

1

Randomized Graph Data-Structures for Approximate Shortest Paths

	1.1	Introduction.....	1-1
	1.2	A Randomized Data-Structure for Static APASP : Approximate Distance Oracles	1-2
		3-approximate Distance Oracle • Preliminaries • ($2k - 1$)-approximate Distance Oracle • Computing Approximate Distance Oracles	
	1.3	A Randomized Data-Structure for Decremental APASP	1-10
Surender Baswana <i>Indian Institute of Technology Delhi, INDIA</i>		Main Idea • Notations • Hierarchical Distance Maintaining Data-structure • Bounding the Size of $\mathbf{B}_u^{d,S}$ under Edge-deletions • Improved Decremental Algorithm for APASP up to Distance d	
Sandeep Sen <i>Indian Institute of Technology Delhi, INDIA</i>	1.4	Further Reading and Bibliography.....	1-17

1.1 Introduction

Let $G = (V, E)$ be an undirected weighted graph on $n = |V|$ vertices and $m = |E|$ edges. Length of a path between two vertices is the sum of the weights of all the edges of the path. The shortest path between a pair of vertices is the path of least length among all possible paths between the two vertices in the graph. The length of the shortest path between two vertices is also called the distance between the two vertices. An α -approximate shortest path between two vertices is a path of length at-most α times the length of the shortest path.

Computing all-pairs exact or approximate distances in G is one of the most fundamental graph algorithmic problem. In this chapter, we present two randomized graph data-structures for all-pairs approximate shortest paths (APASP) problem in static and dynamic environments. Both the data-structures are hierarchical data-structures and their construction involves random sampling of vertices or edges of the given graph.

The first data-structure is a randomized data-structure designed for efficiently computing APASP in a given static graph. In order to answer a distance query in constant time, most of the existing algorithms for APASP problem output a data-structure which is an $n \times n$ matrix that stores the exact/approximate distance between each pair of vertices explicitly. Recently a remarkable data-structure of $o(n^2)$ size has been designed for reporting all-pairs approximate distances in undirected graph. This data-structure is called *approximate distance oracle* because of its ability to answer a distance query in constant time in spite of

its sub-quadratic size. We present the details of this novel data-structure and an efficient algorithm to build it.

The second data-structure is a dynamic data-structure designed for efficiently maintaining APASP in a graph that is undergoing deletion of edges. For a given graph $G = (V, E)$ and a distance parameter $d \leq n$, this data-structure provides the first $o(nd)$ update time algorithm for maintaining α -approximate shortest paths for all pairs of vertices separated by distance $\leq d$ in the graph.

1.2 A Randomized Data-Structure for Static APASP : Approximate Distance Oracles

There exist classical algorithms that require $O(mn \log n)$ time for solving all-pairs shortest paths (APSP) problem. There also exist algorithms based on fast matrix multiplication that achieve sub-cubic time. However, there is still no combinatorial algorithm that could achieve $O(n^{3-\epsilon})$ running time for APSP problem. In recent past, many simple combinatorial algorithms have been designed that compute all-pairs approximate shortest paths (APASP) for undirected graphs. These algorithms achieve significant improvement in the running time compared to those designed for APSP, but the distance reported has some additive or/and multiplicative error. An algorithm is said to compute all pairs α -approximate shortest paths, if for each pair of vertices $u, v \in V$, the distance reported is bounded by $\alpha\delta(u, v)$, where $\delta(u, v)$ denotes the actual distance between u and v .

Among all the data-structures and algorithms designed for computing all-pairs approximate shortest paths, the approximate distance oracles are unique in the sense that they achieves simultaneous improvement in running time (sub-cubic) as well as space (sub-quadratic), and still answers any approximate distance query in constant time. For any $k \geq 1$, it takes $O(kmn^{1/k})$ time to compute $(2k - 1)$ -approximate distance oracle of size $O(kn^{1+1/k})$ that would answer any $(2k - 1)$ -approximate distance query in $O(k)$ time.

1.2.1 3-approximate Distance Oracle

For a given undirected graph, storing distance information from each vertex to all the vertices requires $\theta(n^2)$ space. To achieve sub-quadratic space, the following simple idea comes to mind.

\mathcal{I} : *From each vertex, if we store distance information to a small number of vertices, can we still be able to report distance between any pair of vertices ?*

The above idea can indeed be realized using a simple random sampling technique, but at the expense of reporting approximate, instead of exact, distance as an answer to a distance query. We describe the construction of 3-approximate distance oracle as follows.

1. Let $R \subset V$ be a subset of vertices formed by picking each vertex randomly independently with probability γ (the value of γ will be fixed later on).
2. For each vertex $u \in V$, store the distances to all the vertices of the sample set R .
3. For each vertex $u \in V$, let $p(u)$ be the vertex nearest to u among all the sampled vertices, and let S_u be the set of all the vertices of the graph G that lie closer to u than the vertex $p(u)$. Store the vertices of set S_u along with their distance from u .

For each vertex $u \in V$, storing distance to vertices S_u helps in answering distance query to vertices in locality of u , whereas storing distance from all the vertices of the graph to all

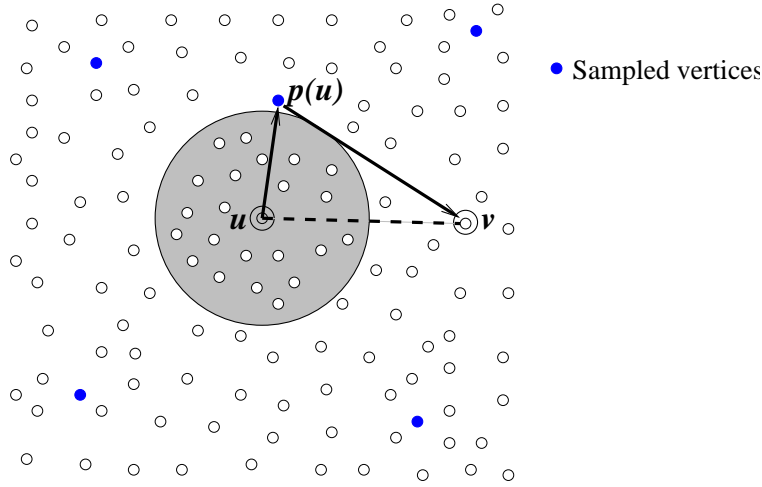


FIGURE 1.1: v is farther to u than $p(u)$, bounding $\delta(p(u), v)$ using triangle inequality

the sampled vertices will be required (as shown below) to answer distance query for vertices that are not present in locality of each other. In order to extract distance information in constant time, for each vertex $u \in V$, we use two *hash tables* (see [4], chapter 12), for storing distances from u to vertices of sets S_u and R respectively. The size of each hash-table is of the order of the size of corresponding set (S_u or R). A typical hash table would require $O(1)$ expected time to determine whether $w \in S_u$, and if so, report the distance $\delta(u, w)$. In order to achieve $O(1)$ worst case time, the *2-level hash table* (see Fredman, Komlos, Szemerédi, [8]) of optimal size is employed.

