I N F O R M A T I K

Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs

Surender Baswana   Kavitha Telikepalli

MPI–I–2005–1–003   September 2005

FORSCHUNGSBERICHT   RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Stuhlsatzenhausweg 85   66123 Saarbrücken   Germany

Authors' Addresses

Surender Baswana
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany.
Email : sbaswana@mpi-sb.mpg.de

Telikepalli Kavitha
Department of Computer Sc. & Automation
Indian Institute of Science
Bangalore 560012, India.
E-mail : kavitha@csa.iisc.ernet.in

Abstract

The all-pairs approximate shortest-paths problem is an interesting variant of the classical all-pairs shortest-paths problem in graphs. The problem aims at building a data-structure for a given graph with the following two features. Firstly, for any two vertices, it should report an *approximate* shortest path between them, that is, a path which is longer than the shortest path by some *small* factor. Secondly, the data-structure should require less preprocessing time (strictly sub-cubic) and occupy optimal space (sub-quadratic), at the cost of this approximation.

In this paper, we present algorithms for computing all-pairs approximate shortest paths in a weighted undirected graph. These algorithms significantly improve the existing results for this problem.

Keywords

Shortest Path, Distance, Spanner

1 Introduction

The all-pairs shortest paths problem is one of the most fundamental algorithmic graph problems. This problem is commonly phrased as follows : *Given a graph G on n vertices and m edges, compute shortest-paths/distances between each pair of vertices.* In many applications the aim is not to compute *all* distances, but to have a mechanism (data structure) through which we can extract the distance/shortest-path for any pair of vertices efficiently. Therefore, the following is a useful alternate formulation of the APSP problem.

Preprocess a given graph efficiently to build a data structure that can answer a shortest-path query or a distance query for any pair of vertices.

Throughout this paper, we stick to the above formulation of the APSP problem. The objective is to construct a data structure for this problem such that it is efficient both in terms of the space and the preprocessing time. There is a lower bound of $\Omega(n^2)$ on the space requirement of any data structure for the APSP problem, and space requirement of all the existing algorithms for APSP match this bound. However, there is a huge gap in the preprocessing time. In its most generic version, that is, for directed graph with real edge weights, the best known algorithm [10] for APSP requires $O(mn + n^2 \log\log n)$ time. However, for graphs with $m = \theta(n^2)$, this algorithm has a running time of $\theta(n^3)$ which matches that of the old and classical algorithm of Floyd and Warshal. In fact the best known upper bound on the worst case time complexity of this problem is $O(n^3/\log n)$ due to Chan [5], which is marginally sub-cubic. Surprisingly, despite the fundamental importance of the problem, there does not exist any algorithm at present that can solve APSP in truly sub-cubic time, i.e., $O(n^{3-\epsilon})$ for some $\epsilon > 0$. The existing lower bound on the time complexity of APSP is the trivial lower bound of $\Omega(n^2)$. This has motivated researchers to explore ways to achieve (truly) sub-cubic preprocessing time and/or sub-quadratic space data structures that can report *approximate* instead of exact shortest-paths/distances.

Let $\delta(u, v)$ denote the distance between vertices u and v in a given graph. A data structure is said to compute t-approximate distances for all-pairs of vertices, if for any pair of vertices $u, v \in V$, the distance that it would report, is at least $\delta(u, v)$ and not greater than $t\delta(u, v)$. A data structure for all-pairs t-approximate shortest paths is defined in a similar way. In the last ten years, many novel algorithms have been designed for the all-pairs approximate shortest paths (APASP) problem for undirected graphs that achieve sub-cubic running time and/or sub-quadratic space. This paper presents significantly improved algorithms for the APASP problem in weighted graphs. The new algorithms are either based on a new scheme for APASP or they involve a more careful analysis of simple modifications of the existing algorithms. We summarize below our new results for the APASP problem and provide comparisons with the existing results.

1. *All-pairs* $(2, \omega)$*-approximate shortest paths* :

It is well known (see [7]) that the all-pairs $(2 - \epsilon)$-approximate shortest paths problem for any arbitrary $\epsilon > 0$ is as hard as boolean matrix multiplication. The only algorithm for computing all-pairs 2-approximate shortest paths has a running time $\tilde{O}(n^{3/2}\sqrt{m})$, and is given by Cohen and Zwick [6]. For dense graphs, their algorithm would require $\tilde{O}(n^{2.5})$ in the worst case. In this paper we present the first algorithm that computes *nearly* 2-approximate shortest paths in $O(n^2 \log n)$ time. For any two vertices $u, v \in V$, the distance reported by the algorithm is bounded by $2\delta(u, v) + \omega_{uv}$, where ω_{uv} is the weight of the heaviest edge on the shortest path between u and v.

2. *All-pairs 3-approximate shortest paths in quadratic time & optimal space:*

There exist two algorithms for 3-approximate shortest paths. The first algorithm is by Cohen and Zwick [6], and requires $O(n^2 \log n)$ running time and $\theta(n^2)$ space. The second algorithm is due to Thorup and Zwick [12]. In [12], they designed a data structure that requires $O(n^{3/2})$ space and can answer any distance query in $O(1)$ time. However, it takes expected $O(m\sqrt{n})$ time to build the data structure. The first algorithm is superior in running time, whereas the second algorithm is superior in the space requirement (which is in fact optimal). *Can we have an algorithm which has the better features of both the algorithms?* This was an important open problem raised by Thorup and Zwick. We have *almost* solved this problem. We show that we can build a data structure of size $O(n^{3/2})$ in expected time $O(\min(n^2 \log n,\ m\sqrt{n}))$ to answer any 3-approximate distance query in $O(\log n)$ time. So the only place where our algorithm falls short of combining the best features of the algorithms in [12] and [6] is in the query answering time, which is $O(\log n)$ now.

3.*Approximate distance oracles in quadratic time :*

Generalizing their data structure for 3-approximate distances, Thorup and Zwick [12] showed that any weighted graph can be processed to build a data structure of $O(kn^{1+1/k})$ size that can answer any $(2k - 1)$-approximate distance query in $O(k)$ time. They named this data structure a $(2k - 1)$-approximate distance oracle due to the feature that it takes essentially constant time to answer any distance query in spite of the fact that it does not store all-pairs distances explicitly. Having achieved essentially optimal query time as well as the space requirement for approximate distance oracles, the only aspect that can be potentially improved is the preprocessing time. Currently, the expected preprocessing time of the $(2k - 1)$-approximate distance oracle is $O(kmn^{1/k})$ which is certainly sub-cubic, but not quadratic for dense graphs. Baswana and Sen [4] show that, for unweighted graphs, a $(2k-1)$-approximate distance oracles can be computed in expected $O(\min(n^2,\ kmn^{1/k}))$ time. Recently Roditty, Thorup and Zwick [11] show that the same can be done deterministically also. However, for weighted graphs, there does not exist any algorithm to compute a $(2k-1)$-approximate distance oracle in quadratic time (expected or deterministic). In this paper we show that it is possible to compute a $(2k-1)$ approximate distance oracle in expected $O(\min(n^2,\ kmn^{1/k}))$ for any $k > 2$.

1.1 Brief summary of the past work on APASP

Except the two algorithms, [6] and [12] as mentioned above, which are for weighted graphs, all the existing algorithms for APASP are for unweighted graphs only. These algorithms fall in two categories, depending upon whether the error in the distance is additive or multiplicative. The algorithm that computes all-pairs t-approximate shortest paths, as defined earlier, is actually an algorithm that achieves a multiplicative error t. An algorithm is said to report distances with additive error k, if for any pair of vertices $u, v \in V$, the distance, that it reports, is at least $\delta(u,v)$ and at most $\delta(u,v)+k$. The first algorithm for reporting distances with additive error was given by Aingworth *et al.* [1]. Their algorithm reports distances with additive error 2. Dor *et al.* [7] improved and extended this algorithm for arbitrary additive error. Their algorithm requires $O(kn^{2-\frac{1}{k}}m^{\frac{1}{k}} \operatorname{polylog} n)$ time to report distances with additive error $2(k-1)$ for any pair of vertices. They also showed that all-pairs 3-approximate shortest paths can be computed in $O(n^2 \operatorname{polylog} n)$ time. There are also two algorithms [2, 8] that compute approximate distances with both multiplicative as well as additive errors simultaneously. However, the techniques of neither of these two algorithms lead to similar results for weighted graphs. All the algorithms for reporting approximate distances that we mentioned above, including the new results of this paper, can also report the corresponding approximate shortest path, and the time for doing so is proportional to the number of edges in the approximate shortest path.

1.2 New techniques and overview of the paper

As mentioned above, there exist two algorithms for the problem of computing all-pairs 3-approximate distances : one by Cohen and Zwick [6] and another by Thorup and Zwick [12]. These two algorithms appear to be based on quite different schemes. This suggests that it would be a nontrivial task to combine the two schemes suitably to achieve a quadratic time and $O(n^{3/2})$ space algorithm for the problem. We present a new generic scheme for APASP that basically captures the essence of both these schemes simultaneously. The new scheme $\mathcal{A}(S,k)$ has two parameters : a subset $S \subset V$ and an integer k. The crucial features of the new scheme, that form the basis of our first two results, are the following.

- The new scheme $\mathcal{A}(S,k)$, with $S = V$ and $k = \log n$, turns out to be quite similar to the scheme used by Cohen and Zwick [12] for all-pairs 3-approximate shortest paths. However, we provide a different analysis that leads to a stretch *close* to 2. In particular, the new scheme leads to computing all-pairs $(2,\omega)$-approximate distances in $O(n^2 \log n)$ time. This forms the first result of the paper.

- The new scheme, with S defined as a random sample of $\sqrt{n}$ vertices and $k = \log n$, leads directly to a data structure for all-pairs 3-approximate distances with $O(\min(n^2 \log n,\ m\sqrt{n}))$ preprocessing time and $O(n^{3/2})$ space. The data structure takes $O(\log n)$ time to answer a 3-approximate distance query between any pair of vertices. This forms the second result of the paper.

For our third result achieving a worst case quadratic time for the construction of the approximate distance oracles, we employ a graph spanner like in [4]. In addition, we perform small but crucial changes to the algorithm of Thorup and Zwick [12]. The approximate distance oracle is a hierarchical data structure that, for each vertex, stores distances to a small subset of vertices in its *vicinity*, called *balls* (see Definition 2.1). Thorup and Zwick gave an expected $O(kmn^{1/k})$ time algorithm for their computation. We present a more efficient algorithm for computing these *balls*. This algorithm is designed by slightly modifying the algorithm of Thorup and Zwick, and requires a more careful probabilistic analysis for bounding its running time. This algorithm is also employed as one of the tool for our second result (mentioned above).

