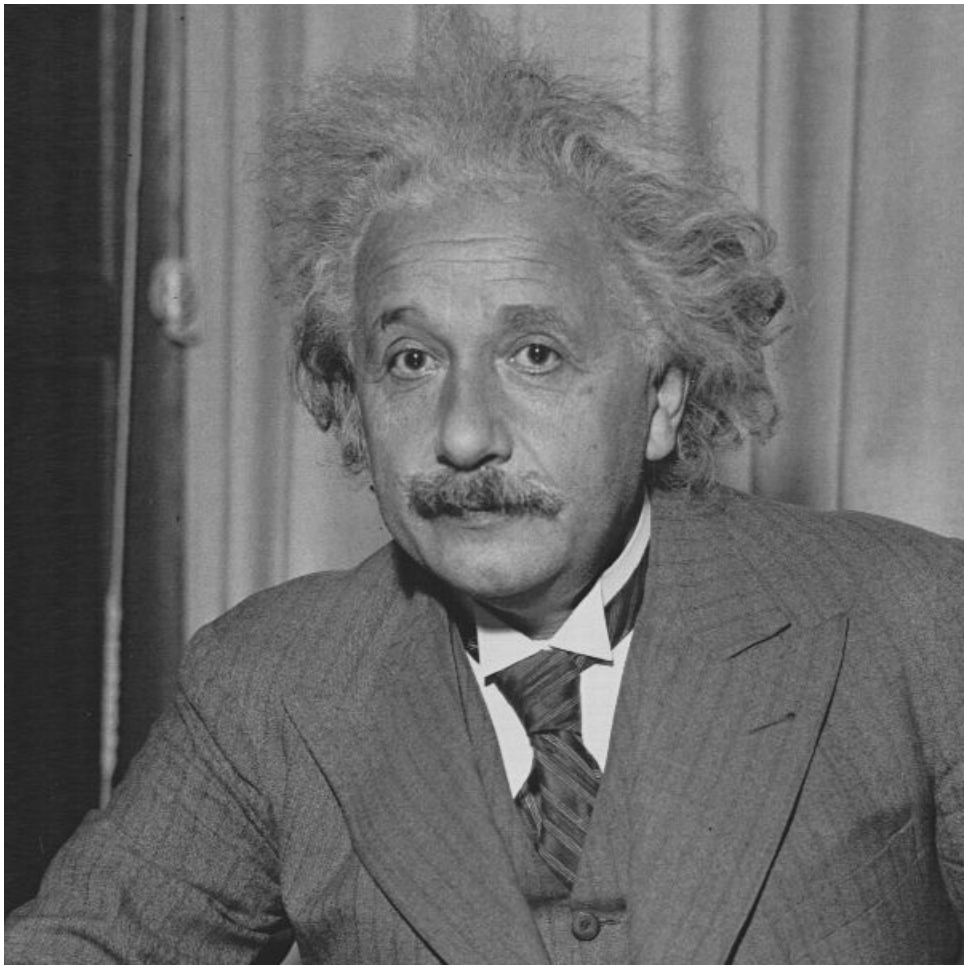
Basic Shortest Path Algorithms

DIKU Summer School on Shortest Paths

Andrew V. Goldberg
Microsoft Research
Silicon Valley

http://research.microsoft.com/users/goldberg

Einstein Quote



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

Shortest Path Problem

Variants

- Point to point, single source, all pairs.
- Nonnegative and arbitrary arc lengths.
- Integer lengths, word RAM model.
- Static, dynamic graphs, dynamic (arrival-dependent) lengths.
- Directed and undirected.

Unless mentioned otherwise, study directed graphs.

- Nonnegative len. undirected problem = symmetric directed.
- General undirected problem complicated *(matching)*.

SSSP Problem

Single-Source Shortest Paths (SSSP) problem:

Input: Digraph $G = (V, A)$, $\ell : A \to \mathbf{R}$, source $s \in V$.
Goal: Find shortest paths and distances from s to all vertices.

Special case: Nonnegative lengths (NSSSP).

W.l.g. assume all vertices reachable from s.
(In linear time can find unreachable vertices.)

One of the most fundamental problems:

- A point-to-point problem is no harder.
- n SSSP problems give all pairs problem.
  (In fact, one SSSP and $n - 1$ NSSSPs.)

Time Bounds

Bounds/currently best for some parameters

year	bound	due to	comments
1955	$O(n^4)$	Shimbel	$n = \|V\|$
1958	$O(nm)$	Bellman, Ford, Moore	$m = \|A\|$
1983	$O(n^{3/4}m \log U)$	Gabow	ℓ int. in $[-U, U]$
1989	$O(\sqrt{n}m \log(nU))$	Gabow & Tarjan	
1993	$O(\sqrt{n}m \log N)$	Goldberg	ℓ int. in $[-N, \infty)$
2005	$\tilde{O}(n^w U)$	Sankowski Yuster & Zwick	$w \approx 2.38$ (matrix mult. exp.)

Will cover $O(nm)$ and $O(\sqrt{n}m \log N)$ results.

Shortest path algorithms are 50 years old!

General Lengths: Outline

- Structural results.
- Scanning method.
- Negative cycle detection.
- Bellman-Ford-Moore (BFM) algorithm.
- Practical relatives of BFM.
- The scaling algorithm.

Definitions and Notation

- $G = (V, A)$, $n = |V|$, $m = |A|$, connected implies $n = O(m)$.
- $\ell : A \to \mathbf{R}$ is the length function.
  Sometimes integer, with range $[-U, U]$ or $[-N, \infty)$.
- dist(v, w) denotes distance from v to w.
  dist$_\ell$ if the length function is ambiguous.
- $d(v)$ is the potential of v.
- Reduced cost: $\ell_d(v, w) = \ell(v, w) + d(v) - d(w)$.
- SSSP is feasible iff the graph has no negative cycles.

Potential Transformation

Replace ℓ by ℓ_d.

Lemma (reduced cost of a path): For $P = (v_1, \ldots, v_k)$, $\ell_d(P) = \ell(P) + d(v_1) - d(v_k)$.

Proof: Recall $\ell_d(v_i, v_{i+1}) = \ell(v_i, v_{i+1}) + d(v_i) - d(v_{i+1})$.

Corollary: Cycle cost unchanged.

Corollary: For a fixed s, t pair, all s-t path lengths change by the same amount, $d(s) - d(t)$.

Equivalence Theorem: For any $d : V \to \mathbf{R}$, ℓ and ℓ_d define equivalent problems.

Feasibility Condition: The problem is feasible iff $\exists d : \forall (v, w) \in A,\ \ell_d(v, w) \geq 0$ (feasible d).

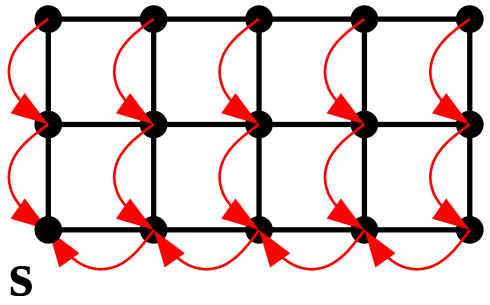
Proof: Only if: no negative cycles (for ℓ_d and thus for ℓ).

If: negative cycle implies no feasible d exists.

Shortest Path Tree

- Naive SSSP representation: $O(n^2)$ arcs.
- Tree representation: rooted at s, tree paths corresponds to shortest paths.
- Only $n - 1$ arcs.

A shortest path tree T of a graph (V_T, A_T) is represented by the parent pointers: $\pi(s) = \texttt{null}$, $(v, w) \in A_T$ iff $\pi(w) = v$.
Can "read" the shortest path in reverse.

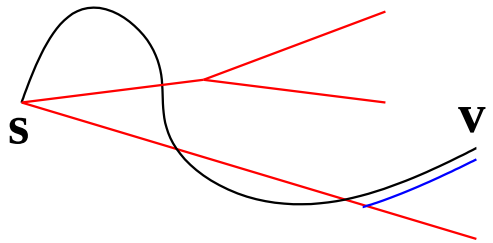


Shortest Path Tree Theorem

Subpath Lemma: A subpath of a shortest path is a shortest path.

SP Tree Theorem: If the problem is feasible, then there is a shortest path tree.

Proof: Grow T iteratively. Initially $T = (\{s\}, \emptyset)$. Let $v \in V - V_T$. Add to T the portion of the s-v shortest path from the last vertex in V_T on the path to v.



Correctness follows from the Subpath Lemma.

Zero Cycles

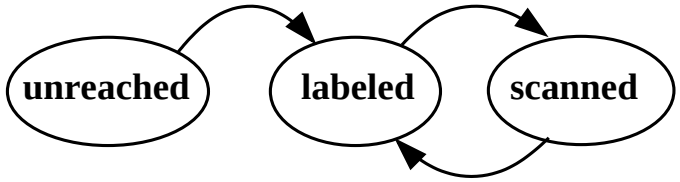
- Suppose G has a zero-length cycle U.
- Make all cycle arcs zero length by a potential transformation.
- Contract the cycle.
- Eliminate self-loops; may find a negative cycle.
- Solve the problem on contracted graph, extend solution to the full graph.

Contracting strongly connected components of zero-lengths arcs is an efficient way to contract all negative cycles.

Scanning Method

For every v maintain

- Potential $d(v)$: length of the best s-v path seen, initially ∞.
- Parent $\pi(v)$, initially `null`.
- Status $S(v)$, initially `unreached`.
- v is `labeled` after $d(v)$ decreases, `scanned` after a scan.



Scanning Method (cont.)

a-scan(v, w) $\{$
  if $(d(w) > d(v) + \ell(v, w))$
  then $\{$ $d(w) = d(v) + \ell(v, w)$; $\pi(w) = v$; $S(v) =$ `labeled`;$\}\}$

scan(v) $\{$
  $\forall (v, w) \in A$ do $\{$
    a-scan(v, w); $S(v) =$ `scanned`; $\}\}$

Intuition: try to extend a shorter path to v to the neighbors.

Startup: $d(s) = 0$; $S(s) =$ `labeled`;

Main loop: while $\exists$ labeled v pick and scan one;

Operation ordering unspecified!

Scanning Method: Termination

Path Lemma: $(d(v) < \infty) \Rightarrow \exists$ an s-v path of length $d(v)$.
Proof: Induction on the number of a-scan operations.

Simple Path Lemma: no negative cycles $\Rightarrow$ simple path.

Proof: Left as an exercise.

Termination Theorem: If there are no negative cycles, the method terminates.
Proof: Each time v becomes labeled, $d(v)$ decreases and we "use" a new simple s-v path. The number of simple paths is finite. Each scan operation makes a vertex `scanned`, which can happen finitely many times.

Scanning Method: Correctness

Lemma: Vertex distances are monotonically decreasing.

Negative Reduced Cost Lemma: $v = \pi(w) \Rightarrow \ell_d(v, w) \leq 0$.

Proof: Last time $\pi(w)$ set to v, $d(w) = d(v) + \ell(v, w)$. After that $d(w)$ unchanged, $d(v)$ nonincreasing.

Tree Lemma: If there are no negative cycles, then G_π is a tree on vertices v with $d(v) < \infty$ rooted in s.

Proof: Induction on the number of a-scans. Consider a-scan(v, w), note $d(v) < \infty$. Nontrivial case $d(w) < \infty$. If we create a cycle in G_π, then before the scan w was an ancestor of v in G_π. The w-v path in G_π has nonnegative reduced cost and $\ell_d(v, w) < 0 \Rightarrow$ negative cycle.

Correctness (cont.)

Lemma: $\ell_d(v,w) < 0 \Rightarrow v$ is labeled.
Proof: $d(v) < \infty$ so v is labeled or scanned. Immediately after scan(v), $\ell_d(v,w) \geq 0$. $d(v)$ must have decreased.

Correctness Theorem: If the method terminates, then $d(v)$'s are correct distances and G_π is a shortest path tree.
Proof: No labeled vertices implies $\ell_d(v,w) \geq 0 \quad \forall (v,w) \in E$. For (v,w) in G_π, we have $\ell_d(v,w) = 0$. Thus G_π is a shortest path tree. For the path P from s to v in G_π, $0 = \ell_d(P) = d(s) + \ell(P) - d(v)$. $d(s) = 0$ implies $d(v) = \ell(P)$.

Have termination and correctness if no negative cycles.

Negative Cycle Detection

Currency arbitrage.

