Advanced Data Structures Jan-Apr 2012

Lecture 16 - March 22, 2012

Lecturer: Sudeshna Kolay *Scribe: Ashutosh Rai*

1 Overview

In the last lecture we talked about how to support dynamic graphs with respect to operations which included insertion and deletion of edges and looking for existence of path between two vertices. In this lecture we will talk about supporting all pair shortest paths along with addition, deletion and updation of edges in $O(n^2 \log^3 n)$ amortized time on the graphs with non negative real-valued edge weights.

2 Introduction

We need a dynamic graph algorithm which processes the queries quickly, and performs the update operations faster than recomputing the solution from scratch. We formulate edge deletion and edge additions as edge weight updates, setting the weights to $+\infty$ for the edges not in the graph.

2.1 Problem

We wish to maintain a directed graph supporting the following operations-

- *update* (v, w') – Update the weights of all edges incident to v according to the edge weight function $w' : E(G) \to \mathbb{R}_{\geq 0}$.
- *distance* (x, y) – Return the distance from x to y.
- *path* (x, y) – Report a shortest path from x to y, if there exists one.

2.2 Notations and Definitions

We call an algorithm *fully dynamic* if it can handle both edge weight increases and edge weight decreases and *partially dynamic* if it can handle edge weight increases or edge weight decreases, but not both. First we will show a partially dynamic algorithm and then extend it to a fully dynamic one.

Throughout the discussion, we will take n to be the number of vertices in the graph and m to be number of edges with weight $< +\infty$ in the graph.

Let $G = (V, E)$ be a directed graph with non-negative real edge weights.

- $\pi_{xy} = \langle x, \ldots, y \rangle$ represents a path from x to y.
- $\pi_{xv} \cdot \pi_{vy} = \langle x, \ldots, x', v, y', \ldots, v \rangle$ represents the path obtained by concatenating π_{xv} and π_{vy} at v.
- w_{uv} represents weight of the edge (u, v).
- $w(\pi_{xy})$ represents weight of the path π_{xy}.
- $l(\pi_{xy})$ represents a subpath π_{xb} of π_{xy} such that $\pi_{xy} = \pi_{xb} \cdot \langle b, y \rangle$.
- $r(\pi_{xy})$ represents a subpath π_{ay} of π_{xy} such that $\pi_{xy} = \langle x, a \rangle \cdot \pi_{ay}$.
- $\sum = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ represents a sequence of update operations.
- t_σ represents the time at which the update σ occurs.
- v_σ represents the vertex affected by update σ.

3 Locally Shortest Paths

We define locally shortest paths as follows-

Definition 1. *A path π_{xy} is* Locally Shortest *in G if either:*

- *π_{xy} consists of a single vertex or a single edge, or*
- *every proper subpath of π_{xy} is a shortest path in G.*

Now we prove a few properties abut locally shortest paths.

Lemma 2. *Let SP and LSP denote the set of shortest paths and locally shortest paths respectively, then $SP \subseteq LSP$.*

Proof. Statement follows from the fact that every shortest path is also a locally shortest path. □

Lemma 3. *If shortest paths are unique in G, then for each pair of vertices x and y, the locally shortest paths connecting x and y in G are internally vertex disjoint.*

Proof. Follows from the observation that if they are not disjoint, then we will get more than one shortest path between (x, v) or (v, y) for some internal vertex v, which is a contradiction to the assumption. □

Lemma 4. *If shortest paths are unique in G, then there can be at most mn locally shortest paths in G.*

Proof. It is easy to see that fixing an edge (u, v) and a vertex y, there can be at most one path starting from (u, v) and ending at vertex y. This follows from the fact that there is at most one shortest path from v to y, and that, along with the edge (u, v), uniquely determines a locally shortest path. □

Lemma 5. *Let G be a graph subject to a sequence Σ of vertex updates. If shortest paths are unique in G, then in the worst case at most $O(n^2)$ paths can stop being locally shortest due to an increase update.*

