6.889 — Lecture 12: Exact Distance Oracles (a.k.a. Shortest-Path *Queries*)

Christian Sommer csom@mit.edu
(figures extracted from [Dji96, FR06])

October 24, 2011

Distance Oracle: given a graph $G = (V, E)$, preprocess it into a data structure such that we can compute shortest-path distances $d_G(v, w)$ (*distance queries*) efficiently (and output path if desired).
two algorithms: one algorithm to *preprocess* the graph, one algorithm to *query* the data structure.

Assumption (all of Lecture 12) *planar* G, non-negative edge lengths $\ell : E \to \mathbb{R}^+$

Lazy strategy run SSSP algorithm for every query $d_G(v, w)$
preprocessing time 0, space $\mathcal{O}(n)$ (store the graph), query time $\mathcal{O}(n)$

Eager strategy precompute APSP, complete distance matrix, one table lookup to answer query $d_G(v, w)$
preprocessing time and space $\mathcal{O}(n^2)$, query time $\mathcal{O}(1)$

Oracle something "between" SSSP and APSP? applications: route planning, traffic simulations, etc.
main concern: *preprocessing time* (running time of the first algorithm), *space* consumption of the data structure (size of the output of the first algorithm), and *query time* (running time of the second algorithm) — in particular, *tradeoffs* between these quantities

Recall: MSSP data structure preprocessing time and space $\mathcal{O}(n \log n)$, query time $\mathcal{O}(\log n)$ (queries however restricted to source on single face)

r**–division approach** pieces of size $\mathcal{O}(r)$ with boundary $\mathcal{O}(\sqrt{r})$ per piece (total boundary $\mathcal{O}(n/\sqrt{r})$)
precompute APSP for all nodes on the boundary, space $\mathcal{O}(n^2/r)$; at query time, explore piece of v (say P_v) and piece of w (P_w) and find best connection pair $\partial P_v \times \partial P_w$, piece sizes $\mathcal{O}(r)$ and $\mathcal{O}(\sqrt{r})^2$ connection pairs, total query time $\mathcal{O}(r) \rightsquigarrow$ smoothly interpolates between SSSP and APSP (only separators used, extends to minor-free graphs). can we do better?

Connections between pieces $\mathcal{O}(\sqrt{r})^2$ connection pairs in $\partial P_v \times \partial P_w$, not independent!
(assume boundary ∂P on $\mathcal{O}(1)$ cycles, r–division with $\mathcal{O}(1)$ *holes* per piece P)

Non-crossing property and bipartite Monge search

Main idea *store* distance from v to all nodes in ∂P_v with respect to G (not just P_v). space $\mathcal{O}(n\sqrt{r})$.
between boundary nodes, precompute distance in $G \setminus (P_v \cup P_w)$. space $\mathcal{O}(n^2/r)$.
at query time, **divide and conquer** for the best pair in $\partial P_v \times \partial P_w$. FOR EACH boundary node $y \in \partial P_w$,

- find the *best* boundary node $x \in \partial P_v$, minimizing $\min\limits_{x \in \partial P_v} d_G(v,x) + d_{G\setminus(P_v \cup P_w)}(x,y)$

if query algo already found pairs $(x_1, y_2), (x_2, y_2)$, the $\min$ for y is restricted to all x between x_1 and x_2

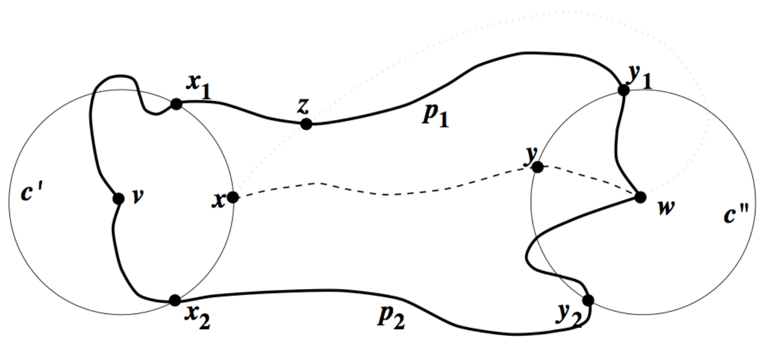

Figure 1: Non-crossing property

also known as the *Monge* property: $\forall u \leqslant v \forall x \leqslant y$: $d(u,x) + d(v,y) \leqslant d(u,y) + d(v,x)$