The collection of these hash-tables (two tables per vertex) constitute a data-structure that we call approximate distance oracle. Let $u, v \in V$ be any two vertices whose intermediate distance is to be determined approximately. If either u or v belong to set R , we can report exact distance between the two. Otherwise also exact distance $\delta(u, v)$ will be reported if v lies in S_u or vice versa. The only case, that is left, is when neither $v \in S_u$ nor $u \in S_v$. In this case, we report $\delta(u, p(u)) + \delta(v, p(u))$ as approximate distance between u and v . This distance is bounded by $3\delta(u, v)$ as shown below.

$$\begin{aligned}
 \delta(u, p(u)) + \delta(v, p(u)) &\leq \delta(u, p(u)) + (\delta(v, u) + \delta(u, p(u))) \quad \{\text{using triangle inequality}\} \\
 &= 2\delta(u, p(u)) + \delta(u, v) \quad \{\text{since graph is undirected}\} \\
 &\leq 2\delta(u, v) + \delta(u, v) \\
 &\quad \{\text{since } v \text{ lies farther to } u \text{ than } p(u), \text{ see Figure 1.1}\} \\
 &= 3\delta(u, v)
 \end{aligned}$$

Hence distance reported by the approximate distance oracle described above is no more than three times the actual distance between the two vertices. In other words, the oracle is a 3-approximate distance oracle. Now, we shall bound the expected size of the oracle. Using linearity of expectation, the expected size of the sample set R is $n\gamma$. Hence storing the distance from each vertex to all the vertices of sample set will take a total of $O(n^2\gamma)$ space. The following lemma gives a bound on the expected size of the sets $S_u, u \in V$.

LEMMA 1.1 Given a graph $G = (V, E)$, let $R \subset V$ be a set formed by picking each

vertex independently with probability γ . For a vertex $u \in V$, the expected number of vertices in the set S_u is bounded by $1/\gamma$.

Proof Let $\{v_1, v_2, \dots, v_{n-1}\}$ be the sequence of vertices of set $V \setminus \{u\}$ arranged in non-decreasing order of their distance from u . The set S_u consists of all those vertices of the set $V \setminus \{u\}$ that lie closer to u than any vertex of set R . Note that the vertex v_i belongs to S_u if none of the vertices of set $\{v_1, v_2, \dots, v_{i-1}\}$ (i.e., the vertices preceding v_i in the sequence above) is picked in the sample R . Since each vertex is picked independently with probability p , therefore the probability that vertex v_i belongs to set S_u is $(1 - \gamma)^{i-1}$. Using linearity of expectation, the expected number of vertices lying closer to u than any sampled vertex is

$$\sum_{i=1}^{n-1} (1 - \gamma)^{i-1} \leq \frac{1}{\gamma}$$

Hence the expected number of vertices in the set S_u is no more than $1/\gamma$.

So the total expected size of the 3-approximate distance oracle is $O(n^2\gamma + n/\gamma)$. Choosing $\gamma = 1/\sqrt{n}$ to minimize the size, we conclude that there is a 3-approximate distance oracle of expected size $n^{3/2}$.

1.2.2 Preliminaries

In the previous subsection, 3-approximate distance oracle was presented based on the idea \mathcal{I} . The $(2k - 1)$ -approximate distance oracle is a k -level hierarchical data-structure. An important construct of the data-structure is $Ball(\cdot)$ defined as follows.

DEFINITION 1.1 For a vertex u , and subsets $X, Y \subset V$, the set $Ball(u, X, Y)$ is the set consisting of all those vertices of the set X that lie closer to u than any vertex from set Y . (see Figure 1.2)

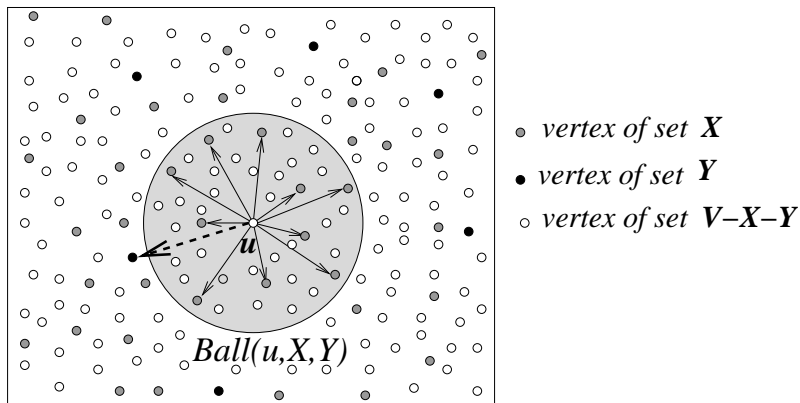


FIGURE 1.2: The vertices pointed by solid-arrows constitute $Ball(u, X, Y)$

It follows from the definition given above that $Ball(u, X, \emptyset)$ is the set X itself, whereas $Ball(u, X, X) = \emptyset$. It can also be seen that the 3-approximate distance oracle described in the previous subsection stores $Ball(u, V, R)$ and $Ball(u, R, \emptyset)$ for each vertex $u \in V$.

If the set Y is formed by picking each vertex of set X independently with probability γ , it follows from Lemma 1.1 that the expected size of $Ball(u, X, Y)$ is bounded by $1/\gamma$.

LEMMA 1.2 Let $G = (V, E)$ be a weighted graph, and $X \subset V$ be a set of vertices. If $Y \subset X$ is formed by selecting each vertex independently with probability γ , the expected number of vertices in $Ball(u, X, Y)$ for any vertex $u \in V$ is at-most $1/\gamma$.

1.2.3 $(2k - 1)$ -approximate Distance Oracle

In this subsection we shall give the construction of a $(2k - 1)$ -approximate distance oracle which is also based on the idea \mathcal{I} , and can be viewed as a generalization of 3-approximate distance oracle.

The $(2k - 1)$ -approximate distance oracle is obtained as follows.

1. Let $\mathcal{R}_k^1 \supset \mathcal{R}_k^2 \supset \dots \mathcal{R}_k^k$ be a hierarchy of subsets of vertices with $\mathcal{R}_k^1 = V$, and $\mathcal{R}_k^i, i > 1$ is formed by selecting each vertex of set \mathcal{R}_k^{i-1} independently with probability $n^{-1/k}$.
2. For each vertex $u \in V$, store the distance from u to all the vertices of $Ball(u, \mathcal{R}_k^k, \emptyset)$ in a hash table.
3. For each $u \in V$ and each $i < k$, store the vertices of $Ball(u, \mathcal{R}_k^i, \mathcal{R}_k^{i+1})$ along with their distance from u in a hash table.
For sake of conciseness and without causing any ambiguity in notations, henceforth we shall use $Ball^i(u)$ to denote $Ball(u, \mathcal{R}_k^i, \mathcal{R}_k^{i+1})$ or the corresponding hash-table storing $Ball(u, \mathcal{R}_k^i, \mathcal{R}_k^{i+1})$ for $i < k$.

The collection of the hash-tables $Ball^i(u) : u \in V, i \leq k$ constitute the data-structure that will facilitate answering of any approximate distance query in constant time. To provide a better insight into the data-structure, Figure 1.3 depicts the set of vertices constituting $\{Ball^i(u) | i \leq k\}$.

Reporting distance with stretch at-most $(2k - 1)$

Given any two vertices $u, v \in V$ whose intermediate distance has to be determined approximately. We shall now present the procedure to find approximate distance between the two vertices using the k -level data-structure described above.

Let $p^1(u) = u$ and let $p^i(u), i > 1$ be the vertex from the set \mathcal{R}_k^i nearest to u . Since $p^i(u) \in Ball^i(u)$ for each $u \in V$, so distance from each u to $p^i(u)$ is known for each $i \leq k$.