In the following section we introduce the definitions, lemmas and notations to be used in the rest of the paper. We describe our new scheme $\mathcal{A}(S,k)$ in Section 3. In Section 4, we present our efficient algorithm for computing balls. We describe the algorithm for all-pairs $(2,\omega)$-approximate distances in section 5. In Section 6, we describe the new algorithm for all-pairs 3-approximate distances that requires $O(\min(n^2 \log n,\ m\sqrt{n}))$ time, $O(n^{3/2})$ space and $O(\log n)$ query time. In Section 7, we describe the $O(\min(n^2,\ kmn^{1/k}))$ algorithm for computing $(2k-1)$-approximate distance oracles for any $k > 2$.

2 Preliminaries

Throughout the paper, we deal with undirected graphs whose edge weights are positive real numbers. We shall use the following notations :

- $\delta(u, v)$: distance between u and v in the original graph.
- $p_S(v)$: the vertex from the set S that is nearest to v. Break the tie arbitrarily in case of a tie. In case v belongs to S, $p_S(v) = v$.
- $\delta(v, S)$: the distance from v to the vertex from S that is nearest to it. If $S = \emptyset$, define $\delta(v, S) = \infty$.
- $E(v)$: The set of edges incident on vertex v.
- $\mathcal{E}_S(v)$: If $v \notin S$, then the set of edges incident on vertex v with weight less than $\delta(v, S)$, otherwise $\emptyset$.
- $\mathcal{E}_S$: $\cup_{v \in V} \mathcal{E}_S(v)$

Given a set $S \subseteq V$, it is quite easy to compute $p_S(v)$ and $\delta(v, S)$ for all $v \in V$ in $O(m + n \log n)$ time as follows. Connect a dummy vertex to all the vertices of the set S with edge weight zero on all these edges, and run Dijkstra's algorithm starting from this vertex on the graph. Thus the following lemma holds.

Lemma 2.1 *Given a set $S \subseteq V$, it requires $O(m + n \log n)$ time to compute $p_S(v)$ for all vertices $v \in V$.*

Lemma 2.2 *Let $G = (V, E)$ be a given undirected weighted graph and $S \subseteq V$. Let $\hat{\delta}(u, v)$ denote the distance between u and v in the subgraph $G_S = (V, \mathcal{E}_S)$. For each vertex $u \in V$, the following property holds : $\delta(u, v) < \delta(u, p_S(u)) \implies \hat{\delta}(u, v) = \delta(u, v)$, that is, the subgraph $G_S = (V, \mathcal{E}_S)$ preserves the <u>exact</u> distance between u and v.*

Proof: Consider the shortest path $u(= v_0), v_1, \cdots, v_j(= v)$ between u and v. It follows that

$$\delta(u, p_S(u)) > \delta(u, v_k) \;\; \forall k \leq j \tag{1}$$

All we need to show is that all the edges of the shortest path are present in $\mathcal{E}_S$. We shall prove this assertion by contradiction. Let (v_i, v_{i+1}) be the first edge (from side of u) on the path which is absent. Since this edge is not present in $\mathcal{E}_S$, it follows from the definition of $\mathcal{E}_S(v_i)$ that there is some vertex $x \in S$

such that $\delta(v_i, x) < \omega(v_i, v_{i+1})$. Combining this fact with Equation (1), we infer

$$\begin{aligned}
\delta(u, p_S(u)) &> \delta(u, v_{i+1}) \\
&= \delta(u, v_i) + \delta(v_i, v_{i+1}) \\
&\qquad\qquad \{ \text{ since } u \rightsquigarrow v_i \rightarrow v_{i+1} \rightsquigarrow v \text{ is the shortest path} \} \\
&> \delta(u, v_i) + \delta(v_i, x) \quad \geq \quad \delta(u, x)
\end{aligned}$$

Thus, for the vertex u, the vertex $x \in S$ is closer than $p_S(u)$ (a contradiction!). □

Corollary 2.1 *Let $G = (V, E)$ be a given undirected weighted graph and $S \subset V$. Let (x, y) be an edge on the shortest path from x to some vertex $v \in V$. If $(x, y) \notin \mathcal{E}_S$, then $\delta(v, S) \leq \delta(v, x)$.*

Lemma 2.3 *If the set $S \subseteq V$ is formed by selecting each vertex with probability q, the expected size of the set $\mathcal{E}_S$ will be $O(n/q)$.*

Proof: Let us calculate the expected number of edges in $\mathcal{E}_S(v)$ for any $v \in V$. If vertex v belongs to the set S, then $\mathcal{E}_S(v) = \emptyset$. Let us now estimate $\mathcal{E}_S(v)$ when $v \notin S$. Consider the sequence $\langle v_1, v_2, \cdots \rangle$ of neighbors of v arranged in non-decreasing order of the weight of their edges (v, v_i). An edge (v, v_i) will belong to $\mathcal{E}_S(v)$ if none of $v_1, \cdots, v_i$ is present in S. Since each vertex is selected in S independently with probability q, $\mathbf{Pr}[(v, v_i) \in \mathcal{E}_S(v)] \leq (1-q)^i$. Using linearity of expectation, if $v \notin S$, the expected number of edges in $\mathcal{E}_S(v)$ is at most $\sum_i (1-q)^i = O(1/q)$. Hence the expected number of edges in $\mathcal{E}_S$ is $O(n/q)$. □

An important construct of the approximate distance oracles is a *ball* which is defined as follows.

Definition 2.1 *For a vertex v in a graph $G = (V, E)$, and subsets X, Y of vertices, the set $ball(v, X, Y)$ of vertices is defined as*

$$ball(v, X, Y) = \{x \in X | \delta(v, x) < \delta(v, Y)\}$$

Let $B(v, X, Y)$ denote the number of vertices in $ball(v, X, Y)$.

In simple words, $ball(v, X, Y)$ consists of all those vertices of the set X whose distance from v is less than $\delta(v, Y)$. As a simple observation, it can be noted that $ball(v, X, \emptyset)$ is the set X itself, whereas $ball(v, X, X) = \emptyset$.

Given a graph $G = (V, E)$, let X, Y be two subsets of vertices. Using Lemma 2.2 and the definition of ball, it follows that the graph $G_Y = (V, \mathcal{E}_Y)$

preserves the exact distance from u to all vertices of $ball(u, V, Y)$, and hence to all vertices of $ball(u, X, Y)$ (the latter being the subset of the former). If $u(= v_0), v_1, v_2, \cdots, v_i, p_Y(u)$ is the shortest path between u and $p_Y(u)$, it also implies that every edge of this path, excluding the last edge, must be present in $G_Y = (V, \mathcal{E}_Y)$. Hence the following Lemma is obvious.

Lemma 2.4 *Given a graph $G = (V, E)$, let X, Y be two subsets of vertices. The graph $G_Y = (V, \mathcal{E}_Y \cup E(p_Y(u))$ preserves the exact distance from u to $p_Y(u)$ and to all the vertices of the set $ball(u, V, Y)$.*

The following lemma from [12] suggests that if the set Y is formed by sampling vertices from the set X with suitable probability, then $\mathbf{E}[B(v, X, Y)]$, i.e., the expected size of $ball(u, X, Y)$ will be sublinear in terms of $|X|$. This observation plays the key role in achieving subquadratic bound on the size of the approximate distance oracles of [12]. These oracles basically store distances to a hierarchy of balls around each vertex.

Lemma 2.5 *[12] Given a graph $G = (V, E)$, let the set Y be formed by picking each vertex of a set $X \subset V$ independently with probability q. Then $\mathbf{E}[B(v, X, Y)]$ is at most $1/q$.*

3 $\mathcal{A}(S, k)$: a hierarchical scheme for APASP

Given an undirected weighted graph $G = (V, E)$, a subset $S \subseteq V$, and a positive integer k, we describe a hierarchical scheme $\mathcal{A}(S, k)$ for APASP as follows. Let $p_i(v)$ denote $p_{S_i}(v)$ for $0 \le i < k$.

The scheme $\mathcal{A}(S, k)$

$S_0 \longleftarrow S, \quad S_k \longleftarrow \emptyset.$ Let $S_i, 0 < i < k$ contain each element of S_{i-1}, independently, with probability $|S|^{-1/k}$.

For $0 \le i < k$

1. Compute $p_i(u)$ for all $u \in V$.
2. For each $s \in S_i$, run Dijkstra's algorithm with source s in the subgraph $G_{i+1} = (V, \mathcal{E}_{S_{i+1}} \cup E(s))$.

3.1 Preprocessing time and space requirement for $\mathcal{A}(S,k)$

Let us now analyze the time and space requirement for the scheme $\mathcal{A}(S,k)$ when the set S is formed by selecting each vertex independently with probability $n^{-\alpha}$, for some $0 \le \alpha < 1$.

