CS 420: Advanced Algorithm Design and Analysis
Spring 2015 – Lecture 14

Department of Computer Science
University of British Columbia

February 26, 2015

Announcements

Assignments...

- Sample solutions to Asst 4 (and Midterm I) have been posted
- Asst5...due today

Midterm II...

- Q/A session...next Tuesday (March 03); 5:30-7:00; DMPT 310
- Exam...Wednesday (March 04); 5:30-7:00; DMPT 310
- ...on material up to and including today's class

Readings...

- minimum-cost path problems [Erickson, Chapt 21, 22; Cormen+, Chapt 25 26; ...]
- edit-distance [Erickson, Chapt. 5.5, 6; Kleinberg&Tardos, Chapt. 6.6 & 6.7]
- Goldberg et al,. "Efficient point-to-point shortest path algorithms"

Last class...

Min-cost path problems

- all-pairs of endpoints (cont.)
  - algorithms for dense graphs, using dynamic programming
    - matrix min-sum product approach (generalized Bellman-Ford): $O(n^3 \lg n)$
    - relax constraints on intermediate vertices (Floyd-Warshall): $O(n^3)$

Edit distance problems

- dynamic programming solutions
- reformulation as (single source, single destination) min-cost path problem

Today...

Edit distance problems (cont.)

- reformulation as (single source, single destination) min-cost path problem
- solution by reweighted Dijkstra

Min-cost path problems (cont.)

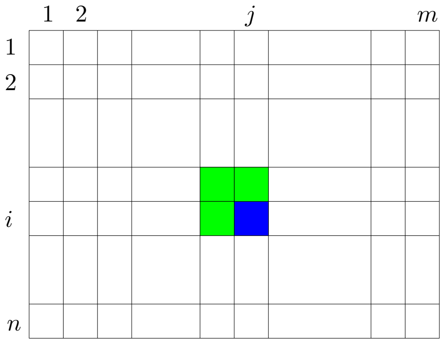
- Dijkstra modifications for single-source single-destination problems
  - bi-directional Dijkstra
  - goal-directed Dijkstra

Edit distance Problem

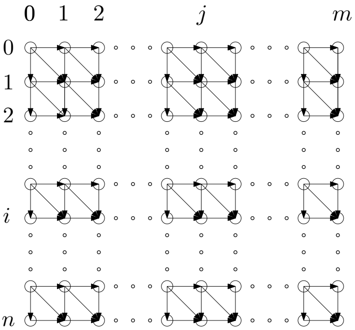
dynamic programming solution:

1. Total cost is $O(nm)$
   - each ED-table entry is computed in $O(1)$ time
2. Space can be reduced to $O(n + m)$
   - it suffices to keep only *two* active columns of ED-table
3. Optimal edit script can be reproduced efficiently
   - by divide and conquer