Proof. Let the vertex updated be v. From lemma 3, it is clear that for any pair of vertices (x, y), v can be an internal vertex of at most one locally shortest path between them. So, the number of locally shortest paths having v as internal vertex is $O(n^2)$. Again, from lemma 4, if we consider edges incident on v then each of them determines uniquely a locally shortest path along with one of the remaining vertices in G. So, the number of locally shortest paths starting from/ending at v is also bounded by $O(n^2)$. Hence, the total number of locally shortest paths passing through v is $O(n^2)$. We also observe that path can stop being locally shortest path by an "increase" update only if v occurs on that path. Hence, in the worst case at most $O(n^2)$ paths can stop being locally shortest due to an increase update. □

Theorem 6. *Let G be a graph subject to a sequence Σ of increase only vertex update operations and let m be the maximum number of edges in G throughout the sequence Σ. If shortest paths are unique in G, then the number of paths that start being locally shortest after each update is*

- *$O(mn)$ in worst case.*
- *$O(n^2)$ amortized over $\Omega(m/n)$ update operations.*

Proof. The first statement follows from lemma 4. For proving the second statement, let T be the total number of paths which were locally shortest paths at some point of time in the update sequence Σ. Since at any point of time the number of locally shortest paths is at most mn and during any update the number of paths which can stop being locally shortest paths is $O(n^2)$, we get

$$T - (m/n * cn^2) \leq mn$$

$$T \leq (c+1)mn$$

Which gives $O(n^2)$ newly created paths per operation amortized over $\Omega(m/n)$ update operations. This concludes the proof of the theorem. □

4 Partially Dynamic Shortest Paths

In this section, we will describe an algorithm which works on *increase-only* updates in $O(n^2 \log n)$ amortized time per operation. The algorithm can be easily modified to run for *decrease-only* updates in the same time complexity per operation. But in decrease-only case, it is not an improvement from a build-from-scratch algorithm, which takes $O(n^2)$ time as discussed in class.

4.1 Data Structures Needed

Now, we will describe the data structures needed for supporting partially dynamic shortest paths. For each pair of vertices x and y in G, we maintain the weight $w_{xy} \geq 0$ of an edge (x, y) (or $+\infty$ if no such edge exists and the following two data structures-

$P_{xy} = \{\pi_{xy} : \pi_{xy}$ is a locally shortest path in $G\}$

$P^*_{xy} = \{\pi_{xy} : \pi_{xy}$ is a shortest path in $G\}$

We maintain each of the P_{xy} as priority queue where the key value of an item $\pi_{xy} \in P_{xy}$ is $w(\pi_{xy})$. If shortest paths are unique, then $|P^*_{xy}| \leq 1$. Each path π_{xy} in P_{xy} and P^*_{xy} is stored as two pointers to $l(\pi_{xy})$ and $r(\pi_{xy})$ and hence take constant space. Notice that $l(\pi_{xy})$ and $r(\pi_{xy})$ are shortest paths as well as locally shortest paths, so they will also appear in the queues P_{xy} and P^*_{xy} and are stored in the same way implicitly.

Also, for each path $\pi_{xy} \in P_{xy}$, we maintain $w(\pi_{xy})$ and the following lists:

$L(\pi_{xy}) = \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} : (x', x) \in E$ and $\pi_{x'y}$ is a locally shortest path in $G\}$

$L^*(\pi_{xy}) = \{\pi_{x'y} = \langle x', x\rangle \cdot \pi_{xy} : (x', x) \in E$ and $\pi_{x'y}$ is a shortest path in $G\}$

$R(\pi_{xy}) = \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle : (y, y') \in E$ and $\pi_{xy'}$ is a locally shortest path in $G\}$

$R^*(\pi_{xy}) = \{\pi_{xy'} = \pi_{xy} \cdot \langle y, y'\rangle : (y, y') \in E$ and $\pi_{xy'}$ is a shortest path in $G\}$

4.2 Implementation

The $distance(x, y)$ and $path(x, y)$ operations can be implemented by accessing the minimum weight path in P_{xy}. The correctness follows from Lemma 2.