Computation of the scheme $\mathcal{A}(S,k)$ involves k iterations, where the ith iteration executes Dijkstra's algorithm from each vertex s of set S_i in the graph $G_{i+1} = (V, \mathcal{E}_{S_{i+1}} \cup E(s))$. A vertex v would belong to set S_i with probability $n^{-\alpha-i/k}$. Hence the expected running time of ith iteration is bounded by

$$\sum_{v\in V} (\deg(v) + \mathbf{E}[X_v|\ v \in S_i]) \cdot \mathbf{Pr}[v \in S_i] \tag{2}$$

Where X_v is the random variable denoting the number of all those edges of the set $\mathcal{E}_{S_{i+1}}$ excluding those which are incident on v. Now consider the graph $G^{\backslash v} = (V\backslash\{v\}, E\backslash E(v))$ formed by eliminating the vertex v and all its edges from the original graph $G = (V,E)$. Let Y be a random variable denoting the number of all edges $\mathcal{E}^{\backslash v}_{S_{i+1}}$ in the graph $G^{\backslash v}_{i+1}$. We shall show that X_v is stochastically dominated by Y as follows. For any choice of S_{i+1}, let $(u,w), u \neq v \neq w$ be an edge from $\mathcal{E}_{S_{i+1}}$. It follows that $\omega(u,w) < \delta(u, S_{i+1})$ or $\omega(u,w) < \delta(w, S_{i+1})$ in the original graph G. Since $G^{\backslash v}$ is a subgraph of G, the distance between any pair of vertices in $G^{\backslash v}$ is at least the distance between them in G. Hence $\omega(u,w) < \delta(u, S_{i+1}\backslash\{v\})$ or $\omega(u,w) < \delta(w, S_{i+1}\backslash\{v\})$ holds in $G^{\backslash v}$. Thus (u,w) must belong to $\mathcal{E}^{\backslash v}_{S_{i+1}}$ also. Hence every edge $(u,w), u \neq v \neq w$ present in $\mathcal{E}_{S_{i+1}}$ is surely present in $\mathcal{E}^{\backslash v}_{S_{i+1}}$ as well for each possible choice of S_{i+1}. So it follows that the random variable X_v is stochastically dominated by Y. Furthermore, note that Y is independent of whether or not $v \in S_i$ because v is not present in $G^{\backslash v}$. Combined together these observations, and using Lemma 2.3,

$$\begin{aligned} \mathbf{E}[X_v|\ v \in S_i] \le \mathbf{E}[Y|\ v \in S_i] &= \mathbf{E}[Y] \\ &= \min\left(\frac{n-1}{n^{-\alpha-(i+1)/k}},\ m - \deg(v)\right) \end{aligned}$$

Plugging the above values in Equation 2, the expected time for the ith iteration during the computation of $\mathcal{A}(S,k)$ is upper bounded by

$$\sum_{v\in V} \min\left(\frac{n-1}{n^{-\alpha-(i+1)/k}} + \deg(v),\ m\right) \cdot \mathbf{Pr}[v \in S_i]$$

$$< \min\left(\frac{2n}{n^{-\alpha-(i+1)/k}},\ m\right) \sum_{v\in V} \mathbf{Pr}[v \in S_i]$$

$$= \min\left(\frac{2n}{n^{-\alpha-(i+1)/k}},\ m\right)\ n^{1-\alpha-i/k} = \min(2n^{2+1/k}, mn^{1-\alpha-i/k})$$

Since there are total k iterations, it follows that the computation time involved in constructing the scheme $\mathcal{A}(S,k)$ is $O(\min(kn^{2+1/k},\ mn^{1-\alpha}))$. Let us now analyze the space requirement for the scheme $\mathcal{A}(S,k)$. We essentially keep a shortest-path tree from each $s \in S$. This will require $O(|S|n)$ space. In addition, we keep $p_i(u)$ for every $v \in V$, $\forall i < k$. So the total space used for storing all the hash tables is $O(nk + |S|n)$. Hence we can state the following lemma.

Lemma 3.1 *Given a graph $G = (V,E)$ and an integer k, let S be a set formed by selecting each vertex independently with probability $n^{-\alpha}$, for some $0 \le \alpha < 1$. It requires expected $O(\min(kn^{2+1/k},\ mn^{1-\alpha}))$ preprocessing time for the scheme $\mathcal{A}(S,k)$. The expected space requirement of the scheme $\mathcal{A}(S,k)$ will be $O(nk + n^{2-\alpha})$*

3.2 $\mathcal{A}(S,k)$ and all-pairs approximate distances

The scheme $\mathcal{A}(S,k)$ with suitable choices of the subset $S \subseteq V$ and integer k will form the foundations for the our two algorithms for all-pairs approximate distances.

The first algorithm computes all-pairs $(2,\omega)$-approximate distances and is described in Section 5. This algorithm is based on the following important property of $\mathcal{A}(S,k)$. *For any two vertices $u, v \in S$, the scheme $\mathcal{A}(S,k)$ stores $(2,\omega)$-approximate distance between them.* This crucial property of $\mathcal{A}(S,k)$ is proved in Theorem 5.1. Exploiting this property fully, the first algorithm is essentially the scheme $\mathcal{A}(S,k)$ with $S = V$ and $k = \log n$.

The second algorithm computes a data structure for all-pairs 3-approximate distances. The important property of the data structure is its $O(n^{3/2})$ size and $O(\min(n^2 \log n,\ m\sqrt{n}))$ preprocessing time. The sub-quadratic size of the data-structure suggests that we cannot afford to have the scheme $\mathcal{A}(S,k)$ with $S = V$. So we employ another important property of the scheme $\mathcal{A}(S,k)$, which says that even a small size set S would suffice to keep all-pairs 3-approximate distances. In precise words, this property of $\mathcal{A}(S,k)$ is stated as follows. *Even for vertices $u \in V \setminus S$, the scheme $\mathcal{A}(S,k)$ stores implicitly 3-approximate distance between u and all those vertices in the graph that are farther than $p_S(u)$, that is, to all the v that satisfy $\delta(u,v) \ge \delta(u, p_S(u))$.* This observation is proved in Theorem 6.2. The second algorithm will perform two main tasks. First it will employ scheme $\mathcal{A}(S,k)$ for a random subset $S \subset V$ of sufficiently small size. Secondly, it will compute for each vertex $u \in V \setminus S$, the distance to all those vertices from V that are closer than $p_S(u)$

(this set is $ball(u, S, V)$). Based on Lemmas 3.1 and 2.5, it turns out that in order to achieve $O(n^{3/2})$ size, it suffices to form S by selecting each vertex independently with probability $1/\sqrt{n}$.

In the following section, we describe an efficient algorithm for computing $ball(v, X, Y), \forall v \in V$. The results of the following section will be required for the new algorithm for all-pairs 3-approximate distances in Section 6 and for the efficient construction of $(2k-1)$-approximate distance oracles in Section 7. However, these results are not required for the algorithm for all-pairs $(2, \omega)$-approximate distances in Section 5.

4 Efficient computation of balls

Recall that $ball(v, X, Y)$ consists of all those vertices from the set X whose distance to v is less than $\delta(v, Y)$. We present an improved algorithm for computing these balls. The starting point is the following lemma from [12].

Lemma 4.1 *[12] If a vertex $x \in X$ belongs to $ball(v, X, Y)$, then for every vertex u lying on a shortest path between x and v, the vertex x will also belong to $ball(u, X, Y)$.*

Proof: The proof is by contradiction. Given that x belongs to $ball(v, X, Y)$, let u be a vertex lying on a shortest path between v and x. The vertex x would not belong to $ball(u, X, Y)$ if and only if there is some vertex $w \in Y$ such that $\delta(u, w) \leq \delta(u, x)$. On the other hand, using triangle inequality it follows that $\delta(v, w) \leq \delta(v, u) + \delta(u, w)$. Therefore, $\delta(u, w) \leq \delta(u, x)$ would imply that $\delta(v, w) \leq \delta(v, x)$. In other words, for the vertex v, the vertex $w \in Y$ is not farther than the vertex x. Hence x does not belong to $ball(v, X, Y)$ (a contradiction !). $\square$

For any $x \in X$, let *cluster* $C(x, Y)$ be the set containing all vertices $v \in V$ that satisfy $x \in ball(v, X, Y)$. It is easy to see that $v \in C(x, Y)$ if and only if $x \in ball(v, X, Y)$, that is, balls and clusters are "inverses" of each other. Thus computing $C(x, Y), \forall x \in X$ yields $ball(v, X, Y), \forall v \in V$. The task of computing clusters is easier since Lemma 4.1 shows that the set of vertices in $C(x, Y)$ appears as a truncated subtree in the shortest path tree rooted at x. Thorup and Zwick gave an efficient algorithm to compute the subtree corresponding to $C(x, Y)$, and the algorithm is essentially a modified Dijkstra's algorithm. Their algorithm is quite efficient since it processes only those edges whose at least one endpoint belongs to the set $C(x, Y)$. However, their algorithm for computing balls does not lead to a quadratic time algorithm for oracles.

We present an improved modified Dijkstra's algorithm to compute $C(x, Y)$ that will process relatively fewer edges of the graph. This new algorithm will process an edge (u, v) if u belongs to $C(x, Y)$ and $\delta(x, u) + \omega(u, v) < \delta(v, Y)$, that is, the path $x \rightsquigarrow u \rightarrow v$ is an *evidence* for "$v \in C(x, Y)$". In order to avoid processing of edges that do not satisfy these two conditions, our algorithm requires some preprocessing on the graph. For each neighbor v of a vertex u, let $\Delta_u(v)$ be defined as

$$\Delta_u(v) = \delta(v, Y) - \omega(u, v)$$

We preprocess the graph as follows. For each vertex $u \in V$, we sort its adjacency list so that all its neighbors appear in the non-increasing order of their Δ_u values in the list. This task will require $O(m \log n)$ time and needs to be done just once. Now we present the new algorithm *Modified_ Dijkstra(x, Y)* that computes all the vertices of the cluster $C(x, Y)$.

Modified_ Dijkstra(x, Y)

Similar to the ordinary Dijkstra's algorithm, the modified Dijkstra's algorithm begins with a priority queue $\mathcal{Q}$ on labels $\{\mathcal{L}[v] | v \in V\}$, with the initialization : $\mathcal{L}[x] \leftarrow 0$ and $\mathcal{L}[v] \leftarrow \infty$ for each $v \in V \setminus \{x\}$.

`While` $\min(\mathcal{Q}) \neq \infty$ `do`

1. `extract_min`$(\mathcal{Q})$: Extract the vertex with the smallest label $\mathcal{L}[\cdot]$ from $\mathcal{Q}$. Let this vertex be u.

2. Scan the adjacency list of u until either we reach a vertex w with $\Delta_u(w) \leq \mathcal{L}[u]$ or we reach the end of the list. For each neighbor v of u with $\Delta_u(v) > \mathcal{L}[u]$ visited during this scan, decrease its label in $\mathcal{Q}$ as follows.
   `decrease_key`$(v, \mathcal{Q})$: $\quad \mathcal{L}[v] \leftarrow \min\{\mathcal{L}[v],\ \mathcal{L}[u] + \omega(u, v)\}$

We shall now analyze the correctness and efficiency of the *Modified_ Dijkstra*(x, Y) algorithm. We may first note the following simple observation about shortest paths in a graph with positive edge weights :

$\mathcal{O}$: *Let $y, z \in V$ be any two vertices such that $\delta(x, y) < \delta(x, z)$. On any shortest path from x to some vertex v that passes through y and z, the vertex y will precede the vertex z on the path.*

Let there be j vertices in the set $C(x, Y)$. Consider the sequence $\langle x(= v_1), v_2, \cdots v_j \rangle$ with x as the first vertex and followed by vertices of $C(x, Y)$ arranged in the non-decreasing order of their distances from x. Let $\mathcal{P}^i_{xv}$ be the set of paths from x to v, each of length $< \delta(v, Y)$ such that all the intermediate vertices belong to the set $\{v_1, \cdots, v_i\}$. Let $\delta_i(x,v)$ denote the length of the shortest path from set $\mathcal{P}^i_{xv}$. We shall show that following assertion $\mathcal{H}_i$ holds for the ith iteration during Modified_Dijkstra(x, Y).