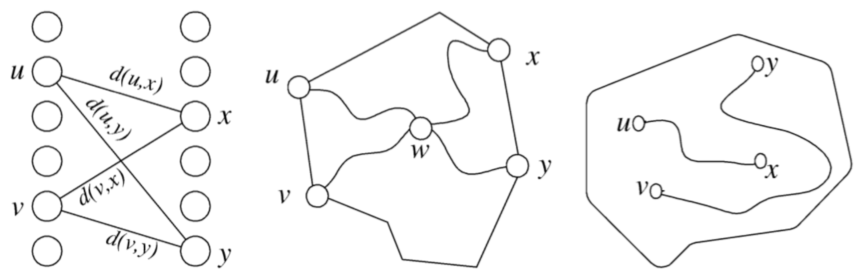

Figure 2: Non-crossing and Monge properties: $d(u,x) + d(v,y) \leqslant d(u,y) + d(v,x)$. Monge property is satisfied for pairwise distances between nodes on a single face of a planar graph (paths cross at node w). Not necessarily satisfied for pairwise distances between general nodes in a planar graph.

Bipartite dense distance graph complete bipartite graph on node set $X \cup Y$, edge lengths correspond to shortest-path distances in some graph H, i.e. $\ell(x,y) := d_H(x,y)$

Bipartite Monge search given $d(x,y)$ (*dense distance graph* for $X \times Y$), compute *parent* $x \in X$ for all $y \in Y$ (directed *matching*, one-sided, x can be parent of multiple y).
If $d(\cdot,\cdot)$ from planar G (Monge) then parent *intervals*: x parent of y_i and $y_k \Rightarrow x$ parent of y_j ($\forall i \leqslant j \leqslant k$).
If $d(\cdot,\cdot)$ satisfies Monge property, divide and conquer computes matching in time $\mathcal{O}((|X|+|Y|)\log(|X|+|Y|))$.
Significantly faster than $\mathcal{O}(|X| \cdot |Y|)$, algorithm does *not* read all of $d(\cdot,\cdot)$. Can even do $\mathcal{O}(|X| + |Y|)$.
Note: still Monge with **initialization** $D(\cdot)$ for each $x \in X$ (think of $D(\cdot)$ as v-to-∂P_v shortest-path distance)

Oracle Problem need to store v-to-∂P_v distances, requires $\mathcal{O}(n\sqrt{r})$ space (dominates space for $r > n^{2/3}$)

Use MSSP? know how to compute (and compactly represent) distances in P_v and $G \setminus P_v$ but not in G
shortest paths may use several nodes of ∂P_v, leaving and re-entering P_v arbitrarily!

On-line bipartite Monge search

Important subroutine in efficient implementation of Dijkstra's algorithm running on dense distance graphs. The algorithm computes shortest paths by combining several instances of the following problem:

On-line bipartite Monge search given $d(x, y)$ (*dense distance graph* for $X \times Y$), **maintain** *parent* $x \in X$ for all $y \in Y$, while initialization $D(\cdot)$ for each $x \in X$ is *revealed on-line*, one node at a time.
Can compute matching on-line in overall time $\mathcal{O}((|X|+|Y|) \log(|X|+|Y|))$ (same time as divide and conquer).

Maintaining a matching means (herein) managing

- set of *active nodes* $A \subseteq X$ and
- growing set of *matched nodes* $M \subseteq Y$ (more importantly: shrinking set of yet unmatched nodes $Y \setminus M$)

while supporting three operations (to be used by Dijkstra's algorithm):

- FINDMIN() returns the $\min$ unmatched node $y \in Y \setminus M$, functions as priority queue for right-hand side
- EXTRACTMIN() adds the current $\min$ to M
- ACTIVATELEFT(x, δ) reveals initialization $D(x) = \delta$ for some $x \in X$ (and updates preliminary matches!)

