All Pairs Almost Shortest Paths *

Dorit Dor † *Shay Halperin* † *Uri Zwick* †

September 10, 1997

Abstract

Let $G = (V, E)$ be an unweighted undirected graph on n vertices. A simple argument shows that computing all distances in G with an additive one-sided error of at most 1 is as hard as Boolean matrix multiplication. Building on recent work of Aingworth, Chekuri and Motwani, we describe an $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ time algorithm $\mathbf{APASP_2}$ for computing all distances in G with an additive one-sided error of at most 2. Algorithm $\mathbf{APASP_2}$ is simple, easy to implement, and faster than the fastest known matrix multiplication algorithm. Furthermore, for every even $k > 2$, we describe an $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ time algorithm $\mathbf{APASP}_k$ for computing all distances in G with an additive one-sided error of at most k.

We also give an $\tilde{O}(n^2)$ time algorithm $\mathbf{APASP}_\infty$ for producing stretch 3 estimated distances in an unweighted and undirected graph on n vertices. No constant stretch factor was previously achieved in $\tilde{O}(n^2)$ time.

We say that a weighted graph $F = (V, E')$ *k-emulates* an unweighted graph $G = (V, E)$ if for every $u, v \in V$ we have $\delta_G(u, v) \leq \delta_F(u, v) \leq \delta_G(u, v) + k$. We show that every unweighted graph on n vertices has a 2-emulator with $\tilde{O}(n^{3/2})$ edges and a 4-emulator with $\tilde{O}(n^{4/3})$ edges. These results are asymptotically tight.

Finally, we show that any *weighted* undirected graph on n vertices has a 3-spanner with $\tilde{O}(n^{3/2})$ edges and that such a 3-spanner can be built in $\tilde{O}(mn^{1/2})$ time. We also describe an $\tilde{O}(n(m^{2/3} + n))$ time algorithm for estimating all distances in a *weighted* undirected graph on n vertices with a stretch factor of at most 3.

Keywords: Graph Algorithms, Shortest Paths, Approximation Algorithms, Spanners, Emulators
AMS Subject Classification: 05C85 68Q25 68R10 05C38

1 Introduction

The all-pairs shortest paths (APSP) problem is one of the most fundamental algorithmic graph problems. The complexity of the fastest known algorithm for solving the problem for weighted directed graphs, without negative cycles, is $O(mn + n^2 \log n)$, where n and m are the number of vertices and edges in the graph (Johnson [Joh77], see also [CLR90]). Algorithms for the APSP problem which work on directed graphs with non-negative edge weights and whose running times are $O(m^*n + n^2 \log n)$, where m^* is the number of edges participating in shortest paths, were obtained by Karger, Koller and Phillips [KKP93] and by McGeoch [McG95]. Karger *et al.* [KKP93] also obtain an $\Omega(mn)$ lower bound on any *path-comparison*

*A preliminary version of this paper appeared in the proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, Vermont, 1996, pages 452–461.

†Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail addresses: {ddorit,hshay,zwick}@math.tau.ac.il

based algorithm for the APSP problem. Takaoka [Tak92], slightly improving a result of Fredman [Fre76], obtained an algorithm for the APSP problem, whose running time is $O(n^3((\log\log n)/\log n)^{1/2})$. These algorithms work again on directed graphs with non-negative edge weights.

The special case of the all-pairs shortest paths problem in which the input graph is unweighted is closely related to matrix multiplication. It is fairly easy to see that solving the APSP problem exactly, even on unweighted graphs, is at least as hard as Boolean matrix multiplication. Recent works, by Alon, Galil and Margalit [AGM97], Alon, Galil, Margalit and Naor [AGMN92], Galil and Margalit [GM93],[GM97] and Seidel [Sei95] have shown that if matrix multiplication can be performed in $O(M(n))$ time, then the APSP problem for unweighted directed graphs can be solved in $\tilde{O}(\sqrt{n^3 M(n)})$ time and the APSP problem for unweighted undirected graphs can be solved in $\tilde{O}(M(n))$ time ($\tilde{O}(f)$ means $O(f\,\mathrm{polylog}\,n)$). The currently best upper bound on matrix multiplication is $M(n) = O(n^{2.376})$ (Coppersmith and Winograd [CW90]).

While the above results are extremely interesting from the theoretical point of view, they are of little use in practice as the fast matrix multiplication algorithms are better than the naive $O(n^3)$ time algorithm only for very large values of n. There is interest therefore in obtaining fast algorithms for the APSP problem that do *not* use fast matrix multiplication. The currently best *combinatorial* algorithm for the unweighted APSP problem is an $O(n^3/\log n)$ time algorithm obtained by Feder and Motwani [FM95] (see also [BKM95]). This offers only a marginal improvement over the naive $O(n^3)$ time algorithm.

As an algorithm for the APSP problem would yield an algorithm with a similar time bound for Boolean matrix multiplication, obtaining a combinatorial $O(n^{3-\epsilon})$ time algorithm for the APSP problem would be a major breakthrough. Here we obtain such combinatorial algorithms for the all-pairs *almost* shortest paths (APASP) problem.

Awerbuch *et al.* [ABCP93] and Cohen [Coh93] considered the problem of finding stretch t all-pairs paths, where t is some fixed constant and a path is of stretch t if its length is at most t times the distance between its endpoints. Cohen [Coh93], improving the results of Awerbuch *et al.* [ABCP93], obtains, for example, an $\tilde{O}(n^{5/2})$ time algorithm for finding stretch $4+\epsilon$ paths and distances in weighted undirected graphs, for any $\epsilon > 0$ (all weights from now on are assumed to be positive). She also exhibits a tradeoff between the running time of the algorithm and the obtained stretch factor. For any even t, stretch $t+\epsilon$ paths between all pairs of vertices can be found in $\tilde{O}(n^{2+2/t})$ time. The works of Awerbuch *et al.* [ABCP93] and Cohen [Coh93] are based on the construction of sparse spanners (Awerbuch [Awe85], Peleg and Schäffer [PS89]). A t-spanner of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G such that for every $u, v \in V$ we have $\delta_{G'}(u, v) \leq t \cdot \delta_G(u, v)$, where $\delta_G(u, v)$ is the distance between the vertices u and v in the (possibly weighted) graph G.

A different approach all together was employed recently by Aingworth, Chekuri, Indyk and Motwani [ACIM96]. They describe a simple and elegant $\tilde{O}(n^{5/2})$ time algorithm for finding all distances in unweighted and undirected graphs with an *additive* one-sided error of at most 2. They also make the very important observation that the small distances, and not the long distances, are the hardest to approximate. Based on the ideas of Aingworth *et al.* [ACIM96], Orlin (unpublished) obtained an $\tilde{O}(n^{7/3})$ time algorithm for finding all distances with an additive one-sided error of at most 4.

In this work we improve and extend the result of Aingworth *et al.* [ACIM96], and of Orlin, and obtain an $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ time algorithm, called $\mathbf{APASP}_2$, for finding all distances in unweighted and undirected graphs with an additive one-sided error of at most 2. Algorithm $\mathbf{APASP}_2$ is just the first in a sequence of algorithms $\mathbf{APASP}_k$, for even $k \geq 2$, that exhibits a trade-off between running time and accuracy. For any even $k > 2$, algorithm $\mathbf{APASP}_k$ runs in $\tilde{O}(\min\{n^{2-\frac{2}{k+2}}m^{\frac{2}{k+2}}, n^{2+\frac{2}{3k-2}}\})$ time and has a one-sided error of at most k. Algorithm $\mathbf{APASP}_4$, for example, runs in $\tilde{O}(n^{5/3}m^{1/3}, n^{11/5})$ time. All algorithms described in this paper can be easily adapted to find almost shortest paths whose lengths are equal to the estimated distances.

In addition, we show that for any $k \geq 2$, the stretch of the estimates produced by algorithm $\mathbf{APASP}_k$ is at most 3. As k increases, the running time of algorithm $\mathbf{APASP}_k$ decreases. For $k = \Theta(\log n)$, the running time becomes $\tilde{O}(n^2)$. We let $\mathbf{APASP}_\infty$ be the algorithm $\mathbf{APASP}_k$ with $k = 2\lfloor \log n \rfloor$. Algorithm $\mathbf{APASP}_\infty$ produces stretch 3 distances in unweighted, undirected graphs in $\tilde{O}(n^2)$ time. As mentioned, no fixed stretch factor was previously achieved in $\tilde{O}(n^2)$ time.