The operation $update(v, w')$ works in two stages, namely *cleanup* and $fixup$. The pseudocodes of these operations can be found in [1]. We will describe both the stages one by one.

4.2.1 *cleanup*

In this stage, we remove every path π_{xy} containing the vertex v from P_{xy}, P^*_{xy}, $L(r(\pi_{xy}))$, $L^*(r(\pi_{xy}))$, $R(l(\pi_{xy}))$ and $R^*(l(\pi_{xy}))$. By doing so, we want to remove all paths that would stop being locally shortest paths if we deleted v from the graph. To achieve this, we first remove the paths of the form $\langle u, v\rangle$ and $\langle v, u\rangle$ and then iteratively remove all paths listed in $L(\pi_{xy})$ and $R(\pi_{xy})$ for each path π_{xy} which has already been removed in previous iterations.

4.2.2 $fixup$

This stage adds *new* locally shortest paths and shortest paths to the data structure. We call a locally shortest path (or shortest path) *new*, if it was not locally shortest (or shortest) before the update or it contains the updated vertex v. The $fixup$ stage works in three phases:

Phase 1. In this phase, we update the edge weights of all the edges of the form (u, v) and (v, u) according to w', and add all the edges of the form (u, v) and (v, u) to P_{uv} and P_{vu} respectively.

Phase 2. Initialize a priority queue H with minimum edge path π_{xy} in P_{xy} for each pair of vertices (x, y).

Phase 3. We repeatedly extract paths π_{xy} from H in increasing order of their weights till $H = \phi$. For a pair (x, y), let the first occurrence of a path be π_{xy}, all the later occurrences of a path between

(x, y) are ignored. We see if $\pi_{xy} \in P^*_{xy}$. If not, we add it to P^*_{xy}, $L^*(r(\pi_{xy}))$ and $R^*(l(\pi_{xy}))$. Also, we combine π_{xy} with existing shortest paths to get new locally shortest paths and add them to P, L, R and H.

4.3 Analysis

To prove the correctness of *update*, we assume that P and P^* are correct before the operation, and we show that they are also correct afterwards. We first discuss an invariant maintained by procedure $fixup$.

Invariant 1. *If shortest paths are unique and edge weights are non-negative, then for each pair of vertices x and y in G, the first path connecting x and y extracted from H in Phase 3 of $fixup$ is a shortest path.*

Proof. Suppose that the statement of the invariant is not true. Let $\hat{\pi}_{xy}$ be the first such path extracted, such that even though it is the first path extracted for some $x, y \in V(G)$, it is not the shortest path between them. Let π_{xy} be the shortest path between x and y, such that $w(\pi_{xy}) < w(\hat{\pi}_{xy})$. Now, π_{xy} can not be in H, since then it would have been extracted earlier, hence neither it could have been in P_{xy} before the second phase, otherwise it would have been put in H in the second phase. Hence π_{xy} has to be necessarily a *new* locally shortest path. It can not have only one edge since all the locally shortest paths consisting of only one edge were added to P_{xy} in Phase 2. So, it has at least two edges and either one of $l(\pi_{xy})$ or $r(\pi_{xy})$ is a new shortest path and was not in P^* at the beginning of $fixup$. Since all edge weights are positive, $w(l(\pi_{xy})), w(r(\pi_{xy})) < w(\hat{\pi}_{xy})$. Also, both of them being new shortest paths, they should have been extracted from H and after combining them with respective vertices, π_{xy} should have been added back to H and hence extracted before $\hat{\pi}_{xy}$, which is a contradiction to the assumption made.

□

Theorem 7. *If the operation is an increase, shortest paths are unique and edge weights are non-negative, then algorithm update correctly updates P_{xy} and P^*_{xy} for each pair of vertices x and y.*

Proof. We have argued that *cleanup* removes every path from the data structures which can turn out to be not a locally shortest path after the update.