Efficient implementation use *heap* and *intervals* in ordered set Y. Assume that, for each $x \in X$, precomputed (together with dense distance graph), data structure supporting queries $\min_{i_- \leqslant i \leqslant i_+} d(x, y_i)$ for any i_-, i_+ (note: data structure independent of $D(x)$, query using LCA in $\mathcal{O}(1)$).
Maintain binary search tree for active nodes $x \in A$.
For active $x \in A$, maintain interval $[i_-(x), i_+(x))$ of children $y \in Y$ (x is *current* parent of y, may change).
Maintain priority queue (heap) containing, for each active $x \in A$, its shortest edge to unmatched $y \in Y \setminus M$.

- FINDMIN() returns $\min$ from heap, $\mathcal{O}(1)$
- EXTRACTMIN() adds the current $\min$ to M (found by FINDMIN()); suppose the $\min$ was $y_j \in Y \setminus M$ with parent $x \in A \rightsquigarrow$ need to insert second-shortest edge from x into heap; instead, create two dummy nodes x', x'' spanning intervals $[i_-(x), j)$ and $[j+1, i_+(x))$, respectively. find $\min$ in these intervals using above LCA data structure (for x) and insert into heap (time $\mathcal{O}(\log |Y|)$).
  how many new dummy nodes? at most two for each $\min$ in Y extracted from heap, $\mathcal{O}(|X|+|Y|)$ overall
- ACTIVATELEFT(x, δ): new node $x \in X$ is activated.
  compute its interval $[i_-(x), i_+(x))$. all of Y if first x, otherwise
  - "walk up" in A (non-empty intervals only) until first x' whose $d(x', y_{i_-(x')}) < d(x, y_{i_-(x')})$,
  - binary search for $i_-(x)$ in the range $[i_-(x'), i_+(x'))$ (time $\mathcal{O}(\log |Y|)$)
  - analogously, "walk down" in A (non-empty int.) until first x'' w. $d(x'', y_{i_+(x'')-1}) < d(x, y_{i_+(x'')-1})$,
  - binary search for $i_+(x)$ in the range $[i_-(x''), i_+(x''))$ (time $\mathcal{O}(\log |Y|)$)

  update other intervals and heap

  - for nodes in A between x and x' (and between x and x''), interval becomes empty, "deactivate" and remove corresponding $\min$ in $Y \setminus M$ from heap (note that sequential search ("walk up/down") visits such nodes at most once, amortized $\mathcal{O}(\log |Y|)$ per $x \in X$)
  - for x' and x'', interval may shrink to $[i_-(x'), i_-(x))$ and $[i_+(x)+1, i_+(x''))$, respectively (could also become empty), update corresponding $\min$ in $Y \setminus M$ in heap (time $\mathcal{O}(\log |Y|)$)