$\mathcal{H}_i$: 1. At the beginning of the iteration, vertex v_i is extracted from $\mathcal{Q}$, and $\mathcal{L}[v_i] = \delta(x, v_i)$.
2. Only those edges (v_i, v) incident on v_i are processed that satisfy $\delta(x, v_i) + \omega(v_i, v) < \delta(v, Y)$.
3. At the end of the iteration, for each $v \in V$, $\mathcal{L}[v] = \delta_i(x, v)$.

We shall prove the assertion $\mathcal{H}_i$ by induction on i. The initialization of $\mathcal{L}[\,]$ ensures that $\mathcal{H}_0(3)$ holds. Moreover $\mathcal{H}_0(1)$ and $\mathcal{H}_0(2)$ hold vacuously. Hence, the base case $\mathcal{H}_0$ holds. Let the assertion $\mathcal{H}_k$ hold for all $k < i$. We shall first prove $\mathcal{H}_i(1)$. Let $\mathcal{P}_{xv_i}$ be the shortest path from x to v_i in the given graph. It follows from Lemma 4.1 and the observation $\mathcal{O}$ that all the intermediate vertices of the path belong to the set $\{v_1, \cdots, v_{i-1}\}$. So $\delta_{i-1}(x, v_i) = \delta(x, v)$. So $\mathcal{L}[v_i] = \delta(x, v_i)$ using the induction hypothesis, and vertices $v_1, \cdots, v_{i-1}$ have already been extracted from $\mathcal{Q}$. Also note that $\mathcal{L}[v_{i'}] = \delta_{i-1}(x, v_{i'})$ is greater than $\delta(x, v_i)$ for all $i < i' \leq j$. And for the vertices $v \notin C(x, Y)$, $\mathcal{L}[v]$ is ∞. So $\mathcal{L}[v_i] = \delta(x, v_i)$ is the smallest label in $\mathcal{Q}$ at the beginning of the ith iteration, and hence the vertex v_i gets extracted from $\mathcal{Q}$. So $\mathcal{H}_i(1)$ holds. Since the adjacency list of v_i is sorted in the non-increasing order of $\Delta_{v_i}(\cdot)$, it follows that only those edges (v_i, v) will be processed that satisfy $\Delta_{v_i}(v) > \delta(x, v_i)$. From the definition of $\Delta_{v_i}(v)$, therefore, $\mathcal{H}_i(2)$ holds. In order to show that $\mathcal{H}_i(3)$ holds at the end of the ith iteration, we shall show that $\mathcal{L}[v] = \delta_i(x, v)$ for any vertex v. Consider the set $\mathcal{P}^i_{xv}$. It follows from the definition that $\mathcal{P}^i_{xv} \supseteq \mathcal{P}^{i-1}_{xv}$. There are the following two cases : $\mathcal{P}^i_{xv} \supset \mathcal{P}^{i-1}_{xv}$ and $\mathcal{P}^i_{xv} = \mathcal{P}^{i-1}_{xv}$. To prove $\mathcal{L}[v] = \delta_i(x, v)$, we shall analyze these two cases separately. Note that $\mathcal{P}^i_{xv} \supset \mathcal{P}^{i-1}_{xv}$ iff there are one or more paths from x to v passing through v_i with intermediate vertices from set $\{v_1, \cdots, v_{i-1}\}$ with length $< \delta(v, Y)$. Let P^i_{xv} be the shortest such path. It follows from Observation $\mathcal{O}$ that vertex v_i must appear as the last intermediate vertex on this path. Hence P^i_{xv} will be of the form $x \rightsquigarrow v_i \rightarrow v$. Therefore, $\mathcal{P}^i_{xv} \supset \mathcal{P}^{i-1}_{xv}$ iff (v_i, v) is an edge and $\delta(x, v_i) + \omega(v_i, v) < \delta(v, Y)$. In this case, $\delta_i(x, v) = \min(\delta_{i-1}(x, v),\ \delta(x, v_i) + \omega(v_i, v))$, and the reader may verify that the ith iteration updates $\mathcal{L}[v]$ to $\delta_i(x, v)$. Conversely $\mathcal{P}^i_{xv} = \mathcal{P}^{i-1}_{xv}$ iff either v is not a neighbor of v_i or $\delta(x, v_i) + \omega(v_i, v) \geq \delta(v, Y)$. If either of the two conditions fails, the ith iteration does not update $\mathcal{L}[v]$. But $\mathcal{P}^i_{xv} = \mathcal{P}^{i-1}_{xv}$ implies $\delta_i(x, v) = \delta_{i-1}(x, v)$. So in this case the induction hypothesis $\mathcal{H}_{i-1}$ itself ensures that $\mathcal{L}[v] = \delta_{i-1}(x, v) = \delta_i(x, v)$, and we are done.

Based on the assertion $\mathcal{H}_i$ that we have just proved, we can conclude the following theorem.

Theorem 4.1 *The algorithm* Modified_Dijkstra(x, Y) *algorithm computes the vertices (and their distances from x) of the set $C(x, Y)$. During this computation, it processes an edge (u, v) if and only if $x \in ball(u, X, Y)$ and the path $x \overset{P_{xu}}{\rightsquigarrow} u \to v$ is shorter than $\delta(v, Y)$.*

4.1 Analyzing the running time of Modified Dijkstra's algorithm with source x

The algorithm basically performs two kinds of operations : `extract_min` and `decrease_key` on the priority queue $\mathcal{Q}$. If we employ a Fibonacci heap, it takes time $O(1)$ for each `decrease_key` operation and $\log n$ for each `extract_min` operation. The number of `extract_min` operations is equal to the number of vertices satisfying $x \in ball(v, X, Y)$, and the number of `decrease_key` operations is of the order of the number of edges processed. Let us assign the cost of processing an edge (u, v) to the vertex v. Refer to Theorem 4.1 and observe that if edge (u, v) is such that "the path $x \overset{P_{xu}}{\rightsquigarrow} u \to v$ is shorter than $\delta(v, Y)$", then $(u, v) \in \mathcal{E}_Y(v)$. So vertex v will be charged $\mathcal{E}_Y(v) + \log n$ amount of computation cost.

Lemma 4.2 *Given a graph $G = (V, E)$, and sets X, Y of vertices, we can compute $ball(v, X, Y), \forall v \in V$ by running* Modified_Dijkstra*(x, Y) on each $x \in X$. During this computation, each vertex $v \in V$ will be charged $(\mathcal{E}_Y(v) + \log n)$ computation cost once by every vertex of $ball(v, X, Y)$.*

5 An algorithm for all-pairs $(2, \omega)$-approximate shortest paths

Given an undirected weighted graph $G = (V, E)$, the algorithm for computing all-pairs $(2, \omega)$-approximate distances is described in Figure 1. The expected running time of the algorithm will be $O(n^2 \log n)$. Previously [6], it was possible to compute only all-pairs 3-approximate distances in this time.

Theorem 5.1 *Let u and v be any two vertices in set S, and let ω_{uv} be the weight of the maximum weight edge on the shortest path between u and v. Given the scheme $\mathcal{A}(S, k)$, we have*

$$\min_{0 \le j < k} \left(\delta(u, p_j(u)) + \hat{\delta}(p_j(u), v) \quad , \quad \delta(v, p_j(v)) + \hat{\delta}(p_j(v), u) \right) \quad \le \quad 2\delta(u, v) + \omega_{uv}$$

1. $S \leftarrow V$, $k \leftarrow \log n$

2. Construct the scheme $\mathcal{A}(S, k)$.

3. For each $u, v \in V$,

$$d[u,v] \leftarrow \min_{0 \le i < k} \{\delta(u, p_i(u)) + \hat{\delta}(p_i(u), v) \ \ , \ \ \delta(v, p_i(v)) + \hat{\delta}(p_i(v), u)\}$$

where $\hat{\delta}(p_i(x), y)$ denotes the distance between $p_i(x)$ and y in the subgraph $(V, \mathcal{E}_{S_{i+1}} \cup E(p_i(x)))$. This distance is retrieved from $\mathcal{A}(S, k)$ in $O(1)$ time.

Figure 1: Algorithm for computing all-pairs $(2, \omega)$-shortest paths

Proof: Let $i < k$ be such that $u \in S_i$ but $u \notin S_{i+1}$. Note that such a unique $i < k$ must exist since $S_0 = S$ and $S_k = \emptyset$. Similarly let $i' < k$ be such that $v \in S_{i'}$ but $v \notin S_{i'+1}$. Without loss of generality assume that $i \ge i'$. Now consider the shortest path P_{uv} between u and v. If the entire path P_{uv}, that is, all its edges are present in $\mathcal{E}_{S_{i+1}} \cup E(u)$, then (the length of) this path will be computed by the scheme $\mathcal{A}(S, k)$ when we run Dijkstra's algorithm from u in the graph $(V, \mathcal{E}_{S_{i+1}} \cup E(u))$. Otherwise let $i + 1 \le j < k$ be such that $P_{uv} \notin \mathcal{E}_{S_j} \cup E(u)$ and $P_{uv} \in \mathcal{E}_{S_{j+1}} \cup E(u)$. Such a j must exist since $(V, \mathcal{E}_{S_k})$ is the entire graph. Traversing the path P_{uv} in the direction from u to v, let (x, y) be the first edge which is not present in $\mathcal{E}_{S_j}$. Using Corollary 2.1, we can state the following inequalities.

$$\delta(u, p_j(u)) \le \delta(u, y) \qquad \text{and} \qquad \delta(v, p_j(v)) \le \delta(v, x)$$

Since the shortest path P_{uv} is of the form $u \rightsquigarrow x \rightarrow y \rightsquigarrow v$, adding the above two inequalities, we get

$$\delta(u, p_j(u)) + \delta(v, p_j(v)) \le \delta(u, y) + \delta(x, v) = \delta(u, v) + \omega(x, y)$$

