Reach for A^*: an Efficient Point-to-Point Shortest Path Algorithm

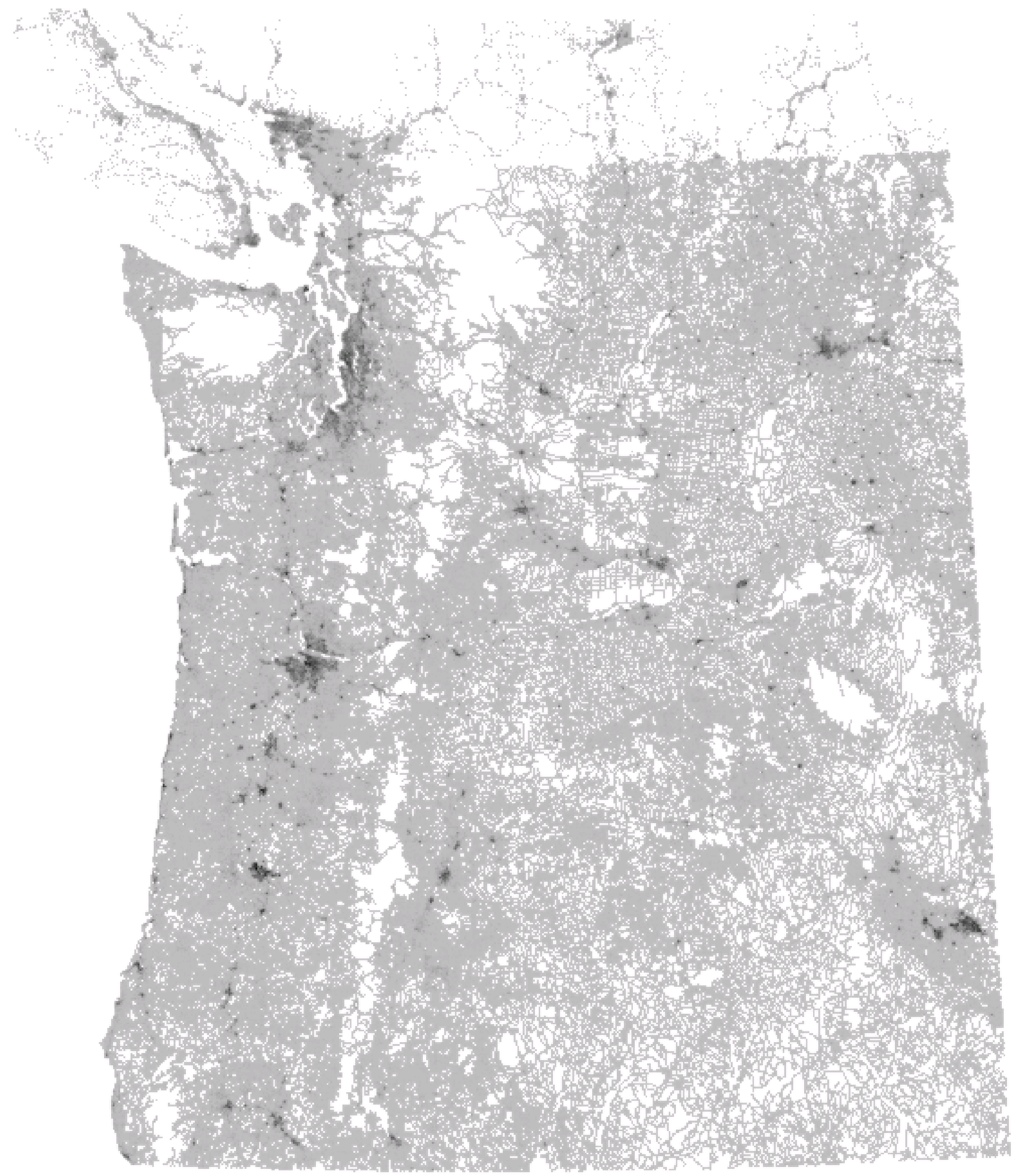
Andrew V. Goldberg
Microsoft Research
www.research.microsoft.com/∼goldberg/

