Advanced algorithms for the $s - t$ shortest path problem: landmarks

Giovanni Righini

University of Milan

Based on: A.V. Goldberg, C. Harrelson, *Computing the Shortest Path: A^* Search Meets Graph Theory*, SODA, 2005.

The $s - t$ shortest path problem

Data:

- a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$;
- a cost function $c : \mathcal{A} \mapsto \Re_+$;
- two nodes s and t (origin and destination).

Problem: find a shortest (minimum cost) path from s to t.

Several applications require computation of $s - t$ shortest paths on very large weighted digraphs (millions of nodes and arcs) in almost real-time (milliseconds).

However the queries concern

- the same digraph
- with different s and t.

The idea is to precompute useful pieces of information that depend on the digraph, but not on s and t.

Landmarks

Landmarks are a technique to compute lower bounds π owing to pre-computed shortest distances.

Consider a landmark L (typically, a node of the digraph) and let $dist(i, L)$ and $dist(L, i)$ be the shortest distances from $i \in \mathcal{N}$ to L and from L to $i \in \mathcal{N}$.

Then, by the triangle inequality,

$$dist(i, L) - dist(j, L) \leq dist(i, j) \quad dist(L, j) - dist(L, i) \leq dist(i, j) \quad \forall i, j \in \mathcal{N}.$$

This holds for any landmark L. Therefore one can select $\pi_t(i) = \max_L\{dist(i, L) - dist(t, L), dist(L, t) - dist(L, i)\}$ (and the same for π_s).

- Pre-compute shortest distances from/to several (e.g. 16) landmarks (independently of s and t).
- Given an (s, t) pair select some landmarks (e.g. 4) providing the largest lower bounds on $dist(s, t)$.

Landmarks selection

Landmark selection techniques to select k landmarks in a given digraph:

- Random: select k nodes at random in $\mathcal{N}$.
- Farthest: iteratively select the node that maximizes the minimum distance from/to all selected landmarks. Variant: consider the number of arcs, instead of the distance (run BFS instead of Dijkstra).
- Planar (for road networks): find a node c closest to the median of the graph. Partition the region into k sectors centered at c, each containing approximately the same number of nodes. For each sector, pick a node farthest away from c (in the sense of the number of arcs).
- Optimized farthest: repeatedly remove a landmark and replace it with the farthest one from the remaining set of landmarks.
- Optimized planar: repeatedly remove a landmark and replace it by the best landmark in a set of candidates. To rate the candidates, compute a score for each one, based on lower bounds tightness for some randomly chosen pairs of nodes.

- Avoid. Given a set S of already selected landmarks, compute a shortest-path tree T_r rooted at some node r.
  Then, for each $v \in \mathcal{N}$ compute its weight, defined as the difference between $dist(r, v)$ and the lower bound for $dist(r, v)$ given by S.
  For each $v \in \mathcal{N}$ compute its size $s(v)$, which depends on T_v, the subtree of T_r rooted at v.
  If T_v contains a landmark, then $s(v) = 0$; otherwise, $s(v)$ is the sum of the weights of all vertices in T_v.
  Let w be the vertex of maximum size. Traverse T_w, starting from w and always following the child with the largest size, until a leaf is reached.
  Make this leaf a new landmark.
  A natural way of selecting r is uniformly at random. Better results are obtained by selecting r with higher probability from the nodes that are far from S.

- Max cover. Define $\overline{c}^L(i,j) = c(i,j) - d(L,j) + d(L,i)$.
  If $\overline{c}^L(i,j) = 0$, then L *covers* (i,j).
  Define $Cost(S) = |\{(i,j) \in \mathcal{A} : \min_{L \in S}\{\overline{c}^L(i,j)\} > 0\}|$.
  Initialize a set C of k *candidate landmarks* by *Avoid*.
  Iteratively remove each landmark from C with probability $1/2$ and generate more landmarks (using *Avoid*) until they are k again.
  Add all newly generated landmarks to C.
  Repeat until either $|C| = 4k$ or *Avoid* is executed $5k$ times.
  Interpreting each landmark as the set of arcs that it covers, solve an instance of the maximum cover problem (NP-hard).
  <u>Multistart heuristic</u>: each iteration starts with a random subset S of C with k landmarks and runs a local search procedure.
  Return the best solution found after $\lfloor \log_2 k + 1 \rfloor$ iterations.
  <u>Local search</u>: iteratively replace a candidate landmark $u \in S$ with $v \in C \backslash S$. Among swaps with positive profit $Cost(S) - Cost(S \backslash \{u\} \cup \{v\})$, pick one at random with probability proportional to the profit. Stop when no improving swaps exist. Each local search iteration takes $O(km)$ time.

Active landmarks

Static landmarks. Select *h* landmarks providing the best lower bounds of the $s - t$ distance.

Trade-off between:

- number of labelled nodes,
- number of landmarks to be examined for each label extension.

Active landmarks

Dynamic landmarks. Initially select 2 landmarks L_1 and L_2, providing the best lower bounds of the $s - t$ distance to L_1 and from L_2.
The search reaches a *checkpoint* when the lower bound for completing the $s - t$ path from the current node is 90%, 80%, 70% and so on of the initial $s - t$ lower bound and at least 100 nodes have been labelled since the last checkpoint.
At a checkpoint at a node v, all landmarks are considered to test whether some of the inactive landmarks provide a lower bound from node v that is larger than $1 + \epsilon$ times the current lower bound (e.g. $\epsilon = 0.01$).
If this is the case, the new landmark is made active (at most 6 active landmarks are accepted) and the potentials are updated.

When π_t and π_s are updated because the active landmarks have been updated, the keys of all labeled vertices are updated and the heaps are updated. This takes $O(|F| + |B|)$ time.

Bounding

Consider a forward iteration in which A^* scans a permanently labelled node i (the same holds symmetrically for backward iterations). Consider one of the outgoing arcs, (i, j). The algorithm should check whether $d_s(i) + c(i, j) < d_s(j)$. If so, $d_s(j)$ is updated in the forward priority queue.

Using lower bounds, the algorithm also checks if $d_s(i) + c(i, j) + \pi_t(j) < U$, where π_t is a feasible forward lower bounding function. When the test fails, the shortest $s - t$ path through (i, j) cannot improve upon the current shortest path. Therefore, there is no need to store an updated value of $d_s(j)$.

The lower bound functions π_t and π_s used for bounding in either direction do not need to be consistent.