Without loss of generality assume that $\delta(u, p_j(u)) \le \delta(v, p_j(v))$. Then we have

$$2\delta(u, p_j(u)) \le \delta(u, v) + \omega(x, y) \tag{3}$$

Recall that during the construction of scheme $\mathcal{A}(S, k)$, we execute Dijkstra's algorithm from $p_j(u)$ in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$. Since $\mathcal{E}_{S_j} \subseteq \mathcal{E}_{S_{j+1}}$, it follows from Lemma 2.4 that the shortest path from $p_j(u)$ to u is preserved in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$. It follows from the definition of j that the shortest path between u and v is also preserved in this graph. Using these observations and applying the triangle inequality, the distance $\hat{\delta}(p_j(u), v)$ between $p_j(u)$ and v in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$ can be bounded from above as follows.

$$\hat{\delta}(p_j(u), v) \leq \delta(p_j(u), u) + \delta(u, v) \tag{4}$$

Using Inequalities (3) and (4), we can bound $\delta(u, p_j(u)) + \hat{\delta}(p_j(u), v)$ as follows.

$$\begin{aligned}
\delta(u, p_j(u)) + \hat{\delta}(p_j(u), v) &\leq \delta(u, p_j(u)) + \delta(p_j(u), u) + \delta(u, v) \\
&\leq 2\delta(u, p_j(u)) + \delta(u, v) \\
&\leq \delta(u, v) + \omega(x, y) + \delta(u, v) \quad \{\text{using Inequality (3)}\} \\
&= 2\delta(u, v) + \omega(x, y)
\end{aligned}$$

So if ω_{uv} is the weight of the heaviest edge on the shortest path between u and v, then the distance $\delta(u, p_j(u)) + \hat{\delta}(p_j(u), v)$ does not exceed $2\delta(u, v) + \omega_{uv}$. $\square$

Since we employ the scheme $\mathcal{A}(S, k)$ with $S = V$ in the algorithm, it follows from Theorem 5.1 that $d[u, v] \leq 2\delta(u, v) + \omega_{uv}$, for all $u, v \in V$. In other words, the algorithm computes all-pairs $(2, \omega)$-approximate distances. Furthermore, since $k = \log n$, it follows from Lemma 3.1 that the expected running time of the algorithm is $O(n^2 \log n)$.

Theorem 5.2 *All-pairs $(2, \omega)$-approximate distances in a graph on n vertices can be computed in expected $O(n^2 \log n)$ time.*

6 All-pairs 3-approx. distances in quadratic time and sub-quadratic space

Now we present an efficient data structure $\mathcal{D}$ of size $O(n^{3/2})$ that can answer a 3-approximate distance query for any pair of vertices in $O(\log n)$ time.

$\mathcal{D}$: a data structure for all-pairs 3-approximate distances

Let S be a set formed by selecting each $v \in V$ independently with probability $1/\sqrt{n}$.

1. For each $v \in V \setminus S$,
   – store distances to all the vertices of the set $ball(v, V, S) \cup \{p_S(v)\}$.
2. Construct the scheme $\mathcal{A}(S, \log n)$.

In order to answer an approximate distance query efficiently, the data structure $\mathcal{D}$ will keep each $ball(v, V, S)$ in a hash table (see [9]) so that one can determine in worst case constant time whether or not any vertex u belongs to $ball(v, V, S)$, and if so, report the distance $\delta(v, u)$. Once we have the set $ball(v, V, Y)$, the computation time as well as the space requirement of the corresponding hash table would be of the order of $|ball(v, V, S)|$. Let us now analyze the time required for constructing the data structure $\mathcal{D}$. There are two main tasks. The first task is the computation of $ball(v, V, S)$, for each v in $V \setminus S$, and the second task is the construction of the scheme $\mathcal{A}(S, \log n)$. Note that S is formed by selecting each vertex independently with probability $1/\sqrt{n}$. So it follows from Lemma 3.1 that the expected time to construct the scheme $\mathcal{A}(S, \log n)$ is $O(\min(n^2 \log n,\ m\sqrt{n}))$. Let us analyze the running time of the first task which involves the computation of $ball(v, V, S), \forall v \in V$. Thorup and Zwick presented an algorithm that takes expected $O(m\sqrt{n})$ time for this task. We shall now show that, by employing our modified Dijkstra's algorithm from Section 4, we can execute the first task in expected $O(\min(n^2,\ m\sqrt{n}))$ time.

Lemma 6.1 *Let S contain each vertex of the set V independently with probability $q = 1/\sqrt{n}$. Then we can compute $ball(v, V, S), \forall v \in V \setminus S$ in expected $O(\min(n^2,\ m\sqrt{n})$ time.*

Proof: We compute $ball(v, V, S), \forall v \in V \setminus S$ by executing *Modified_Dijkstra*(x, S) on each $x \in V$. Note that we also get the distance from v to every $x \in ball(v, V, S)$ from this computation. Let us refer to Lemma 4.2 for analyzing the total computation cost (time) incurred in this task. It follows from Lemma 4.2 that $(|\mathcal{E}_S(v)| + \log n)$ computation cost will be charged to v by each vertex in $ball(v, V, S)$. So the expected cost charged to v is $\mathbf{E}[|\mathcal{E}_S(v)| \cdot B(v, V, S)]$. Observe that $|\mathcal{E}_S(v)| \leq \deg(v)$ and $\mathbf{E}[B(v, V, S)] = 1/q = \sqrt{n}$ using Lemma 2.5. Hence the cost charged to v is bounded by $O(\deg(v).\sqrt{n})$. However, we shall provide a tighter analysis, showing that expected cost charged to any vertex is bounded by $O(n)$ irrespective of the large degree that the vertex v may have. First note that all the neighbors of v corresponding to the edges of $\mathcal{E}_S(v)$ are present in $ball(v, V, S)$. So $|\mathcal{E}_S(v)| \leq B(v, V, S)$ holds always. Hence the expected cost charged to v is bounded from above by $\mathbf{E}[B(v, V, S)|^2]$. To calculate $\mathbf{E}[B(v, V, S)^2]$, consider the sequence $v_1, v_2, v_3, \cdots, v_{n-1}$ of vertices of the given graph arranged in non-decreasing order of their distances from v. It can be observed that $B(v, V, S) = i$ if none of $v_1, \cdots, v_i$ is selected in S, but v_{i+1} is selected in S. The probability associated with this event is $(1-q)^i q$. Hence we can bound $\mathbf{E}[B(v, V, S)^2]$ as follows.

$$
\begin{aligned}
\mathbf{E}[B(v,V,S)^2] &= \sum_{i=1}^{i=n-1} i^2(1-q)^i q \\
&\leq q\sum_{i=1}^{i=\infty} i^2(1-q)^i \\
&= q\frac{(1+(1-q))(1-q)}{(1-(1-q))^3} \qquad \left\{\text{using } \sum_{i\geq 1} i^2x^i = \frac{(1+x)x}{(1-x)^3}\right\} \\
&\leq q\frac{2}{q^3} = \frac{2}{q^2} = 2n \qquad \{\text{since } q = 1/\sqrt{n}\}
\end{aligned}
$$

Hence the expected cost charged to vertex v is $O(\min(n,\ \deg(v)\sqrt{n}))$. So the expected cost incurred in the computation of $ball(v,V,S), \forall v \in V\setminus S$ is $O(\min(n^2,\ m\sqrt{n}))$. □

Based on Lemma 6.1 and the discussion preceding it, we can conclude that the expected time for constructing the data structure $\mathcal{D}$ is $O(\min(n^2\log n,\ m\sqrt{n}))$. Let us analyze its space requirement. The expected size of $ball(u,V,S)$ is $\sqrt{n}$ using Lemma 2.5. So the total expected space required for storing hash tables for all the balls will be $O(n^{3/2})$. It follows from Lemma 3.1 that the expected space requirement for the scheme $\mathcal{A}(S,\log n)$ is also $O(n^{3/2})$. Hence the expected space requirement of the data structure $\mathcal{D}$ is $O(n^{3/2})$. We shall rebuild the data structure $\mathcal{D}$ if its space requirement exceeds the expected bound by a factor larger than two. It follows from Markov's Inequality that the probability of this event will be at most 1/2. So the expected number of repetitions will be $O(1)$. Hence we can state the following theorem.

Theorem 6.1 *For any graph on n vertices and m edges, the data structure $\mathcal{D}$ would occupy $O(n^{3/2})$ space and can be built in expected $O(\min(n^2\log n,\ m\sqrt{n}))$ time.*

6.1 Answering distance query using the new data structure $\mathcal{D}$

We present the query answering algorithm in Figure 2. Recall that the scheme $\mathcal{A}(S,k)$ forms a hierarchy of subsets $S_0 = S \supseteq S_1 \supseteq \cdots \supseteq S_{k-1} \supseteq \emptyset$, and let $p_i(u)$ denote the vertex in S_i that is nearest to u.

Theorem 6.2 *Given the scheme $\mathcal{A}(S,k)$, if u and v are two vertices such that $\delta(u,S) \leq \delta(u,v)$, then there exists some $0 \leq j < k$ such that*

$$\delta(u,p_j(u)) + \hat{\delta}(p_j(u),v) \leq 3\delta(u,v)$$

Reporting approximate distance between u and v

If $v \in ball(u, V, S)$, report
the exact distance $\delta(u, v)$.
Else report using $\mathcal{A}(S, \log n)$

$$\min_{0 \le i < \log n} \{\delta(u, p_i(u)) + \hat{\delta}(p_i(u), v)\}$$

where $\hat{\delta}(p_i(x), y)$ denotes the distance between $p_i(x)$ and y in the subgraph $(V, \mathcal{E}_{S_{i+1}} \cup E(p_i(x)))$.