Joint with Chris Harrelson, Haim Kaplan, Renato Werneck

Outline

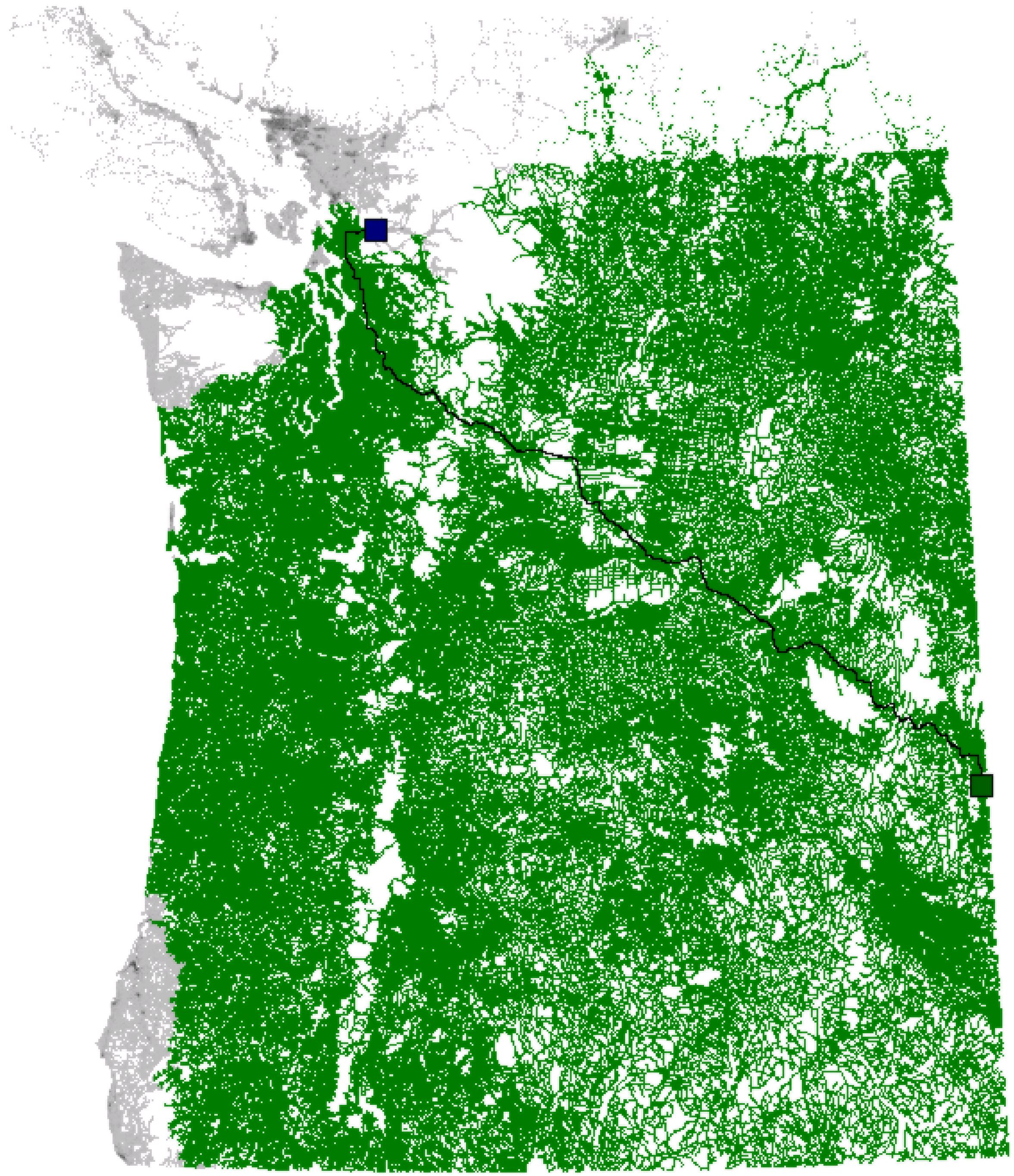
- Scanning method and Dijkstra's algorithm.
- Bidirectional Dijkstra's algorithm.
- A^* search.
- ALT Algorithm
- Definition of reach
- Reach-based algorithm
- Reach for A^*

