Bidirectional search and Goal-directed Dijkstra

Ferienakademie im Sarntal — Course 2
Distance Problems: Theory and Praxis

Kozyntsev A.N.

Fakultät für Informatik
TU München

26. September 2010

Friedrich-Alexander-Universität Erlangen-Nürnberg

Technische Universität München

Universität Stuttgart

Outline

Introduction

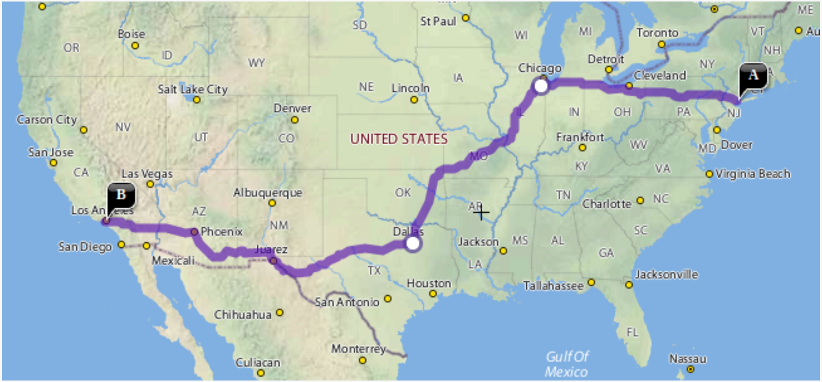
Problems

- Network flow
- Approximations to the traveling salesman problem
- Problem-solving systems in artificial intelligence

Realization: providing driving directions

- Mapquest
- Yahoo! Maps
- Microsoft MapPoint
- Some GPS devices

Introduction



Yahoo! Maps

http://maps.yahoo.com/

```
Q.insert(s, 0)
while not Q.isEmpty() do
    v ← Q.dequeue()
    if v = t then
        // shortest path found
        return
    end
    forall outgoing edges e = (v, w) do
        if w is a new node then
            Q.insert(w, dist(v) + w(e))
            pre(w) ← v
        end
        else
            if dist(v) + w(e) ≤ dist(w) then
                Q.decreaseKey(w, dist(w) + w(e))
                pre(w) ← v
            end
        end
    end
```

Definition of a Random Graph

For expository purposes we use the following probabilistic model:

1. The graph is a complete directed graph with n nodes.
2. Using an exponential distribution: the mean length of an edge is $1/\lambda$ for all n, or $Pr[\ell_e \leq x] = 1 - \exp^{-\lambda x}$
3. Exponential distribution is *memoryless*: $Pr[\ell_e \leq x + y | \ell_e \geq x] = Pr[\ell_e \leq y]$ for all $x, y \geq 0$.

Analysis of Unidirectional Search

We need to study the distribution of the number of edges discovered before d is added to S.

- Consider a point when $L = t$ and k edges $e_1, e_2, \ldots, e_k$ are active.
- Suppose that e_i was activated when $L = x_i$
- Then it's known that $\ell_i \geq t - x_i$.
- Of these edges, the one next discovered is the one that minimize $\ell_i + x$

Analysis of Unidirectional Search

We need to study the distribution of the number of edges discovered before d is added to S.

- Consider a point when $L = t$ and k edges $e_1, e_2, \ldots, e_k$ are active.
- Suppose that e_i was activated when $L = x_i$
- Then it's known that $\ell_i \geq t - x_i$.
- Of these edges, the one next discovered is the one that minimize $\ell_i + x$
- But, for any $y \geq 0$:

$$\begin{aligned} Pr[x_i + \ell_i \leq y | \ell_i \geq t - x_i] &= Pr[\ell_i \leq (t - x_i) + y - t | \ell_i \geq t - x_i] \\ &= Pr[\ell_i \leq y - t] \end{aligned}$$

Analysis of Unidirectional Search

We need to study the distribution of the number of edges discovered before d is added to S.

- Consider a point when $L = t$ and k edges $e_1, e_2, \ldots, e_k$ are active.
- Suppose that e_i was activated when $L = x_i$
- Then it's known that $\ell_i \geq t - x_i$.
- Of these edges, the one next discovered is the one that minimize $\ell_i + x$
- But, for any $y \geq 0$:

$$\begin{aligned} Pr[x_i + \ell_i \leq y | \ell_i \geq t - x_i] &= Pr[\ell_i \leq (t - x_i) + y - t | \ell_i \geq t - x_i] \\ &= Pr[\ell_i \leq y - t] \end{aligned}$$

- This probability is the same for all edges e_i because each ℓ_i is independently chosen from a common exponential distribution.
- Thus each e_i is equally likely to be the next one discovered.