Figure 2: Query answering algorithm based on the data structure $\mathcal{D}$

Proof: Consider the set of vertices $\{p_j(u) | 0 \le j < k\}$. It follows from the construction of S_j's that

$$\delta(u, p_j(u)) \le \delta(u, p_{j+1}(u))$$

Since $\delta(u, S_0) \le \delta(u, v)$ and $\delta(u, S_k) = \infty$, there must be some unique $j < k$, such that

$$\delta(u, p_j(u)) \le \delta(u, v) < \delta(u, p_{j+1}(v)) \tag{5}$$

Recall that during the construction of $\mathcal{A}(S, k)$ we compute distances from the vertex $p_j(u)$ to all the vertices in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$. Since $\mathcal{E}_{S_j} \subseteq \mathcal{E}_{S_{j+1}}$, it follows from Lemma 2.2 that the shortest path between $p_j(u)$ to u is preserved in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$. It follows from the definition of j that the shortest path between u and v is also preserved in this graph. Using these observations and applying the triangle inequality, the distance $\hat{\delta}(p_j(u), v)$ between $p_j(u)$ and v in the graph $(V, \mathcal{E}_{S_{j+1}} \cup E(p_j(u)))$ can be bounded from above as follows.

$$\hat{\delta}(p_j(u), v) \le \delta(p_j(u), u) + \delta(u, v)$$

Combining the above inequality with Equation (5), we conclude that

$$\delta(u, p_j(u)) + \hat{\delta}(p_j(u), v) \le 3\delta(u, v).$$

□

Lemma 6.2 *For any $u, v \in V$, the distance reported by the data structure $\mathcal{D}$ is at most $3\delta(u, v)$.*

Proof: There are two cases depending upon whether or not $\delta(u,v) < \delta(u,p_S(u))$ is true. If $\delta(u,v) < \delta(u,p_S(u))$, v must be present in $ball(u,V,S)$, and we report the exact distance $\delta(u,v)$ using the hash table associated with $ball(u,V,S)$. So let us consider the case when $\delta(u,v) \geq \delta(u,p_S(u)))$. Using Theorem 6.2, it follows that the distance reported using scheme $\mathcal{A}(S,\log n)$ is at most $3\delta(u,v)$. $\square$

Combining Theorem 6.1 and Lemma 6.2, we can state the following theorem.

Theorem 6.3 *An undirected weighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2\log n,\ m\sqrt{n}))$ time to build a data structure of size $O(n^{3/2})$ such that for any pair of vertices it can report 3-approximate distance in $O(\log n)$ time.*

7 A (2k − 1)-approximate distance oracle in expected $\mathbf{O}(n^2)$ time

In this section we shall describe the construction of a $(2k-1)$-approximate distance oracle for $k > 2$. It is essentially the same as the algorithm in [12] except the steps 2 and 3 where we employ a 3-spanner. A 3-spanner of a weighted graph is a subgraph such that the distance between any two vertices $u,v \in V$ in the subgraph is at most $3\delta(u,v)$. We shall use the following result concerning 3-spanners from [3].

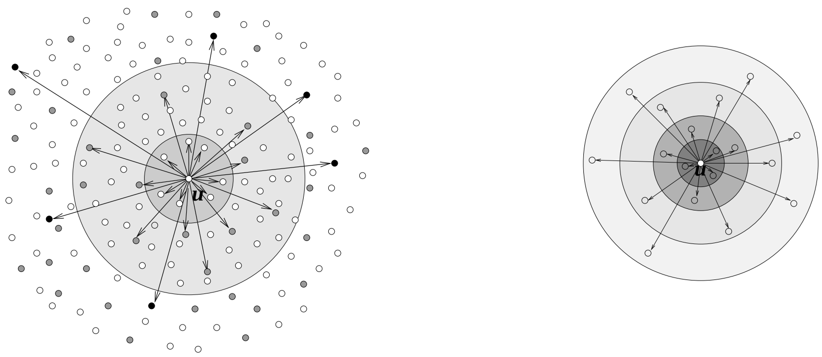
Theorem 7.1 *[3] An undirected weighted graph on n vertices and m edges can be processed in $O(m)$ time to compute its 3-spanner having $O(n^{3/2})$ edges.*

In the following algorithm $p_i(u)$ denotes the vertex $p_{S_i}(u)$ for $0 \leq i < k$.

Computing (2k-1)-approximate distance oracle

$S_0 \longleftarrow V$, $S_k \longleftarrow \emptyset$. Let $S_i, 0 < i < k$ contain each element of S_{i-1}, independently, with probability $n^{-1/k}$.

1. `For` $i \leftarrow 1$ `to` $k-1$
   Compute $p_i(v)$ and $ball(v,S_{i-1},S_i)$ for all $v \in V$.
2. Compute a 3-spanner (V,E_S).
3. Compute $\hat{\delta}(v,x)$ for each $v \in V, x \in S_{k-1}$ in the subgraph $(V,\mathcal{E}_{S_{k-1}} \cup E_S)$.



(i) (ii)

Figure 3: (*i*) Schematic description of $ball^i(u), i < k$. (*ii*) hierarchy of balls around u

7.1 The data-structure

The final data structure for the approximate distance oracle would keep a hash table $B(v)$ for every vertex $v \in V$. The hash table $B(v)$ would store all the vertices of the set $\cup_{i \le k} ball(v, S_{i-1}, S_i)$ and their distances from v. Note that $ball(v, S_{k-1}, S_k) = S_{k-1}$ since $S_k = \emptyset$. It follows from [9] that once we have the sets $ball(v, S_{i-1}, S_i), i \le k$, the construction time as well as the space requirement of the hash-table $B(v)$ would be of the order of $\cup_{i \le k} Ball(v, S_{i-1}, S_i)$. It would take $O(1)$ time to determine whether or not a vertex x is present in $B(v)$, and, if so, the corresponding distance can also be retrieved in $O(1)$ time. Henceforth, we shall use $d[u, x]$ to denote the (approximate/exact) distance between u and x as stored in a the hash table $B(v)$. The following Lemma holds

Lemma 7.1 *If $x \in B(v)$, then*

$$d[v,x] = \begin{cases} \delta(v,x) & \textit{if } \ x = p_{k-1}(v) \textit{ or } \ x \in S_i, \ i < k-1 \\ \hat{\delta}(v,x) & \textit{otherwise} \end{cases}$$

where $\hat{\delta}(v,x)$ is the distance between u and x in the graph $(V, \mathcal{E}_{S_{k-1}} \cup E_S)$.

Note that S_i is formed by selecting each vertex of the set S_{i-1} with probability $n^{-1/k}$. So it follows from Lemma 2.5 that, $\mathbf{E}[B(v, S_{i-1}, S_i]$, i.e., the expected size of $ball(v, S_i, S_{i+1})$ is at most $n^{1/k}$. Hence, the hash table $B(v)$ would occupy expected $O(kn^{1/k})$ space.

7.2 Analyzing the preprocessing time

There are three main tasks in the preprocessing algorithm. The second task involves the computation of a 3-spanner. Using Theorem 7.1, this task will take $O(m)$ time, and there will be $O(n^{3/2})$ edges in the 3-spanner (V, E_S). The third task requires computation of shortest-path trees from each vertex of set S_{k-1} in the graph $(V, \mathcal{E}_{S_{k-1}} \cup E_S)$. Since $|E_S| = O(n^{3/2})$, the number of edges in this graph will be $O(\min(m,\ n^{2-1/k}))$ using Lemma 2.3. Also note that the expected size of S_{k-1} is $O(n^{1/k})$. Hence the third task of the preprocessing algorithm would require expected $O(\min(mn^{1/k},\ n^2))$ time. Now let us analyze the first task that requires computation of $p_i(v)$ and $ball(v, S_{i-1}, S_i)$. Similar to the 3-approximate distance oracle, we can compute $ball(v, S_{i-1}, S_i), \forall v \in V$ by performing *Modified_ Dijkstra*(x, S_i) on every vertex $x \in S_{i-1}$. In the following Lemma, which is basically a generalization of Lemma 6.1, we shall provide a tighter bound on the expected time required to compute $ball(v, S_{i-1}, S_i), \forall v \in V$.

Lemma 7.2 *The expected time required to compute* $ball(v, S_{i-1}, S_i), \forall v \in V$ *is* $O(\min(n^{\frac{k+i+1}{k}},\ mn^{1/k})$.

Proof: see the Appendix. □

The preprocessing algorithm computes $ball(v, S_{i-1}, S_i) \forall v \in V$ for each $1 \leq i < k$. Hence using Lemma 7.2 and the preceding discussion, we can conclude that the expected preprocessing time for the new approximate distance oracle is $O(\min(n^2,\ kmn^{1/k}))$. The expected size of the oracle is of the order of $kn^{1+1/k}$. We shall repeat the algorithm if the size exceeds twice this bound, and using Markov's inequality, the probability of this event will be at most $1/2$. So the expected number of repetitions will be $O(1)$. Hence we can state the following theorem.

Theorem 7.2 *The new approximate distance oracle occupies* $O(kn^{1+1/k})$ *space and can be computed in expected* $O(\min(n^2,\ kmn^{1/k}))$ *time.*

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

The query answering procedures of the new oracle is exactly the same as the previously existing $(2k-1)$-approximate distance of [12]. However, its analysis is slightly different due to slight dissimilarity in the construction. First, we provide an overview of the underlying query answering procedure before we formally state it.

Let u and v be two vertices whose approximate distance is to be reported. In order to explain the query answering procedure, we introduce the following notation.

For a pair of vertices u and v, a vertex w is said to be *t-near* to u if $\delta(u, w) \leq t\delta(u, v)$. It follows from the simple triangle inequality that if a vertex is t-near to u then it is $(t+1)$-near to v also.

The entire query answering process is to search for a t-near vertex of u whose distance to both u and and v is known, and $t < k$. The search is performed iteratively as follows : the ith iteration begins with a vertex $w \in S_{i-1}$ which is $(i-1)$-near to u (and hence i-near to v) and its distance from u is known. It is determined whether or not its distance to v is known. We do so by checking whether or not $w \in B(v)$. Note that $w \notin B(v)$ if and only if $w \notin ball(v, S_{i-1}, S_i)$. Using Definition 2.1, therefore, it follows that $w \notin B^(v)$ if and only if, for the vertex v, the vertex $p_i(v)$ is equidistant or nearer than the vertex w. But this would imply that $p_i(v)$ is also i-near to v, and note that its distance from v is known. Therefore, if the distance $\delta(v, w)$ is not known, we continue in the next iteration with vertex $w = p_i(v)$, and swap the roles of u and v. In this way we proceed gradually, searching over the sets $S_0, \cdots, S_{k-1}$. We are bound to find such a vertex within at most k iterations since the distance from S_{k-1} is known to all the vertices. We now formally state the query answering procedure which is essentially the same as that of [12] except the 'If' loop at the end.

$\mathcal{Q}(u, v)$: *Reporting approximate distance between u and v*

$i \leftarrow 1$
`While` $(p_{i-1}(u)) \notin B(v))$ `do`
  swap(u, v)
  $i \leftarrow i + 1$

`return` $d[u, p_{i-1}(u)] + d[v, p_{i-1}(u)]$

Based on the discussion above, the following lemma holds.