Example Graph



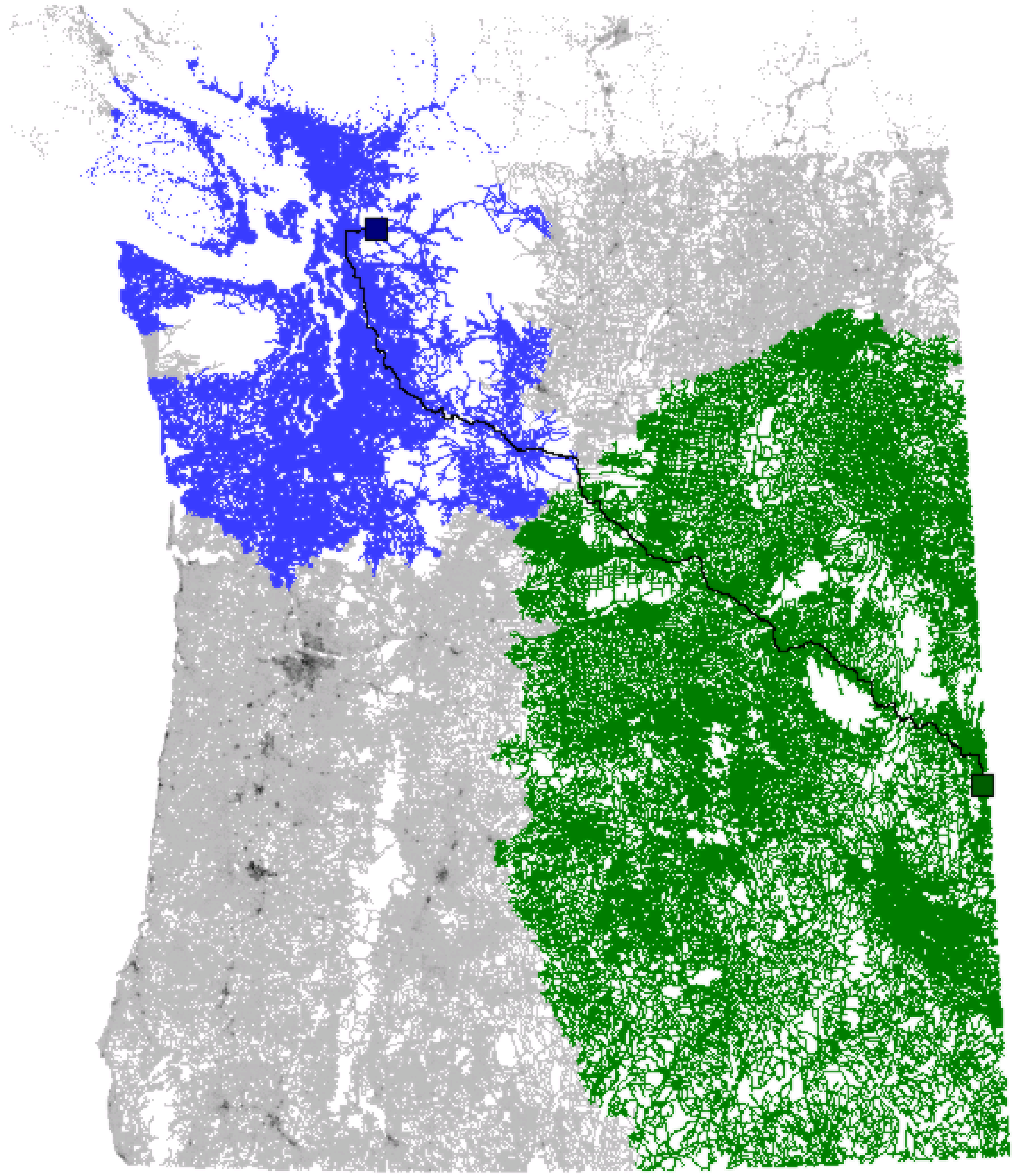
1.6M vertices, 3.8M arcs, travel time metric.

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 throgh v.
- In general, optimality is not guaranteed.

Computing Lower Bounds

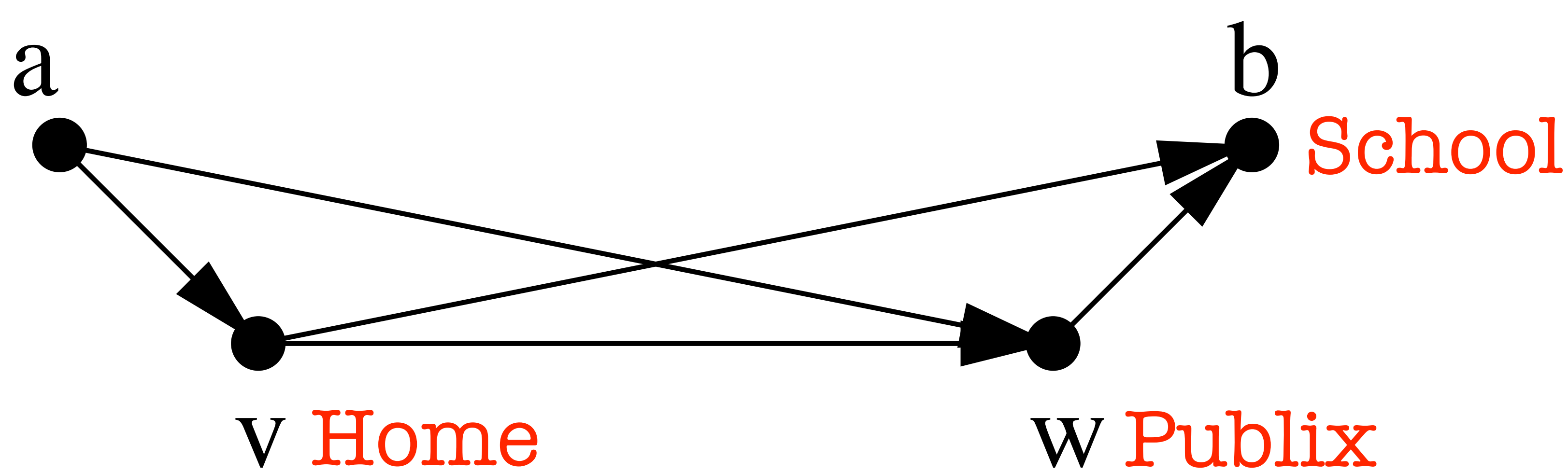
Euclidean bounds:

[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).

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.