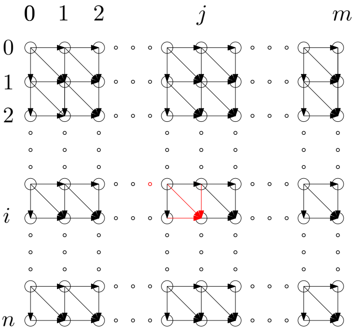
Edit Distance Matrix



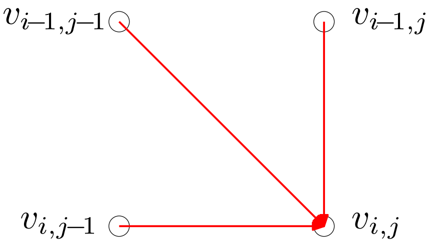
Edit Distance Graph



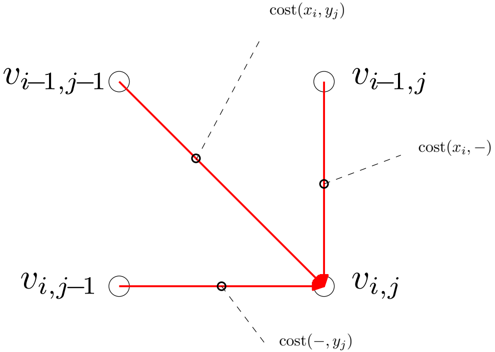
Edit Distance Graph



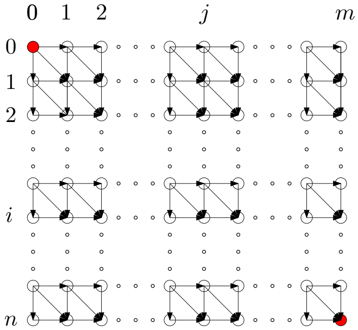
Edit Distance Graph



Edit Distance Graph

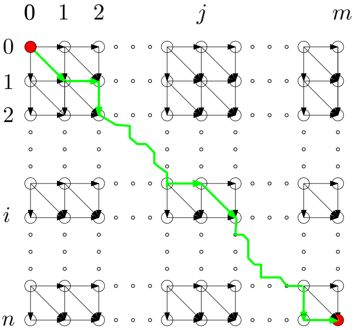


Edit Distance Graph



We want to find the min-cost path from $v_{0,0}$ to $v_{n,m}$.

Edit Distance Graph



We want to find the min-cost path from $v_{0,0}$ to $v_{n,m}$.

Edit Distance Graph

In general, Dijkstra's algorithm will find the min-cost path (and the corresponding edit script) in $O(|E| + |V| \lg |V|) = O(nm \lg(nm))$ time.

In the case where we are interested in the minimum length edit sequence *with no mismatches*:

- we can assign horizontal and vertical edges a cost of 1
- and diagonal edges a cost of 0 (if x_i matches y_j), and ∞ otherwise

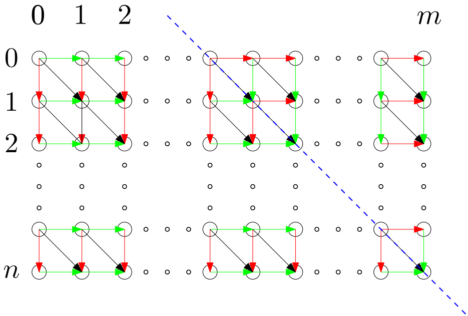
In this case, *any path* from $v_{i,j}$ to $v_{n,m}$ must use at least $|(m-j)-(n-i)|$ horizontal/vertical segments, its cost must be at least $|m-n+i-j|$. Why?

Edit Distance Graph

So, if we use the function $z(v_{i,j}) = |m - n + i - j|$ to reweight the graph, as $\hat{c}(u, v) = c(u, v) - z(u) + z(v)$:

- horizontal and vertical edges that point towards the diagonal through $v_{n,m}$ have their cost reduced (by 1) to zero
- horizontal and vertical edges that point away from the diagonal through $v_{n,m}$ have their cost increased (by 1) to two
- the weight of every diagonal edge is unchanged
- all paths from $v_{i,j}$ to $v_{n,m}$ have their cost reduced by $|m - n + i - j|$

Edit Distance Graph

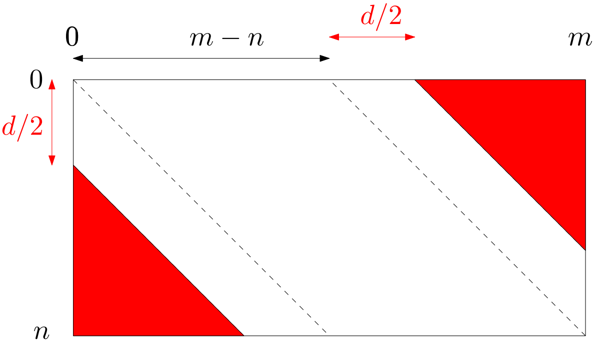


Red edges have cost increased to 2, green edges have cost decreased to 0.

Edit Distance Graph

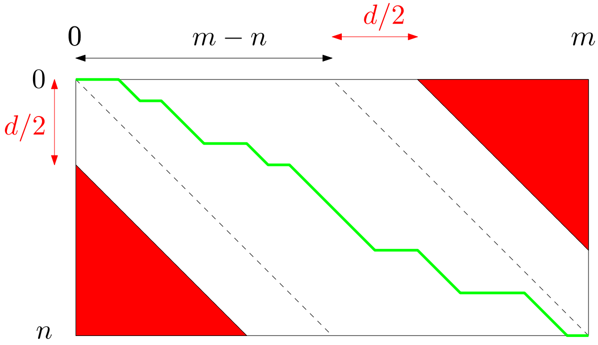
Dijkstra's algorithm will find vertices at (reweighted) cost $1, 2, \ldots, d$ from $v_{0,0}$, where the edit-distance D is $|m - n| + d$. So, explored vertices are confined to $|m - n| + d$ diagonals, each of length $\min\{n, m\}$.