Analysis of Unidirectional Search

Let the random variable X be the number of nodes in S at the end of the algorithm. Then

$$Pr(X = k) = \frac{1}{(n-1)},\ for\ 2 \le k \le n \tag{1}$$

$$E(X) = \Theta(n) \tag{2}$$

Analysis of Unidirectional Search

Proof.

- We say the algorithm is in stage k when $|S| = k$
- In stage k there are $k(n-k)$ external edges leaving S, of which only k go to d
- The probability that d is added to S in stage k is $1/(n-k)$
- Using induction on k: $Pr(X = k) = 1/(n-1)$ for $k = 2, 3, \ldots, n$
- It follows that $E(X) = 1 + (n/2)$

Analysis of Unidirectional Search

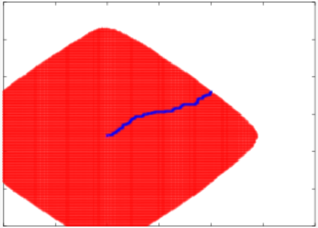
The expected number of edge discoveries in unidirectional search, both internal and external, is $\Theta(n)$.

Analysis of Unidirectional Search

Proof.

- Let Y_i be the number of internal edges discovered in stage i
- In stage i there are $i(n-i)$ external(good) edges active, and at most i^2 internal(bad) edges active
- The probability that a bad edge is next discovered is at most i/n
- $E[Y_i] \leq i(n-i)$
- $E[Z] \leq n$
- The number of edges discovered (internal and external) is $O(n)$

Analysis of Unidirectional Search



What improvements can you suggest?

Basic heuristics

- For the single pair shortest path problem, 2 techniques will be presented:
    1. **Bidirectional search**
    2. **Goal-Directed search**

Bidirectional search

Bidirectional search

- Definition
- Analysis
    - Analysis of **Phase I**
    - Analysis of **Phase II**
- Applications

Bidirectional search

The bidirectional search algorithm proceeds in two phases.

1. **Phase I** alternately adds one node to S and one node to D, continuing until an edge crosing from S to D is drawn
2. **Phase II** finds a minimum path from S to D

Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $L(u)$ is the length of the shortest $s - u - path$.

Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $L(u)$ is the length of the shortest $s - u - path$.
- At the same time we could execute another Dijkstra on the graph with reversed arcs. Now we have the length of the shortest $v - d - path$ for each node v not in this second priority queue too.

Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $L(u)$ is the length of the shortest $s - u - path$.
- At the same time we could execute another Dijkstra on the graph with reversed arcs. Now we have the length of the shortest $v - d - path$ for each node v not in this second priority queue too.
- When a node gets outside both priority queues, we know the shortest path.

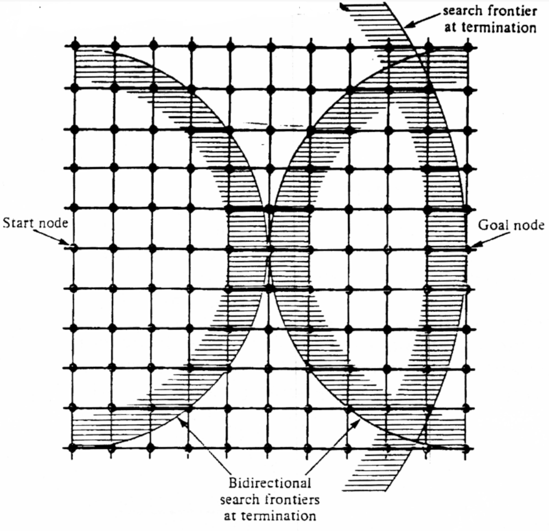
Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $L(u)$ is the length of the shortest $s - u - path$.
- At the same time we could execute another Dijkstra on the graph with reversed arcs. Now we have the length of the shortest $v - d - path$ for each node v not in this second priority queue too.
- When a node gets outside both priority queues, we know the shortest path.
- A degree of freedom in this method is the choice whether a forward or backward iteration is executed.

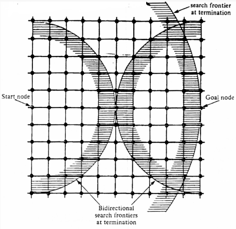
Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $L(u)$ is the length of the shortest $s - u - path$.
- At the same time we could execute another Dijkstra on the graph with reversed arcs. Now we have the length of the shortest $v - d - path$ for each node v not in this second priority queue too.
- When a node gets outside both priority queues, we know the shortest path.
- A degree of freedom in this method is the choice whether a forward or backward iteration is executed.
- Simply alternate or choose the one with lower minimum d in the queue are examples of strategies.