Now, we have to argue that every new locally shortest path is added back to the data structure by $fixup$. New locally shortest paths consisting of one edge are added in Phase 1, so we are left with only new shortest paths π_{xy} with at least two edges. Now, let us look at $l(\pi_{xy})$ or $r(\pi_{xy})$ which are shortest paths. If one of them is extracted from H at some iteration of Phase 3, then after combining with the other, π_{xy} will be added to P_{xy}. Also, if none of them are extracted, then they are part of P^* and hence do not involve v as a vertex. Since $l(\pi_{xy})$ and $r(\pi_{xy})$ do not involve v as a vertex, neither can π_{xy}, and hence it would not have been removed from P_{xy}. □

Theorem 8. *In an increase-only sequence of $\Omega(m/n)$ operations, if shortest paths are unique and edge weights are non-negative, our data structure supports each update operation in $O(n^2 \log n)$ amortized time, and each distance and path query in constant time.*

Proof. The statement about *distance* and *path* queries is obvious. Each iteration of *cleanup* removes a path π_{xy} from a constant number of lists. By lemma 5, at most $O(n^2)$ paths have to be removed, each taking $O(\log n)$ time each, hence the *cleanup* stage running in $O(n^2 \log n)$

Phase 1 of $fixup$ operation adds at most $O(n)$ edges to constant number of lists, and hence takes time $O(n \log n)$. In Phase 2, at most n^2 paths are inserted into H, which takes time $O(n^2 \log n)$. Also, in Phase 3, we perform insertions into the priorities queues for the new locally shortest paths, the number of which is $O(n^2)$ amortized over $\Omega(m/n)$ update operations. Hence, we spend $O(n^2 \log n)$ time in Phase 3 amortized over $\Omega(m/n)$ update operations. So, we have proved that the whole algorithm takes time $O(n^2 \log n)$ amortized over $\Omega(m/n)$ update operations. □

5 Resolving Ties

So far we have assumed that all the shortest paths are unique, but that is not generally the case. There can be more than one shortest path, in which case, when asked for shortest path, the algorithm can output any one of the paths with minimum weight. But the algorithm used here relied heavily on shortest paths being unique. In this section, we will show how to deal with this issue. We start with a few definitions.

Definition 9. *For any path π with at least one edge we define the extended weight of π as $ew(\pi) = \langle w(\pi), ID(\pi) \rangle$, where $w(\pi)$ is the real weight of π in the graph and $ID(\pi)$ is defined as follows-*

$$ID(\pi) = \begin{cases} u + nv, & \text{if } \pi = \langle u, v \rangle \\ \max\{ID(l(\pi)), ID(r(\pi))\}, & \text{otherwise} \end{cases}$$

We also define an ordering among the extended weights as following-

Definition 10. *$ew(\pi_1) \leq ew(\pi_2)$ if and only if either $w(\pi_1) < w(\pi_2)$ or $w(\pi_1) = w(\pi_2)$ and $ID(\pi_1) \leq ID(\pi_2)$*

Now, we define a notion of shortest extended weight paths analogous to the notion of shortest paths-

Definition 11. *Let G be a graph with real-valued edge weights and let ew be the extended weight function of Definition 9. We define S_{ew} as follows:*

$$S_{ew} = \{\pi \in G : \forall \pi_{xy} \subseteq \pi, \forall \pi'_{xy} \in G, ew(\pi_{xy}) \leq ew(\pi'_{xy})\}$$

The definition ensures the optimal substructure property for S_{ew}. Since we want the shortest extended weight paths to be unique, we prove the following lemma-

Lemma 12. *For each pair of vertices x and y in G, there can be at most one path $\pi_{xy} \in S_{ew}$ connecting them.*

Proof. We prove this by induction on number of edges in the paths. The base case is trivially true, since in a simple graph, there can be only one edge connecting two vertices, and there is only one path between them. Now, we assume that the claim holds for paths of length at most k, i.e. there

is at most one path of length at most k between any two vertices in S_{ew}. Now, let us assume that there are two different paths π_{xy} and π'_{xy} of length at most $k+1$ in S_{ew}. Since there extended weights are equal, their ID's are equal, and hence they must share an edge with that particular ID. Let the edge be (u, v). Now, because of the optimal substructure property and induction hypothesis, minimum paths from x to u and from v to y are unique, and must occur as subpaths of π_{xy} and π'_{xy}. Hence, π_{xy} and π'_{xy} are the same, which is a contradiction to our assumption. □

Also, it is clear to see that if $\pi_{xy} \in S_{ew}$, then π_{xy} is a shortest path in G. We prove the following lemma which guarantees existence of one shortest path for every pair of vertices in S_{ew}.