We next introduce the notion of *emulators*. Emulators may be seen as the additive counterparts of spanners. We show that any graph on n vertices has a 2-emulator with $\tilde{O}(n^{3/2})$ edges, and a 4-emulator with $\tilde{O}(n^{4/3})$ edges. These can be constructed in $\tilde{O}(n^{5/2})$ and in $\tilde{O}(n^{7/3})$ time, respectively. We are not able to obtain sparser emulators. We are able, however, to construct 6-emulators of size $\tilde{O}(n^{4/3})$ in $\tilde{O}(n^2)$ time. The bounds on the number of edges in 2-emulators and 4-emulators are asymptotically tight. For any $\epsilon > 0$, there are graphs on n vertices that cannot be 2-emulated by graphs with $n^{3/2-\epsilon}$ edges, and there are graphs on n vertices that cannot be 4-emulated by graphs with $n^{4/3-\epsilon}$ edges.

We are also able to obtain some results for *weighted* undirected graphs. We show that any weighted graph on n vertices has a 3-spanner with $\tilde{O}(n^{3/2})$ edges and that such a 3-spanner can be found in $\tilde{O}(mn^{1/2})$ time. Finally, we describe an $\tilde{O}(n(m^{2/3}+n))$ time algorithm for finding stretch 3 distances in a *weighted* undirected graph on n vertices. Extended and improved results for weighted graphs, including an $\tilde{O}(n^2)$ time algorithm for finding stretch 3 distances and an $\tilde{O}(n^{3/2}m^{1/2})$ time algorithm for finding stretch 2 distances appear in Cohen and Zwick [CZ97].

2 Preliminaries

The work of Aingworth *et al.* [ACIM96] is based on the following simple observation: there is a small set of vertices that dominates all the high degree vertices of a graph. A set of vertices D is said to *dominate* a set U if every vertex in U has a neighbor in D. This observation is also central to our work.

Lemma 2.1 ([ACIM96], see also [AS92], pp. 6-7) *Let $G = (V, E)$ be an undirected graph with n vertices and m edges. Let $1 \leq s \leq n$. A set D of size $O((n \log n)/s)$ that dominates all the vertices of degree at least s in the graph can be found deterministically in $O(m + ns)$ time.*

Note that as $s \leq n$, the running time of this deterministic algorithm is always at most $O(n^2)$. Picking each vertex of V independently at random with probability $(c \log n)/s$, for some large enough $c > 0$, will yield a desired dominating set of size $O((n \log n)/s)$ with high probability. A deterministic algorithm can be obtained using the simple greedy approximation algorithm for the set cover problem. See [ACIM96] for details.

In the subsequent sections, we use an algorithm, called $\mathbf{dominate}(G, s)$, that receives an undirected graph $G = (V, E)$ and a degree threshold s. The algorithm outputs a pair (D, E^*), where D is a set of size $O((n \log n)/s)$ that dominates the set of vertices of degree at least s in G. The set $E^* \subseteq E$ is a set of edges of G of size $O(n)$ such that for every vertex $u \in V$ with degree at least s, there is an edge $(u, u') \in E^*$ such that $u' \in D$. Once a dominating set D is obtained, the set E^* can be easily obtained by adding to E^* a single edge for each vertex of degree at least s.

Another ingredient used by our algorithm is the classical Dijkstra's algorithm.

Lemma 2.2 (Dijkstra's algorithm) *Let $G = (V, E)$ be a weighted directed graph with n vertices and m edges. Let $s \in V$. Dijkstra's algorithm runs in $O(m + n \log n)$ time and finds distances, and a tree of shortest paths, from s to all the vertices of V. Furthermore, if the weights of the edges are integers in the range $\{1, 2, \ldots, n\}$ then the algorithm can be implemented to run in $O(m + n)$ time.*

Dijkstra's algorithm appeared originally in [Dij59], though the running time of the version described there is $O(n^2)$. For a more modern description of Dijkstra's algorithm see [CLR90]. The running time of $O(m+n\log n)$ is obtained by using *Fibonacci heaps* [FT87]. The observation that Dijkstra's algorithm can be implemented to run in $O(m+n)$ time if the weights are in the range $\{1,2,\ldots,n\}$ is a simple exercise.

In all the algorithms described in this paper, except those of Section 7, we start with an *unweighted* undirected graph $G=(V,E)$. We then build many auxiliary *weighted* graphs and run Dijkstra's algorithm on each one of them. The weights of the edges in these auxiliary graphs will always be integers in the range $\{1,2,\ldots,n\}$, so we can use the simple $O(m+n)$ time implementation of Dijkstra's algorithm.

By running Dijkstra's algorithm from every vertex of a graph $G=(V,E)$, we get an $O(mn+n^2\log n)$ time algorithm for solving the all pairs shortest paths problem (APSP). Our goal in this paper is to reduce the running time of APSP algorithms to as close to $\tilde{O}(n^2)$ as possible. To achieve this goal we are willing to settle for *almost* shortest paths instead of genuine shortest paths.

Our algorithms also involve many runs of Dijkstra's algorithm. Most of these runs, however, are performed on graphs with substantially less edges than in the original input graph. A typical step in our algorithms is composed of choosing a vertex $u\in V$, choosing a set of edges F, and then running Dijkstra's algorithm, from u, on the graph $H=(V,F)$. The set of edges F is *not* necessarily a subset of the edge set E of the input graph. Furthermore, the set F *varies* from step to step. The weight of an edge $(u,v)\in F$ is taken to be the currently best upper bound on the distance between u and v in the input graph G. Bounds obtained in a run of Dijkstra's algorithm are used, therefore, in some of the subsequent runs.

In our algorithms, we use a *symmetric* $n\times n$ matrix, denoted $\{\hat{\delta}(u,v)\}_{u,v}$, to hold the currently best upper bounds on distances between all pairs of vertices in the input graph $G=(V,E)$. Initially $\hat{\delta}(u,v)=1$, if $(u,v)\in E$, and $\hat{\delta}(u,v)=+\infty$ otherwise. By $\mathbf{dijkstra}((V,F),\hat{\delta},u)$ we denote an invocation of Dijkstra's algorithm, from the vertex u, on the graph (V,F), where the weight of an edge $(u,v)\in F$ is taken to be $\hat{\delta}(u,v)$. The edges of F are considered to be *undirected*. As mentioned, an edge of F is not necessarily an edge of E. If $(u,v)\in F$ is an edge of the original graph then its weight is 1, otherwise, its weight is greater than 1. A call to $\mathbf{dijkstra}((V,F),\hat{\delta},u)$ updates the row and the column belonging to u in the matrix $\hat{\delta}$ with the distances found during this run, if they are smaller than the previous estimates. Note that the matrix $\{\hat{\delta}(u,v)\}_{u,v}$ serves as both input and output of **dijkstra**.

If the graph (V,F) is a subgraph of the input graph $G=(V,E)$, then a call to $\mathbf{dijkstra}((V,F),\hat{\delta},u)$ amounts to running a BFS on (V,F) from u. When we want to stress this fact, we denote such a call by $\mathbf{bfs}((V,F),\hat{\delta},u)$. In such a call, the matrix $\{\hat{\delta}(u,v)\}_{u,v}$ is only used for output as the weight of each vertex in the input graph is assumed to be 1.

It should be clear from the above discussion that at any time during the run of our algorithms and for any $u,v\in V$ we have $\delta(u,v)\le\hat{\delta}(u,v)$, where $\delta(u,v)$ is the distance between u and v in the input graph G.

3 Additive error 2

Aingworth *et al.* [ACIM96] obtained an $\tilde{O}(n^{5/2})$ algorithm for approximating all distances in an undirected and unweighted graph with a one-sided additive error of 2. We describe two faster algorithms that have the same accuracy. The first algorithm, $\mathbf{apasp}_2$, runs in $\tilde{O}(n^{3/2}m^{1/2})$ time. The second algorithm, $\overline{\mathbf{apasp}}_3$, runs in $\tilde{O}(n^{7/3})$. The first algorithm is faster if the input graph is sufficiently sparse, namely, if $m<n^{5/3}$. By combining these two algorithms we get the algorithm $\mathbf{APASP}_2$ mentioned in the abstract.