Lemma 7.3 *[12] At the beginning of the ith iteration of the* `While` *loop,* $\delta(u, p(u, S_{i-1})) \leq (i-1)\delta(u, v)$.

Theorem 7.3 *The query algorithm $\mathcal{Q}(u, v)$ reports $(2k-1)$-approximate distance between u and v, for any $k > 2$.*

Proof: The query answering algorithm performs iterations of the `While` loop until the condition of the loop fails. As mentioned above, there will be at most $(k-1)$ successful iterations before the condition of the loop fails. Let the condition fail at the beginning of the ith iteration. So the following inequality follows from Lemma 7.3 when we exit the `While` loop.

$$\delta(u, p_{i-1}(u)) \leq (i-1)\delta(u, v) \tag{6}$$

We shall analyze the distance reported by the oracle on the basis of the final value of i.

Case 1 : $i < k$

In this case, using Lemma 7.1 the reported distance is $\delta(u, p_{i-1}(u))+\delta(v, p_{i-1}(u))$, which using the triangle inequality is at most $2\delta(u, p_{i-1}(u)) + \delta(u, v)$. Using Inequality (6), this distance is at most $(2i-1)\delta(u, v)$.

Case 2 : $i = k$.

In this case, it follows from Lemma 7.1 that $d[u, p_{k-1}(u)] = \delta(u, p_{k-1}(u))$, but $d[v, p_{k-1}(u)] = \hat{\delta}(v, p_{k-1}(u))$. In order to ensure that the distance reported is $(2k-1)$, we shall analyze the following two sub-cases as shown in Figure 4.

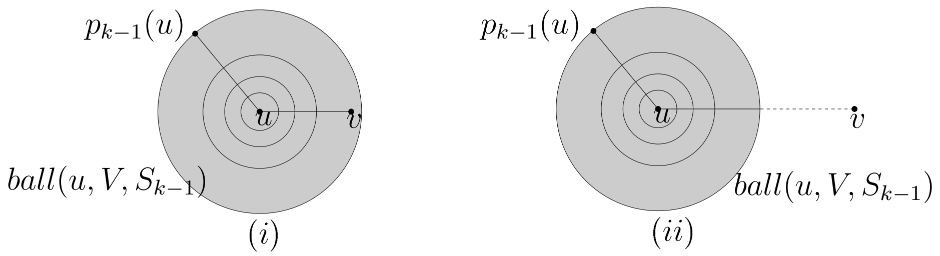


Figure 4: Two sub-cases depending upon whether or not $v \in ball(u, V, S_{k-1})$ Two sub-cases depending upon which of v or $p(u, S_{k-1})$ is closer to u.

In sub-case (i), the vertex v belongs to $ball(u, V, S_{k-1})$. Therefore, using Lemma 2.4, it follows that $\hat{\delta}(v, u) = \delta(v, u)$ and $\hat{\delta}(u, p_{k-1}(u)) = \delta(u, p_{k-1}(u))$. Combining these two equalities together and using the triangle inequality, it follows that

$$d[v, p_{k-1}(u)] = \hat{\delta}(v, p_{k-1}(u)) \leq \delta(v, u) + \delta(u, p_{k-1}(u))$$

Hence the distance reported is at most $(2k-1)\delta(u, v)$ as shown below.

$$\begin{aligned} d[u, p_{k-1}(u)] + d[v, p_{k-1}(u)] &\leq 2\delta(u, p_{k-1}(u)) + \delta(u, v) \\ &\leq 2(k-1)\delta(u, v) + \delta(u, v) \quad \{\text{using Inequality (6)}\} \\ &= (2k-1)\delta(u, v) \end{aligned}$$

In sub-case (ii), the vertex v does not belong to $ball(u, V, S_{k-1})$. Therefore, $p_{k-1}(u)$ is in fact equidistant or closer to u than the vertex v, that is, $\delta(u, p_{k-1}(u)) \leq \delta(u, v)$. Using Theorem 2.4, $\hat{\delta}(u, p_{k-1}(u)) = \delta(u, p_{k-1}(u))$. Since (V, E_S) is a 3-spanner, so $\hat{\delta}(v, u) \leq 3\delta(v, u)$. Combining these two inequalities together and using the triangle inequality, it follows that

$$d[v, p_{k-1}(u)] = \hat{\delta}(v, p_{k-1}(u)) \leq \delta(v, u) + \delta(u, p_{k-1}(u)) \leq 4\delta(v, u)$$

Therefore, in this sub-case, the distance $d[u, p_{k-1}(u)] + d[v, p_{k-1}(u)]$ reported by the oracle will be at most $5\delta(u, v)$. For $k > 2$, the stretch, which is 5, is bounded by $(2k - 1)$. $\square$

Combining Theorems 7.2 and 7.3, we can state the following theorem.

Theorem 7.4 *An undirected weighted graph on n vertices and m edges can be preprocessed in expected $O(\min(n^2,\ kmn^{1/k}))$ time to compute a $(2k-1)$-approximate distance oracle of size $O(kn^{1+1/k})$, for any integer $k > 2$.*

References

[1] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths(without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.

[2] Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in $O(n^2 \operatorname{polylog} n)$ time. In *Proceedings of 22nd Annual Symposium on Theoretical Aspect of Computer Science*, pages 666–679, 2005.

[3] Surender Baswana and Sandeep Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 384–396, 2003.

[4] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (to appear), January 11-13, New Orleans, USA*, 2004.

[5] Timothy Chan. All-pairs shortest paths with real edge weights in $O(n^3/\log n)$ time. In *Proceedings of Workshop on Algorithms and Data Structures (to appear)*, volume 3608, 2005.

[6] Edith Cohen and Uri Zwick. All-pairs small stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

[7] Dorit Dor, Shay Halperin, and Uri Zwick. All pairs almost shortest paths. *Siam Journal on Computing*, 29:1740–1759, 2000.

[8] Michael Elkin. Computing almost shortest paths. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2001.

[9] Michael L. Fredman, Janos Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case time. *Journal of Association of Computing Machinery*, 31:538–544, 1984.

[10] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312:47–74, 2004.

[11] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic construction of approximate distance oracles and spanners. In *Proceedings of 32nd International Colloquim on Automata, Languagaes and Programming (to appear)*, 2005.

[12] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of Association of Computing Machinery*, 52:1–24, 2005.

Appendix

We shall now prove Lemma 7.2. The proof will employ two simple Lemmas that we prove below. Let us introduce the notation $\mathcal{S}_u^v$ defined for any two vertices u, v in a given graph (V, E) as follows.
$\mathcal{S}_u^v$: the set of vertices in the graph that lie within distance $\delta(u, v)$ from v.

The first Lemma gives a bound on $\mathbf{Pr}[u \in ball(v, X, Y)]$ in terms of $\mathbf{Pr}[u \in X]$ and $\mathbf{Pr}[u \in ball(v, V, Y)]$.

Lemma 7.4 *Let X be formed by selecting each vertex from V independently with probability q, and let Y be formed by selecting each vertex of X independently with probability $q' \leq 1$. For any $u, v \in V$,*

$$\mathbf{Pr}[u \in ball(v, X, Y)] \leq \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \in ball(v, V, Y)]$$

Proof: It follows from Definition 2.1 that vertex u will belong to $ball(v, X, Y)$ iff $u \in X$ and none of the vertices of set $\mathcal{S}_u^v$ is present in Y. Now whether or not a vertex belongs to sets X or Y is independent of any other vertices. Because of this independence incorporated in selecting vertices to form sets X and Y, we can observe that

$$\begin{aligned}
\mathbf{Pr}[u \in ball(v, X, Y)] &= \mathbf{Pr}[u \notin Y \wedge u \in X] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y] \\
&= \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \notin Y | u \in X] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y]
\end{aligned}$$

Now note that $\mathbf{Pr}[u \notin Y | u \in X] = (1-q') \leq (1-qq') = \mathbf{Pr}[u \notin Y]$. Hence

$$\begin{aligned} \mathbf{Pr}[u \in ball(v,X,Y)] &< \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \notin Y] \cdot \prod_{w \in \mathcal{S}_u^v \setminus \{u\}} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u \in X] \cdot \prod_{w \in \mathcal{S}_u^v} \mathbf{Pr}[w \notin Y] \\ &= \mathbf{Pr}[u \in X] \cdot \mathbf{Pr}[u \in ball(v,V,Y)] \end{aligned}$$

□

Lemma 7.5 *Let X be a set formed by selecting each vertex of the given graph independently with certain probability, and let Y be formed by selecting each vertex from X independently with certain probability. For any $u, u', v \in V$,*

$$\mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,X,Y)] = \mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,V,Y)]$$

Proof: First note that $u \in ball(v,X,Y)$ iff $u \in X$ and $u \in ball(v,V,Y)$. So if $u' \in \mathcal{S}_u^v$, then $\mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,X,Y)]$ as well as $\mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,V,Y)]$ are 1. Let us now analyze the case when $u' \notin \mathcal{S}_u^v$. First note that $u \in ball(v,X,Y)$ implies that none of the vertices of the set $\mathcal{S}_u^v$ are selected in Y. So conditioned on the event "$u \in ball(v,X,Y)$", the vertex u' will belong to $ball(v,V,Y)$ iff none of the vertices from $\mathcal{S}_{u'}^v \setminus \mathcal{S}_u^v$ is present in set Y. The latter event is independent of $u \in X$ since a vertex from set $V \setminus \{u\}$ is selected in Y independent of whether or not $u \in X$. Hence,

$\mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,X,Y)] =$

$$\prod_{w \in \mathcal{S}_{u'}^v \setminus \mathcal{S}_u^v} \mathbf{Pr}[w \notin Y] \quad = \quad \mathbf{Pr}[u' \in ball(v,V,Y) | u \in ball(v,V,Y)]$$

□

The above Lemma essentially says the following : Conditioned on the event $u \in ball(v,V,Y)$, the probability that some vertex u' also belongs to $ball(v,V,Y)$ is independent of whether or not u was present in X. Based on this Lemma, the following Corollary can be stated.