The query answering process performs at-most k search steps. In the first step, we search $Ball^1(u)$ for the vertex $p^1(v)$. If $p^1(v)$ is not present in $Ball^1(u)$, we move to the next level and in the second step we search $Ball^2(v)$ for vertex $p^2(u)$. We proceed in this way querying balls of u and v alternatively : In i th step, we search $Ball^i(x)$ for $p^i(y)$, where $(x = u, y = v)$ if i is odd, and $(x = v, y = u)$ otherwise. The search ends at i th step if $p^i(y)$ belongs to $Ball^i(x)$, and then we report $\delta(x, p^i(y)) + \delta(y, p^i(y))$ as an approximate distance between u and v .

can be bounded as follows

$$\begin{aligned}
\delta(y, p^{j+2}(y)) &= \delta(u, p^{j+2}(u)) \\
&\leq \delta(u, p^{j+1}(v)) \\
&\leq \delta(u, v) + \delta(v, p^{j+1}(v)) \quad \{\text{using triangle inequality}\} \\
&\leq \delta(u, v) + j\delta(u, v) \quad \{\text{using } \mathcal{A}_j \} \\
&= (j+1)\delta(u, v)
\end{aligned}$$

Thus the assertion \mathcal{A}_{j+1} holds.

THEOREM 1.1 *The algorithm `Distance_Report(u, v)` reports $(2k-1)$ -approximate distance between u and v*

Proof As an approximate distance between u and v , note that the algorithm `Distance_Report(u, v)` would output $\delta(y, p^l(y)) + \delta(x, p^l(y))$, which by triangle inequality is no more than $2\delta(y, p^l(y)) + \delta(x, y)$. Since $\delta(x, y) = \delta(u, v)$, and $\delta(y, p^l(y)) \leq (l-1)\delta(u, v)$ as follows from assertion \mathcal{A}_l . Therefore, the distance reported is no more than $(2l-1)\delta(u, v)$. Since the 'while loop' will execute at-most $k-1$ iterations, so $l = k$, and therefore the distance reported by the oracle is at-most $(2k-1)\delta(u, v)$.

Size of the $(2k-1)$ -approximate distance oracle

The expected size of the set R_k^k is $O(n^{1/k})$, and the expected size of each $Ball^i(u)$ is $n^{1/k}$ using Lemma 1.2. So the expected size of the $(2k-1)$ -approximate distance oracle is $O(n^{1/k} \cdot n + (k-1) \cdot n \cdot n^{1/k}) = O(kn^{1+1/k})$.

1.2.4 Computing Approximate Distance Oracles

In this subsection, a sub-cubic running time algorithm is presented for computing $(2k-1)$ -approximate distance oracles. It follows from the description of the data-structure associated with approximate distance oracle that after forming the sampled sets of vertices \mathcal{R}_k^i , that takes $O(m)$ time, all that is required is the computation of $Ball^i(u)$ along with the distance from u to the vertices belonging to these balls for each u and $i \leq k$.

Since $Ball^i(u)$ is the set of all the vertices of set R_k^i that lie closer to u than the vertex $p^{i+1}(u)$. So, in order to compute $Ball^i(u)$, first we compute $p^i(u)$ for all $u \in V, i \leq k$.

Computing $p^i(u), \forall u \in V$

Recall from definition itself that $p^i(u)$ is the vertex of the set R_k^i that is nearest to u . Hence, computing $p^i(u)$ for each $u \in V$ requires solving the following problem with $X = R_k^i, Y = V \setminus X$.

Given $X, Y \subset V$ in a graph $G = (V, E)$, with $X \cap Y = \emptyset$, compute the nearest vertex of set X for each vertex $y \in Y$.

The above problem can be solved by running a single source shortest path algorithm (Dijkstra's algorithm) on a modified graph as follows. Modify the original graph G by adding a dummy vertex s to the set V , and joining it to each vertex of the set X by an edge of zero weight. Let G' be the modified graph. Running Dijkstra's algorithm from the vertex s as the source, it can be seen that the distance from s to a vertex $y \in Y$ is

indeed the distance from y to the nearest vertex of set X . Moreover, if $e(s, x), x \in X$ is the edge leading to the shortest path from s to y , then x is the vertex from the set X that lies nearest to y . The running time of the Dijkstra's algorithm is $O(m \log n)$, we can thus state the following lemma.

LEMMA 1.3 Given $X, Y \subset V$ in a graph $G = (V, E)$, with $X \cap Y = \emptyset$, it takes $O(m \log n)$ to compute the nearest vertex of set X for each vertex $y \in Y$.

COROLLARY 1.1 Given a weighted undirected graph $G = (V, E)$, and a hierarchy of subsets $\{\mathcal{R}_k^i | i \leq k\}$, we can compute $p^i(u)$ for all $i \leq k, u \in V$ in $O(km \log n)$ time

Computing $Ball^i(u)$ efficiently

In order to compute $Ball^i(u)$ for each vertex $u \in V$ efficiently, we first compute clusters $\{C(v, \mathcal{R}_k^{i+1}) | v \in \mathcal{R}_k^i\}$ which are defined as follows :

DEFINITION 1.2 For a graph $G = (V, E)$, and a set $X \subset V$, the cluster $C(v, X)$ consists of each vertex $w \in V$ for whom v lies closer than any vertex of set X . That is, $\delta(w, v) < \delta(w, x)$ for each $x \in X$.

It follows from the definition given above that $u \in C(v, \mathcal{R}_k^{i+1})$ if and only if $v \in Ball^i(u)$. So, given clusters $\{C(v, \mathcal{R}_k^{i+1}) | v \in \mathcal{R}_k^i\}$, we can compute $\{Ball^i(u) : u \in V\}$ as follows.

For each $v \in \mathcal{R}_k^i$ **do**
 For each $u \in C(v, \mathcal{R}_k^{i+1})$ **do**
 $Ball^i(u) \leftarrow Ball^i(u) \cup \{v\}$

Hence we can state the following Lemma.

LEMMA 1.4 Given the family of clusters $\{C(v, \mathcal{R}_k^{i+1}) | v \in \mathcal{R}_k^i\}$, the time required to compute $\{Ball^i(u)\}$ is bounded by $O(\sum_{u \in V} |Ball^i(u)|)$.

The following property of the cluster $C(v, \mathcal{R}_k^{i+1})$ will be used in its efficient computation.

LEMMA 1.5 If $u \in C(v, \mathcal{R}_k^{i+1})$, then all the vertices on the shortest path from v to u also belong to the set $C(v, \mathcal{R}_k^{i+1})$.

Proof We give a proof by contradiction. Given that $u \in C(v, \mathcal{R}_k^{i+1})$, let w be any vertex on the shortest path from v to u . If $w \notin C(v, \mathcal{R}_k^{i+1})$, the vertex v doesn't lie closer to w than the vertex $p^{i+1}(w)$. See Figure 1.4. In other words $\delta(w, v) \geq \delta(w, p^{i+1}(w))$. Hence

$$\delta(u, v) = \delta(u, w) + \delta(w, v) \geq \delta(u, w) + \delta(w, p^{i+1}(w)) \geq \delta(u, p^{i+1}(w))$$

Thus v does not lie closer to u than $p^{i+1}(w)$ which is a vertex of set \mathcal{R}_k^{i+1} . Hence by definition, $u \notin C(v, \mathcal{R}_k^{i+1})$, thus a contradiction.