A description of the algorithm $\mathbf{apasp}_2$ is given in Figure 1. Throughout the paper, we let $\deg(v)$ denote the degree of a vertex v. The algorithm is extremely simple. It starts by partitioning the vertices of the graph G into two classes, V_1 and $V_2=V\setminus V_1$. The class V_1 includes the high degree vertices, i.e., the

```
Algorithm apasp_2:

input: An unweighted undirected graph G = (V, E).
output: A matrix {δ̂(u, v)}_{u,v} of estimated distances.

s_1 ← (m/n)^{1/2}

V_1 ← {v ∈ V | deg(v) ≥ s_1}
E_2 ← {(u, v) ∈ E | deg(u) < s_1 or deg(v) < s_1}

(D_1, E*) ← dominate(G, s_1)
For every u, v ∈ V let δ̂(u, v) ← { 1    if (u, v) ∈ E,
                                  { +∞   otherwise.

For every u ∈ D_1 run bfs(G, δ̂, u)
For every u ∈ V \ D_1 run dijkstra( (V, E_2 ∪ E* ∪ ({u} × D_1)) , δ̂, u)
```

Figure 1: An $\tilde{O}(n^{3/2}m^{1/2})$ time algorithm for computing surplus 2 distances

vertices of degree at least $s_1 = (m/n)^{1/2}$. The class V_2 includes all the low degree vertices, i.e., the vertices of degree less than $s_1 = (m/n)^{1/2}$. A similar partition is used by Aingworth *et al.* [ACIM96] and also by Alon, Yuster and Zwick [AYZ97]. The edge set E_2 is composed of all the edges that touch a low degree vertex and therefore $|E_2| = O(n \cdot (m/n)^{1/2}) = O((nm)^{1/2})$. The algorithm proceeds by finding a set D_1 of size $\tilde{O}(n^{3/2}/m^{1/2})$ that dominates the vertices of V_1 and an edge set $E^* \subseteq E$ of size $O(n)$ such that for every $u \in V_1$ there exists $v \in D_1$ such that $(u, v) \in E^*$. Finally, the main part of the algorithm is composed of two steps. In the first, a BFS is performed on the graph G from every vertex $u \in D_1$. In the second and final step, Dijkstra's algorithm is run, from every $u \in V \setminus D_1$, on the graph $G_2(u) = (V, E_2 \cup E^* \cup (\{u\} \times D_1))$. It is important to note that Dijkstra's algorithm is not run on the input graph G, that may contain too many edges, and that a slightly different graph is used for each vertex $u \in V \setminus D_1$. The graph $G_2(u)$, on which Dijkstra's algorithm is run from u, includes all the edges that touch low degree vertices, edges that connect each high degree vertex with a vertex of the dominating set, and edges connecting u with all the vertices of the dominating set. The number of edges in $G_2(u)$ is therefore $O((nm)^{1/2})$. It is also important to note that the graph $G_2(u)$ is a *weighted* graph. The weight of the edges in $E_2 \cup E^*$ is 1, as in the unweighted graph. The weight of an edge $(u, v) \in \{u\} \times D_1$, however, is the distance between u and v in G. This distance was found by the BFS that started at $v \in D_1$.

Theorem 3.1 *Algorithm* **apasp$_2$** *runs in $\tilde{O}(n^{3/2}m^{1/2})$ time, where n is the number of vertices and m is the number of edges in the input graph $G = (V, E)$, and for every $u, v \in V$ we have $\delta(u, v) \le \hat{\delta}(u, v) \le \delta(u, v) + 2$.*

Proof: We start with the complexity analysis. Finding the dominating set D_1 requires, according to Lemma 2.1, only $O(n^2)$ time. As $|D_1| = \tilde{O}(n^{3/2}/m^{1/2})$, the total running time of all the BFS's is $\tilde{O}(n^{3/2}/m^{1/2} \cdot m) = \tilde{O}(n^{3/2}m^{1/2})$. As the number of edges in each graph on which Dijkstra's algorithm is applied is $\tilde{O}((nm)^{1/2})$, the total running time of all these calls is also $\tilde{O}(n \cdot (nm)^{1/2}) = \tilde{O}(n^{3/2}m^{1/2})$ and this is also the running time of the whole algorithm.

We now examine the accuracy of the algorithm. The weights attached to the weighted edges in the graphs $G_2(u)$ are distances in the graph G. This implies that the approximations produced by the algorithm cannot be too small. In other words, $\delta(u, v) \le \hat{\delta}(u, v)$ for every $u, v \in V$. We now prove that $\hat{\delta}(u, v) \le \delta(u, v) + 2$

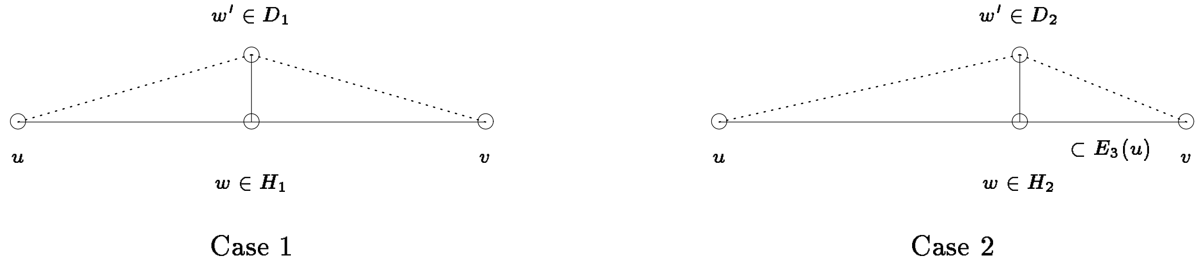


Figure 2: (a) Case 1 in the proof of Theorem 3.1. (b) Case 1 in the proof of Theorem 3.2.

for every $u, v \in V$. Let u and v be two vertices in G. We consider the following two (non-exclusive but exhaustive) cases:

Case 1: There is a shortest path between u and v that contains a vertex from V_1.

Let w be the *last* vertex on the path that belongs to V_1 (see Figure 2(a)). All the edges on the path from w to v touch vertices in V_2 and therefore belong to E_2, and therefore to $G_2(u)$. Let $w' \in D_1$ be such that $(w, w') \in E^*$. The edge (w, w') also belongs to $G_2(u)$. As $w' \in D_1$, a weighted edge (u, w') was added to $G_2(u)$. The weight of this edge is $\delta(u, w')$, the distance between u and w' in G, found by the BFS from $w' \in D_1$. Note that $\delta(u, w') \leq \delta(u, w) + 1$. By running Dijkstra's algorithm from u, on $G_2(u)$, we find therefore that

$$\begin{aligned} \hat{\delta}(u, v) &\leq \delta(u, w') + \delta(w', w) + \delta(w, v) \\ &\leq (\delta(u, w) + 1) + 1 + \delta(w, v) = \delta(u, v) + 2 \;. \end{aligned}$$

Case 2: There is a shortest path between u and v that does not contain any vertex from V_1.

This shortest path is contained in (V, E_2) and therefore $\hat{\delta}(u, v) = \delta(u, v)$. □

Algorithm $\overline{\mathbf{apasp}}_3$, described in Figure 3, is similar to algorithm $\mathbf{apasp}_2$. Instead of dividing the vertices into two classes according to their degrees, we now divide them into three classes. Instead of using the 'threshold' $(m/n)^{1/2}$, we now use the two thresholds $n^{1/3}$ and $n^{2/3}$. Another important difference between $\overline{\mathbf{apasp}}_3$ and $\mathbf{apasp}_2$ is that the graphs on which Dijkstra's algorithm is run now contain the edges $(D_1 \times V) \cup (D_2 \times D_2)$. These edges are weighted. The weight of an edge $(u, v) \in D_1 \times V$ is the distance between u and v in G, found by the BFS from $u \in D_1$. The weight of an edge $(u, v) \in D_2 \times D_2$ is the distance between u and v in the graph (V, E_2). Note that this distance may be larger then the distance between u and v in G.