Lemma 13. *For each pair of connected vertices x and y in G, there is a path $\pi_{xy} \in S_{ew}$.*

Proof. We prove this by constructing a path π^*_{xy} that satisfies definition 11. Let π_{xy} be the minimum extended weight path. We start with an empty π^*_{xy} and look at the edge with the largest ID on π_{xy}. Let the edge be (u, v). We add (u, v) to π^*_{xy} and then recursively build π^*_{xu} and π^*_{vy}.

We prove that each path constructed by the above procedure belongs to S_{ew} by induction on the path length. The base case is trivially true for paths of length 1. Now, let us assume that all the paths of length k constructed belong to S_{ew}. Now, for the induction step, assume by contradiction that there is subpath $\pi_{ab} \in \pi^*_{xy}$ that is not of minimum extended weight and is of length at most $k+1$. There are three possibilities-

1. both vertices a and b are in π^*_{xu}
2. both vertices a and b are in π^*_{vy}
3. vertex a is in π^*_{xu} and vertex b is in π^*_{vy}

Cases 1 and 2 are not possible since π^*_{xu} and π^*_{vy} have at most k edges and hence by induction hypothesis belong to S_{ew}, the so will π_{ab} by Definition 11. Now, we look at the third case, and look at the minimum extended weight path between a and b, say π'_{ab}. But π^*_{xy} contains only the edges which lie on some shortest path from x to y in G, and hence $w(\pi_{ab}) \leq w(\pi'_{ab})$, but since π'_{ab} has less extended weight, we conclude that $w(\pi_{ab}) = w(\pi'_{ab})$ and $ID(\pi'_{ab}) < ID(\pi_{ab})$. So, we construct a new extended weight shortest path from x to y, avoiding (u, v) which has the same weight but smaller ID. This path differs from π_{xy} in the sense that for going from a to b (or from b to a), it uses π'_{ab} instead of π_{ab}. This path having less extended weight than π_{xy} is a contradiction to our assumption. This concludes proof of the lemma. □

Lemma 12 and Lemma 13 tell us that S_{ew} contains exactly one representative shortest path between each pair of vertices. Also, it is easy to see that the algorithm, namely the procedure *update* can be made to work with extended weights instead of actual weights and hence the assuming that shortest paths are unique is reasonable.

Now, to make use of this approach of extended weights to resolve the ties and have unique shortest paths, we keep an additional field ID for each path π stored in the data structure. Also, while assigning new weights to edges we assign the ID as $(u+nv)$ and while combining two paths, we just take the ID to be maximum of the two paths.Now, the priority of a path is the extended weight (combination of weight and ID instead of the original weights. Also, because of the weights and

ID's of paths being stored with them ,the extended weights can be compared in constant time. We conclude the analysis of resolving ties by the following theorem.

Theorem 14. *If we compare paths according to the extended weight function ew, then* $P* = S_{ew}$ *after each execution of update.*

Proof. Because of the original edge weights being non-negative it is easy to see that extended weights satisfy monotone property, i.e. $ew(l(\pi)) \leq ew(\pi)$ and $ew(r(\pi)) \leq ew(\pi)$. Also, by Lemma 12 and Lemma 13, there is exactly one path in S_{ew} for each pair of vertices. So, Invariant 1 continues to hold for extended weights, which implies that each path added to P^* has minimum extended weight. Also, p^* satisfies the optimal substructure property by construction. So, if π is in P, every subpath of π is also in P. Thus, $P* = S_{ew}$ by definition 11. □

References

[1] C. Demetrescu, G. F. Italiano, *A New Approach To Dynamic All Pair Shortest Paths*, Journal of the ACM, 51(6):968-992, November 2004.