From Lemma 1.5, it follows that the graph induced by the vertices of the cluster $C(v, \mathcal{R}_k^{i+1})$ is connected (hence the name cluster). Moreover, the entire cluster $C(v, \mathcal{R}_k^{i+1})$ appears as

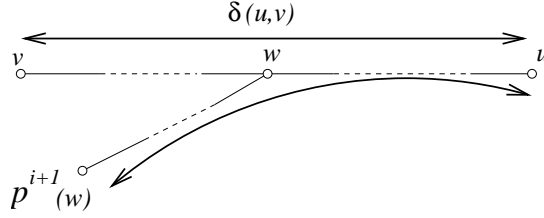


FIGURE 1.4: if w does not lie in $C(v, \mathcal{R}_k^{i+1})$, then $p^{i+1}(w)$ would lie closer to u than v .

a sub-tree of the shortest path tree rooted at v in the graph. As follows from the definition, for each vertex $x \in C(v, \mathcal{R}_k^{i+1})$, $\delta(v, x) < \delta(x, p^{i+1}(x))$. Based on these two observations, here follows an efficient algorithm that computes the set $C(v, \mathcal{R}_k^{i+1})$. The algorithm performs a *restricted* Dijkstra's algorithm from the vertex v , wherein we don't proceed along any vertex that does not belong to the set $C(v, \mathcal{R}_k^{i+1})$.

A restricted Dijkstra's algorithm : Note that the Dijkstra's algorithm starts with singleton tree $\{v\}$ and performs $n - 1$ steps to grow the complete shortest path tree. Each vertex $x \in V \setminus \{v\}$ is assigned a label $L(x)$, which is infinity in the beginning, but eventually becomes the distance from v to x . Let V_i denotes the set of i nearest vertices from v . The algorithm maintains the following invariant at the end of l th step :

$\mathcal{I}(l)$: For all the vertices of the set V_l , the label $L(x) = \delta(v, x)$, and for every other vertex $y \in V \setminus V_l$, the label $L(y)$ is equal to the length of the shortest path from v to y that passes through vertices of V_l only.

During the $(j + 1)$ th step, we select the vertex, say w from set $V - V_j$ with least value of $L(\cdot)$. Since all the edge weights are positive, it follows from the invariant $\mathcal{I}(j)$ that $L(w) = \delta(w, v)$. Thus we add w to set V_j to get the set V_{j+1} . Now in order to satisfy the invariant $\mathcal{I}(j + 1)$, we relax each edges $e(w, y)$ incident from w to a vertex $y \in V - V_{j+1}$ as follows : $L(y) \leftarrow \min\{L(y), L(w) + \text{weight}(w, y)\}$. It is easy to observe that this ensures the validity of the invariant $\mathcal{I}(j + 1)$.

In the restricted Dijkstra's algorithm, we will put the following restriction on relaxation of an edge $e(w, y)$: we relax the edge $e(w, y)$ only if $L(w) + \text{weight}(w, y)$ is less than $\delta(y, p^i(y))$. This will ensure that a vertex $y \notin C(v, \mathcal{R}_k^{i+1})$ will never be visited during the algorithm. The fact that the vertices of the cluster $C(v, \mathcal{R}_k^{i+1})$ form a sub-tree of the shortest path tree rooted at v , ensures that the above restricted Dijkstra's algorithm indeed finds all the vertices (along with their distance from v) that form the cluster $C(v, \mathcal{R}_k^{i+1})$. Since the running time of Dijkstra's algorithm is dominated by the number of edges relaxed, and each edge relaxation takes $\log(n)$ time only, therefore, the restricted Dijkstra's algorithm will run in time of the order of $\sum_{x \in C(v, \mathcal{R}_k^{i+1})} \text{degree}(x) \log n$. Thus the total time for computing all the clusters $\{C(v, \mathcal{R}_k^{i+1}) | v \in \mathcal{R}_k^i\}$ is given by :

$$\begin{aligned} \sum_{v \in \mathcal{R}_k^i, x \in C(v, \mathcal{R}_k^{i+1})} \text{degree}(x) \log n &= \left(\sum_{x \in V, v \in \text{Ball}^i(x)} \text{degree}(x) \right) \log n \\ &= \left(\sum_{x \in V} |\text{Ball}^i(x)| \cdot \text{degree}(x) \right) \log n \end{aligned}$$

By Lemma 1.2, the expected size of $\text{Ball}^i(x)$ is bounded by $n^{1/k}$, hence using linearity of

expectation, the total expected cost of computing $\{C(v, \mathcal{R}_k^{i+1}) | v \in \mathcal{R}_k^i\}$ is asymptotically bounded by

$$\sum_{x \in V} n^{1/k} \cdot \text{degree}(x) \log n = 2mn^{1/k} \log n$$

Using the above result and Lemma 1.4, we can thus conclude that for a given weighted graph $G = (V, E)$ and an integer k , it takes a total of $\tilde{O}(kmn^{1/k} \log n)$ time for computing $\{Ball^i(u) | i < k, u \in V\}$. If we use Fibonacci heaps instead of binary heaps in implementation of the restricted Dijkstra's algorithm, we can get rid of the logarithmic factor in the running time. Hence the total expected running time for building the data-structure is $O(kmn^{1/k})$. As mentioned before, the expected size of the data-structure will be $O(kn^{1+1/k})$. To get $O(kn^{1+1/k})$ bound on the worst case size of the data-structure, we repeat the preprocessing algorithm. The expected number of iterations will be just a constant. Hence, we can state the following theorem.

THEOREM 1.2 *Given a weighted undirected graph $G = (V, E)$ and an integer k , a data-structure of size $O(kn^{1+1/k})$ can be built in $O(kmn^{1/k})$ expected time so that given any pair of vertices, $(2k - 1)$ -approximate distance between them can be reported in $O(k)$ time.*

1.3 A Randomized Data-Structure for Decremental APASP

There are a number of applications that require efficient solutions of the APASP problem for a dynamic graph. In these applications, an initial graph is given, followed by an on-line sequence of queries interspersed with updates that can be insertion or deletion of edges. We have to carry out the updates and answer the queries on-line in an efficient manner. The goal of a dynamic graph algorithm is to update the solution efficiently after the dynamic changes, rather than having to re-compute it from scratch each time.

The approximate distance oracles described in the previous section can be used for answering approximate distance query in a static graph. However, there does not seem to be any efficient way to dynamize these oracles in order to answer distance queries in a graph under deletion of edges. In this section we shall describe a hierarchical data structure for efficiently maintaining APASP in an undirected unweighted graph under deletion of edges. In addition to maintaining approximate shortest paths for all-pairs of vertices, this scheme has been used for efficiently maintaining approximate shortest paths for pair of vertices separated by distance in an interval $[a, b]$ for any $1 \leq a < b \leq n$. However, to avoid giving too much details in this chapter, we would outline an efficient algorithm for the following problem only.

APASP- d : Given an undirected unweighted graph $G = (V, E)$ that is undergoing deletion of edges, and a distance parameter $d \leq n$, maintain approximate shortest paths for all-pairs of vertices separated by distance at-most d .

1.3.1 Main Idea

For an undirected unweighted graph $G = (V, E)$, a breadth-first-search (BFS) tree rooted at a vertex $u \in V$ stores distance information with respect to the vertex u . So in order to maintain shortest paths for all-pairs of vertices separated by distance $\leq d$, it suffices to maintain a BFS tree of depth d rooted at each vertex under deletion of edges. This is the

approach taken by the previously existing algorithms.