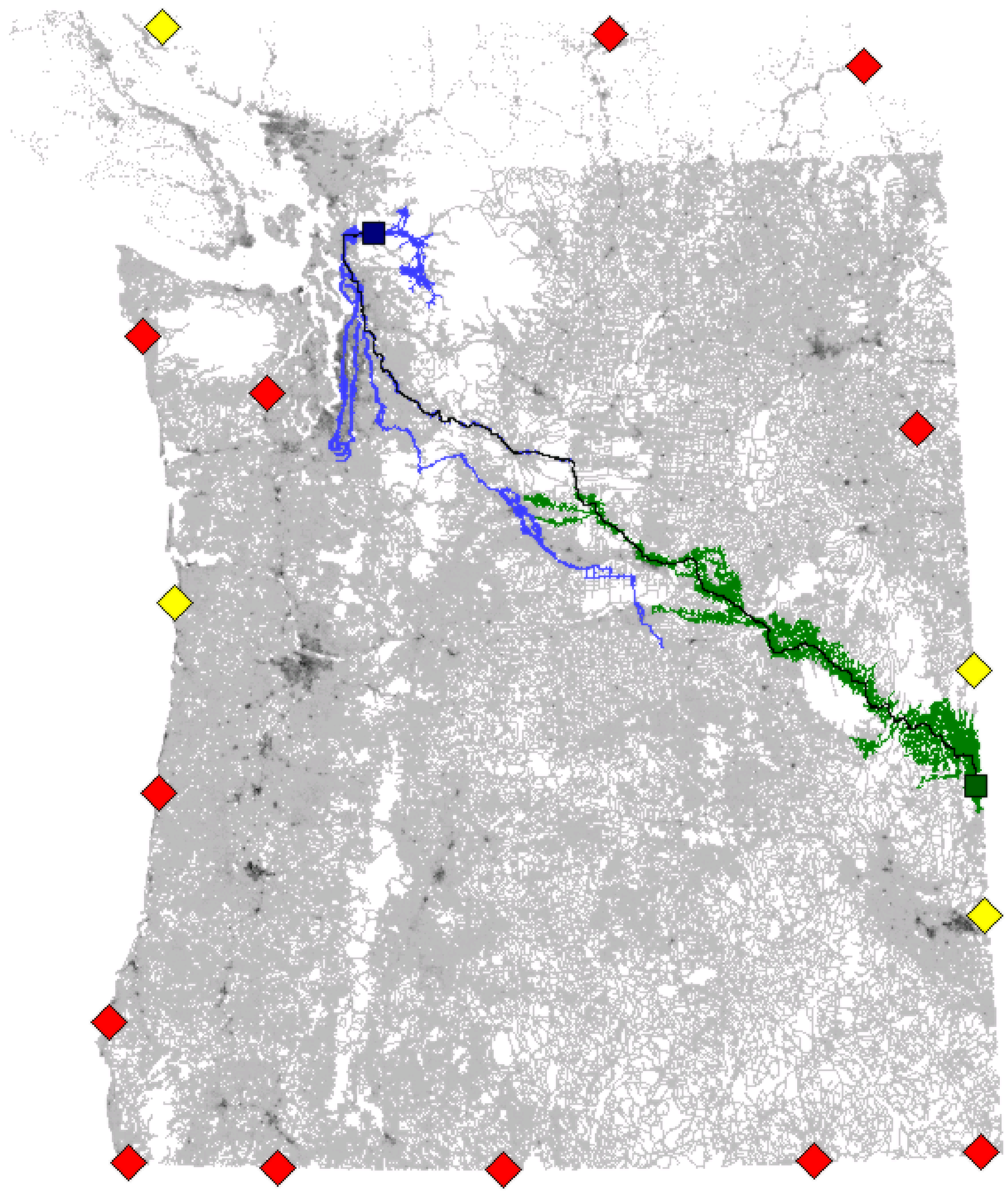
Preprocessing/query trade-off.

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).

Allows using many landmarks with small time overhead.

Bidirectional ALT Example



Experimental Results

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

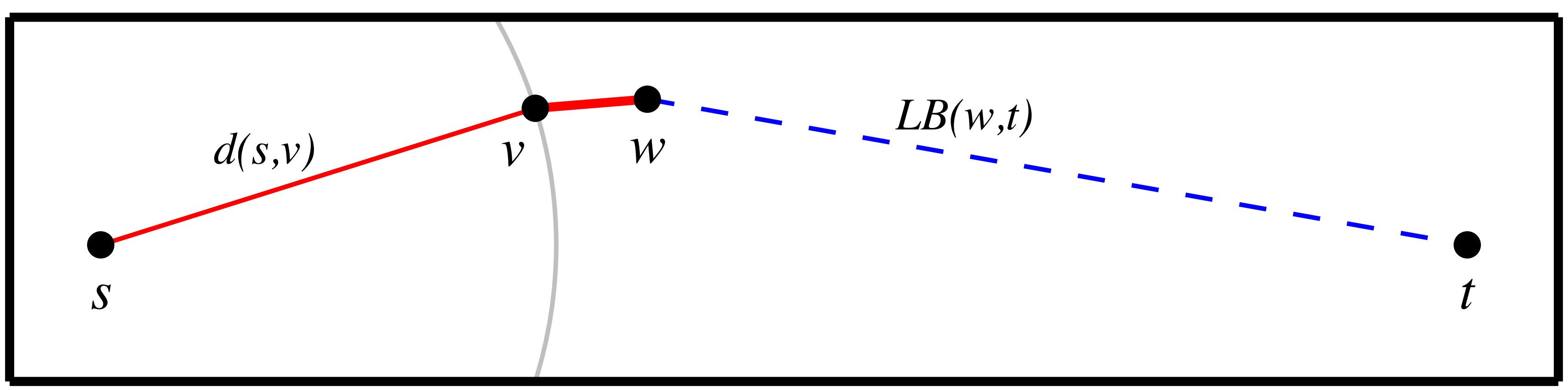
method	preprocessing		query		
	minutes	MB	avgscan	maxscan	ms
Bidirectional Dijkstra	—	28	518 723	1 197 607	340.74
ALT	4	132	16 276	150 389	12.05