Bidirectional search: **Phase I**



Bidirectional search: **Phase I**



- Let X be the number of stages in Phase I
- **Then** $E[X] = \Theta(\sqrt{n})$
- The total number of edges discovered in Phase I is bounded by $2X$ plus the number of internal edges that are discovered
- **The expected number of internal edges discovered in Phase I is** $O(1)$

Bidirectional search: **Phase II**

In general, the $s - d$ path P found at the end of **Phase I** *is not necessarily the shortest $s - d$ path.*

- The shortest $s - d$ path lies entirely within the search trees associated with S and D except for at most one cross-edge.
- The aim of **Phase II** is to find the shortest path among this restricted set of paths.

Bidirectional search: **Phase II**

Phase II *is a process of node elimination.*

1. Let v be the last node added to D at **Phase I** and t_v be the value of L_D
2. We increase L_S until $L_S + t_v \geq U$, where $U = L_S + L_D$ at the end of **Phase I**
3. At this point, the length of any undiscovered path from s to d via v is at least $L_S + t_v \Rightarrow$ we can eliminate v from D
4. We then increase L_D until we can eliminate the last node added to S in **Phase I**

Bidirectional search: **Phase II**

The expected number of edges discovered during Phase II is $O(\sqrt{n})$.

Bidirectional search: **Implementation Details**

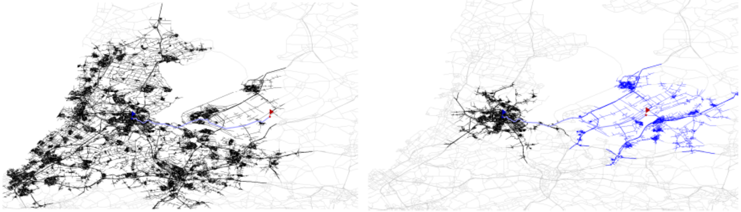
Unidirectional Search

1. Each queue operation takes $O(\log n)$ time
2. The expected running time is $O(n \log n)$

Bidirectional Search

1. Each queue operation takes $O(\log n)$ time
2. The expected running time is $O(\sqrt{n} \log n)$

Bidirectional search: **Implementation Details**



(a) Unidirectional Dijkstra (b) Bidirectional Dijkstra

Unidirectional Search

1. Algorithm searches a ball with s in the center and d on the boundary

Bidirectional Search

1. Algorithm searches two touching balls centered at s and d.

Goal-directed search

Goal-directed search

- Definitions
- Computing lower bounds
- Landmark selection

Goal-directed search

Keynotes

1. Modifies the priority of active nodes to change the order in which the nodes are processed.
2. Adds to the priority $dist(u)$ a *potential* $\rho_t : V \rightarrow \Re_0^+$ depending on the target t of the search.
3. Changes the edge lengths such that the search is given towards the target t
4. The weight of an edge $(u, v) \in E$ is replaced by $\ell'(u, v) := \ell(u, v) - \rho_t(u) + \rho_t(v)$
5. Use Dijkstra with the new weights.

Goal-directed search: **Computing lower bounds**

Obtaining feasible potentials

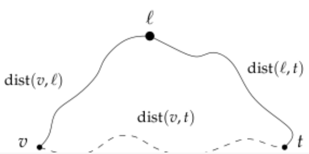
- *Euclidean Distances*
- *Landmarks*
- *Distances from Graph Condensation*

Goal-directed search: **Computing lower bounds**

Euclidean Distances

- Assume a layout $L : V \to \Re^2$ of the graph is available where the length of an edge is correlated with the Euclidean distance of its end nodes.
- A **feasible potential** for a node v can be obtained using the Euclidean distance $\|L(v) - L(t)\|$ to the target t

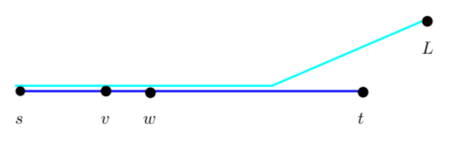
Goal-directed search: **Computing lower bounds**



Landmarks

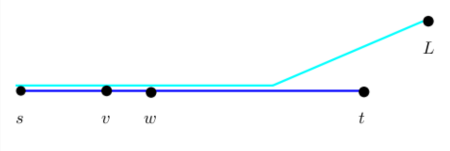
- A small fixed-sized subset $L \subset V$ of **landmarks** is chosen
- The distance $d(v,\ell)$ to all nodes $\ell \in L$ is precomputed and stored, for all $v \in V$.
- The potential for each landmark $\rho_t^{(\ell)}(v) = max\{dist(v,\ell) - dist(t,\ell), dist(\ell,t) - dist(\ell,v)\} \leq dist(v,t)$.
- The potential $\rho_t(v) := \max\{\rho_t^{(\ell)}(v); l \in L\}$.

Goal-directed search: **Computing lower bounds**



Why our ALT algorithms work well?