Theorem 3.2 *Algorithm $\overline{\mathbf{apasp}}_3$ runs in $\tilde{O}(n^{7/3})$ time, where n is the number of vertices in the input graph $G = (V, E)$, and for every $u, v \in V$ we have $\delta(u, v) \leq \hat{\delta}(u, v) \leq \delta(u, v) + 2$.*

Proof: We start again with the complexity analysis. Finding the two dominating sets D_1 and D_2 requires only $O(n^2)$ time. As $|D_1| = \tilde{O}(n^{1/3})$, $|D_2| = \tilde{O}(n^{2/3})$, $|E_2| = O(n^{5/3})$ and $|E_3| = O(n^{4/3})$, the BFS's from the vertices of D_1 take $\tilde{O}(n^{1/3} \cdot n^2) = \tilde{O}(n^{7/3})$ time and the BFS's from the vertices of D_2 take $\tilde{O}(n^{2/3} \cdot n^{5/3}) = \tilde{O}(n^{7/3})$ time. As $|D_1 \times V| = \tilde{O}(n^{4/3})$ and $|D_2 \times D_2| = \tilde{O}(n^{4/3})$, each graph on which Dijkstra's algorithm is run has only $\tilde{O}(n^{4/3})$ edges. The total time taken by all these runs is therefore $\tilde{O}(n \cdot n^{4/3}) = \tilde{O}(n^{7/3})$.

It is again clear that $\delta(u, v) \leq \hat{\delta}(u, v)$, for every $u, v \in V$. It remains to show therefore that $\hat{\delta}(u, v) \leq \delta(u, v) + 2$, for every $u, v \in V$. Let u and v be two vertices in G. We consider the following three cases:

Case 1: There is a shortest path between u and v that contains a vertex w from V_1.

```
Algorithm apasp_3 (overlined):

input: An unweighted undirected graph G = (V, E).
output: A matrix {δ̂(u, v)}_{u,v} of estimated distances.
```

Algorithm $\overline{\mathbf{apasp}}_3$:

input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\{\hat{\delta}(u,v)\}_{u,v}$ of estimated distances.

$s_1 \leftarrow n^{2/3}$; $s_2 \leftarrow n^{1/3}$

For $i \leftarrow 1$ to 2 let $V_i \leftarrow \{v \in V \mid \deg(v) \geq s_i\}$
For $i \leftarrow 2$ to 3 let $E_i \leftarrow \{(u,v) \in E \mid \deg(u) < s_{i-1}$ or $\deg(v) < s_{i-1}\}$

For $i \leftarrow 1$ to 2 let $(D_i, E_i^*) \leftarrow \mathbf{dominate}(G, s_i)$
$E^* \leftarrow E_1^* \cup E_2^*$
For every $u, v \in V$ let $\hat{\delta}(u,v) \leftarrow \begin{cases} 1 & \text{if } (u,v) \in E, \\ +\infty & \text{otherwise.} \end{cases}$

For every $u \in D_1$ run $\mathbf{bfs}(G, \hat{\delta}, u)$
For every $u \in D_2$ run $\mathbf{bfs}((V, E_2), \hat{\delta}, u)$
For every $u \in V$ run $\mathbf{dijkstra}(\,(V, E_3 \cup E^* \cup (D_1 \times V) \cup (D_2 \times D_2) \cup (\{u\} \times D_2))\,, \hat{\delta}, u)$

Figure 3: An $\tilde{O}(n^{7/3})$ time algorithm for computing surplus 2 distances

Let $w' \in D_1$ such that $(w, w') \in E$ (see Figure 2(b)). (Note that we do not require $(w, w') \in E^*$, although we could.) The edges (u, w') and (w', v) belong to the graph on which Dijkstra's algorithm is run from u. The weights of these edges are $\delta(u, w')$ and $\delta(w', v)$, the distances found by the BFS on G from $w' \in D_1$. Note that $\delta(u, w') \leq \delta(u, w) + 1$ and $\delta(w', v) \leq 1 + \delta(w, v)$. By running Dijkstra's algorithm from u, we find therefore that

$$\hat{\delta}(u,v) \leq \delta(u,w') + \delta(w',v) \leq \delta(u,v) + 2 \; .$$

Case 2: There is a shortest path between u and v that contains vertices from V_2 but not from V_1.

This case is very similar to case 1 in the proof of Theorem 3.1. Let w be the *last* vertex on the path that belongs to V_2. All the edges on the path from w to v touch vertices that do not belong to V_2 and therefore belong to the set E_3 and to the graph $G_3(u)$ on which Dijkstra's algorithm is run from u. Let $w' \in D_2$ be such that $(w, w') \in E^*$. The graph $G_3(u)$ contains weighted edges connecting u to all the vertices of D_2. It contains in particular a weighted edge (u, w'). The weight of this edge is the distance $\delta_2(u, w')$ between u and w' in the graph $G_2 = (V, E_2)$, found by the BFS from $w' \in D_2$ on G_2. As all the edges on the path from u to w, as well as (w, w') belong to E_2, we get that $\delta_2(u, w') \leq \delta(u, w) + 1$. By running Dijkstra's algorithm from u we find therefore that

$$\hat{\delta}(u,v) \leq \delta_2(u,w') + \delta(w',w) + \delta(w,v) \leq \delta(u,v) + 2 \; .$$

Case 3: There is a shortest path between u and v that does not contain any vertex from V_2.

This shortest path is then contained in (V, E_3) and therefore $\hat{\delta}(u,v) = \delta(u,v)$. □

The above proof remains correct even if the edge set $D_2 \times D_2$ is not added to the graphs on which Dijkstra's algorithm is run. We have added it as it may improve the accuracy of the algorithm, in certain cases, and as it is used in the proof of Theorem 6.3. We can also replace the edge set $\{u\} \times D_1$, in algorithm $\mathbf{apasp}_2$, and the edge set $\{u\} \times D_1$, in algorithm $\overline{\mathbf{apasp}}_3$, by the larger edge set $\{u\} \times V$ without increasing the running times of the algorithms.

We can easily get a randomized version of algorithm $\overline{\mathbf{apasp}_3}$ which has the property that *all* reported distances greater than $n^{2/3}$ are, with high probability, correct. Similarly, we can get a randomized version of $\mathbf{apasp}_2$ for which all reported distances greater than $n^{3/2}/m^{1/2}$ are, with high probability, correct. We use the following simple observation (a similar idea is used by Ullman and Yannakakis [UY91]):

Theorem 3.3 *Let $G = (V, E)$ be a weighted directed graph with n vertices and m edges. Let $1 \le s \le n$. There is an $\tilde{O}(n^3/s)$ time randomized algorithm that finds, with high probability, the exact distance between any pair of vertices connected by a shortest path that uses at least s edges.*

Proof: Let D be a random set of vertices obtained by picking each vertex independently with probability $(c \log n)/s$, for some large enough $c > 0$. The expected size of D is $\tilde{O}(n/s)$. Run Dijkstra's algorithm from each vertex of D in both G, and the graph obtained from G by reversing the edges. The complexity of this step is $\tilde{O}(nm/s)$. For every $u \in D$ and $v \in V$, we now know $\delta(u, v)$ and $\delta(v, u)$ exactly. For every pair of vertices $u, v \in V$, let $\hat{\delta}(u, v) = \min_{w \in D}\{\delta(u, w) + \delta(w, v)\}$. The complexity of this step is $\tilde{O}(n^3/s)$. It is easy to see that $\hat{\delta}(u, v) = \delta(u, v)$ if and only if there is a shortest path between u and v that passes through a vertex of D. If there is a shortest path between u and v of length s, then with high probability, at least one of the vertices on the path will belong to D. As there are $O(n^2)$ pairs of vertices connected by shortest paths that use at most s edges, and as we can focus on one such path for each pair, we get that, with high probability, each one of these $O(n^2)$ paths will pass through a vertex of D, and the exact distances between all these pairs will be found. $\square$

It follows that if the set D_1 in algorithm $\overline{\mathbf{apasp}_3}$ is chosen at random, by picking each element with probability $cn^{-2/3} \log n$, then, with high probability, if $\delta(u, v) \ge n^{2/3}$ then $\hat{\delta}(u, v)$ is the exact distance between u and v. Long distances are therefore easier to compute.

4 Trading time and accuracy

Aingworth *et al.* [ACIM96] obtained their $\tilde{O}(n^{5/2})$ algorithm by splitting the vertices into two classes according to their degree. We obtain our $\tilde{O}(n^{7/3})$ time algorithm by dividing them into three classes. It is natural to try to divide the vertices into more classes. In Figure 4 we describe an algorithm $\mathbf{apasp}_k$ that divides the vertices into k classes and runs in $\tilde{O}(n^{2-1/k}m^{1/k})$ time. Algorithm $\mathbf{apasp}_2$ described in Figure 1 is a special case of this more general algorithm. We next show that algorithm $\mathbf{apasp}_k$ has an additive error of at most $2(k-1)$.