Reaches

[Gutman 04]

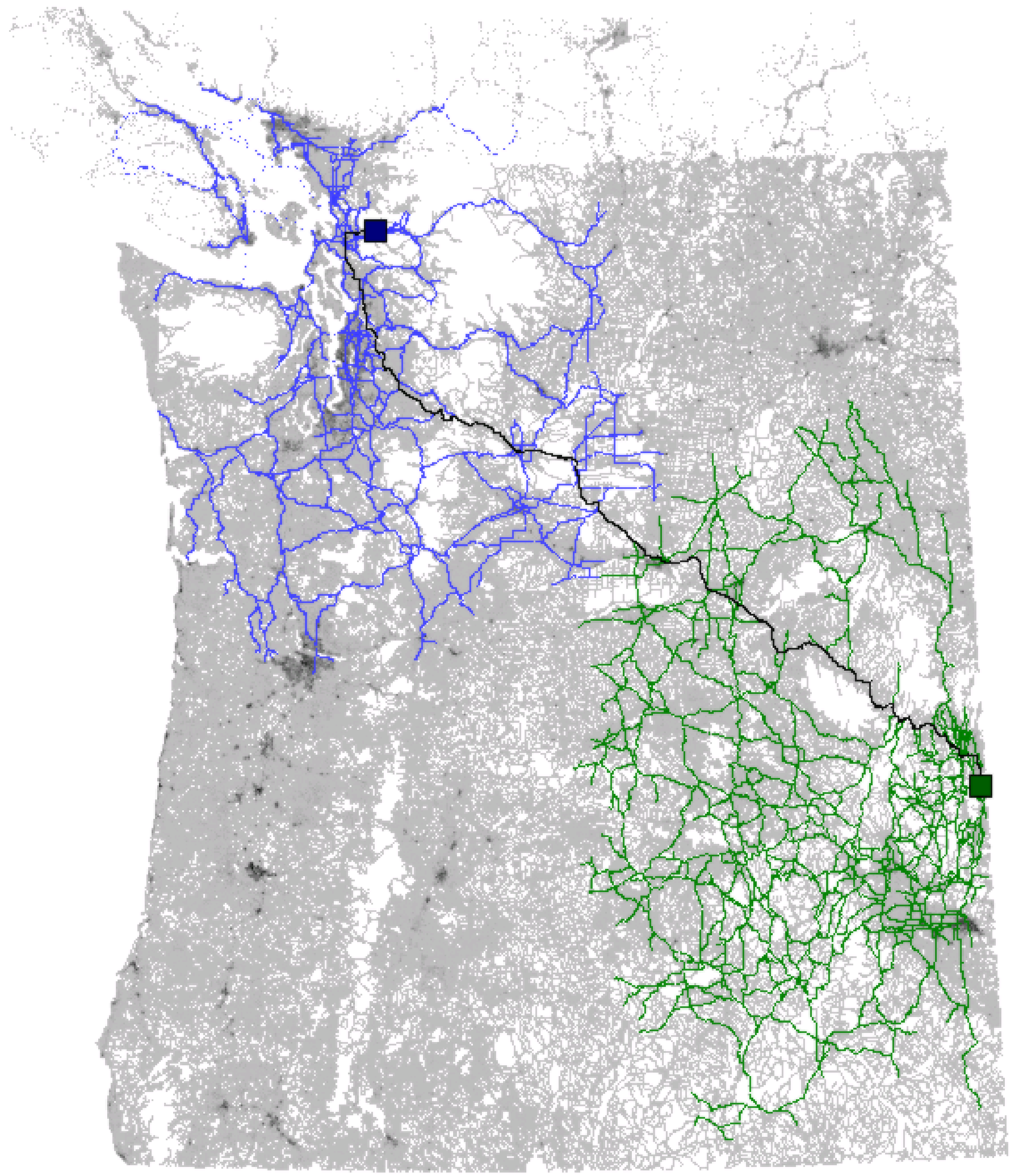
- Consider a vertex v that splits a path P into P_1 and P_2. $r_P(v) = \text{min}(\ell(P_1), \ell(P_2))$.
- $r(v) = \text{max}_P(r_P(v))$ over all shortest paths P through v.