Corollary 7.1 *Let X be formed by selecting each vertex from V independently with some probability, and let Y be formed by selecting each vertex of X independently with some probability. For any vertices $u, u', v \in V$,*

$$\mathbf{E}[B(v,V,Y) | u \in ball(v,X,Y)] = \mathbf{E}[B(v,V,Y) | u \in ball(v,V,Y)]$$

Proof of Lemma 7.2 : We compute $ball(v, S_{i-1}, S_i), \forall v \in V$ by executing the subroutine *Modified_ Dijkstra*(x, S_i) on each $x \in S_{i-1}$. Following Lemma 4.2, we just need to show that the expected computation cost charged to a vertex v during all these subroutines is bounded by $O(\min(n^{\frac{i+1}{k}},\ \deg(v)n^{1/k})$. Let W_v denote the random variable for the computation cost charged to v. A vertex will charge $\mathcal{E}_{S_i}(v)$ to vertex v iff it belongs to $ball(v, S_{i-1}, S_i)$ and 0 otherwise. So the expected cost charged to v is

$$\mathbf{E}[W_v] = \mathbf{E}[\mathcal{E}_{S_i}(v) \cdot B(v, S_{i-1}, S_i)] \tag{7}$$

Since $|\mathcal{E}_{S_i}(v)| \leq \deg(v)$, so $\mathbf{E}[W_v]$ is bounded by $\deg(v)\mathbf{E}[B(v, S_{i-1}, S_i)]$. It follows from Lemma 2.5 that $\mathbf{E}[B(v, S_{i-1}, S_i)] = n^{1/k}$. Hence the expected cost charged to v is bounded by $O(\deg(v)n^{1/k})$. Now we shall show that it is bounded by $O(n^{\frac{i+1}{k}})$ as well.

Note that all the neighbors of v corresponding to the edges $\mathcal{E}_{S_i}(v)$ are present in $ball(v, V, S_i)$, hence $|\mathcal{E}_{S_i}(v)| \leq B(v, V, S_i)$ always. Using this bound and Equation 7, we bound $\mathbf{E}[W_v]$ as follows.

$$\begin{aligned}
\mathbf{E}[W_v] &\leq \mathbf{E}[B(v, V, S_i) \cdot B(v, S_{i-1}, S_i)] \\
&= \sum_{\forall v' \in V} \mathbf{E}[B(v, V, S_i) | v' \in ball(v, S_{i-1}, S_i)]\ \mathbf{Pr}[v' \in ball(v, S_{i-1}, S_i] \\
&\leq \sum_{\forall v' \in V} \mathbf{E}[B(v, V, S_i) | v' \in ball(v, S_{i-1}, S_i)] n^{\frac{-i+1}{k}} \mathbf{Pr}[v' \in ball(v, V, S_i] \\
&\qquad \{\text{using Lemma 7.4}\} \\
&= n^{\frac{-i+1}{k}} \sum_{\forall v' \in V} \mathbf{E}[B(v, V, S_i) | v' \in ball(v, V, S_i)] \mathbf{Pr}[v' \in ball(v, V, S_i] \\
&\qquad \{\text{using Corollary 7.1 }\} \\
&= n^{\frac{-i+1}{k}} \mathbf{E}[B(v, V, S_i)^2]
\end{aligned}$$

Using arguments similar to those used in Lemma 6.1, it follows that $\mathbf{E}[B(v, V, S_i)^2] = 1/q^2$ with $q = n^{\frac{-i}{k}}$. So $\mathbf{E}[W_v] \leq n^{\frac{i+1}{k}}$.

INFORMATIK

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-2005-5-002	S. Siersdorfer, G. Weikum	Automated Retraining Methods for Document Classification and their Parameter Tuning
MPI-I-2005-4-006	C. Fuchs, M. Goesele, T. Chen, H. Seidel	An Emperical Model for Heterogeneous Translucent Objects
MPI-I-2005-4-005	G. Krawczyk, M. Goesele, H. Seidel	Photometric Calibration of High Dynamic Range Cameras
MPI-I-2005-4-004	C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A.,. Magnor, H. Seidel	Joint Motion and Reflectance Capture for Creating Relightable 3D Videos
MPI-I-2005-4-003	T. Langer, A.G. Belyaev, H. Seidel	Analysis and Design of Discrete Normals and Curvatures
MPI-I-2005-4-002	O. Schall, A. Belyaev, H. Seidel	Sparse Meshing of Uncertain and Noisy Surface Scattered Data
MPI-I-2005-4-001	M. Fuchs, V. Blanz, H. Lensch, H. Seidel	Reflectance from Images: A Model-Based Approach for Human Faces
MPI-I-2005-2-004	Y. Kazakov	A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures
MPI-I-2005-2-003	H.d. Nivelle	Using Resolution as a Decision Procedure
MPI-I-2005-2-002	P. Maier, W. Charatonik, L. Georgieva	Bounded Model Checking of Pointer Programs
MPI-I-2005-2-001	J. Hoffmann, C. Gomes, B. Selman	Bottleneck Behavior in CNF Formulas
MPI-I-2005-1-007	I. Katriel, M. Kutz	A Faster Algorithm for Computing a Longest Common Increasing Subsequence
MPI-I-2005-1-003	S. Baswana, K. Telikepalli	Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs
MPI-I-2005-1-002	I. Katriel, M. Kutz, M. Skutella	Reachability Substitutes for Planar Digraphs
MPI-I-2005-1-001	D. Michail	Rank-Maximal through Maximum Weight Matchings
MPI-I-2004-NWG3-001	M. Magnor	Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae
MPI-I-2004-NWG1-001	B. Blanchet	Automatic Proof of Strong Secrecy for Security Protocols
MPI-I-2004-5-001	S. Siersdorfer, S. Sizov, G. Weikum	Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning
MPI-I-2004-4-006	K. Dmitriev, V. Havran, H. Seidel	Faster Ray Tracing with SIMD Shaft Culling
MPI-I-2004-4-005	I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel	Neural Meshes: Surface Reconstruction with a Learning Algorithm
MPI-I-2004-4-004	R. Zayer, C. Rössl, H. Seidel	r-Adaptive Parameterization of Surfaces

MPI-I-2004-4-003	Y. Ohtake, A. Belyaev, H. Seidel	3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs
MPI-I-2004-4-002	Y. Ohtake, A. Belyaev, H. Seidel	Quadric-Based Mesh Reconstruction from Scattered Data
MPI-I-2004-4-001	J. Haber, C. Schmitt, M. Koster, H. Seidel	Modeling Hair using a Wisp Hair Model
MPI-I-2004-2-007	S. Wagner	Summaries for While Programs with Recursion
MPI-I-2004-2-002	P. Maier	Intuitionistic LTL and a New Characterization of Safety and Liveness
MPI-I-2004-2-001	H. de Nivelle, Y. Kazakov	Resolution Decision Procedures for the Guarded Fragment with Transitive Guards
MPI-I-2004-1-006	L.S. Chandran, N. Sivadasan	On the Hadwiger's Conjecture for Graph Products
MPI-I-2004-1-005	S. Schmitt, L. Fousse	A comparison of polynomial evaluation schemes
MPI-I-2004-1-004	N. Sivadasan, P. Sanders, M. Skutella	Online Scheduling with Bounded Migration
MPI-I-2004-1-003	I. Katriel	On Algorithms for Online Topological Ordering and Sorting
MPI-I-2004-1-002	P. Sanders, S. Pettie	A Simpler Linear Time 2/3 - epsilon Approximation for Maximum Weight Matching
MPI-I-2004-1-001	N. Beldiceanu, I. Katriel, S. Thiel	Filtering algorithms for the Same and UsedBy constraints
MPI-I-2003-NWG2-002	F. Eisenbrand	Fast integer programming in fixed dimension
MPI-I-2003-NWG2-001	L.S. Chandran, C.R. Subramanian	Girth and Treewidth
MPI-I-2003-4-009	N. Zakaria	FaceSketch: An Interface for Sketching and Coloring Cartoon Faces
MPI-I-2003-4-008	C. Roessl, I. Ivrissimtzis, H. Seidel	Tree-based triangle mesh connectivity encoding
MPI-I-2003-4-007	I. Ivrissimtzis, W. Jeong, H. Seidel	Neural Meshes: Statistical Learning Methods in Surface Reconstruction
MPI-I-2003-4-006	C. Roessl, F. Zeilfelder, G. Nürnberger, H. Seidel	Visualization of Volume Data with Quadratic Super Splines
MPI-I-2003-4-005	T. Hangelbroek, G. Nürnberger, C. Roessl, H.S. Seidel, F. Zeilfelder	The Dimension of C^1 Splines of Arbitrary Degree on a Tetrahedral Partition
MPI-I-2003-4-004	P. Bekaert, P. Slusallek, R. Cools, V. Havran, H. Seidel	A custom designed density estimation method for light transport
MPI-I-2003-4-003	R. Zayer, C. Roessl, H. Seidel	Convex Boundary Angle Based Flattening
MPI-I-2003-4-002	C. Theobalt, M. Li, M. Magnor, H. Seidel	A Flexible and Versatile Studio for Synchronized Multi-view Video Recording
MPI-I-2003-4-001	M. Tarini, H.P.A. Lensch, M. Goesele, H. Seidel	3D Acquisition of Mirroring Objects
MPI-I-2003-2-004	A. Podelski, A. Rybalchenko	Software Model Checking of Liveness Properties via Transition Invariants
MPI-I-2003-2-003	Y. Kazakov, H. de Nivelle	Subsumption of concepts in $DL\ \mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete
MPI-I-2003-2-002	M. Jaeger	A Representation Theorem and Applications to Measure Selection and Noninformative Priors
MPI-I-2003-2-001	P. Maier	Compositional Circular Assume-Guarantee Rules Cannot Be Sound And Complete
MPI-I-2003-1-018	G. Schaefer	A Note on the Smoothed Complexity of the Single-Source Shortest Path Problem
MPI-I-2003-1-017	G. Schäfer, S. Leonardi	Cross-Monotonic Cost Sharing Methods for Connected Facility Location Games
MPI-I-2003-1-016	G. Schäfer, N. Sivadasan	Topology Matters: Smoothed Competitive Analysis of Metrical Task Systems
MPI-I-2003-1-015	A. Kovács	Sum-Multicoloring on Paths

MPI-I-2003-1-014	G. Schäfer, L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, T. Vredeveld	Average Case and Smoothed Competitive Analysis of the Multi-Level Feedback Algorithm
MPI-I-2003-1-013	I. Katriel, S. Thiel	Fast Bound Consistency for the Global Cardinality Constraint
MPI-I-2003-1-012		- not published -