The main idea underlying the hierarchical data-structure that would provide efficient update time for maintaining APASP can be summarized as follows : Instead of maintaining exact distance information separately from each vertex, keep *small* BFS trees around each vertex for maintaining distance information within locality of each vertex, and some what *larger* BFS trees around *fewer* vertices for maintaining global distance information.

We now provide the underlying intuition of the above idea and a brief outline of the new techniques used.

Let B_u^d denote the BFS tree of depth d rooted at vertex $u \in V$. There exists a simple algorithm for maintaining a BFS tree B_u^d under deletion of edges that takes a total of $\mu(B_u^d) \cdot d$ time, where $\mu(t)$ is the number of edges in the graph induced by tree t . Thus the total update time for maintaining shortest path for all-pairs separated by distance at-most d is of the order of $\sum_{u \in V} \mu(B_u^d) \cdot d$. Potentially $\mu(B_u^d)$ can be as large as $\theta(m)$, and so the total update time over any sequence of edge deletions will be $O(mnd)$. Dividing this total update cost uniformly over the entire sequence of edge deletions, we can see that it takes $O(nd)$ amortized update time per edge deletion, and $O(1)$ time for reporting exact distance between any pair of vertices separated by distance at-most d .

In order to achieve $o(nd)$ bound on the update time for the problem APASP- d , we closely look at the expression of total update time $\sum_{u \in V} \mu(B_u^d) \cdot d$. There are n terms in this expression each of potential size $\theta(m)$. A decrease in either the total number of terms or the size of each term would give an improvement in the total update time. Thus the following simple ideas come to mind.

- Is it possible to solve the problem APASP- d by keeping very *few* depth- d BFS trees ?
- Is there some other alternative t for depth bounded BFS tree B_u^d that has $o(m)$ bound on $\mu(t)$?

While it appears difficult for any of the above ideas to succeed individually, they can be combined in the following way : *Build and maintain BFS trees of depth $2d$ on vertices of a set $S \subset V$ of size $o(n)$, called the set of special vertices, and for each remaining vertex $u \in V \setminus S$, maintain a BFS tree (denoted by B_u^S) rooted at u and containing all the vertices that lie closer to u than the nearest special vertex, say $\mathcal{N}(u, S)$.*

Along the above lines, we present a 2-level data-structure (and its generalization to k -levels) for the problem APASP- d .

It can be seen that unlike the tree B_u^d , the new BFS tree B_u^S might not contain all the vertices lying within distance d from u . In order to ensure that our scheme leads to a solution of problem APASP- d , we use the following observation similar to that of 3-approximate distance oracle in the previous section. If v is a vertex lying within distance d from u but not present in B_u^S , an *approximate* distance from u to v can be extracted from the tree rooted at the nearest special vertex $\mathcal{N}(u, S)$. This is because (by triangle inequality) the distance from $\mathcal{N}(u, S)$ to v is at most twice the distance from u to v .

For our hierarchical scheme to lead to improved update time, it is crucial that we establish sub-linear upper bounds on $\mu(B_u^S)$. We show that if the set S is formed by picking each vertex independently with *suitable* probability, then $\mu(B_u^S) = \tilde{O}(m/|S|)$ with probability arbitrarily close to 1.

1.3.2 Notations

For an undirected unweighted graph $G = (V, E)$, $S \subset V$, and a distance parameter $d \leq n$,

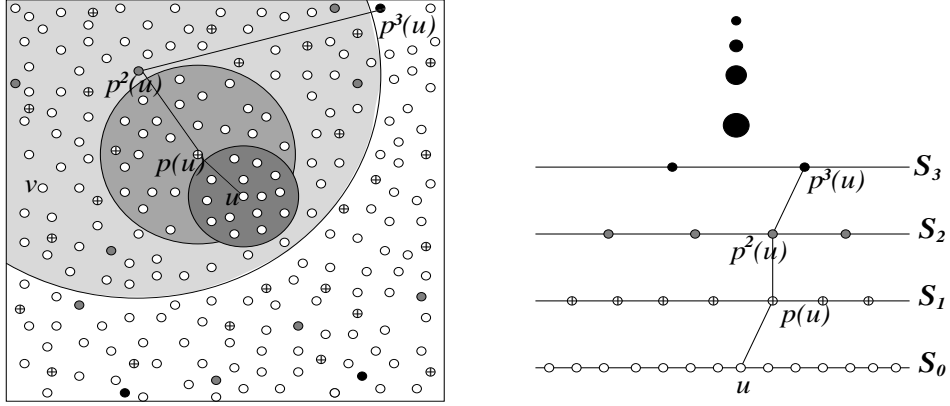


FIGURE 1.5: Hierarchical scheme for maintaining approximate distance

- $\delta(u, v)$: distance between u and v .
- $\mathcal{N}(v, S)$: the vertex of the set $S \subset V$ nearest to v .
- B_v^d : The BFS tree of depth d rooted at $v \in V$.
- B_v^S : The BFS tree of depth $(\delta(u, \mathcal{N}(u, S)) - 1)$ rooted at v
- $B_v^{d,S}$: The BFS tree of depth $\min\{d, \delta(v, \mathcal{N}(v, S)) - 1\}$ rooted at v .
- $\mu(t)$: the number of edges in the sub-graph (of G) induced by the tree t .
- $\nu(t)$: the number of vertices in tree t .
- For a sequence $\{S_0, S_1, \dots, S_{k-1}\}$, $S_i \subset V$, and a vertex $u \in S_0$, we define
 - $p^0(u) = u$.
 - $p^{i+1}(u)$ = the vertex from set S_{i+1} nearest to $p^i(u)$.
- $\bar{\alpha}$: the smallest integer of the form 2^i which is greater than α .

1.3.3 Hierarchical Distance Maintaining Data-structure

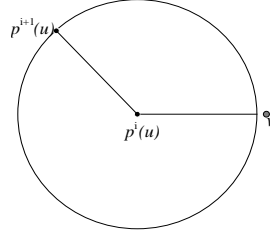
Based on the idea of “keeping *many small trees*, and a few *large trees*”, we define a k -level hierarchical data-structure for efficiently maintaining approximate distance information as follows. (See Figure 1.5)

Let $\mathcal{S} = \{S_0, S_1, \dots, S_{k-1} : S_i \subset V, |S_{i+1}| < |S_i|\}$ be a sequence. For a given distance parameter $d \leq n$ and $i < k - 1$, let \mathcal{F}_i be the collection $\{B_u^{2^i d, S_{i+1}} : u \in S_i\}$ of BFS trees, and \mathcal{F}_{k-1} be the collection of BFS trees of depth $2^{k-1}d$ rooted at each $u \in S_{k-1}$. We shall denote the set $\{(S_0, \mathcal{F}_0), (S_1, \mathcal{F}_1), \dots, (S_{k-1}, \mathcal{F}_{k-1})\}$ as the k -level hierarchy \mathcal{H}_d^k induced by the sequence \mathcal{S} .

Let v be a vertex within distance d from u . If v is present in B_u^{d, S_1} , we can report exact distance between them. Otherwise, (as will soon become clear) we can extract the approximate distance between u and v from the collection of the BFS trees rooted at the vertices $u, p(u), \dots, p^{k-1}(u)$ (see Figure 1.5). The following Lemma is the basis for estimating the distance between two vertices using the hierarchy \mathcal{H}_d^k .

