reaches and ALT

- ALT computes transformed and original distances.
- ALT can be combined with reach pruning.
- Careful: Implicit lower bounds do not work, but landmark lower bounds do.
- Shortcuts do not affect landmark distances and bounds.

Reach with Shortcuts and ALT



Experimental Results

Northwest (1.6M vertices), random queries, 16 landmarks.

method	preprocessing		query		
	minutes	MB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	28	518 723	1 197 607	340.74
ALT	4	132	16 276	150 389	12.05
Reach	1 100	34	53 888	106 288	30.61
Reach+Short (RE)	17	100	2 804	5 877	2.39
Reach+Short+ALT (REAL)	21	204	367	1 513	0.73

The North America Graph

North America (30M vertices), random queries, 16 landmarks.

method	preprocessing		query		
	hours	GB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	0.5	10 255 356	27 166 866	7 633.9
ALT	1.6	2.3	250 381	3 584 377	393.4
Reach	impractical				
Reach+Short (RE)	11.3	1.8	14 684	24 618	17.4
Reach+Short+ALT (REAL)	12.9	3.6	1 595	7 450	3.7

Further Improvements

Improved locality: sort by reach.

Reach-aware landmarks:

- Store landmark distances only for high-reach vertices (e.g., 5%).
- For low-reach vertices, use the closest high-reach vertex to compute lower bounds.
- Can use freed space for more landmarks, improve both space and time.

Practical even on the North America graph (30M vertices):

- $\approx$ 1ms. query time on a server.
- $\approx$ 6sec. query time on a Pocket PC with 4GB flash card.
- Better for local queries.

Reach-Aware Landmarks

- Most time is spend searching high-reach vertices.
- To save space, maintain landmark distances only for high-reach vertices.
- For a low-reach vertex, use a nearby proxy high-reach vertex to compute distance bounds.
- Trade efficiency for space.
- Use more landmarks to improve efficiency.

REAL(i, j): i landmarks, distances maintained for $\frac{n}{j}$ vertices.

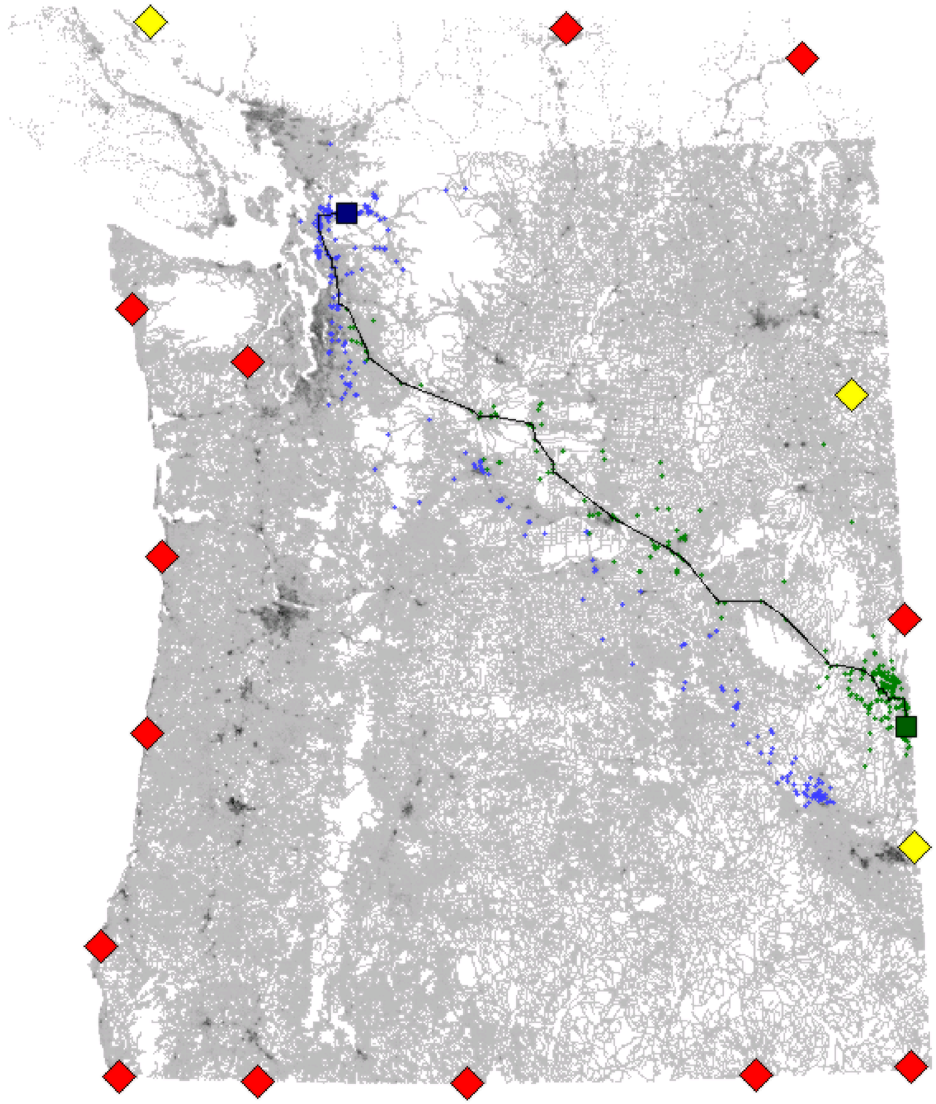
Random Queries – USA Graph

metric	method	prep. tm (min)	disk sp (MB)	query avg sc.	query max sc.	query time (ms)
time	ALT(16)	18.6	2563	187968	2183718	400.51
	RE	44.3	890	2317	4735	1.81
	REAL(16,1)	63.9	3028	675	3011	1.14
	REAL(64,16)	121.0	1575	540	1937	1.05
distance	ALT(16)	14.5	2417	276195	2910133	530.35
	RE	70.8	928	7104	13706	5.97
	REAL(16,1)	87.8	2932	892	4894	1.80
	REAL(64,16)	138.1	1585	628	4076	1.48
unit	ALT(16)	14.2	1992	240801	3922923	414.06
	RE	82.7	821	3455	6849	2.52
	REAL(16,1)	99.5	2277	847	2684	1.31
	REAL(64,16)	147.0	1270	617	2484	1.16

Random Queries – Europe Graph

metric	method	prep. tm (min)	disk sp (MB)	query avg sc.	query max sc.	query time (ms)
time	ALT(16)	13.2	1597	82348	993015	160.34
	re	82.7	626	4643	8989	3.47
	real(16,1)	96.8	1849	814	4709	1.22
	real(64,16)	140.8	1015	679	2955	1.11
distance	ALT(16)	10.1	1622	240750	3306755	430.02
	re	49.3	664	7045	12958	5.53
	real(16,1)	60.3	1913	882	5973	1.52
	real(64,16)	89.8	1066	583	2774	1.16
unit	ALT(16)	11.5	1488	140291	2137518	247.79
	re	184.9	579	4312	11198	2.95
	real(16,1)	196.5	1674	1097	5025	1.38
	real(64,16)	229.4	917	756	4175	1.14

Demo



Concluding Remarks

- Recent progress: the DIMACS Challenge, [Bast et. al 06], [Sanders and Schultes 06].
- Preprocessing heuristics work well on road networks.
- How to select good shortcuts? (Road networks/grids.)
- For which classes of graphs do these techniques work?
- Need theoretical analysis for interesting graph classes.
- Interesting problems related to reach, e.g.
  - Is exact reach as hard as all-pairs shortest paths?
  - Constant-ratio upper bounds on reaches in $\tilde{O}(m)$ time.
- Dynamic graphs.