Nontermination Lemma: If there is a negative cycle, the method does not terminate.
Proof: Negative with respect to ℓ_d for any d. Thus $\exists(v, w)$: $\ell_d(v, w) < 0$, and v is labeled.
Unbounded Distance Lemma: If there is a negative cycle, for some vertex v, $d(v)$ is unbounded from below.
Proof: Left as an exercise.

Lemma: If there is a negative cycle, then after some point G_π always has a cycle.
Proof: Let $-N$ be the most negative arc length. At some point, $d(v) < -N \cdot (n-1)$. Follow parent pointers from v. Either find a cycle or reach s and $d(s) = 0$. The latter impossible because the tree path lengths is at most $d(v)$.

Shortest Paths in DAGs

PERT application.

Linear time algorithm:

1. Topologically order vertices.
2. Scan in topological order.

Correctness: When scanning i, all of its predecessors have correct distances.
Running time: Linear.

Bellman-Ford-Moore Algorithm

The BFM algorithm processes labeled vertices in FIFO order.
Use a queue with constant time enqueue/dequeue operations.

Definition: Initialization is pass zero. Pass $i+1$ consists of processing vertices on the queue at the end of pass i.
Lemma: No negative cycles $\Rightarrow$ termination in less then n passes.

Proof: By induction: if a shortest path to v has k arcs, then after pass k, $d(v) = \text{dist}(s, v)$.

Theorem: No negative cycles $\Rightarrow$ BFM runs in $O(nm)$ time.
Proof: A pass takes $O(n+m)$ time as a vertex and an arc are examined at most once.

Remark: Can abort after $n-1$ passes and conclude that there is a negative cycle.

Heuristics

- BFM performs poorly in practice.
- Many heuristics with poor time bounds have been proposed.
- These perform well on some, but not all, problem classes.
- Robust algorithms always competitive with heuristics, better in the worst case.

Pape's Algorithm: Use `dequeue` Q. Remove vertices from the head. Add first-time labeled vertices to the tail, others to the head.

Exercise: Pape's algorithm exponential in the worst case.

Immediate Cycle Detection

Immediate cycle detection: stop the first time G_π is about to get a cycle; throughout G_π is a tree.
a-scan(v, w) creates a cycle in G_π iff w is an ancestor of v.
Naive implementation within BFM

- Walk to the root from v, stop if find w.
- Traverse the subtree rooted at w, stop if find v.
  Needs augmented tree data structure.

Both methods increase a-scan complexity to $O(n)$, and BFM complexity to $O(n^2m)$.

With no negative cycle, cycle-checking operations are wasteful and dominate the work.
Tarjan 1981: a beautiful use of amortization. Aimed at a "free" immediate cycle detection; later discovered to drastically improve practical performance.

Tarjan's Algorithm

Subtree disassembly: do subtree traversal; if v is not in the subtree, delete all vertices $u \neq w$ of the subtree from G_π, set $\pi(u) = \texttt{NULL}$ and $S(u) = \texttt{unreached}$.
Remark: This variation of the scanning method allows unreached vertices with finite d.

Correctness: Since $d(w)$ just decreased, removed vertices have $d(u) > \text{dist}(s, u)$, so they will become labeled again.

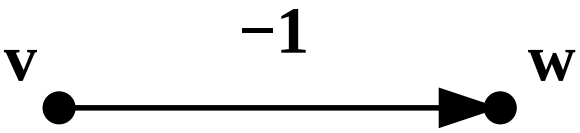
Analysis: Similar to BFM, except note that subtree disassembly work can be amortized over the construction of the subtree. $O(nm)$ time.

Practical improvement: Some labeled vertices u with $d(u) > \text{dist}(s, u)$ become unreached and not scanned; direct and (potentially bigger) indirect savings.

GOR Algorithm

[Goldberg & Radzik 93]

Idea: which scan first?



An admissible graph G_A is the graph induced by $(v, w) \in E$: $\ell_d(v, w) \leq 0$.

GOR algorithm works in passes.

L : the set of labeled vertices at the beginning of a pass.

1. If G_A has a negative cycle, stop. Contract zero cycles in G_A.
2. $\forall v \in L$, if $\forall (v, w) \in E$, $\ell_d(v, w) \geq 0$, $L = L - \{v\}$.
3. W: the set of vertices reachable from L in G_A.
4. Topologically order W and scan in topological order.

All vertices in L processed in a pass, $O(n)$ passes, $O(nm)$ time.

Build-in cycle detection, heuristic performance improvement.

Experiments/No Negative Cycles

Pallottino's algorithm PAL is more robust than Pape's

PALT is PAL with subtree disassembly.

SIMP is network simplex (ignore if unfamiliar).

	bfm	tarj	gor	simp	palt	pal
Grid-SSquare	⊗	◯	◯	◯	◯	◯
Grid-SSquare-S	⊗	◯	◯	◯	•	•
Grid-PHard	•	◯	◯	⊙	⊗	•
Grid-NHard	•	◯	◯	⊙	⊙	⊗
Rand-4	◯	◯	◯	◯	◯	◯
Rand-1:4	◯	◯	◯	◯	◯	◯
Acyc-Neg	•	•	◯	⊗	•	•

◯ means good, ⊙ means fair, ⊗ means poor, and • means bad.

GOR is $O(m)$ on acyclic graphs.

Square Grids

time (sec.) / scans per vertex
GOR scans include DFS scans.

nodes/arcs	bfm	tarj	gor	simp	palt	pal
4097	**0.03**	**0.02**	**0.02**	**0.02**	**0.01**	**0.02**
12288	2.74	1.35	2.26	1.34	1.25	1.25
16385	**0.21**	**0.09**	**0.08**	**0.10**	**0.05**	**0.04**
49152	5.05	1.43	2.29	1.42	1.26	1.26
65537	**1.92**	**0.39**	**0.37**	**0.45**	**0.29**	**0.22**
196608	9.66	1.48	2.28	1.46	1.27	1.27
262145	**19.30**	**1.69**	**1.94**	**1.93**	**1.20**	**0.97**
786432	19.68	1.52	2.29	1.50	1.27	1.27
1048577	**165.57**	**7.50**	**7.52**	**8.52**	**5.16**	**4.08**
3145728	41.78	1.57	2.30	1.54	1.27	1.27

Compare BFM to TARJ.

Square Grids/Artificial Source

Artificial source connects to s with a zero length arc, to all other vertices with very long arcs.
Ideally get an extra scan per vertex (two for GOR).

nodes/arcs	bfm	tarj	gor	simp	palt	pal
4098	**0.05**	**0.03**	**0.05**	**0.03**	**0.32**	**0.26**
16385	4.78	2.37	4.51	2.37	29.10	38.14
16386	**0.40**	**0.16**	**0.18**	**0.18**	**2.14**	**2.24**
65537	9.19	2.48	4.57	2.46	39.21	71.31
65538	**3.35**	**0.67**	**0.87**	**0.81**	**16.49**	**26.28**
262145	17.43	2.52	4.59	2.50	71.46	166.50
262146	**33.66**	**2.88**	**4.08**	**3.37**	**130.42**	**387.80**
1048577	34.12	2.56	4.62	2.54	124.35	489.18
1048578	**279.75**	**12.55**	**16.58**	**14.72**		
4194305	70.97	2.62	4.62	2.58		

Pallottino's (and Pape's) algorithm is not robust.

Experiments/Negative Cycles

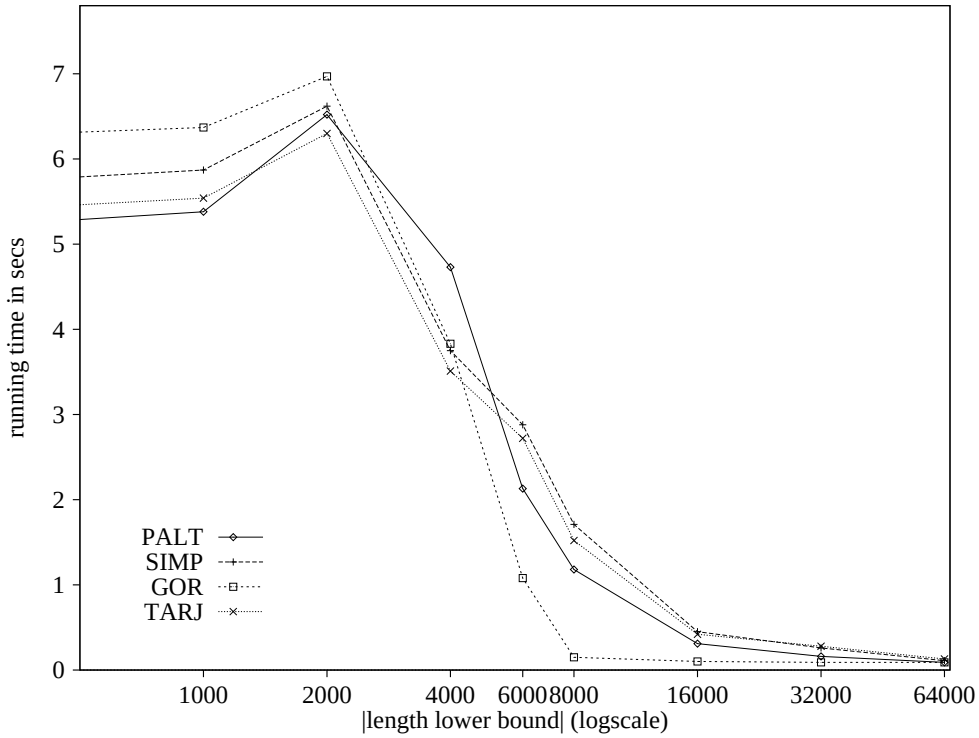
These are most robust algorithms; really bad ones excluded

	tarj	gorc	palt	simp
Rand-5	⊙	◯	⊙	⊙
SQNC01	◯	◯	◯	⊗
SQNC02	◯	◯	◯	⊙
SQNC03	◯	◯	◯	◯
SQNC04	◯	◯	◯	◯
SQNC05	◯	◯	◯	◯
PNC01	◯	◯	⊙	⊙
PNC02	◯	◯	⊙	⊙
PNC03	◯	◯	◯	◯
PNC04	◯	◯	◯	◯
PNC05	◯	◯	◯	◯

◯ means good, ⊙ means fair, and ⊗ means poor.

Table limited to better algorithms.

Random Graphs



Lengths in $[-L, 32,000]$, L changes.

Similar performance except around $L = 8,000$.

Experiments vs. Analysis

In theory, there is no difference between theory and practice.

- Both are important, complement each other.
- Worst-case analysis is too pessimistic, ignores system issues.
- Experimental analysis is incomplete, machine-dependent.
- Best implementations are robust, competitive on easy problems and do not get embarrassed on hard problems.
- 20% running time difference is not very important.
- Importance of machine-independent performance measures.

Scaling Algorithm

Finds a feasible potential function d in $O(\sqrt{n}m \log N)$ time.

Scaling loop:

- Integral costs $> -N$, $N = 2^L$ for integer $L \geq 1$.
- $\ell^i(v,w) = \left\lceil \frac{\ell(v,w)}{2^i} \right\rceil$.
- Note: $\ell^L \geq 0$, $\ell^0 = \ell$, no new negative cycles.
- Iteration i takes d feasible for ℓ^{L-i+1} and produces d feasible for ℓ^{L-i}.
- Terminate in L iterations.
- $\ell^{L-i+1}(v,w) + d(v) - d(w) \geq 0 \Rightarrow \ell^{L-i}(v,w) + 2d(v) - 2d(w) \geq -1$.
- Let $d = 2d$ and $c = \ell_d^{L-i}$, note $c \geq -1$.

Basic subproblem: Given integer $c \geq -1$, find a feasible potential function p.

Basic Subproblem Solution

An arc with a negative reduced cost is a bad arc. Each iteration does not create new bad arcs and reduces their number, k, by $\sqrt{k}$.