LEMMA 1.6 Given a hierarchy \mathcal{H}_d^k , if $j < k - 1$ is such that v is not present in any of the BFS trees $\{B_{p^i(u)}^{2^i d, S_{i+1}} \mid 0 \leq i \leq j\}$, then for all $i \leq j$

$$\delta(p^{i+1}(u), p^i(u)) \leq 2^i \delta(u, v) \quad \text{and} \quad \delta(p^{i+1}(u), v) \leq 2^{i+1} \delta(u, v).$$

FIGURE 1.6: Bounding the approximate distance between $p^{i+1}(u)$ and v

Proof We give a proof by induction on j .

Base Case ($j = 0$) : Since v is not present in $B_u^{S_1}$, so the vertex $p(u)$ must be lying equidistant or closer to u than v . Hence $\delta(p(u), u) \leq \delta(u, v)$. Using triangle inequality, it follows that $\delta(p(u), v) \leq \delta(p(u), u) + \delta(u, v) = 2\delta(u, v)$.

Induction Hypothesis :

$\delta(p^{i+1}(u), p^i(u)) \leq 2^i \delta(u, v)$, and
 $\delta(p^{i+1}(u), v) \leq 2^{i+1} \delta(u, v)$, for all $i < l$.

Induction Step ($j = l$) : if $v \notin B_{p^l(u)}^{S_{l+1}}$, then the distance between p^{l+1} and $p^l(u)$ must not be longer than $\delta(p^l(u), v)$, which is less than $2^l \delta(u, v)$ (using induction hypothesis).

Now using triangle inequality (see the Figure 1.6) we can bound $\delta(p^{l+1}(u), v)$ as follows.

$$\begin{aligned} \delta(p^{l+1}(u), v) &\leq \delta(p^{l+1}(u), p^l(u)) + \delta(p^l(u), v) \\ &\leq 2^l \delta(u, v) + \delta(p^l(u), v) \\ &\leq 2^l \delta(u, v) + 2^l \delta(u, v) \quad \{ \text{using I.H.} \} \\ &= 2^{l+1} \delta(u, v) \end{aligned}$$

Since the depth of a BFS tree at $(k - 1)$ th level of hierarchy \mathcal{H}_d^k is $2^{k-1}d$, therefore the following corollary holds true.

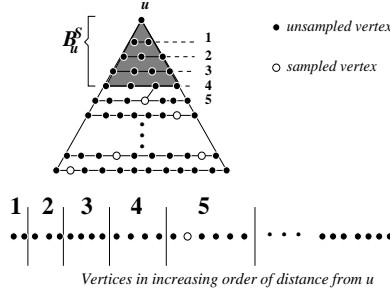
COROLLARY 1.2 If $\delta(u, v) \leq d$, then there is some $p^i(u), i < k$ such that v is present in the BFS tree rooted at $p^i(u)$ in the hierarchy \mathcal{H}_d^k .

LEMMA 1.7 Given a hierarchy \mathcal{H}_d^k , if $j < k - 1$ is such that v is not present in any of the BFS trees $\{B_{p^i(u)}^{2^i d, S_i} \mid 0 \leq i \leq j\}$, then $\delta(p^{i+1}(u), u) \leq (2^{i+1} - 1)\delta(u, v)$, for all $i \leq j$.

Proof Using simple triangle inequality, it follows that

$$\begin{aligned} \delta(p^{i+1}(u), u) &\leq \sum_{l \leq i} \delta(p^{l+1}(u), p^l(u)) \\ &\leq \sum_{l \leq i} 2^l \delta(u, v) = (2^{i+1} - 1)\delta(u, v) \end{aligned}$$

It follows from Lemma 1.6 and Lemma 1.7 that if l is the smallest integer such that v is present in the BFS tree rooted at $p^l(u)$ in the hierarchy \mathcal{H}_d^k , then we can report

FIGURE 1.7: Bounding the size of BFS tree B_u^S

$\delta(p^l(u), u) + \delta(p^l(u), v)$ as an approximate distance between u and v . Along these lines, we shall present an improved decremental algorithms for APASP- d .

1.3.4 Bounding the Size of $B_u^{d,S}$ under Edge-deletions

We shall now present a scheme based on random sampling to find a set $S \subset V$ of vertices that will establish a sub-linear bound on the number of vertices ($\nu(B_u^S)$) as well as the number of edges ($\mu(B_u^S)$) induced by B_u^S under deletion of edges. Since $B_u^{d,S} \subset B_u^S$, so these upper bounds also hold for $B_u^{d,S}$.

Build the set S of vertices by picking each vertex from V independently with probability $\frac{n^c}{n}$. The expected size of S is $O(n^c)$. Consider an ordering of vertices V according to their levels in the BFS tree B_u^S (see Figure 1.7). The set of vertices lying at higher levels than the nearest sampled vertex in this ordering is what constitutes the BFS tree B_u^S . Along similar lines as that of Lemma 1.1, it follows that the expected size of this set (and hence $\nu(B_u^S)$) is $\frac{n^c}{n}$. Moreover, it can be shown that $\nu(B_u^S)$ is no more than $\frac{4n \ln n}{n^c}$ with probability $> 1 - \frac{1}{n^4}$. Now as the edges are being deleted, the levels of the vertices in the tree B_u^S may fall, and so the ordering of the vertices may change. There will be a total of m such orderings during the entire course of edge deletions. Since the vertices are picked randomly and independently, therefore, the upper bound of $\frac{4n \ln n}{n^c}$ holds for $\nu(B_u^S)$ with probability $(1 - \frac{1}{n^4})$ for any of these orderings. So we can conclude that $\nu(B_u^S)$, the number of vertices of tree B_u^S never exceeds $(\frac{4n \ln n}{n^c})$ during the entire course of edge deletions with probability $> 1 - \frac{1}{n^4}$.

To bound the number of edges induced by B_u^S , consider the following scheme. Pick every edge independently with probability $\frac{n^c}{m}$. The set S consists of the end points of the sampled edges. The expected size of S is $O(n^c)$. Consider an ordering of the edges according to their level in B_u^S (level of an edge is defined as the minimum of the levels of its end points). Along the lines of arguments given above (for bounding the the number of vertices of B_u^S), it can be shown that $\mu(B_u^S)$, the number of edges induced by B_u^S remains $\leq \frac{4m \ln n}{n^c}$ with probability $> 1 - \frac{1}{n^4}$ during the entire course of edge deletions.

Note that in the sampling scheme to bound the number of vertices of tree B_u^S , a vertex v is picked with probability $\frac{n^c}{n}$. Whereas in the sampling scheme for bounding the number of edges in the sub-graph induced by B_u^S , a vertex v is picked with probability $\frac{\text{degree}(v) \cdot n^c}{m}$. It can thus be seen that both the bounds can be achieved simultaneously by the following random sampling scheme :

$$\mathcal{R}(c) : \text{Pick each vertex } v \in V \text{ independently with probability } \frac{n^c}{n} + \frac{\text{degree}(v) \cdot n^c}{m}.$$

It is easy to see that the expected size of the set formed by the sampling scheme $\mathcal{R}(c)$ will be $O(n^c)$.