Theorem 4.1 *For every $2 \le k = O(\log n)$, algorithm $\mathbf{apasp}_k$ runs in $\tilde{O}(n^{2-1/k}m^{1/k})$ time, where n is the number of vertices and m is the number of edges in the input graph $G = (V, E)$, and for every $u, v \in V$ we have $\delta(u, v) \le \hat{\delta}(u, v) \le \delta(u, v) + 2(k-1)$.*

Proof: We start again with the complexity analysis. For every $1 \le i \le k$ and $u \in D_i$, let $E_i(u)$ be the edge set of the graph on which Dijkstra's algorithm is run from u. It is easy to check that $|D_i| = \tilde{O}(n^{2-i/k}/m^{1-i/k})$, and that $|E_i| = \tilde{O}(n \cdot (m/n)^{1-(i-1)/k}) = \tilde{O}(n^{(i-1)/k}m^{1-(i-1)/k})$, for every $1 \le i \le k$. The cost of running Dijkstra's algorithm from every $u \in D_i$, where $1 \le i \le k$, is therefore $\tilde{O}(n^{2-i/k}/m^{1-i/k} \cdot n^{(i-1)/k}m^{1-(i-1)/k}) = \tilde{O}(n^{2-1/k}m^{1/k})$. The total running time of the algorithm is therefore $\tilde{O}(k \cdot n^{2-1/k}m^{1/k}) = \tilde{O}(n^{2-1/k}m^{1/k})$, as $k = O(\log n)$.

For every $1 \le i \le k$ and $u, v \in V$, let $\delta_i(u, v)$ be the value of $\hat{\delta}(u, v)$ after running **dijkstra** from all the vertices of D_i. We now prove by induction on i, that if $u \in D_i$ and $v \in V$, then $\delta(u, v) \le \delta_i(u, v) \le \delta(u, v) + 2(i-1)$. Recall that $D_k = V$. For every $u, v \in V$, we get that $\hat{\delta}(u, v) = \delta_k(u, v) \le \delta(u, v) + 2(k-1)$, as required.

```
Algorithm apasp_k:

input: An unweighted undirected graph G = (V, E).
output: A matrix {δ̂(u,v)}_{u,v} of estimated distances.

For i ← 1 to k − 1 let s_i ← (m/n)^{1−i/k}

For i ← 1 to k − 1 let V_i ← {v ∈ V | deg(v) ≥ s_i}
For i ← 2 to k let E_i ← {(u,v) ∈ E | deg(u) < s_{i−1} or deg(v) < s_{i−1}}

For i ← 1 to k − 1 let (D_i, E_i^*) ← dominate(G, s_i)
E_1 ← E ; D_k ← V ; E^* ← ∪_{1≤i<k} E_i^*

For every u, v ∈ V let δ̂(u,v) ← { 1    if (u,v) ∈ E,
                                  { +∞   otherwise.

For i ← 1 to k do
For every u ∈ D_i run dijkstra( (V, E_i ∪ E^* ∪ ({u} × V)) , δ̂, u)
```

Figure 4: An $\tilde{O}(n^{2-1/k}m^{1/k})$ time algorithm for computing surplus $2(k-1)$ distances

Let $G_i(u) = (V, E_i(u))$ be the graph on which Dijkstra's algorithm is run from $u \in D_i$. For $i = 1$, the claim is clear, as for every $u \in D_1$ we have $G_1(u) = G$, and therefore $\delta_1(u,v) = \delta(u,v)$ for every $u \in D_1$ and $v \in V$. Suppose therefore that $i > 1$ and that the claim is true for $i-1$. Let $u \in D_i$ and let $v \in V$. Consider a shortest path p from u to v. If all the edges on p belong to E_i, then $\delta_i(u,v) = \delta(u,v)$ and we are done. Otherwise, there must be a vertex from V_{i-1} on the path p. The argument that follows is again similar to the argument used in case 1 in the proof of Theorem 3.1 and case 2 in the proof of Theorem 3.2. Let w be the last vertex from V_{i-1} on the path p. Let p' be the subpath of p that connects w and v. As all the vertices on p', except w, do not belong to V_{i-1}, all the edges of p' belong to E_i. Let $w' \in D_{i-1}$ be such that $(w,w') \in E^*$. The graph $G_i(u)$ contains the edge (w,w') and a weighted edge (u,w') whose weight is $\delta_{i-1}(u,w') = \delta_{i-1}(w',u)$. By the induction hypothesis, $\delta_{i-1}(w',u) \leq \delta(w',u) + 2(i-2) \leq \delta(u,w) + (2i-3)$. As a consequence, we get that

$$\begin{aligned}
\delta_i(u,v) &\leq \delta_{i-1}(u,w') + \delta(w',w) + \delta(w,v) \\
&\leq (\delta(u,w) + (2i-3)) + 1 + \delta(w,v) \\
&\leq \delta(u,v) + 2(i-1)\ .
\end{aligned}$$

This completes the proof of the theorem. □

As in the previous section, we can obtain a slightly better algorithm for denser graphs. Algorithm $\overline{\mathbf{apasp}}_k$ is described in Figure 5. There are two differences between $\mathbf{apasp}_k$ and $\overline{\mathbf{apasp}}_k$. The first is that the degree thresholds are now $s_i = n^{1-i/k}$, and not $s_i = (m/n)^{1-i/k}$. The second is that for any $1 \leq j_1, j_2 \leq k$ such that $i + j_1 + j_2 \leq 2k+1$, the edges of $D_{j_1} \times D_{j_2}$ are added to the graph on which Dijkstra's algorithm is run from every vertex $u \in D_i$.

Theorem 4.2 *For every $2 \leq k = O(\log n)$, algorithm $\overline{\mathbf{apasp}}_k$ runs in $\tilde{O}(n^{2+1/k})$ time, where n is the number of vertices in the input graph $G = (V,E)$, and for every $u, v \in V$ we have $\delta(u,v) \leq \hat{\delta}(u,v) \leq \delta(u,v) + 2(\lfloor k/3 \rfloor + 1)$.*

Algorithm $\overline{\textbf{apasp}}_k$:

input: An unweighted undirected graph $G = (V, E)$.
output: A matrix $\{\hat{\delta}(u,v)\}_{u,v}$ of estimated distances.

For $i \leftarrow 1$ to $k-1$ let $s_i \leftarrow n^{1-\frac{i}{k}}$

For $i \leftarrow 1$ to $k-1$ let $V_i \leftarrow \{v \in V \mid \deg(v) \geq s_i\}$
For $i \leftarrow 2$ to k let $E_i \leftarrow \{(u,v) \in E \mid \deg(u) < s_{i-1} \text{ or } \deg(v) < s_{i-1}\}$

For $i \leftarrow 1$ to $k-1$ let $(D_i, E_i^*) \leftarrow$ **dominate**(G, s_i)
$E_1 \leftarrow E$; $D_k \leftarrow V$; $E^* \leftarrow \cup_{1 \leq i < k} E_i^*$

For every $u, v \in V$ let $\hat{\delta}(u,v) \leftarrow \begin{cases} 1 & \text{if } (u,v) \in E, \\ +\infty & \text{otherwise.} \end{cases}$

For $i \leftarrow 1$ to k do
For every $u \in D_i$ run **dijkstra**($(V, E_i \cup E^* \cup (\{u\} \times V)) \cup (\cup_{i+j_1+j_2 \leq 2k+1} D_{j_1} \times D_{j_2})$, $\hat{\delta}, u$)

Figure 5: An $\tilde{O}(n^{2+1/k})$ time algorithm for computing surplus $O(k)$ distances

Proof: We start again with the complexity analysis. It is easy to see that all the preparatory steps of the algorithm take $\tilde{O}(n^2)$ time. For every $1 \leq i \leq k$, and for every $u \in D_i$, we then run Dijkstra's algorithm, from u, on the graph $G_i(u) = (V, E_i(u))$. What is the number of edges in $E_i(u)$? The number of edges in E_i is $O(n^{2-(i-1)/k})$ as each edge of E_i touches a vertex of degree less than $n^{1-(i-1)/k}$. The number of edges in E^* is at most n. This accounts for all the non-weighted edges in $G_i(u)$. The graph $G_i(u)$ contains two types of weighted edges. Edges of the first type connect u with all the other vertices of the graph. There are at most n such edges. Edges of the second type connect vertices of D_{j_1} and D_{j_2} where $1 \leq j_1, j_2 \leq k$ and $i + j_1 + j_2 \leq 2k+1$. The number of edges in $D_{j_1} \times D_{j_2}$ is $n^{j_1/k} \cdot n^{j_2/k}$, which by the condition $i + j_1 + j_2 \leq 2k+1$, is at most $n^{2-(i-1)/k}$. The total number of edges in $E_i(u)$, summing over all valid choices of j_1 and j_2 is therefore $\tilde{O}(n^{2-(i-1)/k})$ (we use the fact that $k = O(\log n)$). For every $1 \leq i \leq k$, we therefore run Dijkstra's algorithm $\tilde{O}(n^{i/k})$ times on graphs with $\tilde{O}(n^{2-(i-1)/k})$ edges. The total running time of step 4 is therefore $\tilde{O}(n^{2+1/k})$. This is also the complexity of the algorithm.