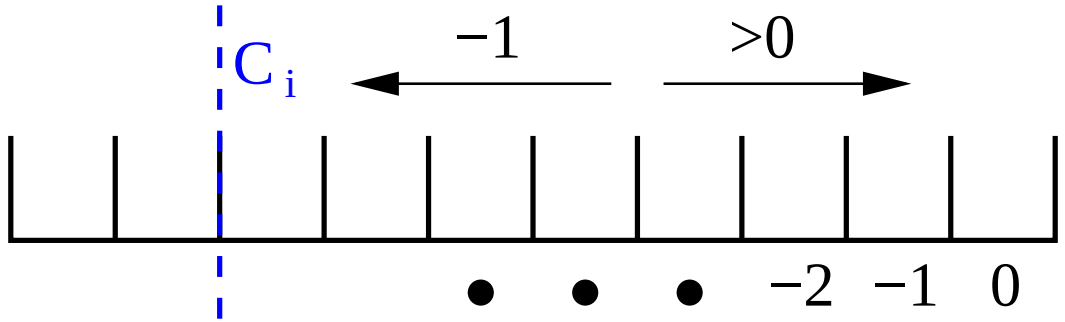
Admissible graph G_A is induced by arcs (v, w) with $c_p(v, w) \leq 0$.

Initialize $p = 0$.

Main loop:

1. DFS admissible graph. Stop if a negative cycle found. Contract zero cycles.
2. Add an artificial source, connect to all vertices by zero arcs.
3. Compute shortest paths in the resulting (acyclic) G'_A.
4. Case 1: $\exists$ a path P of length $\leq -\sqrt{k}$; `fixPath`(P).
   Case 2: $\exists$ a cut C with $\geq \sqrt{k}$ bad arcs crossing it; `fixCut`(C).

A Bucketing Argument



An efficient way to find P or C:

- Put vertices in buckets according to the distance d' in G'_A.
- If use $\geq \sqrt{k}$ buckets, s.p. tree yields P with $\sqrt{k}$ bad arcs.
- Otherwise consider cuts $C_i = \{v : d'(v) \geq -i\}$. The number of different non-trivial cuts $\leq \sqrt{k}$.
- For each i, no admissible arc enters C_i.
- Each bad arcs crosses at least one non-trivial cut.
- There is a cut C with $\sqrt{k}$ bad arcs exiting and no admissible arcs entering.

C or P can be found in $O(m)$ time.

Fixing Cuts

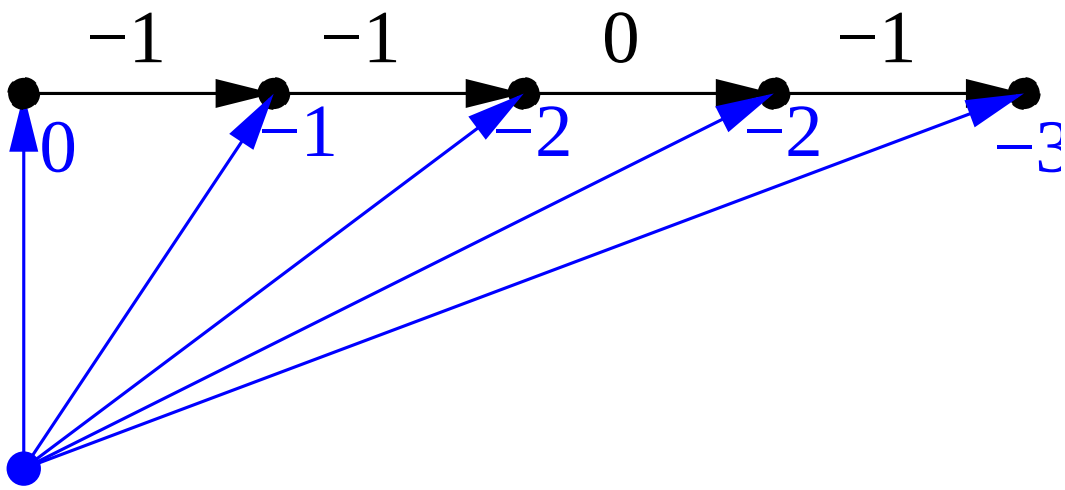
Case 1: C is the cut as above.
`fixCut`(C): $\forall v \in C, p(v) = p(v) + 1$.
Bad arcs out of C are fixed, no bad arcs are created.

Preliminaries for path-fixing.

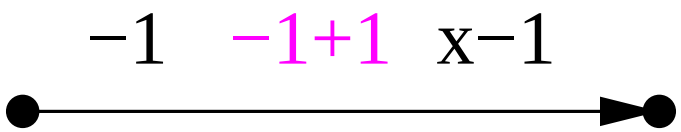
Dilworth Theorem: In a partial order on k elements, there is a chain or an antichain of cardinality $\sqrt{k}$.

Dial's algorithm finds s.p. in linear time if ℓ is integral and $\forall v \in V$, dist$(s, v) \leq n$.

Path Fixing Ingredients

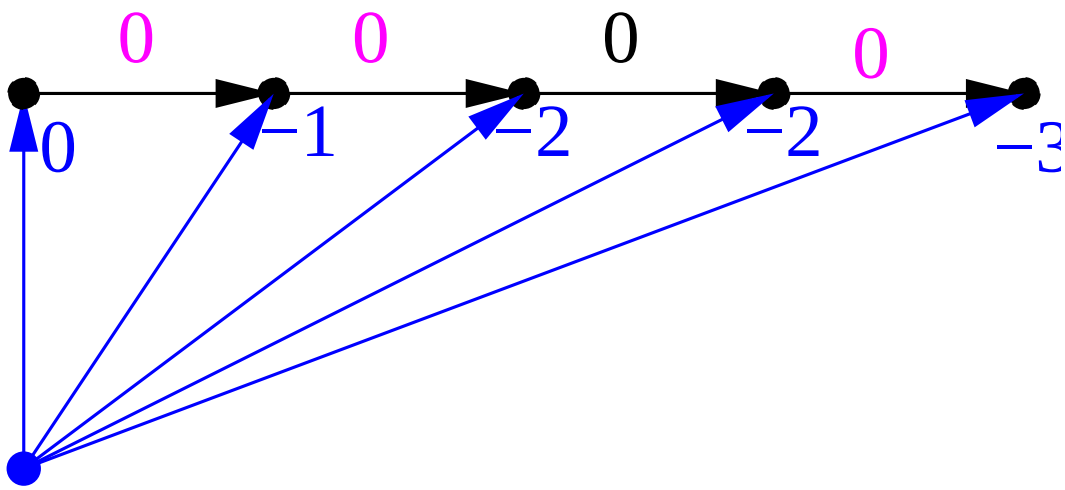


Shortest path arcs have zero reduced costs.

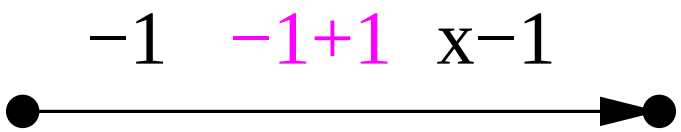


$x \geq 0$, $x - 1 \geq -1$, no new bad arcs.

Path Fixing Ingredients

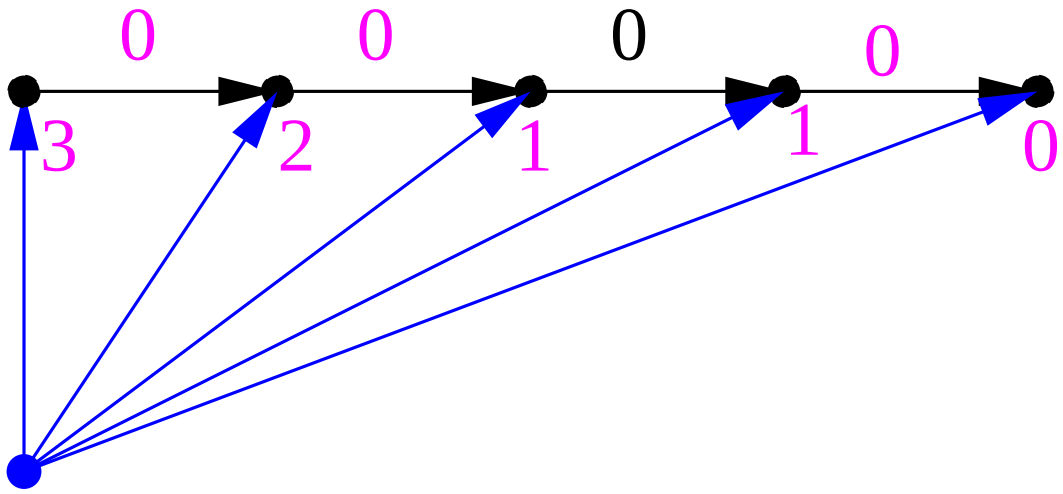


Shortest path arcs have zero reduced costs.

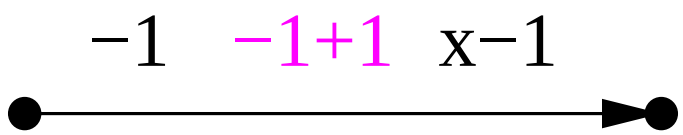


$x \geq 0$, $x - 1 \geq -1$, no new bad arcs.

Path Fixing Ingredients



Shortest path arcs have zero reduced costs.



$x \geq 0$, $x - 1 \geq -1$, no new bad arcs.

Fixing Paths

Case 2: P is the admissible path. Let α give distances on P.

1. Add s', arcs (s', v) with length $\alpha(v) + |\alpha(P)|$ if v on P and zero o.w.
   Add 1 to bad arc length.
2. Use Dial's algorithm to compute shortest path distances p.
3. Subtract 1 from bad arc lengths.
4. Replace c by c_p.

No new negative arcs are created, either fix all bad arcs on P or find a negative cycle.

Analysis and Correctness

Using Dial's Algorithm: All distances from s' are between 0 and $-\alpha(P) < n$. Dial's algorithm runs in linear time.

Lemma: `fixPath`(P) procedure does not create new bad arcs and either finds a negative cycle or fixes all bad arcs on P.

Proof: Left as an exercise.

Theorem: The scaling algorithm runs in $O(\sqrt{n}m \log N)$ time.

Proof: Enough to show $\sqrt{n}$ main loop iterations. $O(\sqrt{k})$ iterations reduce the number of bad arcs by a factor of two. The total is bounded by

$$\sum_{i=0}^{\infty} \sqrt{\frac{n}{2^i}} = \sqrt{n} \sum_{i=0}^{\infty} (\sqrt{2})^{-i} = O(\sqrt{n}).$$

General Lengths: Summary

SSSP problem with negative arcs:

- Structural results (s.p. trees, potentials, subpaths).
- Scanning method (correctness, termination, negative cycles).
- Bellman-Ford-Moore (BFM) algorithm.
- Negative cycle detection (walk to root, subtree disassembly).
- Practical relatives of BFM (Tarjan's and GOR algorithms).
- The scaling algorithm.

Nonnegative Lengths: Outline

- Dijkstra's algorithm and priority queues.
- Dial's algorithm, multilevel buckets, HOT queues.
- Expected linear-time algorithm.
- Experimental results.
- Point-to-Point shortest paths.
- Bidirectional Dijkstra algorithms.
- A^* Search.
- Use of landmarks.
- A demo.

Nonnegative Arc Lengths

$\ell \geq 0$ (NSSSP): a natural and important special case of SSSP.

[Dijkstra 59, Dantzig 63]
Minimum label selection rule: Pick labeled v with minimum $d(v)$.

Theorem: If $\ell \geq 0$, each vertex is scanned once.
Proof: After v is scanned with $d(v) = D$, all labeled vertices w have distance labels $d(w) \geq D$, and v never becomes labeled.

Vertices scanned in the order of distances from s, i.e., grow a ball of scanned vertices around s.

Naive time bound: $O(n^2)$.

Directed NSSSP Bounds

date	discoverer	bounds	note
1959	Dijkstra	$O(n^2)$	min. selection
1964	Williams	$O(m \log n)$	2-heap
1969	Dial	$O(m + nU)$	buckets
1977	Johnson	$O(m \log_{(2+m)/n} n)$	d-heap
1978	Dinitz	$O(m + n(U/delta))$	non-min. selection
1979	Denardo & Fox	$O(m + n\frac{\log U)}{\log\log U})$	MB
1987	Fredman & Tarjan	$O(m + n \log n)$	Fibonacci heap
1990	Ahuja et al.	$O(m + n\sqrt{\log U})$	MB+Fib. heap
1996	Thorup	$O(m \log\log n)$	new heap
1996	Raman	$O(m + n\sqrt{\log n})$	new heap
1997	Cherkassky et al.	$O(m + n(\log U)^{(1/3)+\epsilon})$	HOT q.
1997	Raman	$O(m + n(\log U \log\log U)^{1/4+\epsilon})$	HOT+impr. heap
2001	Meyer	$E(m)$ & $O(nm \log n)$	expected
2001	Goldberg	$E(m)$ & $O(m + n \log U)$	MB variant
2002	Han & Thorup	$E(m\sqrt{\log\log \min(n, U)})$	randomized
2004	Hagerup	$E(m)$& $O(m + n \log n)$	more distributions
2004	Thorup	$O(m + n \log\log \min(n, U))$	new heap

Bucket-based algorithms can work with real-valued lengths.