Edit Distance Graph



Range of vertices explored by Dijkstra

Edit Distance Graph



Typical path of (reweighted) cost 0.

Single-source single-destination min-cost paths

The example of the edit-distance graph illustrates two important *general* modifications of Dijkstra's algorithm that applies to single-source single-destination min-cost path problems:

1. bi-directional Dijkstra: run Dijkstra's algorithm from *both* the source and destination, until the wavefronts collide
2. goal-directed Dijkstra: use some *estimate*, $b(v)$, of the distance from all intermediate nodes v to the destination t, to guide the exploration of nodes (using a *reweighted* graph)

Single-source single-destination min-cost paths

In general, it suffices for the estimate b to satisfy two properties:

1. *admissible*: $b(v)$ is a lower bound (underestimate) on $\delta(v, t)$
2. *consistent*: $b(u) - b(v) \leq c(u, v)$

If these hold then we can *reweight* the graph, by $\hat{c}(u, v) = c(u, v) - b(u) + b(v)$, and all edge-weights remain non-negative (by consistency).

Single-source single-destination min-cost paths

In the reweighted graph Dijkstra's algorithm always explores the next unexplored vertex v that minimizes the current $\hat{d}$-value, which must equal

$$\hat{\delta}(s, v) = \delta(s, v) - b(s) + b(v)$$

This, of course, is equivalent to choosing v that minimizes $d_S[v] + b(v)$
which corresponds to the A^* *heuristic* for graph search.

Single-source single-destination min-cost paths

If we were *clairvoyant*, we could choose $b(v) = \delta(v, t)$. In this case, all edges on any min-cost path from s to t would have their weight reduced to zero. So Dijkstra would find one such path in time proportional to the length of the path.

Without a crystal ball, how can we construct admissible and consistent path-cost estimates?

Single-source single-destination search in massive graphs

What if we want to give point-to point driving directions on a large map?

- use Dijkstra
- use bidirectional Dijkstra
- use goal-directed (bidirectional) Dijkstra (A^*)
  - with path estimates based on Euclidean distance – not very effective in practice
  - with path estimates based on *landmarks*

Single-source single-destination search in massive graphs

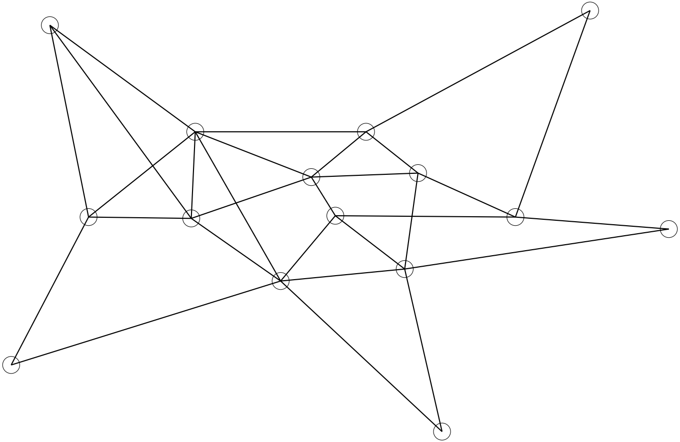
Some figures from a talk by Andrew Goldberg (Microsoft Research):

http://www.slideshare.net/csclub/andrew-goldberg-an-efficient-pointtopoint-shortest-path-algorithm