We now study the accuracy of the distance estimates produced $\overline{\textbf{apasp}}_k$. For every $1 \leq i \leq k$, we let $\delta_i(u,v)$ be the value of $\hat{\delta}(u,v)$ after running **dijkstra** from all the vertices of D_i. We now define recursively the following sequence

$$e_{i_1,j,i_2} = \left\{ \begin{array}{cl} 0 & \text{if } i_1 \leq j \\ 2 & \text{if } i_1 + j + i_2 \leq 2k+1 \\ e_{i_1-1,j,i_1} + 2 & \text{otherwise} \end{array} \right\}$$

and prove by induction the following claim:

Claim: If $u \in D_{i_1}$ and $v \in D_{i_2}$ are connected by a path p of length ℓ in which the vertex of highest degree belongs to V_j, then $\delta_{i_1}(u,v) \leq \ell + e_{i_1,j,i_2}$.

To prove the claim we use essentially the same arguments used in the proofs of Theorems 3.1, 3.2 and 4.1.

If $i_1 \leq j$, then all the edges on the path p are contained in $E_{i_1}(u)$ and therefore $\delta_{i_1}(u,v) \leq \ell$, as required.

Suppose now that $i_1 + j + i_2 \leq 2k+1$ and that $j < i_1$. This means that $D_j \times D_{i_2} \subseteq E_{i_1}(u)$. Let w be a vertex on p that belongs to V_j. Let ℓ_1 be the distance from u to w on the path p. Let ℓ_2 be the distance from w to v on the path p. Clearly $\ell = \ell_1 + \ell_2$. Let $w' \in D_j$ such that $(w, w') \in E^*$. It is easy to see that $\delta_j(u, w') \le \ell_1 + 1$ and $\delta_j(w', v) \le \ell_2 + 1$. As $(u, w'), (w', v) \in E_{i_1}(u)$, we get that $\delta_{i_1}(u, v) \le \ell + 2$, as required.

Finally, suppose that $i_1 + j + i_2 > 2k + 1$ and that $j < i_1$. As $j < i_1$, we get that $V_j \subseteq V_{i_1 - 1}$. Let w be the *last* vertex on p that belongs to V_{i_1-1}. Let $w' \in D_{i_1-1}$ such that $(w, w') \in E^*$. Let ℓ_1 and ℓ_2 be, as before, the distances from u to w, and from w to v on p. Consider the path p' composed of the edge (w', w) and the portion of the path p from w to u. The path p' starts at $w' \in D_{i_1-1}$, ends at $u \in D_{i_1}$, and the vertex of highest degree on it belongs to V_j. The length of p' is $\ell_1 + 1$. By applying the claim inductively to p', we get that $\delta_{i_1-1}(w', u) \le \ell_1 + 1 + e_{i_1-1,j,i_1}$. As w is the last vertex from v_{i_1-1} on p, all the edges on the portion of p from w to v belong to $E_{i_1}(u)$. The set $E_{i_1}(u)$ also contains the edge $(w', w) \in E^*$ and a weighted edge (u, w') of weight $\delta_{i_1-1}(w', u) = \delta_{i_1-1}(u, w')$. After running **dijkstra** from u on $(V, E_i(u))$ we get $\delta_{i_1}(u, v) \le (\ell_1 + 1 + e_{i_1-1,j,i_1}) + (1 + \ell_2) = \ell + (e_{i_1-1,j,i_1} + 2) = \ell + e_{i_1,j,i_2}$, as required. This completes the proof of the claim.

Now let $e_{i,j} = e_{i,j,i+1}$. It is easy to verify that

$$e_{i,j} = \left\{ \begin{array}{cl} 0 & \text{if } \ i \le j \\ 2 & \text{if } \ 2i + j \le 2k \\ e_{i-1,j} + 2 & \text{otherwise} \end{array} \right\} .$$

It is not difficult to unwind this recursion and get that

$$e_{i,j} = \left\{ \begin{array}{cl} 0 & \text{if } \ i \le j \\ 2 & \text{if } \ 2i + j \le 2k \\ \min\{\, 2(i-j)\,,\, 2(i - k + \lceil \frac{j}{2} \rceil + 1) \,\} & \text{otherwise} \end{array} \right\} .$$

Finally, we get

$$e_{k,j,k} = \left\{ \begin{array}{cl} 0 & \text{if } \ j = k \\ 2 & \text{if } \ j = 1 \\ e_{k-1,j} + 2 & \text{otherwise} \end{array} \right\} = \left\{ \begin{array}{cl} 0 & \text{if } \ j = k \\ 2 & \text{if } \ j = 1 \\ \min\{\, 2(k-j)\,,\, 2\lceil \frac{j}{2} \rceil + 2 \,\} & \text{otherwise} \end{array} \right\} .$$

It is now not difficult to verify that for every $1 \le j \le k$, we have $e_{k,j,k} \le 2(\lfloor k/3 \rfloor + 1)$. As $D_k = V$, this completes the proof of the theorem. $\square$

To get an additive error of at most k, where $k > 2$ is even, we can either use algorithm $\mathbf{apasp}_{k/2+1}$, whose running time is $\tilde{O}(n^{2-\frac{2}{k+2}} m^{\frac{2}{k+2}})$, or algorithm $\overline{\mathbf{apasp}}_{(3k-2)/2}$, whose running time is $\tilde{O}(n^{2+\frac{2}{3k-2}})$. The combination of these two algorithms is the algorithm $\mathbf{APASP}_k$ mentioned in the abstract.

Although the additive error of the estimated distances produced by $\mathbf{apasp}_k$ and $\overline{\mathbf{apasp}}_k$ increases as k increases, we can show that the *stretch* of the estimated distances produced is always at most 3.

Theorem 4.3 *For every $2 \le k = O(\log n)$, algorithm* $\mathbf{apasp}_k$ *runs in $\tilde{O}(n^{2-1/k} m^{1/k})$ time, and for every $u, v \in V$ we have $\delta(u, v) \le \hat{\delta}(u, v) \le \min\{\, \delta(u, v) + 2(k-1)\,,\, 3\delta(u, v) - 2 \,\}$.*

Proof: We only have to show that for every $u, v \in V$ we have $\hat{\delta}(u, v) \le 3\delta(u, v) - 2$. All the rest follows from Theorem 4.1. We start with the following claim:

Claim 4.4 *Let p be a path of length ℓ between u and v, where $u \in D_i$. If $\delta_i(u, v) > \ell + 2r$, then there are at least $r + 1$ edges on the path p that do not belong to $E_i \cup E^*$.*

We prove this claim by induction on i. If $i = 1$, then $\delta_1(u,v) = \delta(u,v) \le \ell$ and there is nothing to prove.