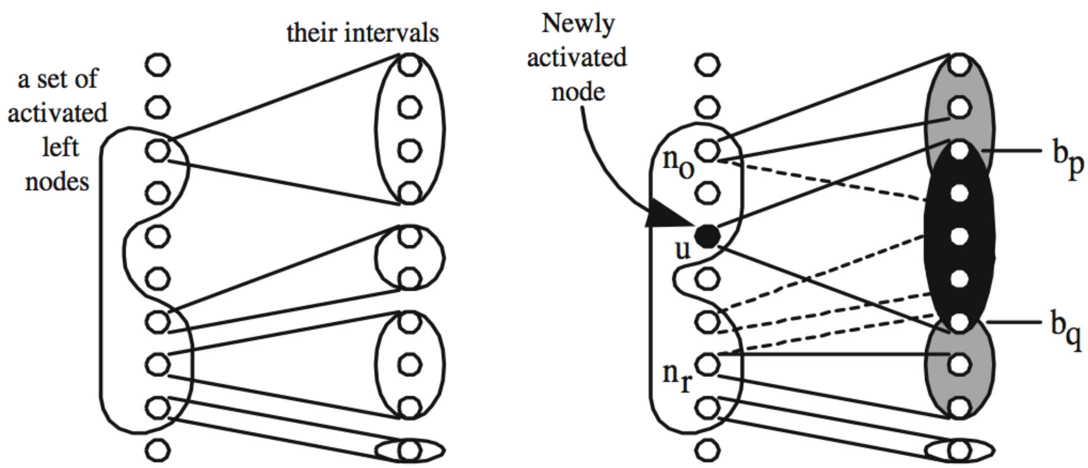

Figure 3: On-line bipartite Monge search

Efficient Dijkstra implementation

called *FR-Dijkstra* due to Fakcharoenphol and Rao

Dense distance graph complete graph on node set X, edge lengths correspond to shortest-path distances in some graph H, i.e. $\ell(x, x') := d_H(x, x')$. Often X is the boundary ∂P_u of a piece P_u and distances are with respect to P_u.

Reduction to bipartite case recursively "cut" X into halves, two bipartite graphs (one from left to right, another one from right to left), each node in $\leqslant 2\lceil \log_2 |X| \rceil$ bipartite graphs

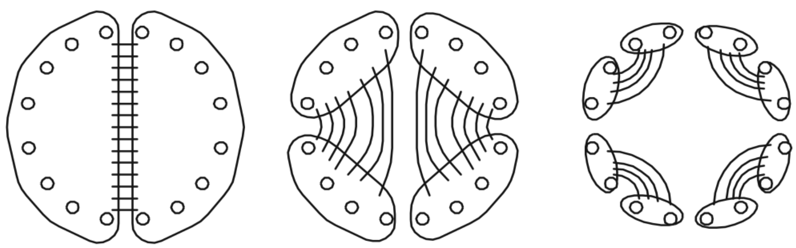

Figure 4: Reduction to bipartite Monge

Note Nodes are on the left-hand-side of some bipartite graphs, right-hand-side of other bipartite graphs. Each arc (directed edge) is in exactly one bipartite graph.

Algorithm compute SSSP tree on n nodes of a planar graph (connected by dense distance graphs) in time $\mathcal{O}(n \log^2 n)$ (note that there may be $\Omega(n^2)$ edges)
The FR-DIJKSTRA algorithm first converts each dense distance graph (DDG) into $2\lceil \log n \rceil$ bipartite DDGs (X_i, Y_i); then it runs on-line bipartite Monge searches for all the instances.
These instances are combined using a global priority queue with one element per instance (X_i, Y_i).
Each instance runs in time $\mathcal{O}((|X_i| + |Y_i|) \log(|X_i| + |Y_i|))$, the overall time is thus $\mathcal{O}(n \log^2 n)$.

DIJKSTRA(G, s)	FR-DIJKSTRA(G, s)
$\forall v \in V(G) : d(v) = \infty$; $d(s) = 0$; $S = \emptyset$	$\forall v \in V(G) : d(v) = \infty$; $d(s) = 0$; $S = \emptyset$
	convert each DDG into bipartite DDG**s** (X_i, Y_i)
	FOR EACH (X, Y) s.t. $s \in Y$
	(X, Y) . INSERT$(s, 0)$
maintain global heap with respect to $d(\cdot)$	GLOBAL . DECREASEKEY$((X, Y), 0)$
WHILE $S \neq V(G)$	WHILE $S \neq V(G)$
	$(X, Y) \leftarrow$ GLOBAL . EXTRACTMIN$()$
$u \leftarrow$ EXTRACTMIN$(V \setminus S)$	$u \leftarrow (X, Y)$. EXTRACTMIN$(Y \setminus M)$
	IF $u \notin S$ (could have been settled in $(\hat{X}, \hat{Y})$)
FOR EACH $e = (u, v) \in E(G)$	FOR EACH (X', Y') s.t. $u \in X'$ ("relax" edges in DDGs)
$d(v) \leftarrow \min\{d(v), d(u) + \ell(u, v)\}$ (relax e)	(X', Y') . ACTIVATELEFT$(u, d(u))$
	$v \leftarrow (X', Y')$. FINDMIN$(Y' \setminus M')$
	GLOBAL . DECREASEKEY$((X', Y'), d(v))$
$S = S \cup \{u\}$	$S = S \cup \{u\}$
	$v \leftarrow (X, Y)$. FINDMIN$(Y \setminus M)$
	GLOBAL . DECREASEKEY$((X, Y), d(v))$

Distance oracles

Quasi-linear space

Preprocessing recursively apply cycle separator and compute dense distance graph for inside and outside of cycle (using MSSP). Time per level is $\mathcal{O}(n \log n)$, recursion depth is $\mathcal{O}(\log n)$; overall preprocessing time $\mathcal{O}(n \log^2 n)$ and space requirement $\mathcal{O}(n \log n)$.