Goal-directed search: **Computing lower bounds**



Why our ALT algorithms work well?

- The shortest $s - L$ route consists of:
  1. a segment from s to a highway
  2. a segment that uses highways only
  3. a segment from a highway to L
- The shortest route to t follows the same path
- The lower bound ρ_t^ℓ has the following property $\ell'(v, w) = 0$
- These arcs will be the first ones the ALT algorithm will scan

Goal-directed search: **Computing lower bounds**

Distances from Graph Condensation

- Just run *Dijkstra Algorithm* on a **condensed graph**.
- The distances of all v to the target t can be obtained by a single run of *Dijkstra's Algorithm*.
- This distances provide a feasible potential for the time-expanded graph.

Goal-directed search: **Landmark selection**

Finding good landmarks is critical for the overall performance of lower-bounding algorithms.

1. *Random Landmark Selection*
2. *Farthest Landmark Selection*
3. *Planar Landmark Selection*

Goal-directed search: **Landmark selection**

Random Landmark Selection

The simplest way of selecting landmarks is to select k landmark verticies at **random**

Goal-directed search: **Landmark selection**

Farthest Landmark Selection

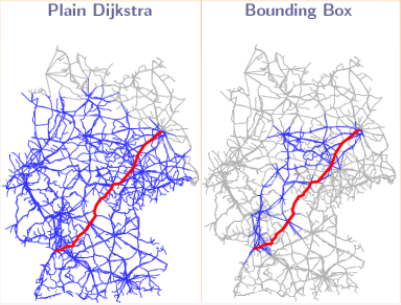
- Pick a start vertex and find a vertex v_1 that is farthest far away from it.
- Add v_1 to the set landmarks.
- Proceed in iterations, finding a vertex that is farthest away from the current set of landmarks.

Goal-directed search: **Landmark selection**

Planar Landmark Selection

- Find a vertex c closest to the center of the embedding.
- Divide the embedding into k pie-slice sectors centered at c.
- For each sector pick a vertex farthest away from the center.

Comparison



Bidirectional lower-bounding algorithms

- Just run the forward and the reverse searches and stop as soon as they meet. This does not work, however.

Bidirectional lower-bounding algorithms

- Just run the forward and the reverse searches and stop as soon as they meet. This does not work, however.
- We say that ρ_t and ρ_s are *consistent* if for all arcs (v, w), $\ell_{\rho_t}(v, w)$ in the original graph is equal to $\ell_{\rho_s}(w, v)$ in the reverse graph.
  $\rightarrow \rho_t + \rho_s = const$

Bidirectional lower-bounding algorithms

- Just run the forward and the reverse searches and stop as soon as they meet. This does not work, however.
- We say that ρ_t and ρ_s are *consistent* if for all arcs (v, w), $\ell_{\rho_t}(v, w)$ in the original graph is equal to $\ell_{\rho_s}(w, v)$ in the reverse graph.
$\rightarrow \rho_t + \rho_s = const$
- If they are not, the forward and the reverse searches use different length functions. Therefore when the searches meet, we have no guarantee that the shortest path has been found.

Bidirectional lower-bounding algorithms

There are two possibilities

1. Develop a new termination condition - **Symmetric Approach**
2. Use consistent potential functions - **Consistent Approach**

Bidirectional lower-bounding algorithms

Symmetric Approach

1. Each time a forward scans an arc (v, w) such that w is already scanned by the reverse search
   - See if the concatenation of the $s - t$ path $(s - v, (v, w), w - t)$ is shorter than best $s - t$ path found so far
   - Update the best path and its length, μ, if needed
2. Do the corresponding updates during the reverse search
3. Stop when one of the searches is about to scan a vertex v with $k(v) \geq \mu$

Bidirectional lower-bounding algorithms

Consistent Approach

1. Let Π_t and Π_s be feasible potential functions giving lower bounds to the source and from the sink.
2. Use $\rho_t(v) = \frac{\Pi_t(v) - \Pi_s(v)}{2}$ and $\rho_s(v) = \frac{\Pi_s(v) - \Pi_t(v)}{2}$
3. These two potential functions are consistent

References

- M.Luby, P.Ragde
  A Bidirectional shortest-path algorithm with good average-case behabior
  Algorithmica, 1989
- A.V.Goldberg, C.Harrelson
  Computing the shortest path: $A^\star$ search meets grapth theory
  Microsoft research, 2003
- T.Pajor
  Goal Directed Speed-Up Techniques for Shortest Path Queries in Timetable Networks
  2008
- R.Bauer, D.Delling, P.Sanders, D.Schieferdecker, D.Schultes, D.Wagner
  Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm
  Universitut Karlsruhe, 2008

Thank you!