THEOREM 1.3 Given an undirected unweighted graph $G = (V, E)$, a constant $c < 1$, and a distance parameter d ; a set S of size $O(n^c)$ vertices can be found that will ensure the following bound on the number of vertices and number of edges in the sub-graph of G induced by $B_u^{d,S}$.

$$\nu(B_u^{d,S}) = O\left(\frac{n \ln n}{n^c}\right), \quad \mu(B_u^{d,S}) = O\left(\frac{m \ln n}{n^c}\right)$$

with probability $\Omega(1 - \frac{1}{n^2})$ during the entire sequence of edge deletions.

Maintaining the BFS tree $B_u^{d,S}$ under edge deletions

Even and Shiloach [7] design an algorithm for maintaining a depth- d BFS tree in an undirected unweighted graph.

LEMMA 1.8 [Even, Shiloach [7]] Given a graph under deletion of edges, a BFS tree $B_u^d, u \in V$ can be maintained in $O(d)$ amortized time per edge deletion.

For maintaining a $B_u^{d,S}$ tree under edge deletions, we shall use the same algorithm of [7] with the modification that whenever the depth of $B_u^{d,S}$ has to be increased (due to recent edge deletion), we grow the tree to its new level $\min\{d, \delta(u, \mathcal{N}(u, S)) - 1\}$. We analyze the total update time required for maintaining $B_u^{d,S}$ as follows.

There are two computational tasks : one extending the level of the tree, and another that of maintaining the levels of the vertices in the tree $B_u^{d,S}$ under edge deletions. For the first task, the time required is bounded by the edges of the new level introduced which is $O(\mu(B_u^{d,S}))$. For the second task, we give a variant of the proof of Even and Shiloach [7] (for details, please refer [7]). The running time is dominated by the processing of the edges in this process. In-between two consecutive processing of an edge, level of one of the end-points of the edge falls down by at least one unit. The processing cost of an edge can thus be charged to the level from which it has fallen. Clearly the maximum number of edges passing a level i is bounded by $\mu(B_u^{d,S})$. The number of levels in the tree $B_u^{d,S}$ is $\min\{d, \nu(B_u^{d,S})\}$. Thus the total cost for maintaining the BFS tree $B_u^{d,S}$ over the entire sequence of edge deletions is $O(\mu(B_u^{d,S}) \cdot \min\{d, \nu(B_u^{d,S})\})$.

LEMMA 1.9 Given an undirected unweighted graph $G = (V, E)$ under edge deletions, a distance parameter d , and a set $S \subset V$; a BFS tree $B_u^{d,S}$ can be maintained in

$$O\left(\frac{\mu(B_u^{d,S})}{m} \cdot \min\{d, \nu(B_u^{d,S})\}\right)$$

amortized update time per edge deletion.

Some technical details

As the edges are being deleted, we need an efficient mechanism to detect any increase in the depth of tree $B_u^{d,S}$. We outline one such mechanism as follows.

For every vertex $v \notin S$, we keep a count $C[v]$ of the vertices of the S that are neighbors of v . It is easy to maintain $C[u], \forall u \in V$ under edge-deletions. We use the count $C[v]$ in order to detect any increase in the depth of a tree $B_u^{d,S}$ as follows. Note that when depth of a tree $B_u^{d,S}$ is less than d , there has to be at-least one vertex w at leaf-level in $B_u^{d,S}$ with $C[w] \geq 1$ (as an indicator that the vertex $p(u)$ is at next level). Therefore, after an edge deletion if

there is no vertex w at leaf level with $C[w] \geq 1$, we grow the BFS tree $B_u^{d,S}$ beyond its previous level until either depth becomes d or we reach some vertex w' with $C[w'] \geq 1$.

Another technical issue is that when an edge $e(x, y)$ is deleted, we must update only those trees which contain x and y . For this purpose, we maintain for each vertex, a set of roots of all the BFS trees containing it. We maintain this set using any dynamic search tree.

1.3.5 Improved Decremental Algorithm for APASP up to Distance d

Let $\{(S_0, \mathcal{F}_0), (S_1, \mathcal{F}_1), \dots, (S_{k-1}, \mathcal{F}_{k-1})\}$ be a k -level hierarchy \mathcal{H}_d^k with $S_0 = V$ and $n^{c_i} = |S_i|$, where each $c_i, i < k$ is a fraction to be specified soon. Each set $S_i, i > 0$ is formed by picking the vertices from set V using the random sampling scheme \mathcal{R} mentioned in the previous subsection.

To report distance from u to v , we start from the level 0. We first inquire if v lies in B_u^{d,S_1} . If v does not lie in the tree, we move to the first level and inquire if v lies in $B_{p(u)}^{2d,S_2}$. It follows from the Corollary 1.2 that if $\delta(u, v) \leq d$, then proceeding in this way, we eventually find a vertex $p^l(u), l \leq k-1$ in the hierarchy \mathcal{H}_d^k such that v is present in the BFS tree rooted at $p^l(u)$. (See Figure 1.5). We then report the sum of distances from $p^l(u)$ to both u and v .

Algorithm for reporting approximate distance using \mathcal{H}_d^k

```

Distance( $u, v$ )
{
   $D \leftarrow 0; l \leftarrow 0$ 
  While ( $v \notin B_{p^l(u)}^{2^l d, S_{l+1}} \wedge l < k-1$ ) do
  {
    If  $u \in B_{p^l(u)}^{2^l d, S_{l+1}}$ , then  $D \leftarrow \delta(p^l(u), u)$ ,
     $D \leftarrow D + \delta(p^l(u), p^{l+1}(u))$ 
     $l \leftarrow l + 1$ ;
  }
  If  $v \notin B_{p^l(u)}^{2^l d, S_{l+1}}$ , then " $\delta(u, v)$  is greater than  $d$ ",
  else return  $\delta(p^l(u), v) + D$ 
}

```

The approximation factor ensured by the above algorithm can be bounded as follows.

It follows from the Lemma 1.7 that the final value of D in the algorithm given above is bounded by $(2^l - 1)\delta(u, v)$, and it follows from Lemma 1.6 that $\delta(p^l(u), v)$ is bounded by $2^l \delta(u, v)$. Since $l \leq k-1$, therefore the distance reported by the algorithm is bounded by $(2^k - 1)\delta(u, v)$ if v is at distance $\leq d$.

LEMMA 1.10 Given an undirected unweighted graph $G = (V, E)$, and a distance parameter d . If α is the desired approximation factor, then there exists a hierarchical scheme \mathcal{H}_d^k with $k = \log_2 \alpha$, that can report α -approximate shortest distance between any two vertices separated by distance $\leq d$, in time $O(k)$.

Update time for maintaining the hierarchy \mathcal{H}_k^d : The update time per edge deletion for maintaining the hierarchy \mathcal{H}_k^d is the sum total of the update time for maintaining the set of BFS trees $\mathcal{F}_i, i \leq k-1$.