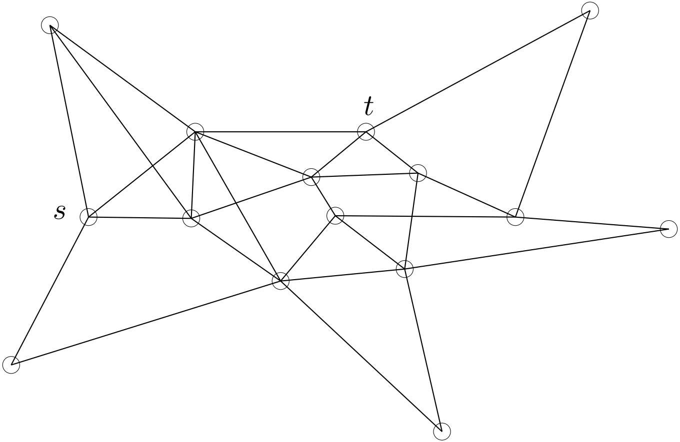
Landmark based estimates

- Choose a small number of well-spaced vertices (*landmarks*) and compute shortest paths from all vertices to all landmarks.
- If $\delta(v, L_i)$ denotes the distance from v to landmark L_i, then estimate $\delta(u, v)$ by $\max_i\{\delta(u, L_i) - \delta(v, L_i)\}$.
- This estimate
  - is guaranteed to be a lower bound on $\delta(u, v)$, by the triangle inequality
  - will be reasonably accurate if some landmark *aligns* well with the shortest path from u to v.

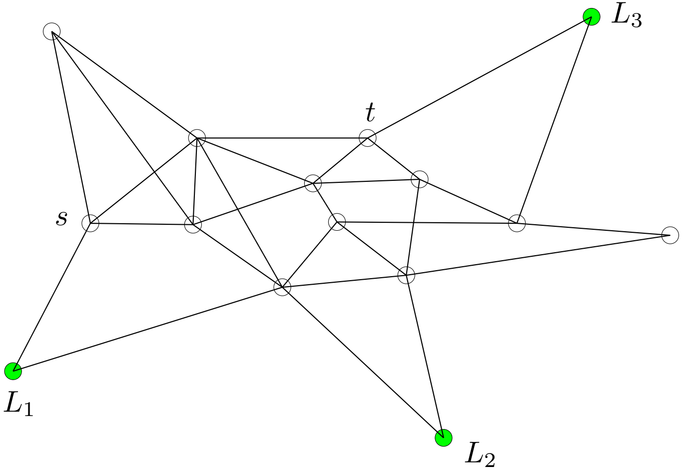
Landmark based estimates



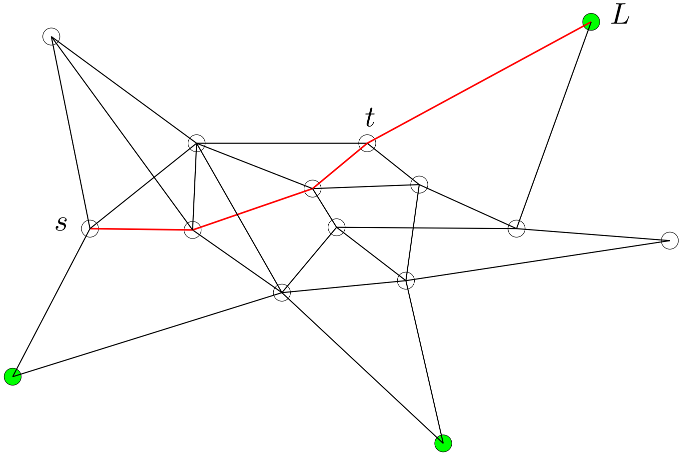
Landmark based estimates



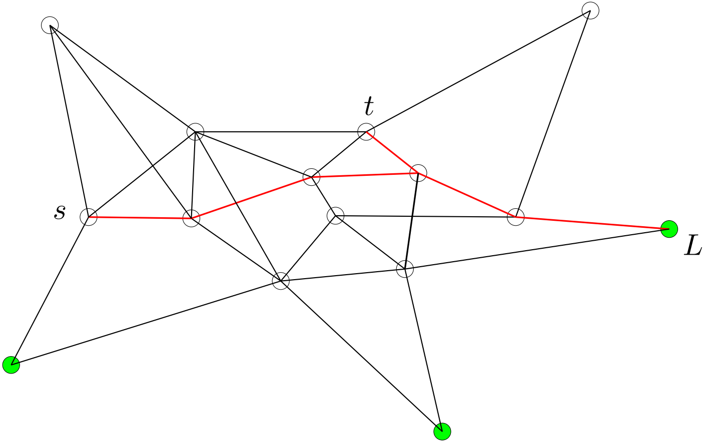
Landmark based estimates



Landmark based estimates



Landmark based estimates



Coming up...

Min-cost path problems

- issues related to real-world constraints
  - robustness (failure tolerance)
  - single-source single destination queries
- issues motion planning (continuous path problems)
  - paths on terrains
  - obstacle avoidance
  - curvature constraints