Use of Priority Queues

Priority queue operations: `insert`, `decreaseKey`, `extractMin` (also `create`, `empty`).

Examples: (with amortized operation times)

Binary heaps:[Williams 64] $O(\log n), O(\log n), O(\log n)$.

Fibonacci heaps:[Friedman & Tarjan 84] $O(1), O(1), O(\log n)$.

```
initialize as in the scanning method;
Q = create(); d[s] = 0; insert(s, Q);
while (!empty(Q) {
  v = Extract-Min(Q);
  for each arc (v,w) { // a-scan
    if (d[w] > d[v] + l(v,w)) {
      d[w] = d[v] + l(v,w);
      if (pi[w] = NULL) insert(w, Q) else decreaseKey(w, Q);
      pi[w] = v;
}}}
```

Heap-Based Time Bounds

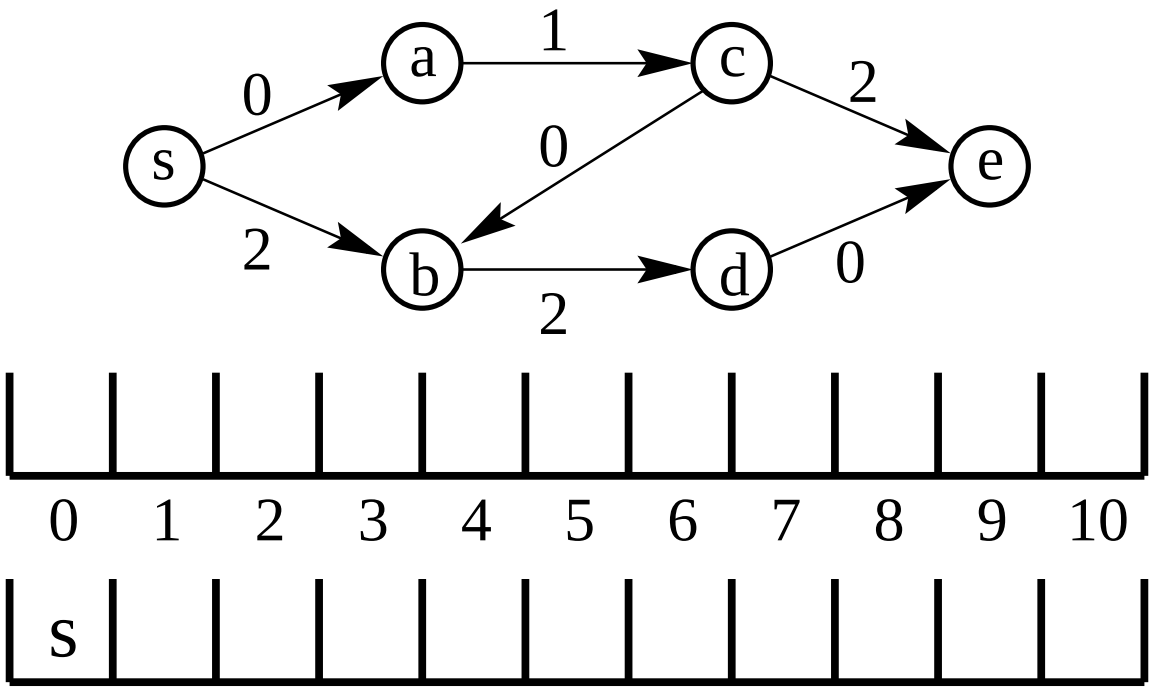
- Naive implementation: $O(n^2)$.
- Binary heaps: $O(m \log n)$. 4-heaps better in practice.
- Fibonacci heaps: $O(m + n \log n)$.
  Linear except for sparse graphs.

In practice, 4-heaps usually outperform Fibonacci heaps.

Monotone heaps: inserted elements no less than the last extracted one.
Sufficient for Dijkstra's algorithm. Better bounds known.

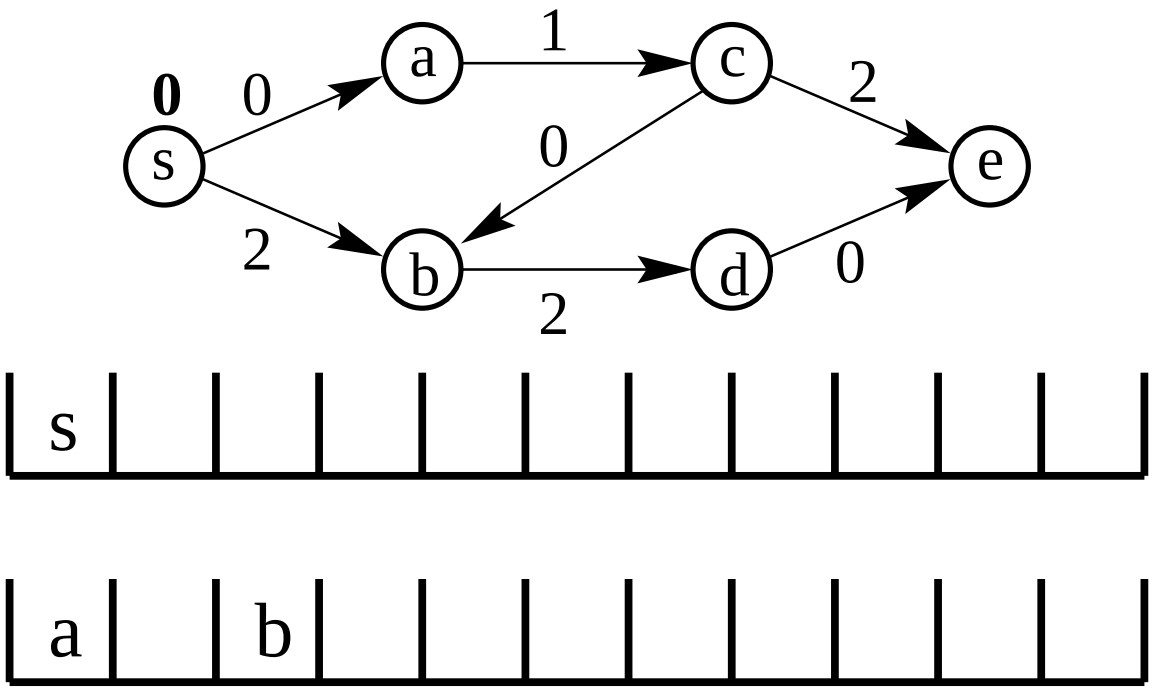
Buckets and Dial's Implementation

- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.



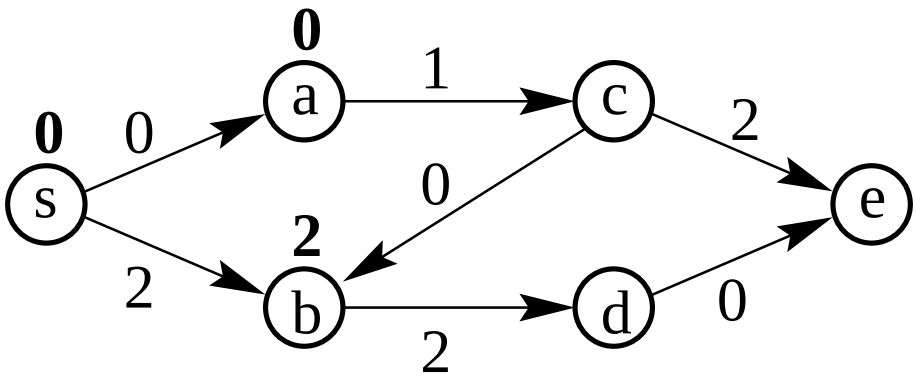
Dial's Algorithm

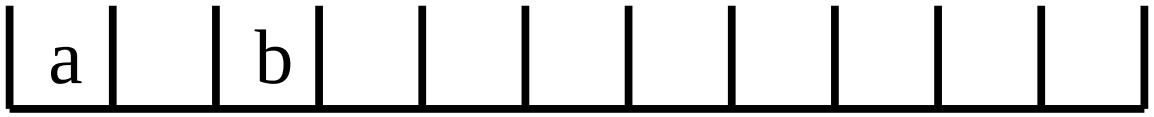
- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.



Dial's Algorithm

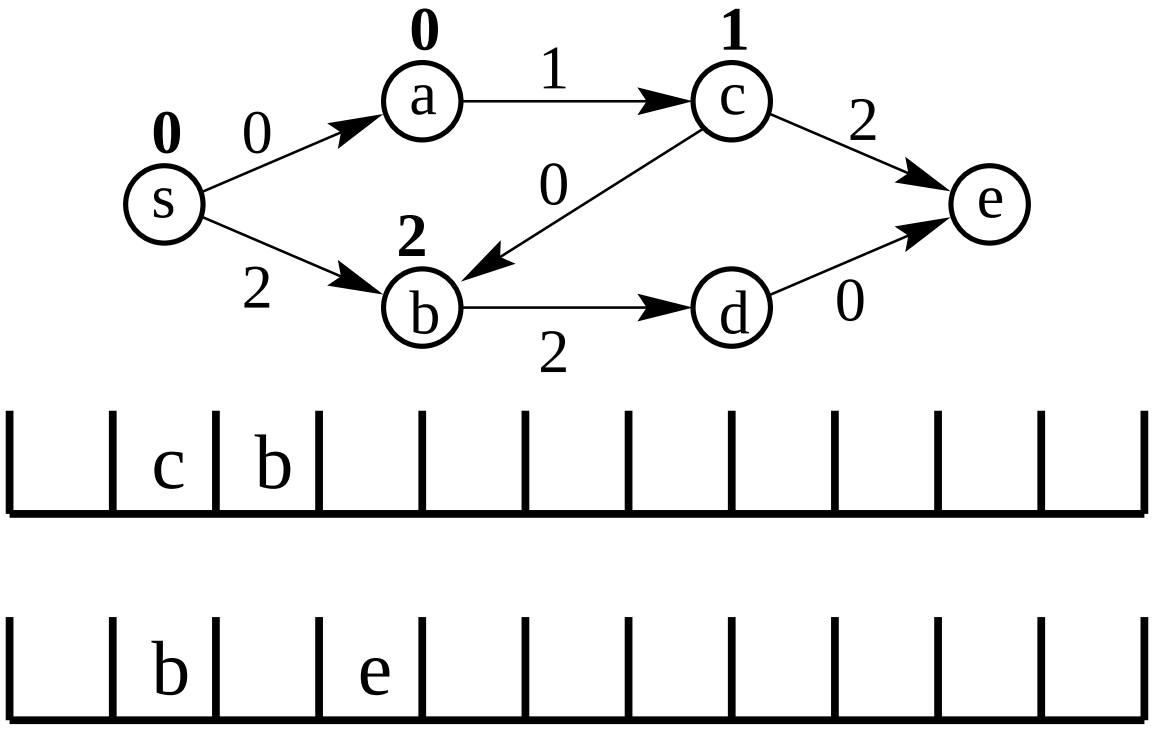
- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.