Using reaches to prune Dijkstra:



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

Reach Algorithm



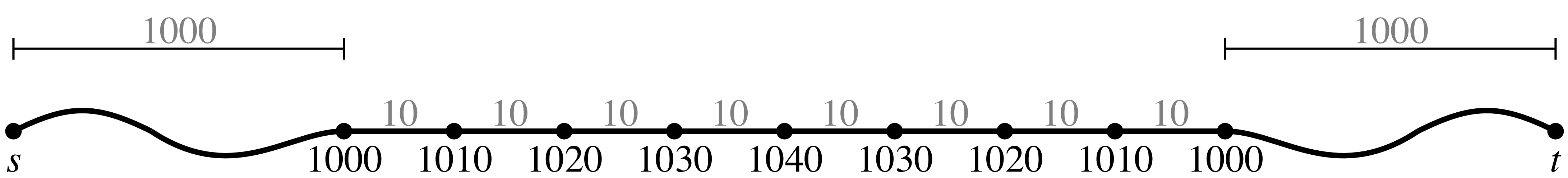
Experimental Results

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

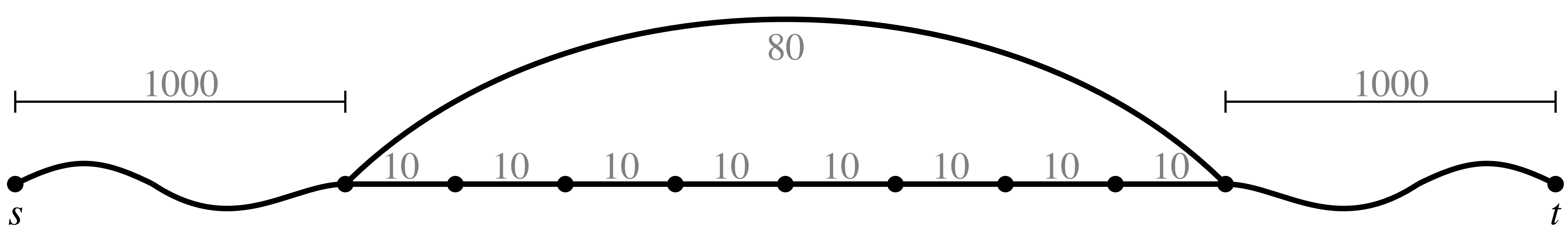
Shortcuts

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



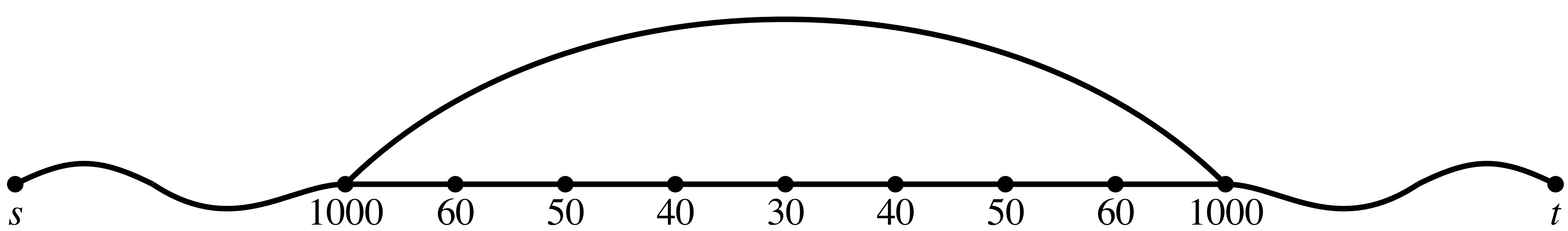
Shortcuts

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