Suppose that the claim is true for every $j < i$. Let p be a path of length ℓ between $u \in D_i$ and v, and suppose that $\delta_i(u,v) > \ell + 2r$. It follows that not all the edges of p belong to $E_i(u)$. This completes the proof if $r = 0$. Suppose therefore that $r > 0$. Let $e = (w_1, w_2)$ be the last edge on p that does not belong to $E_i(u)$. Let $1 \le j < i$ be such that $w_2 \in V_j \setminus V_{j-1}$ (where $V_0 = \phi$). As $w_2 \notin V_{j-1}$, we get that $e \in E_j$. Let $w_2' \in D_j$ such that $(w_2, w_2') \in E^*$. Let ℓ_1 and ℓ_2 be the distances from u to w_2 and from w_2 to v on p. As $\delta_i(u,v) > \ell + 2r$, on the one hand, and $\delta_i(u,v) \le \delta_j(w_2', u) + 1 + \ell_2$, on the other, we get that $\delta_j(w_2', u) > (\ell_1 + 1) + 2(r-1)$. Let p' be the path from w_2' to u composed of the edge (w_2', w_2) and the portion of p from w_2 to u. According to the induction hypothesis, there must be at least r edges on p' that do not belong to $E_j \cup E^*$. As $(w_2', w_2) \in E^*$ we get that all these r edges must be edges of p. As $e \in E_j$, we get that e is not one of these edges. As $e \notin E_i \cup E^*$, we get that there must be at least $r+1$ edges on p that do not belong to $E_i \cup E^*$, as required. This completes the proof of the claim.

A path p of length ℓ may contain at most ℓ edges that do not belong to $E_k \cup E^*$. Thus, if p is a path of length ℓ connecting $u \in D_k$ and v, then it follows from the above claim that $\hat{\delta}(u,v) \le 3\ell$. Note that $D_k = V$. This is almost what we wanted to show, but not quite.

Consider again a path p of length ℓ between two vertices u and v. If at least one of the edges of p belongs to $E_k \cup E^*$, then Claim 4.4 implies that $\hat{\delta}(u,v) \le 3\ell - 2$. If the path p is of length one, i.e., $\ell = 1$, then $\hat{\delta}(u,v) = \delta(u,v) = 1 = 3\ell - 2$. Otherwise, we know that $e_1 = (u, u')$ and $e_2 = (v', v)$, the first and last edges on the path p, do not belong to $E_k \cup E^*$. As a consequence, we get that $u, v \in V_{k-1}$. Let $1 \le j_1 < k$ and $1 \le j_2 < k$ be such that $u \in V_{j_1} \setminus V_{j_1 - 1}$ and $v \in V_{j_2} \setminus V_{j_2-1}$ (where again $V_0 = \phi$). Assume, without loss of generality, that $j_2 \le j_1$. For brevity, let $j = j_2$. As $u, v \notin V_{j-1}$, we get that $e_1, e_2 \in E_j$. Let $w \in D_j$ such that $(v, w) \in E^*$. Let p' be the path from w to u composed of the edge (w, v) and the path p in reversed order. As the number of edges on the path p' that do not belong to $E_j \cup E^*$ is at most $\ell - 2$, we get, by Claim 4.4, that $\delta_j(w,u) \le (\ell+1) + 2(\ell-2) = 3\ell - 3$. Thus, $\delta_k(u,v) \le \delta_j(w,u) + 1 \le 3\ell - 2$, as required. $\square$

As an immediate corollary, we get that the estimated distances produced by **apasp**$_k$ satisfy $\delta(u,v) \le \hat{\delta}(u,v) \le (3 - \frac{2}{k})\delta(u,v)$, for every $u, v \in V$. For $k = 2$, the distances are stretched by a factor of at most 2. For $k = 3$, the distances are stretched by a factor of at most 7/3. For any k, the distances are stretched by a factor of at most 3.

By taking $k = \Theta(\log n)$, we get an $\tilde{O}(n^2)$ time algorithm for finding stretch 3 approximate distances. An extension of this algorithm for weighted graph is presented in [CZ97].

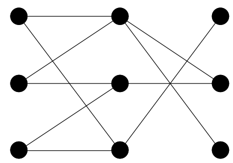
5 Boolean Matrix Multiplication

Let A and B be two Boolean $n \times n$ matrices. Construct a graph $G_{A,B} = (V, E)$ with

$$\begin{aligned} V &= \{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_n\} \cup \{w_1, \ldots, w_n\} , \\ E &= \{(u_i, v_k) \mid a_{ik} = 1\} \cup \{(v_k, w_j) \mid b_{kj} = 1\} . \end{aligned}$$

The graph corresponding to two 3×3 matrices is depicted in Figure 6. Let $C = A \times B$ (Boolean matrix multiplication). Clearly, $c_{ij} = 1$ if and only if $\delta_G(u_i, w_j) = 2$. Furthermore, as the graph $G_{A,B}$ is bipartite, $c_{ij} = 1$ if and only if $\delta_G(u_i, w_j) \le 3$. As an consequence we get

Theorem 5.1 *If all the distances in an undirected n vertex graph can be approximated with a one-sided additive error of at most one in $O(A(n))$ time, then Boolean matrix multiplication can also be performed in $O(A(n))$ time.*



$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Figure 6: Boolean matrix multiplication.

By adding a disjoint path of length $k-2$ ending at each u_i, we get that, for any *fixed* $k \geq 2$, distinguishing between distance k and $k+2$ in graphs with n vertices, i.e., deciding for each pair of vertices u, v whether $\delta(u,v) \leq k$, or $\delta(u,v) \geq k+2$ (if $\delta(u,v) = k+1$, then either decision is fine), is at least as hard as multiplying two Boolean matrices of size $n \times n$. Note, in contrast, that if $k \geq n^{2/3}$ then by Theorem 3.3, we can distinguish, with high probability, between distance k and $k+2$ in graphs with n vertices in $\tilde{O}(n^{7/3})$ time, i.e., faster than then the fastest known matrix multiplication algorithm.

Similarly, as any approximation algorithm that finds approximated distances of stretch strictly less than 2 can distinguish between distance 2 and distance 4, we get that getting approximate distances of stretch *less* than 2, between all pairs of vertices, is also as hard as Boolean matrix multiplication.

By turning the graph $G_{A,B}$ into a directed graph, where edges are directed to the right, we get that $c_{ij} = 1$ iff $\delta_G(u_i, w_j) < \infty$. Approximating distances in *directed* graphs, to within any *multiplicative* factor, not necessarily bounded, is therefore as hard as Boolean matrix multiplication. Note that such an approximation is equivalent to the computation of the *transitive closure* of the graph. It is proved in [AHU74] (see Theorems 5.6 and 5.7) that the computation of the transitive closure of a directed graph is equivalent to Boolean matrix multiplication. We end this section with another simple observation:

Theorem 5.2 *If two $n \times n$ Boolean matrices could be multiplied in $O(M(n))$ time, then for any fixed $\epsilon > 0$, all the distances in an unweighted directed graph on n vertices can be estimated with stretch $1+\epsilon$ in $\tilde{O}(M(n))$ time.*

Proof: Let $G = (V, E)$ be an unweighted directed graph on n vertices. Let A be the adjacency matrix of the graph with self loops added to all the vertices. Note that $\delta_G(u,v) \leq d$, if and only if $(A^d)_{u,v} = 1$. Let $r_i = \lfloor (1+\epsilon)^i \rfloor$, for $1 \leq i \leq k = \lceil \log_{1+\epsilon} n \rceil$. We compute A^{r_i}, for $1 \leq i \leq k$. We let $\hat{\delta}(u,v) = r_{i+1}$ if and only if $(A^{r_i})_{u,v} = 0$ but $(A^{r_{i+1}})_{u,v} = 1$. It is easy to verify that for every $u, v \in V$ we have $\delta(u,v) \leq \hat{\delta}(u,v) \leq (1+\epsilon) \cdot \delta(u,v)$, and that the running time of this algorithm is $\tilde{O}(M(n))$. □

6 Distance Emulators

Closely related to the algorithms of Sections 3 and 4 is the notion of emulators.

Definition 6.1 (Emulators) *Let $G = (V, E)$ be an unweighted undirected graph. A weighted graph $H = (V, F)$ is said to be a k-*emulator *of G if and only if for every $u, v \in V$ we have $\delta_G(u,v) \leq \delta_H(u,v) \leq \delta_G(u,v) + k$.*

There is one significant difference, however, between emulators and the auxiliary graphs used in the algorithms of Sections 3 and 4. There, we constructed for each vertex u an auxiliary graph $G_k(u)$ that supplied good approximations to the distances from u to all the vertices of the graph. Here we want a single graph that will supply good approximations of *all* distances. Constructing a sparse k-emulator is therefore harder than computing surplus k distances.