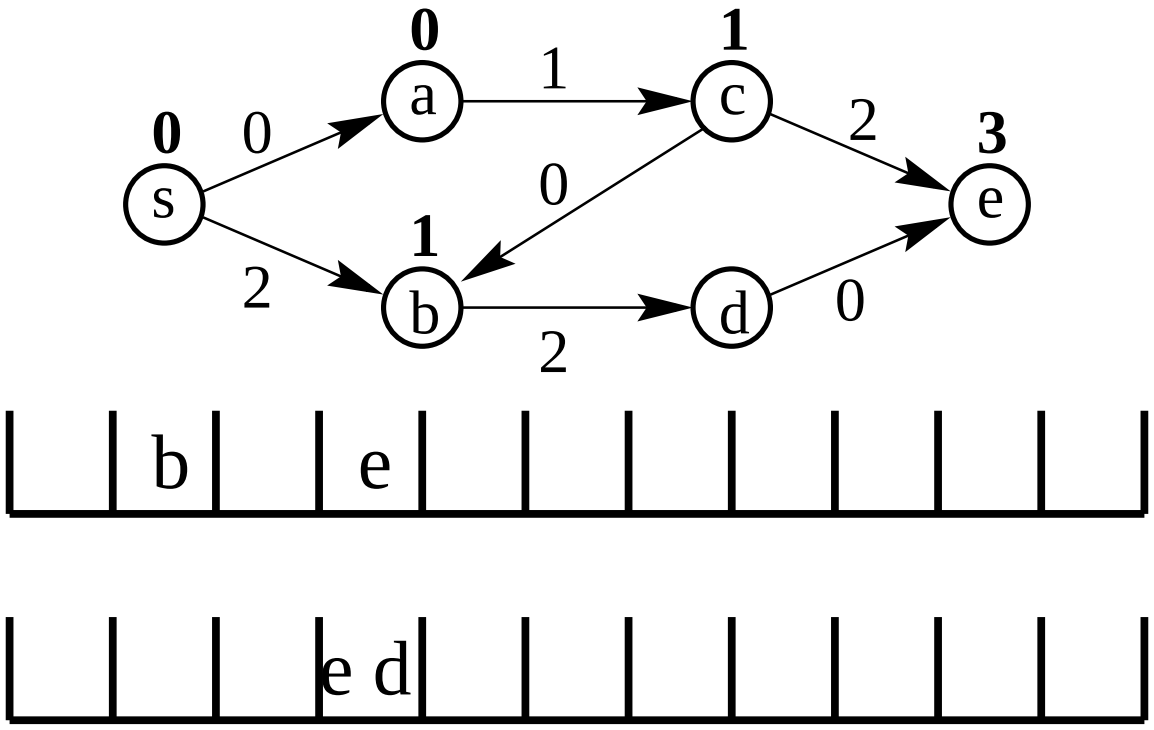
Dial's Algorithm

- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.



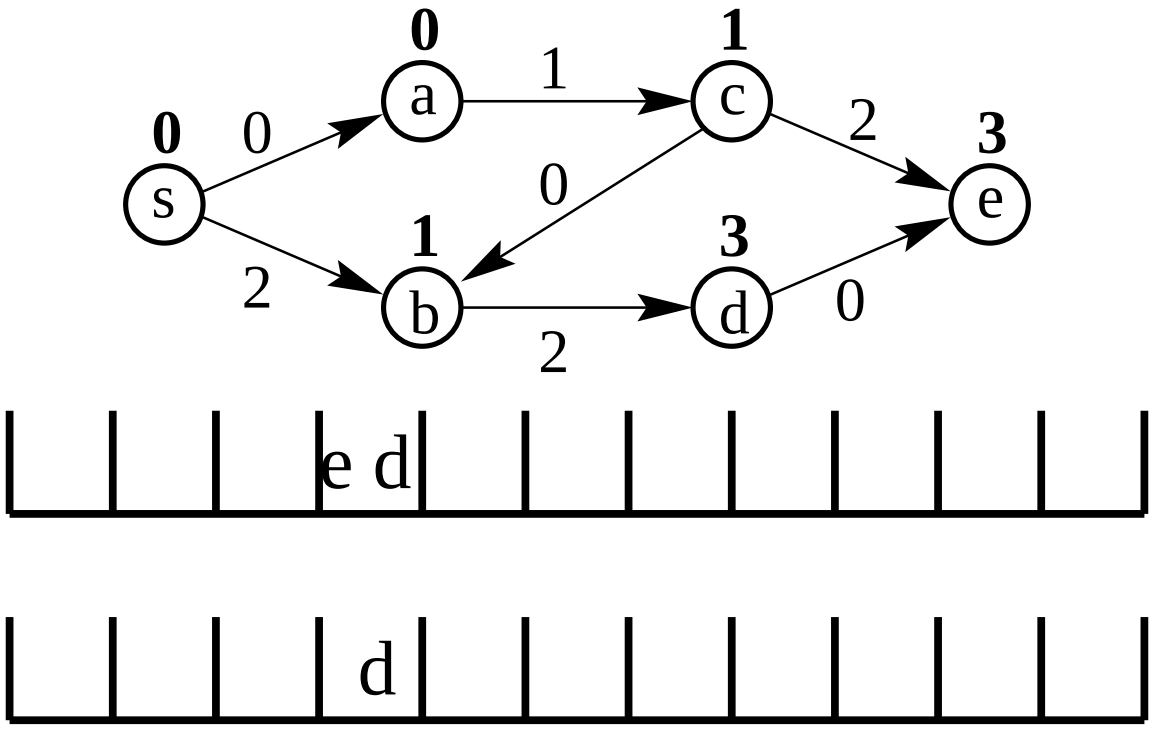
Dial's Algorithm

- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.



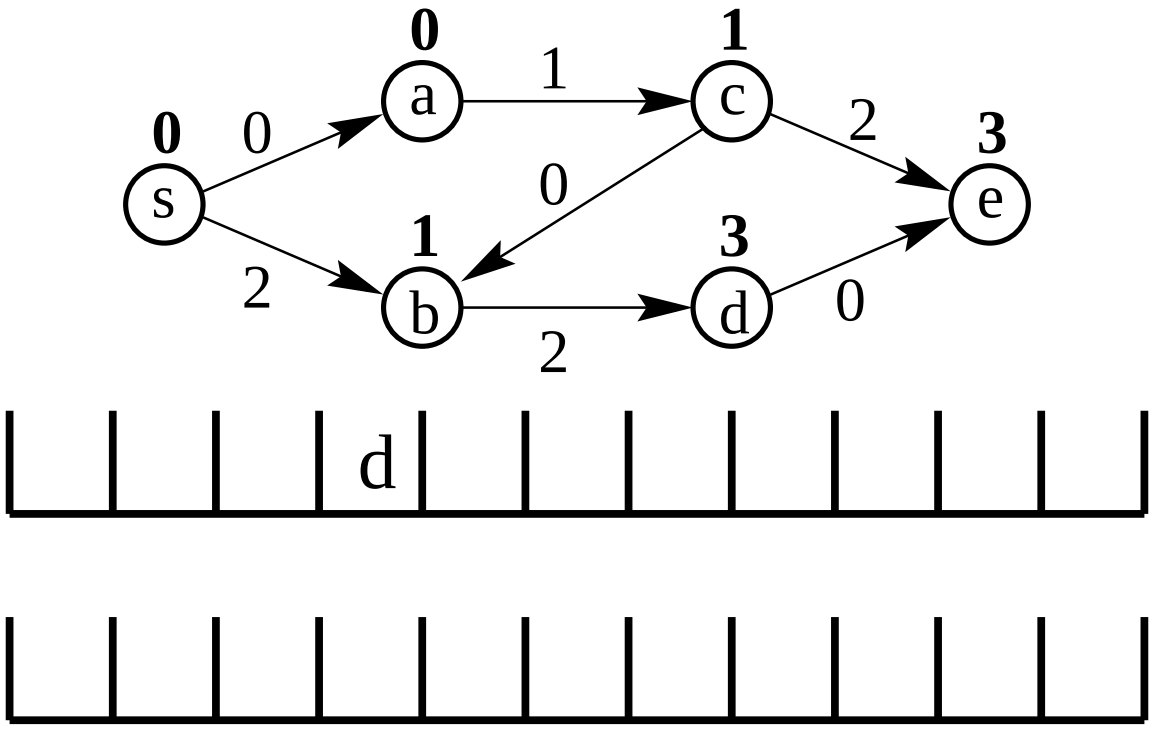
Dial's Algorithm

- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.



Dial's Algorithm

- Maintain an array $B[0 \ldots (n-1)U]$ of buckets (vertex sets).
- Keep a labeled vertex v in $B[d(v)]$.
- The smallest-labeled vertex is in the first nonempty (*active*) bucket.

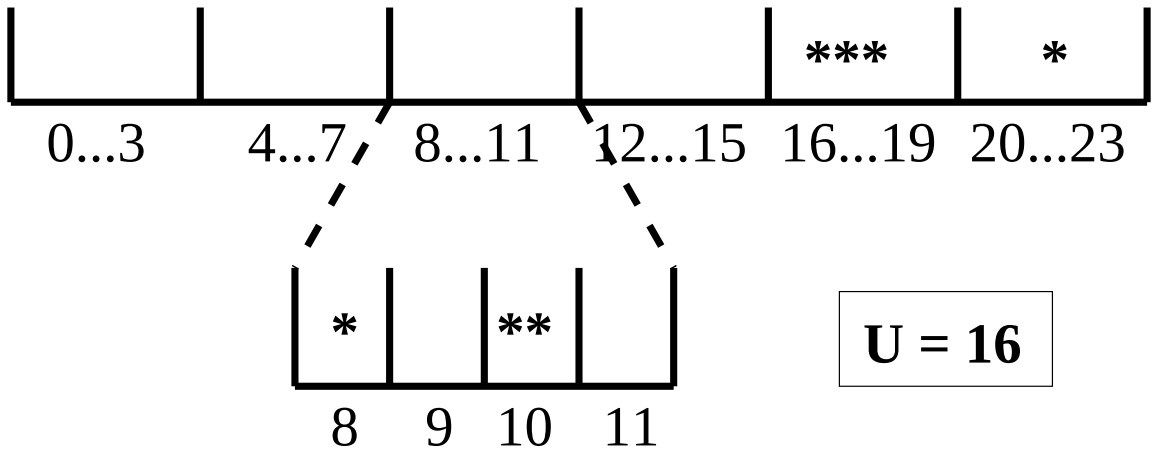


Dial's Performance

- Call the first nonempty bucket active.
- The active bucket index is monotone.
- `insert`, `decreaseKey` take $O(1)$ time, `extractMin` takes $O(U)$.
- Time "wasted" scanning empty buckets.
- $O(m + nU)$ time, $O(nU)$ additional space.
  - Improve space bound to $O(U)$ by working mod $(U + 1)$.
- Alternative time: $O(m + D)$ (D is the biggest distance).

Simple, works well for small values of U.

Two-Level Buckets



- Make (upper level) bucket width $\sqrt{U}$.
- If nonempty, *expand* the active upper-level bucket into $\sqrt{U}$ low-level buckets by finding and scanning a minimum labeled vertex and bucket-sorting the remaining vertices.
- Scan the low-level buckets, then return to the upper level.

Two-Level Bucket Analysis

- Do not expand empty buckets.
- Vertices can move only down, bucket expansions charged to these moves; $O(n)$ work.
- At most $\sqrt{U}$ consecutive empty buckets can be scanned without a vertex scan.
- Charge the bucket scans to the vertex scan. $O(\sqrt{U})$ charges per vertex.

$O(m + n\sqrt{U})$ time.

Multi-Level Buckets

MB algorithm [Denardo & Fox 1979]

- Generalization to k levels.
- Δ buckets at each level, $k = O(\log_\Delta U)$ levels.
- Key to the analysis: Vertices can move only down in the bucket structure.
- $O((\log_\Delta U + \Delta)n)$ data structure manipulation time.
  Number of levels + number of buckets in a level.
- The term-balancing value is $\Delta = \frac{\log U}{\log \log U}$.

$O(m + n\frac{\log U}{\log \log U})$ shortest path algorithm.

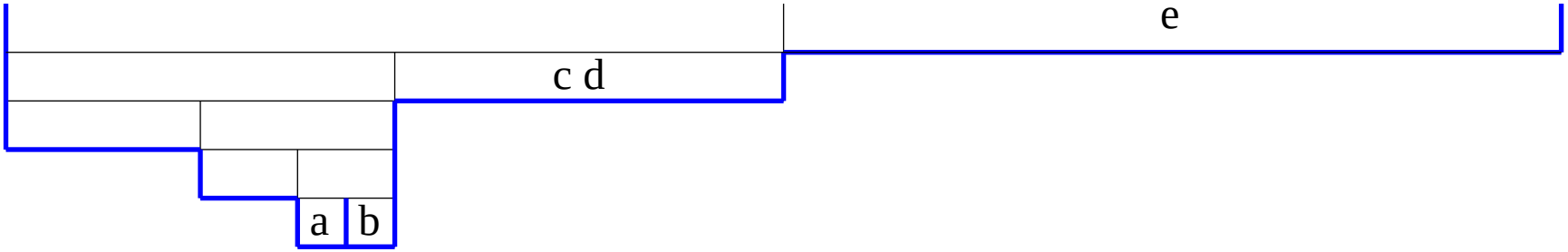
Low-Level Details

- Word RAM model, AC^0 word operations.
- $B(i, j)$: j-th bucket at level i.
- μ is the last extracted key value (initially zero).
- Bucket positions are w.r.t. μ, which is a the bottom (level 0)
- μ_t denotes μ base Δ with t lowest bits replaced by 0's.