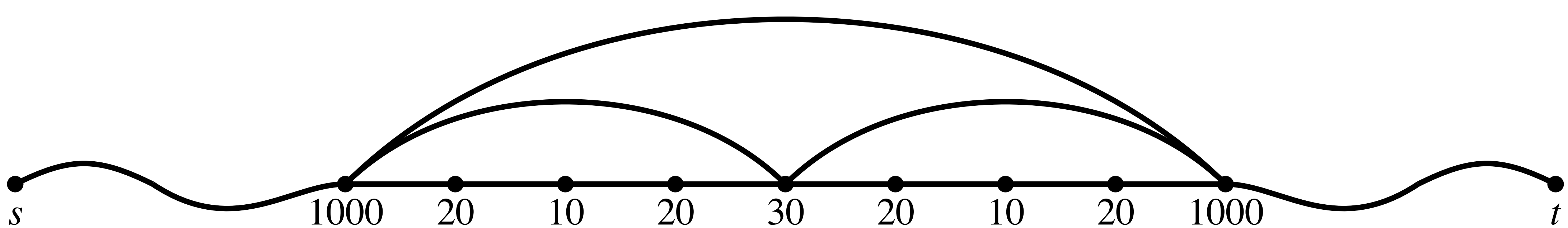
Shortcuts

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



Shortcuts

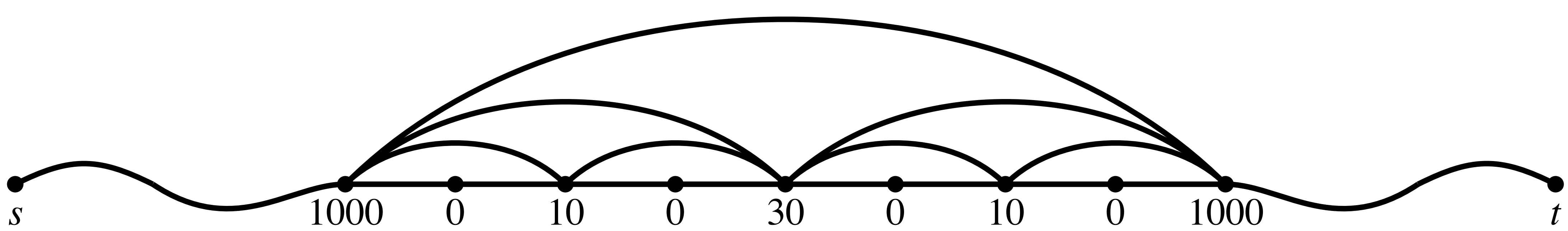
- Consider the graph below.
- Many vertices have large reach.
- Add a shortcut arc, break ties by the number of hops.
- Reaches decrease.
- Repeat.



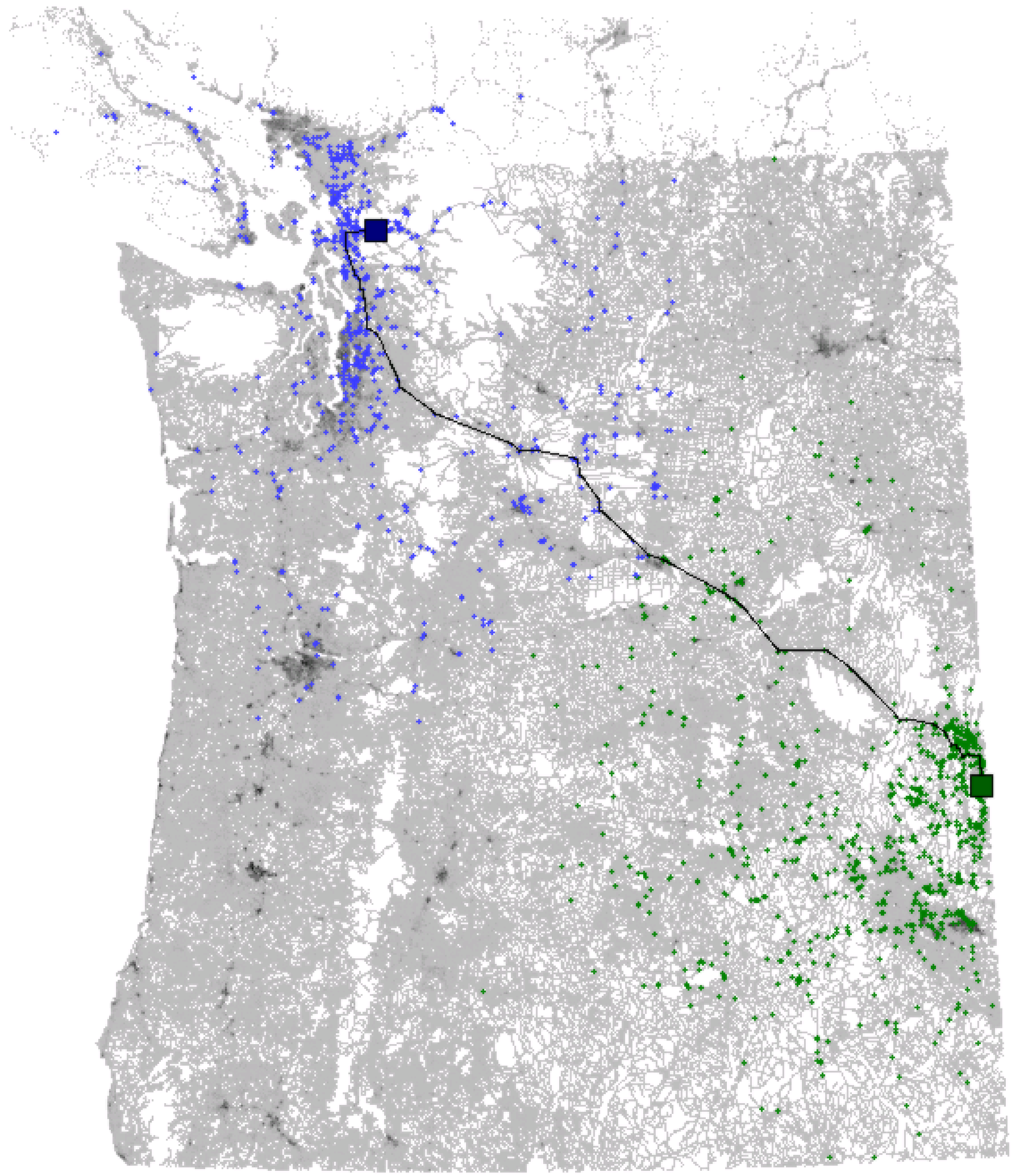
Wednesday, January 26, 2011

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.