Each BFS tree from the set \mathcal{F}_{k-1} has depth $2^{k-1}d$, and edges $O(m)$. Therefore, using Lemma 1.8, each tree from set \mathcal{F}_{k-1} requires $O(2^{k-1}d)$ amortized update time per edge deletion. So, the amortized update time T_{k-1} per edge deletion for maintaining the set \mathcal{F}_{k-1} is

$$T_{k-1} = O(n^{c_{k-1}} 2^{k-1}d)$$

It follows from the Theorem 1.3 that a tree t from a set $\mathcal{F}_i, i < (k-1)$, has $\mu(t) = m \ln n/n^{c_{i+1}}$, and depth $= \min\{2^i d, n \ln n/n^{c_{i+1}}\}$. Therefore, using the Lemma 1.9, each tree $t \in \mathcal{F}_i, i < k-1$ requires $O(\min\{2^i d/n^{c_{i+1}}, n \ln n/n^{2c_{i+1}}\})$ amortized update time per edge deletion. So the amortized update time T_i per edge deletion for maintaining the set \mathcal{F}_i is

$$T_i = O\left(\min\left\{2^i d \frac{n^{c_i}}{n^{c_{i+1}}} \ln n, \frac{n^{1+c_i}}{n^{2c_{i+1}}} \ln^2 n\right\}\right), \quad i < k-1$$

Hence, the amortized update time T per edge deletion for maintaining the hierarchy \mathcal{H}_k^d is

$$\begin{aligned} T &= T_{k-1} + \sum_{i < k-1} T_i \\ &= O(n^{c_{k-1}} 2^{k-1}d) + \sum_{i=0}^{i=k-2} O\left(\min\left\{2^i d \frac{n^{c_i}}{n^{c_{i+1}}} \ln n, \frac{n^{1+c_i}}{n^{2c_{i+1}}} \ln^2 n\right\}\right) \end{aligned}$$

To minimize the sum on right hand side in the above equation, we balance all the terms constituting the sum, and get

$$T = \tilde{O}\left(2^{k-1} \cdot \min\left\{\sqrt[k]{nd}, (nd)^{\frac{2^{(k-1)}}{2^{k-1}-1}}\right\}\right)$$

If α is the desired approximation factor, then it follows from Lemma 1.10 that the number of levels k , in the hierarchy are $\log_2 \bar{\alpha}$. So the amortized update time required is $\tilde{O}(\alpha \cdot \min\{\sqrt[\log_2 \bar{\alpha}]{nd}, (nd)^{\frac{2}{2(\bar{\alpha}-1)}}\})$.

THEOREM 1.4 *Let $G = (V, E)$ be an undirected unweighted graph undergoing edge deletions, d be a distance parameter, and $\alpha > 2$ be the desired approximation factor. There exists a data-structure $\tilde{D}_\alpha(1, d)$ for maintaining α -approximate distances for all-pairs separated by distance $\leq d$ in $\tilde{O}(\alpha \cdot \min\{\sqrt[\log_2 \bar{\alpha}]{nd}, (nd)^{\frac{2}{2(\bar{\alpha}-1)}}\})$ amortized update time per edge deletion, and $O(\log \bar{\alpha})$ query time.*

Based on the data-structure of [7], the previous best algorithm for maintaining all-pairs exact shortest paths of length $\leq d$ requires $O(nd)$ amortized update time. We have been able to achieve $o(nd)$ update time at the expense of introducing approximation as shown in Table 1.1 on the following page.

1.4 Further Reading and Bibliography

Zwick [10] presents a very recent and comprehensive survey on the existing algorithms for all-pairs approximate/exact shortest paths. Based on the fastest known matrix multiplication algorithms given by Coppersmith and Winograd [3], the best bound for computing all-pairs shortest paths is $O(n^{2.575})$ [11].

Data-structure	α (the approximation factor)	Amortized update time per edge deletion
$\dot{D}_3(1, d)$	3	$\tilde{O}(\min(\sqrt{nd}, (nd)^{2/3}))$
$\dot{D}_7(1, d)$	7	$\tilde{O}(\min(\sqrt[3]{nd}, (nd)^{4/7}))$
$\dot{D}_{15}(1, d)$	15	$\tilde{O}(\min(\sqrt[4]{nd}, (nd)^{8/15}))$

TABLE 1.1 Maintaining α -approximate distances for all-pairs of vertices separated by distance $\leq d$

Approximate distance oracles are designed by Thorup and Zwick [9]. Based on a 1963 girth conjecture of Erdős [6], they also show that $\Omega(n^{1+1/k})$ space is needed in the worst case for any oracle that achieves stretch strictly smaller than $(2k + 1)$. The space requirement of their approximate distance oracle is, therefore, essentially optimal. Also note that the preprocessing time of $(2k - 1)$ -approximate distance oracle is $O(mn^{1/k})$, which is sub-cubic. However, for further improvement in the computation time for approximate distance oracles, Thorup and Zwick pose the following question : *Can $(2k - 1)$ -approximate distance oracle be computed in $\tilde{O}(n^2)$ time?* Recently Baswana and Sen [2] answer their question in affirmative for unweighted graphs. However, the question for weighted graphs is still open.

For maintaining fully dynamic all-pairs shortest paths in graphs, the best known algorithm is due to Demetrescu and Italiano [5]. They show that it takes $O(n^2)$ amortized time to maintain all-pairs exact shortest paths after each update in the graph. Baswana et al. [1] present a hierarchical data-structure based on random sampling that provides efficient decremental algorithm for maintaining APASP in undirected unweighted graphs. In addition to achieving $o(nd)$ update time for the problem APASP- d (as described in this chapter), they also employ the same hierarchical scheme for designing efficient data-structures for maintaining approximate distance information for all-pairs of vertices separated by distance in an interval $[a, b]$, $1 \leq a < b \leq n$.

Acknowledgment

The work of the first author is supported, in part, by a fellowship from Infosys Technologies Limited, Bangalore.

References

- [1] Baswana, S., Hariharan R., and Sen, S., Maintaining all-pairs approximate shortest paths under deletion of edges, in *Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, 394.
- [2] Baswana, S. and Sen S., Approximate distance oracle for unweighted graphs in $\tilde{O}(n^2)$ time, to appear in *Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
- [3] Coppersmith, D. and Winograd, S., Matrix multiplication via arithmetic progressions, *J. symbolic computation*, 9, 251, 1990.
- [4] Cormen, T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms*, the MIT Press, 1990, chapter 12.
- [5] Demetrescu, C., and Italiano, G.F., A new approach to dynamic all pairs shortest

- paths, in *Proc. of 35th ACM Symposium on Theory of Computing (STOC)*, 2003, 159.
- [6] Erdős, P., Extremal problems in graph theory, *Theory of Graphs and its Applications* (Proc. Sympos. Smolenice,1963), Publ. House Czechoslovak Acad. Sci., Prague. 1964, 29.
 - [7] Even, S. and Shiloach, Y., An on-line edge-deletion problem, *J. ACM*, 28, 1, 1981.
 - [8] Fredman, M.L., Komlós, J., and Szemerédi, E., Storing a sparse table with $O(1)$ worst case time, *J. ACM*, 31, 538, 1984.
 - [9] Thorup, M. and Zwick, U., Approximate distance oracles, in *Proc. of 33rd ACM symposium on theory of computing (STOC)*, 2001, 183.
 - [10] Zwick, U., Exact and approximate distances in graphs - a survey, in *Proc. of the 9th European Symposium on Algorithms (ESA)*, 2001, 33.
 - [11] Zwick, U., All-pairs shortest paths in weighted directed graphs - exact and almost exact algorithms, in *Proc. of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1998, 310.

Index

Ball(·), 1-4

approximate distance oracle, 1-2

approximate shortest path, 1-1

BFS tree, 1-10

decremental, 1-10

Dijkstra's algorithm, 1-7

distance, 1-1

dynamic, 1-10–1-18

hash table, 1-3

induction, 1-13

random sampling, 1-2

randomized, 1-1

shortest path tree, 1-9