Computing positions in B:

- Consider values base Δ (power of two).
- Buckets correspond to value ranges:
  $B(i, j) \rightarrow [\mu_i + j\Delta^i, \mu_i + (j + 1)\Delta^i)$.
- Position of u: i is the index of the first digit u and μ differ in, j is the digit value.
- Can be computed in constant time.

Example



- Base $\Delta = 2$, $\mu = 00110$.
- $a = 00110$, $b = 00111$, $c = 01000$, $d = 01111$, $e = 10010$.

Details (cont.)

Insert: Find the position (i, j) of u and insert u into $B(i, j)$.
decreaseKey: Delete, insert with the new key.

Both operations take constant time.

extractMin:

- Find lowest nonempty level i (constant time).
- Set j to the i-th digit of μ, while $B(i, j)$ empty increment j.
- If $i = 0$ delete u from $B(i, j)$, set $\mu = u$, return u.
- Otherwise
  - Find and delete minimum u in $B(i, j)$, set $\mu = u$.
  - Expand $B(i, j)$ by inserting $v \in B(i, j)$ into new positions.
  - return u.
- Positions change only for $v \in B(i, j)$, vertex levels decrease.

Incrementing j charged to extracted vertex;
expand work charged to vertex level decreases.

HOT Queues

Heap On Top of buckets.

- Maintain active buckets with at most t elements as a heap.
- Can use "black-box" monotone heaps.
- Assume `extractMin` on the heap takes $T(N)$ time, other operations take $O(1)$ time.
- Hot queue operations `insert` and `decreaseKey` take $O(1)$ time.
- `extractMin` takes $O(k + T(t) + \frac{kU^{1/k}}{t})$ amortized time.
- Work to find a nonempty bucket is charged to t elements; each element charged $O(k)$ times.

Hot Queue Bounds

Using Fibonacci heaps: $T(N) = \log N$.
Set $k = \sqrt{\log U}$, $t = 2^{\sqrt{\log U}} = U^{1/\sqrt{\log U}}$.
$O(m + n\sqrt{\log U})$ time bound.

Better heaps lead to better bounds.

Real-world numbers: For $U = 2^{36}$, we have $t = 64$, $\sqrt{\log U} = 6$ (and $\log\log U \approx 5$).

HOT queues are practical (without fancy heaps).

Dinitz Algorithm

- Special case with arc lengths at least $\delta > 0$.
- Use single-level buckets of width δ, scan any active vertex.
- Any vertex v in the first non-empty bucket has $d(v) = \text{dist}(s, v)$.
- Works for real-valued lengths; $O(m + n(U/\delta))$ running time.

Relaxed selection: may pick vertices with non-minimal, but exact label.

Calibers

Definition: A vertex *caliber* $c(v)$ is the minimum length of its incoming arc.

Caliber theorem: Suppose that ℓ is nonnegative, for any labeled v, $d(v) \geq \mu$, and for some u, $d(u) \leq \mu + c(u)$. Then $d(u)$ is exact, i.e., $d(u) = \text{dist}(s, u)$.

Proof: Replace $d(u)$ by $d(u) - c(u)$, $\ell(x, u)$ by $\ell(x, u) - c(u)$ and $\ell(u, y)$ by $\ell(u, y) + c(u)$ for all arcs in and out of u, respectively. Get an equivalent problem with non-negative lengths, and u is the minimum-labeled vertex.

If we scan u, we will never have to scan it again.

Smart Queue Algorithm

Early detection of vertices with correct distances.

SQ is MB with the caliber heuristic:

- Maintain a set F in addition to buckets B; a labeled vertex is either in F or in B. Recall positions in B are relative to μ.
- Initially F contains s.
- When inserting a labeled vertex u into $B \cup F$, insert into F if $d(u) \leq \mu + c(u)$ and into B otherwise.
- Select from F if not empty, otherwise select from B.

F contains vertices v with $d(v)$ equal to the distance from s.

Worst-Case Time

- Overhead for the caliber heuristic is amortized over the other work of the MB algorithm.
- The worst-case time bound is the same as for MB.

Caliber heuristic never hurts much. Does it help?

Yes in many cases.

Lemma: The MB and SQ algorithms run in time $O(m + n + \Phi_1 + \Phi_2)$, where Φ_1 is the number of empty bucket scans and Φ_2 is the number of vertex moves during bucket expansions. Φ's balance for $\Delta = \Theta(\frac{\log U}{\log \log U})$.

Average-Case Analysis

Probabilistic model: Assume that arc lengths are integers uniformly distributed on $[1, M]$. Similar analysis works for some other distributions, e.g., reals on $[0, 1]$.

Expected Time Theorem: The SQ algorithm with $\Delta = 2$ runs in linear expected time.
Compare to $\Theta(m + n \log M)$ worst-case time ($\Delta = 2$).
$\Phi_1 = O(n)$ because $\Delta = 2$.
$\mathbf{E}[\Phi_2] = O(m)$ due to the caliber heuristic.

High Probability Theorem: If arc lengths are independent, the SQ algorithm with $\Delta = 2$ runs in linear time w.h.p.

Expected Time Analysys

Recall that the top level is k.

Lemma: vertex v never gets below level $\lfloor \log c(v) \rfloor$.

Definition: $w(u,v) = k - \lfloor \log c(u) \rfloor$; $w(v) = \max w(u,v)$.

$\sum_v (w(v))$ bounds the number of vertex down moves.

$\sum_V (w(v)) \leq \sum_A w(u,v)$.

$Pr[w(u,v) = t] = 2^{k-t}/M \leq 2^{k-t}/U \leq 2^{-t}$ for $t = 1, \ldots, k$.

$\sum_v (w(v)) = O(m)$.

This implies the theorem.

Experimental Evaluation

Address the following issues:

- How well MB performs?
- How much can the caliber heuristic save/cost?
- Worst-case behavior.
- Room for improvement?

Codes we compare:

- MB2D ($\Delta = 2$), MB2L ($k = 2$), MB-A (k adaptive).
- SQ2D ($\Delta = 2$), SQ2L ($k = 2$), SQ-A (k adaptive).
- H2 (2-heap), H4 (4-heap).
- PAL (Pallottino's algorithm).

Engineering Aspects

Careful coding to keep constants small.

Tuning. In MB-A and SQ-A, set asymptotic values of k and Δ so that $\Delta \approx 64k$. (Expanding a vertex is much more expensive than scanning an empty bucket.)

Explore locality: Place arcs out of each vertex in adjacent memory locations.

Measure time relative to breadth-first search (intuitive lower bound, machine-independent).

Problems with Uniform Lengths

Random graphs: degree 4, arc lengths independent and uniformly distributed on $[1, M]$.
Many labeled vertices most of the time.

- RAND-I: $M = n$, n grows.
- RAND-C: n fixed, M grows.

Long grids: Width 8, arc lengths independent and uniformly distributed on $[1, M]$.
Few labeled vertices at any time.

- LONG-I: $M = n$, n grows.
- LONG-C: n fixed, M grows.

RAND-I Data

Data: time (relative to BFS); empty scans /n; moves/n;

$M = n$	BFS	MB2L	SQ2L	MB2D	SQ2D	MB-A	SQ-A	H2	H4	PAL
2^{17}	0.15 sec.	1.55 3.01 1.09	1.56 1.09 1.04	3.86 0.74 7.14	2.26 0.01 1.56	1.88 2.05 2.00	1.63 0.06 1.05	4.33	3.60	1.67
2^{18}	0.30 sec.	1.73 3.03 1.45	1.64 0.79 1.19	4.15 0.74 7.65	2.26 0.01 1.56	1.93 2.04 2.36	1.73 0.04 1.19	5.03	4.30	1.64
2^{19}	0.62 sec.	1.68 3.34 1.06	1.63 1.12 1.01	4.41 0.74 8.15	2.31 0.00 1.58	1.89 2.36 1.96	1.71 0.05 1.01	5.58	4.60	1.84
2^{20}	1.30 sec.	1.83 3.34 1.41	1.79 0.79 1.20	4.64 0.74 8.65	2.35 0.00 1.58	1.94 2.37 2.09	1.79 0.03 1.06	5.99	4.86	1.85
2^{21}	2.90 sec.	1.83 3.67 1.16	1.77 1.13 1.03	4.73 0.74 9.13	2.29 0.00 1.56	2.00 2.36 2.33	1.78 0.03 1.16	6.54	5.07	1.84

Note: SQ2D vs. MB2D; "2L" codes work well: SQ-A is competitive: PAL similar to MB-A; heaps are not competitive.

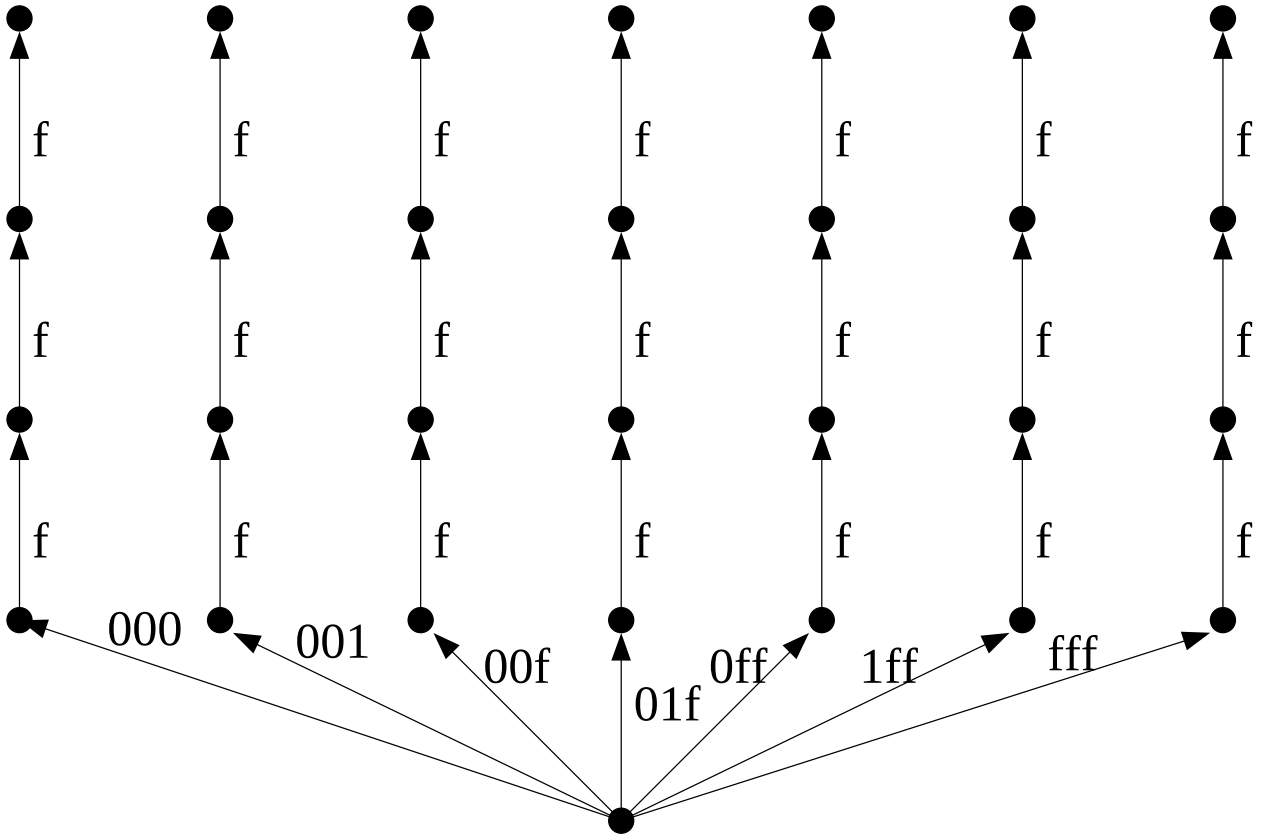
LONG-I Data

Data: time (relative to BFS); empty scans /n; moves/n;

$M = n$	BFS	MB2L	SQ2L	MB2D	SQ2D	MB-A	SQ-A	H2	H4	PAL
2^{17}	0.08 sec.	1.71 8.59 0.46	1.71 4.77 0.39	2.71 1.00 1.51	2.14 0.29 0.83	1.86 4.88 0.61	1.71 1.00 0.41	2.50	2.50	1.63
2^{18}	0.17 sec.	1.66 12.45 0.27	1.61 5.10 0.22	2.80 1.00 1.51	2.13 0.29 0.83	1.81 4.17 0.63	1.68 1.31 0.46	2.35	2.35	1.58
2^{19}	0.35 sec.	1.65 13.65 0.42	1.65 9.43 0.38	2.74 1.00 1.51	2.17 0.29 0.83	1.73 8.89 0.52	1.61 1.40 0.34	2.29	2.29	1.51
2^{20}	0.75 sec.	1.59 17.89 0.23	1.60 10.09 0.21	2.72 1.00 1.51	2.10 0.29 0.83	1.64 6.98 0.45	1.60 1.47 0.31	2.09	2.08	1.41
2^{21}	1.61 sec.	1.60 23.59 0.40	1.63 18.84 0.37	2.65 1.00 1.51	2.06 0.29 0.83	1.62 5.88 0.52	1.59 2.44 0.42	1.96	1.94	1.34

Note: Easy problems, especially for PAL and heaps; SQ2D vs. MB2D; "2L" and SQ-A perform OK.