Reach with Shortcuts

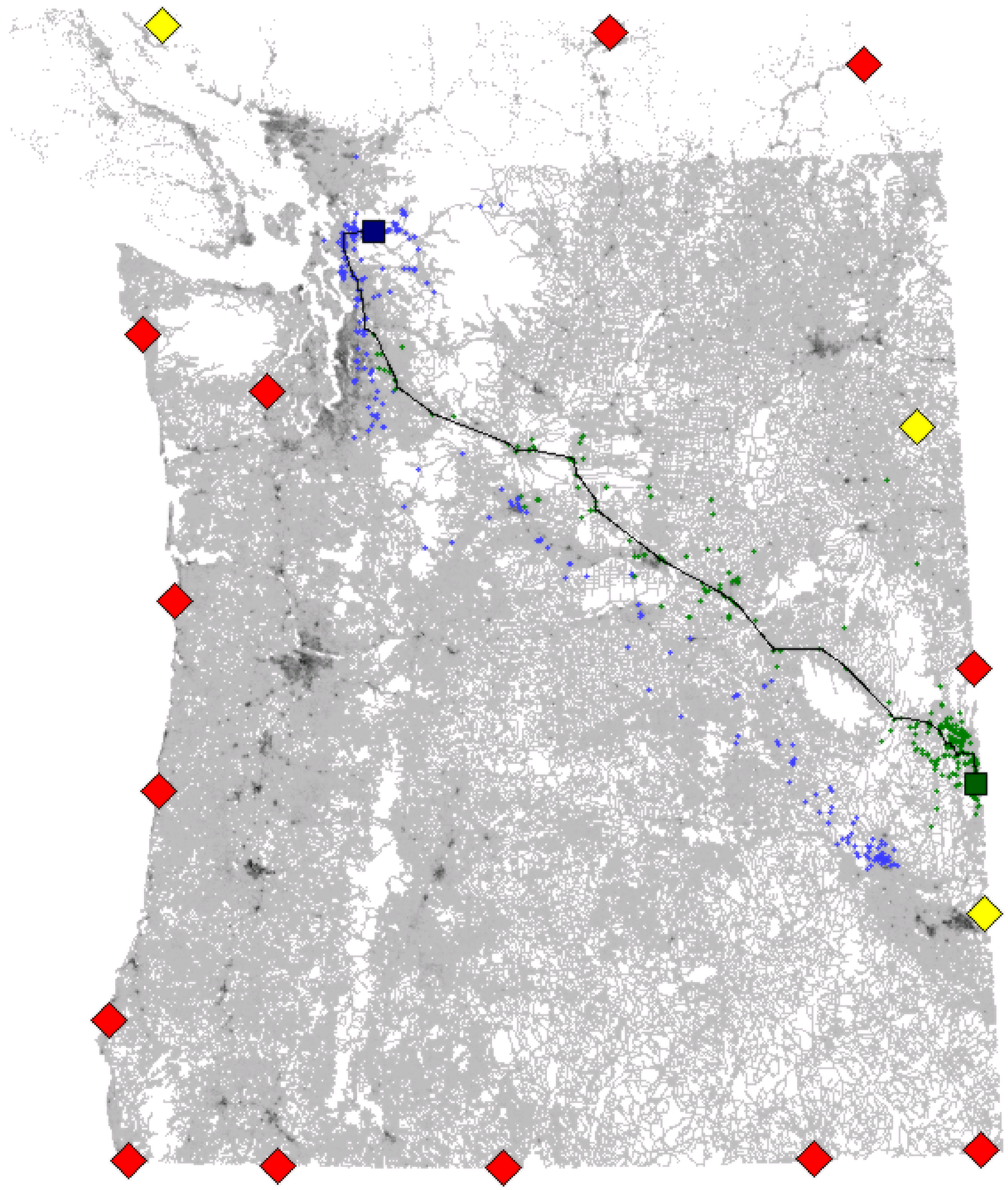


Experimental Results

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	17	100	2 804	5 877	2.39

Reach with Shortcuts and ALT



Wednesday, January 26, 2011

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	17	100	2 804	5 877	2.39
Reach+Short+ALT	21	204	367	1 513	0.73

The North America Graph

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

method	preprocessing hours	GB	query 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	11.3	1.8	14 684	24 618	17.4
Reach+Short+ALT	12.9	3.6	1 595	7 450	3.7

Concluding Remarks

- Our heuristics work well on road networks.
- Have improvements for query time and space requirements.
- 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.