Query given query pair (v, w), run efficient implementation of Dijkstra's algorithm on dense distance graphs for all cycles enclosing v or w. Cycles on $\mathcal{O}(\sqrt{n})$ nodes, cycle lengths decrease geometrically with recursion, total $\mathcal{O}(\sqrt{n})$ nodes, overall query time $\mathcal{O}(\sqrt{n} \log^2 n)$.

"Arbitrary" space

Preprocessing for any $r \in [1, n]$, compute r–division, and, for each piece P,

- compute and store MSSP data structure for inside and outside of P, respectively,
  (at the same time: compute and store dense distance graphs $DDG(P)$ and $DDG(\bar{P})$ for inside and outside of P, respectively), and
- preprocess quasi-linear-space distance oracle for each piece.

Overall preprocessing $\mathcal{O}(n/r) \cdot \mathcal{O}(n \log n) + \mathcal{O}(n/r) \cdot \mathcal{O}(r \log^2 r)$ and space $\mathcal{O}((n^2/r) \log n)$.

Query given query pair (v, w),

- if in the same piece, query distance oracle in time $\mathcal{O}(\sqrt{r} \log^2 r)$
- in any case (even if in the same piece P, shortest path could leave and re-enter P),
  - query MSSP of P_v for v-to-∂P_v distances (in P_v),
  - query MSSP of $\bar{P}_v$ for ∂P_v-to-w distances (in $\bar{P}_v := (G \setminus P_v) \cup \partial P_v$),
  - run efficient Dijkstra implementation on $\{v\} \times \partial P_v$, $\partial P_v \times \{w\}$, $DDG(P_v)$, and $DDG(\bar{P}_v)$

Overall query time $\mathcal{O}(\sqrt{r} \log^2 r + \sqrt{r} \log n)$.

References

Exact distance oracles and shortest-path queries for planar graphs have been investigated by several researchers [FMS91, Dji96, ACC+96, CX00, FR06, Cab06, Nus11, MS12]. Djidjev [Dji96] first exploited the non-crossing property for pairs of boundary nodes. The efficient implementation of Dijkstra's algorithm (using on-line Monge search) is by Fakcharoenphol and Rao [FR06] (named *Monge* search due to [Mon81]). The quasi-linear-space construction is by [FR06]; the construction for arbitrary space can be found in [MS12].

[ACC+96] Srinivasa Rao Arikati, Danny Ziyi Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *4th European Symposium on Algorithms (ESA)*, pages 514–528, 1996.

[Cab06] Sergio Cabello. Many distances in planar graphs. In *17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1213–1220, 2006. A preprint of the journal version is available in the University of Ljubljana preprint series, Vol. 47 (2009), 1089.

[CX00] Danny Ziyi Chen and Jinhui Xu. Shortest path queries in planar graphs. In *32nd ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

[Dji96] Hristo Djidjev. Efficient algorithms for shortest path problems on planar digraphs. In *22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 151–165, 1996.

[FMS91] Esteban Feuerstein and Alberto Marchetti-Spaccamela. Dynamic algorithms for shortest paths in planar graphs. In *17th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 187–197, 1991.

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Announced at FOCS 2001.

[Mon81] Gaspard Monge. Mémoire sur la théorie des déblais et de remblais. *Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année*, pages 666–704, 1781.

[MS12] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012. To appear, preprint available online at http://arxiv.org/abs/1011.5549.

[Nus11] Yahav Nussbaum. Improved distance queries in planar graphs. In *12th International Symposium on Algorithms and Data Structures (WADS)*, pages 642–653, 2011.