Hard Problems



A hard problem instance; $k = 3$ and $\Delta = 16$. Arc lengths are given in hexadecimal. Arcs designed to manipulate vertex calibers omitted.

Caliber Heuristic Effect

k		BFS	MB	SQ
2	time	0.63	1189.20	1.38
	emp.	sec.	52428.94	0.40
	exp.		0.60	0.30
3	time	0.63	10.66	1.39
	emp.	sec.	1170.14	0.15
	exp.		1.14	0.57
4	time	0.62	2.68	1.44
	emp.	sec.	170.44	0.11
	exp.		1.56	0.67
6	time	0.63	2.25	1.53
	emp.	sec.	24.31	0.08
	exp.		2.46	0.77
9	time	0.62	2.87	1.60
	emp.	sec.	6.37	0.05
	exp.		3.89	0.85
12	time	0.63	3.70	1.66
	emp.	sec.	3.12	0.04
	exp.		5.36	0.88
18	time	0.63	5.86	1.82
	emp.	sec.	1.41	0.03
	exp.		8.32	0.93
36	time	0.62	16.32	2.32
	emp.	sec.	0.49	0.01
	exp.		17.75	0.97

k		BFS	MB	SQ
2	time	0.62	1199.67	1201.13
	emp.	sec.	52428.94	52428.94
	exp.		0.60	0.60
3	time	0.62	9.38	9.39
	emp.	sec.	1170.14	1170.14
	exp.		1.14	1.14
4	time	0.63	2.67	2.82
	emp.	sec.	170.44	170.44
	exp.		1.56	1.56
6	time	0.62	2.25	2.45
	emp.	sec.	24.31	24.31
	exp.		2.46	2.46
9	time	0.62	2.87	3.08
	emp.	sec.	6.37	6.37
	exp.		3.89	3.89
12	time	0.62	3.72	3.93
	emp.	sec.	3.12	3.12
	exp.		5.36	5.36
18	time	0.63	5.86	6.05
	emp.	sec.	1.41	1.41
	exp.		8.32	8.32
36	time	0.63	16.24	16.43
	emp.	sec.	0.49	0.49
	exp.		17.75	17.75

Hard problems; 36 bits; calibers large (left) and zero (right).

Best emp./exp. tradeoff: $\times 10$ to $\times 100$.

Adaptive codes use 6 levels.

Worst-Case Performance

bits	log Δ	k		BFS	SQ-A
4	4	1	time	0.62	1.37
			emp.	sec.	0.33
			exp.		0.04
6	3	2	time	0.63	1.50
			emp.	sec.	1.60
			exp.		0.80
8	4	2	time	0.62	1.51
			emp.	sec.	3.20
			exp.		0.80
15	5	3	time	0.62	1.73
			emp.	sec.	9.00
			exp.		1.14
18	6	3	time	0.62	1.78
			emp.	sec.	18.14
			exp.		1.14
24	6	4	time	0.62	1.98
			emp.	sec.	21.11
			exp.		1.56
30	6	5	time	0.62	2.18
			emp.	sec.	23.00
			exp.		2.00
35	7	5	time	0.62	2.32
			emp.	sec.	46.27
			exp.		2.00
42	7	6	time	0.62	2.55
			emp.	sec.	48.92
			exp.		2.46
49	7	7	time	0.62	2.80
			emp.	sec.	50.87
			exp.		2.93

Our hardest problems with 49-bit lengths took less then 3×BFS. Bigger costs cause distance overflows.

Much harder problems are unlikely. Problems with 32 or fewer length bits should always stay under 2.5×BFS and will often be below 2×BFS.

Remarks on Experiments

For typical problems, fetching the graph from memory dominates SQ data structure overhead.

- The gap between CPU speed and memory speed grows.
- A cache miss costs about 100 instructions. (1999 machine used.)
- SQ/MB data structure fits in cache (small number of levels with a moderate number of buckets each).
- Amortized cost of SQ operations is less than a 100 instruction per vertex scan.
- MB and SQ will look even better on current machines.

Point-to-Point Problem (P2P)

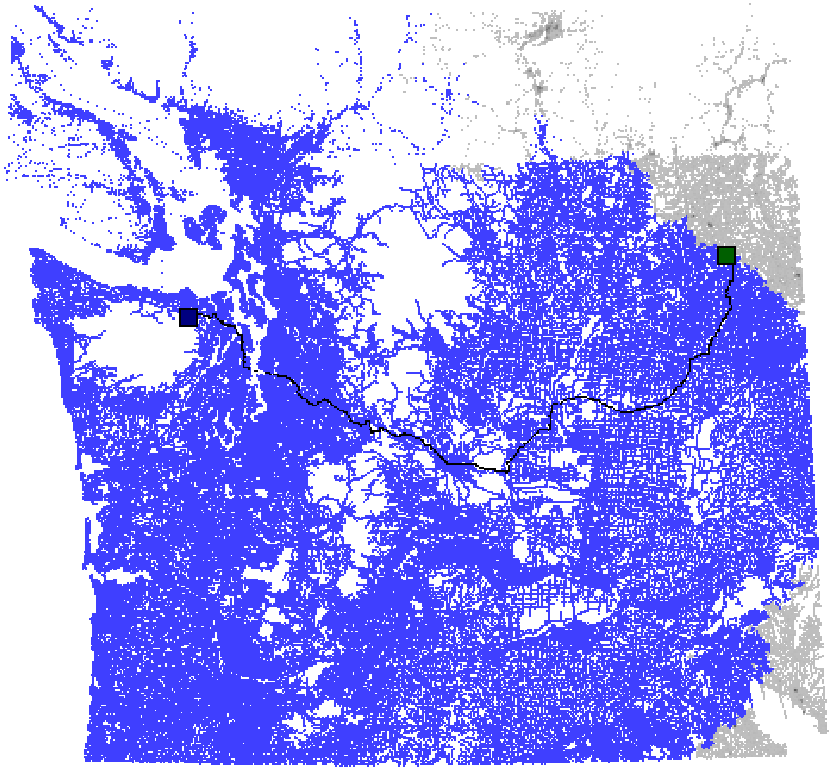
Input: Digraph $G = (V, A)$, $\ell : A \rightarrow \mathbf{R}^+$, source $s, t \in V$.
Goal: Find a shortest path from s to t.
Fundamental problem with many applications.
For arbitrary lengths, little is known.

P2P Dijkstra's algorithm: Run from s, stop when about to scan t. At this point t has correct distance label/path.

Do not need to look at the whole graph. Try to search as little as possible.

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

Example

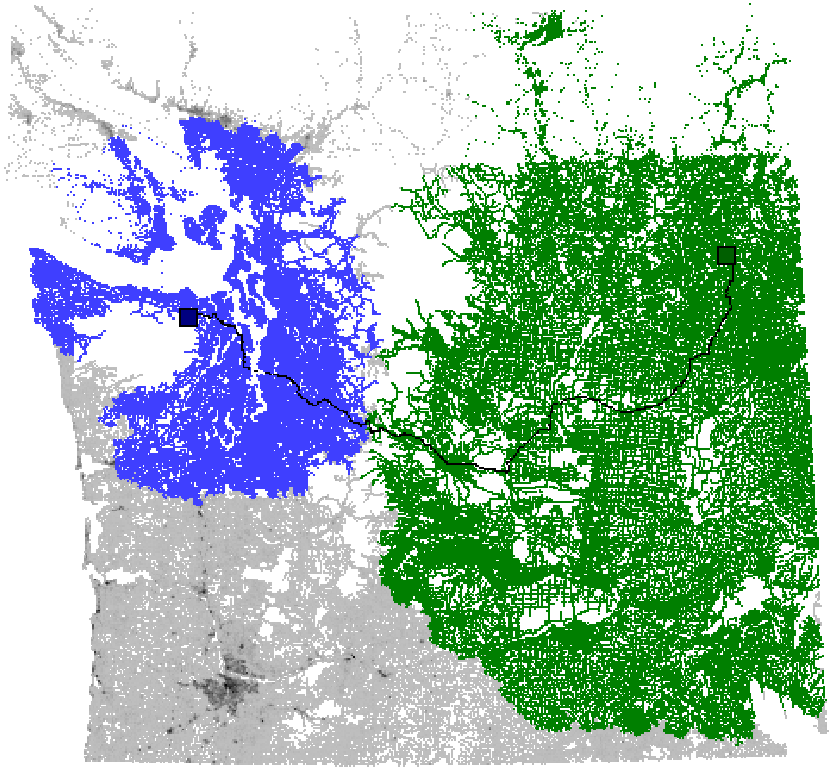


The algorithm grows a ball around s.

Bidirectional Algorithm

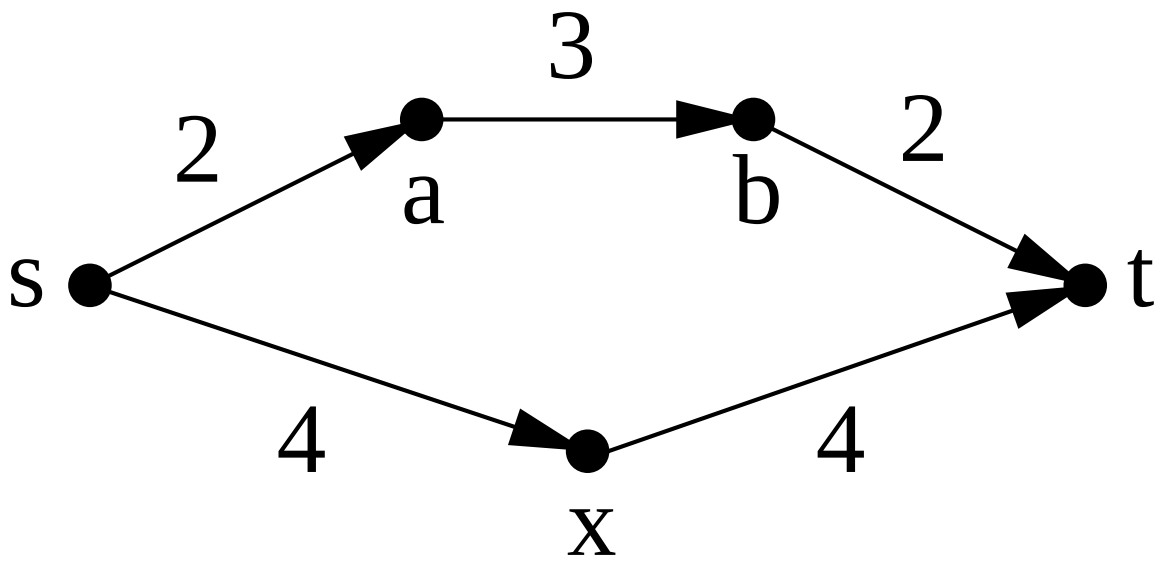
- Run forward Dijkstra from s and backward from t.
- Maintain β, the length of the shortest path seen (initially ∞): when scanning an arc (v, w) such that w has been scanned in the other direction, check the corresponding s-t path.
- Stop when about to scan a vertex x scanned in the other direction.
- Output β and the corresponding path.
- Easy to get wrong.
- Can alternate between the two searches in any way.
- Balancing the work is 2-competitive.

Bidirectional Example



Two balls meet.

Bidirectional Algorithm (cont.)



x need not be on a shortest path.

Alternative stopping criteria: Stop when the sum of the minimum d's for the two search queues is at least β.

Theorem: The alternative stopping condition is correct.

Proof: Left as an exercise.

May stop before the standard algorithm.

Use of Preprocessing

If expect many different s, t queries on the same graph, can preprocess the graph (e.g., map graph). May be unable to store all pairs of shortest paths.

Preprocessing with Bounded Space:

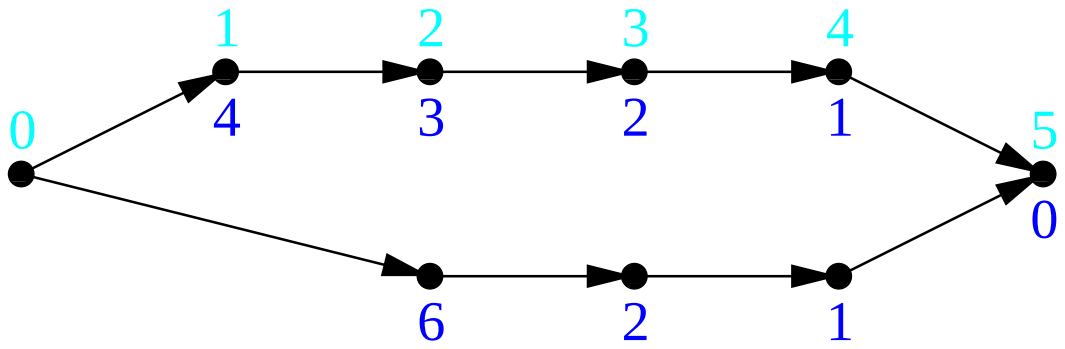
- Theoretical results: [Fakcharoenphol & Rao].
- Approximation algorithms: [Cowen & Wagner 00], [Thorup 01], [Klein 02].
- Using geometry: [Gutman 04], [Lauther 04], [Wagner & Willhalm 03].
- Hierarchical approach: [Schulz, Wagner, Weihe 02], [Sanders & Schultes 05].
- A* search (goal-directed, heuristic) search [Goldberg & Harrelson 04], [Goldberg & Werneck 05].

A* Search

AI motivation: search a small subset of a large space.
[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 vertex with min. $k(v) = d_s(v) + \pi_t(v)$.
- Scan a node on a path with the best length estimate.
- In general, optimality is not guaranteed.



Feasibility and Optimality

Potential transformation: Replace $\ell(v,w)$ by $\ell_{\pi_t}(v,w) = \ell(v,w) - \pi_t(v) + \pi_t(w)$.
Definition: π_t is *feasible* if $\forall (v,w) \in A$, the reduced costs are nonnegative. (Estimates are "locally consistent".)

Optimality: If π_t is feasible, A* search is Dijkstra's algorithm on the network with lengths replaced by reduced costs, which are nonnegative. In this case A* search is optimal.
Proof: $k(v) = d_s(v) + \pi_t(v) = d_{\ell_{\pi_t}}(v) + \pi_t(s)$, $\pi(s)$ is a constant for fixed s.

Different order of vertex scans, different subgraph searched.

Lemma: If π_t is feasible and $\pi_t(t) = 0$, then π_t gives lower bounds on distances to t.
Proof: Left as an exercise.

Bidirectional A* search

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 $\pi_t + \pi_s = \text{const}$.

Need consistent π_t, π_s or a new stopping criteria.

Consistent potentials: [Ikeda et al. 94]
$p_t(v) = \frac{\pi_t(v) - \pi_s(v)}{2}$, $p_s(v) = -p_t(v)$.

Compromise: in general, p_t gives worse lower bounds than π_t.

Computing Lower Bounds

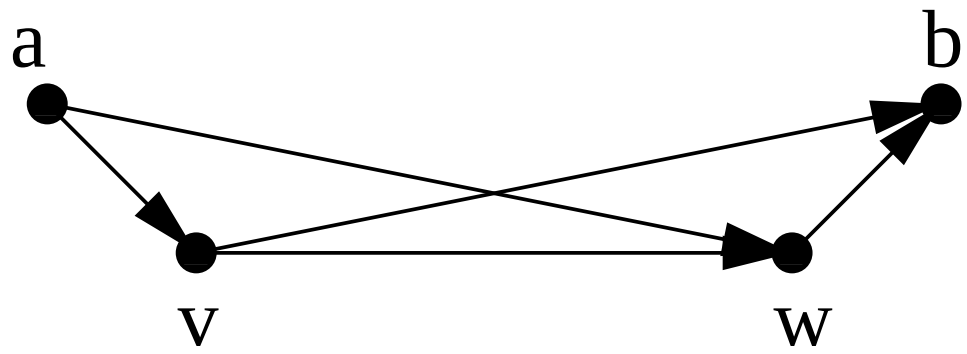
Geometric bounds:

[folklore], [Pohl 69], [Sedgewick & Vitter 86].

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

Limited applicability.

The use of triangle inequality (applies to any graph!)



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

a and b are landmarks (L).

Lower Bounds (cont.)

Lemma: Potentials based on a landmark are feasible.

Proof: $\pi_t(v) = \text{dist}(v, L) - \text{dist}(t, L)$; $\pi_t(w) = \text{dist}(w, L) - \text{dist}(t, L)$

$\ell(v, w) - \pi_t(v) + \pi_t(w) = \ell(v, w) - \text{dist}(v, L) + \text{dist}(w, L) \geq 0$.

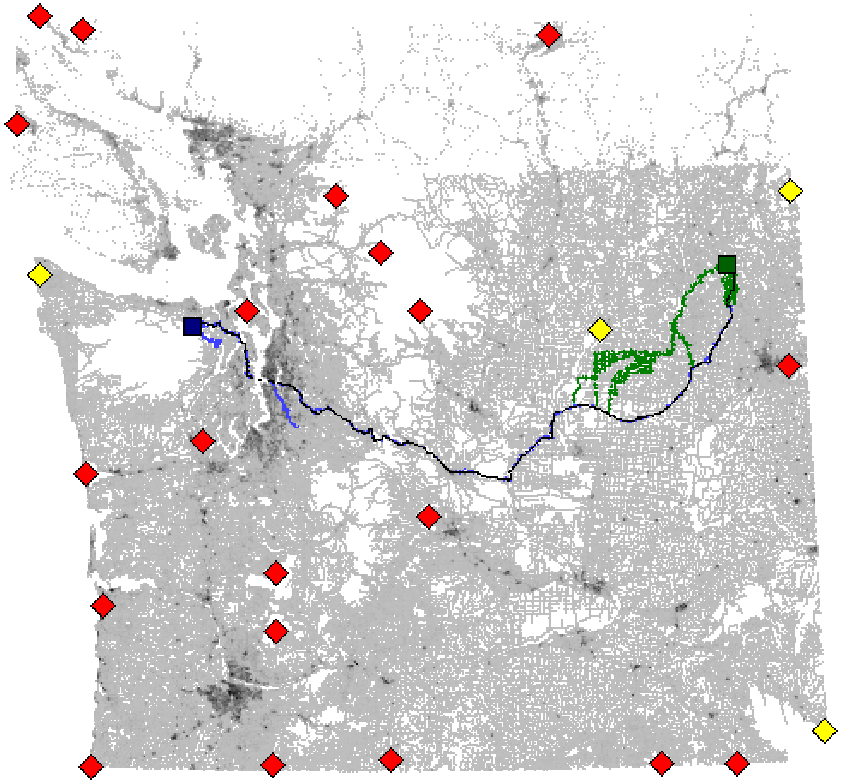
Lemma: Maximum (minimum, average) of feasible potentials is feasible.

Proof: Left as an exercise.

ALT algorithms: A* search with landmark/triangle inequality bounds.

- 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 $\pi_t(v)$.

Bidirectional ALT Example



Note landmarks and active landmarks.

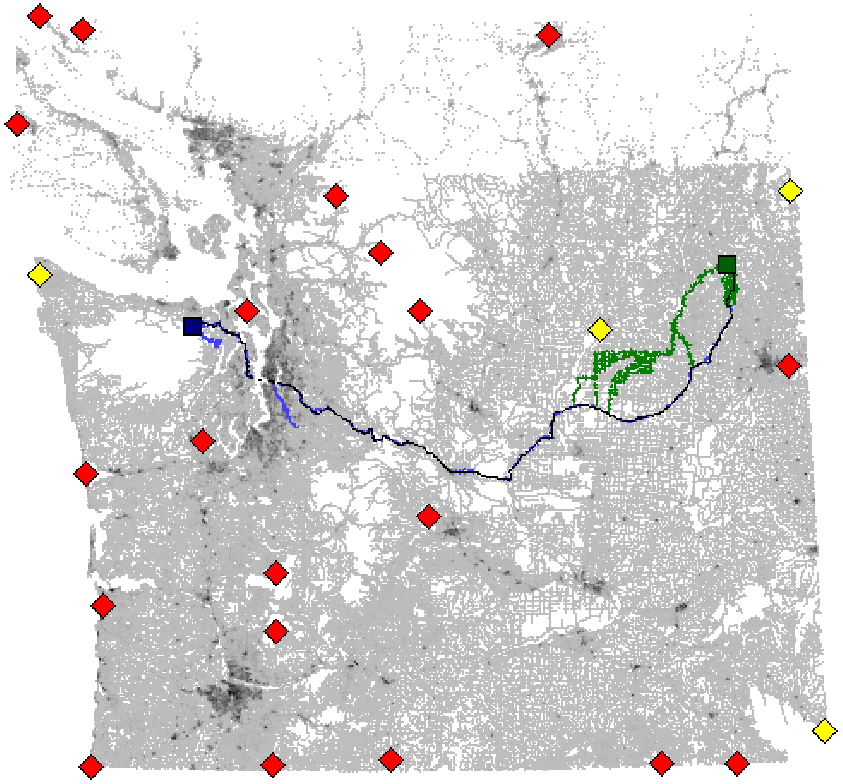
Active Landmarks

- For a given s, t, most landmarks are useless.
- Dynamic selection is better than static selection.
- Start with two landmarks what give the best in and out bounds on the s-t distance.
- When made sufficient progress, check if there are better landmarks and add the best (if any).
- Restart the computation as potentials changed.

Landmark Selection

- The problem is probably NP-hard in any formulation.
- Landmarks can be general or domain-specific.
- Many possible heuristics, even more combinations.
- Iterative improvement, e.g., using local search.
- For very large graphs, preprocessing needs to be fast.
  - Ours takes a few minutes on 1,000,000 vertex graph,
  - several hours on a 30,000,000 vertex graph.
- See the paper for details.
- Better selection may be possible.

Demo



Memory-Efficient Implementation

- Challenge: compute shortest paths in NA on a palmtop.
  Solution: ALT + 4GB flash memory card.
- Toshiba 800e Pocket PC, 128 MB Ram (but $\approx$ 48 MB used by OS), 400 MHz Arm processor.
- Limitations:
  - Small RAM for the task.
  - Cannot read less than 512 bytes from flash.
  - Slow random access read from flash (DSL speed).

Implementation (cont.)

- Input graph and landmark distances on flash.
- In RAM mutable nodes (visited vertices) with
  - ID,
  - parent,
  - distance,
  - heap position index.
- Use hashing to access mutable nodes.

Footprint: 15 MB for 200,000 mutable nodes, 78 MB for 2,000,000.

Nonnegative Lengths: Summary

- Dijkstra's algorithm and priority queues.
- Dial's algorithm, multilevel buckets, HOT queues.
- Relaxed selection rules and expected linear-time algorithm.
- Experimental results.
- Point-to-Point shortest paths.
- Bidirectional Dijkstra algorithms.
- Preprocessing, A* search, use of landmarks.

Remarks

- NSSSP almost solved in theory and in practice.
- Big open question: linear-time algorithm (like [Thorup 99] for undirected graphs).
- Preprocessing may help locality.
- P2P problem with preprocessing: significant recent progress in algorithms for special graph classes. Theoretical work trails behind.
- SSSP problem: widely studied but open questions remain, even for worst-case bounds: e.g., BFM is still the best strongly-polynomial algorithms.
- Other topics include dynamic and all pair algorithms, sophisticated data structures, special networks (planar, small lengths, ...), etc.