The definition of k-emulators is related to the definition of ***k*-spanners** (Awerbuch [Awe85], Peleg and Schäffer [PS89]). Let $G=(V,E)$ be a weighted undirected graph. A subgraph $G'=(V,E')$ of G is said to be a k-spanner of G if and only if for every $u,v\in V$ we have $\delta_{G'}(u,v)\le k\cdot\delta_G(u,v)$. As G' is a subgraph of G, we always have $\delta_G(u,v)\le\delta_{G'}(u,v)$. This definition differs from the definition of emulators in three respects. We require additive error, not multiplicative error. We do not insist on getting a subgraph of the original graph and we allow weighted edges. Althöfer *et al.* [ADD+93] also consider *Steiner spanners* in which vertices and edges may be added to the graph. Steiner spanners are more closely related to emulators. Liestman and Shermer [LS93] consider *additive spanners*. They are able, however, to obtain sparse additive spanners only for specific graphs such as pyramids, grids and hypercubes. Additive spanners, unlike emulators, must be subgraphs of the original graph. Emulators may be described as weighted additive Steiner spanners. The definition of k-emulators is also related to the definition of ***hop sets*** (Cohen [Coh94]).

Implicit in the work of Aingworth *et al.* [ACIM96] is an $\tilde{O}(n^{5/2})$ time algorithm for constructing 2-emulators with $\tilde{O}(n^{3/2})$ edges. We can get the following slightly stronger result.

Theorem 6.2 *Every unweighted undirected graph $G=(V,E)$ on n vertices can be 2-emulated by a* sub*graph $G'=(V,F)$ with $\tilde{O}(n^{3/2})$ edges. Such a subgraph can be constructed in $\tilde{O}(n^2)$ time.*

Proof: We start by proving the existence of such an emulator. Split the vertices of G into two classes: $V_1=\{v\in V\mid \deg(v)\ge n^{1/2}\}$ and $V_2=\{v\in V\mid \deg(v)<n^{1/2}\}$. Find a set D of size $\tilde{O}(n^{1/2})$ that dominates V_1 and a set E^* of at most n edges such that for every $u\in V_1$ there exists $v\in D$ such that $(u,v)\in E^*$. From every $v\in D$ perform a BFS and find its distances to all the vertices of the graph. A 2-emulator of G of size $\tilde{O}(n^{3/2})$ is then obtained by taking all edges that touch vertices of V_2, and weighted edges between any vertex of D and any vertex of V. Instead of adding these weighted edges, we can simply take a tree of shortest paths rooted at each vertex of D. The total number of edges is still $\tilde{O}(n^{3/2})$. It is easy to check that the resulting subgraph is a 2-emulator. The proof is similar to the proofs of Theorems 3.1 and 3.2.

The above construction can be carried out in $\tilde{O}(n^{5/2})$ time. The most time consuming task is running BFS from all the vertices of D. To reduce the running time to $\tilde{O}(n^2)$, we split the vertices into $O(\log n)$ classes, instead of just two. The resulting algorithm $\mathbf{emul}_2$ is given in Figure 7. Note that s_k, the last degree threshold in $\mathbf{emul}_2$, is about $n^{1/2}$.

The complexity analysis of $\mathbf{emul}_2$ is straightforward. Note that $|D_i|=\tilde{O}(n/s_i)$. The graph (V,E_i) on which we run BFS from each vertex of D_i has $O(ns_{i-1})$ edges. The total running time of all these runs is therefore $\tilde{O}(n^2\cdot s_{i-1}/s_i)=\tilde{O}(n^2)$. As $k=O(\log n)$. The total running time of the whole algorithm is also $\tilde{O}(n^2)$.

It is also easy to check that $|F|=\tilde{O}(n^{3/2})$. This follows from the fact that $|E_k|=O(ns_{k-1})=O(n^{3/2})$, and from the fact that each tree of shortest paths contains exactly $n-1$ edges and the total number of vertices from which we run BFS, and therefore the total number of shortest paths trees that we use, is $\tilde{O}(n^{1/2})$.

All that remains is to show that $G'=(V,F)$ is indeed a 2-emulator of G. Note that $G'=(V,F)$ is a subgraph of $G=(V,E)$. The fact that $\delta_G(u,v)\le\delta_{G'}(u,v)$ for every $u,v\in V$ follows from the fact that G' is a subgraph of G. We have to show that for every $u,v\in V$ we have $\delta_G(u,v)\le\delta_{G'}(u,v)\le\delta_G(u,v)+2$. We consider two different cases.

Case 1: There is a shortest path between u and v in G all whose edges are contained in E_k.

In this case, $\delta_{G'}(u,v)=\delta_G(u,v)$.

```
Algorithm emul_2:

input: An unweighted undirected graph G = (V, E).
output: A subgraph 2-emulator (V, F) of G.

Let k ← ⌈(1/2) log_2 n⌉
For i ← 0 to k let s_i ← n/2^i

For i ← 1 to k − 1 let V_i ← {v ∈ V | deg(v) ≥ s_i}
For i ← 1 to k let E_i ← {(u, v) ∈ E | deg(u) < s_{i−1} or deg(v) < s_{i−1}}

For i ← 1 to k − 1 let (D_i, E_i^*) ← dominate(G, s_i)

For i ← 1 to k − 1 do
For every u ∈ D_i run bfs( (V, E_i) , δ̂, u)

The edge set F is composed of E_k and the edges
of all the shortest paths trees found in all the BFS runs.
```

Figure 7: An $\tilde{O}(n^2)$ time algorithm for generating a subgraph 2-emulator

Case 2: Every shortest path between u and v in G contains edges that are not contained in E_k.

Consider a shortest path p between u and v in G. Let w be a vertex of highest degree on p. Let $1 \leq i < k$ be such that $w \in V_i \setminus V_{i-1}$ (where $V_0 = \phi$). Let $w' \in D_i$ be such that $(w, w') \in E$. As all the vertices on p do not belong to V_{i-1}, we get that all the edges on p, as well as the edge (w, w') belong to E_i. The shortest paths tree constructed by running BFS on (V, E_i) from w' contains therefore a path from w' to u of length at most $\delta_G(u, w) + 1$, and a path from w' to v of length at most $\delta_G(w, v) + 1$. It follows that this shortest paths tree, and therefore $G' = (V, F)$, contain a path from u to v of length at most $\delta_G(u, v) + 2$.

This completes the proof of the theorem. □

A subgraph 2-emulator is also an additive 2-spanner and a multiplicative 3-spanner. In Section 7 (Theorem 7.3) we show that weighted graphs also have 3-spanners of size $\tilde{O}(n^{3/2})$. We present there an algorithm, whose running time is $\tilde{O}(mn^{1/2})$, for finding such 3-spanners. As there are bipartite graphs with $\Omega(n^{3/2})$ edges that do not contain cycles of length four [Wen91], this result is tight, up to polylogarithmic factors. We can also show:

Theorem 6.3 *Every unweighted undirected graph $G = (V, E)$ on n vertices has a 4-emulator with $\tilde{O}(n^{4/3})$ edges. Such a graph can be constructed in $\tilde{O}(n^{7/3})$ time.*

Proof: It is not difficult to check that the graph $G_3 = (V, E_3 \cup E^* \cup (D_1 \times V) \cup (D_2 \times D_2))$, in the notations of algorithm $\overline{\mathbf{apasp}}_3$ is a 4-emulator of $G = (V, E)$. □

It is tempting to claim that k-emulators, for $k > 4$, can be similarly obtained by running $\overline{\mathbf{apasp}}_k$ with $k > 4$. Unfortunately, this is not true. The fact that all the edges that touch a vertex u are added to the graph on which distances are found from u seems to be crucial there. We cannot do the same with emulators as we are supposed to use the same graph for all sources.

```
Algorithm emul_6:

input: An unweighted undirected graph G = (V, E).
output: A 6-emulator (V, F) of G.

Let k ← ⌈(2/3) log_2 n⌉
For i ← 0 to k let s_i ← n/2^i

For i ← 1 to k − 1 let V_i ← {v ∈ V | deg(v) ≥ s_i}
For i ← 1 to k let E_i ← {(u, v) ∈ E | deg(u) < s_{i−1} or deg(v) < s_{i−1}}

For i ← 1 to k − 1 let (D_i, E_i^*) ← dominate(G, s_i)
E^* ← ∪_{1≤i<k} E_i^*

For every u, v ∈ V let δ̂(u, v) ← { 1   if (u, v) ∈ E,
                                   { +∞  otherwise.

For i ← 1 to k − 1 do
For every u ∈ D_i run bfs( (V, E_i ∪ E^*) , δ̂, u)

Let F ← E_k ∪ E^* ∪_{i=1}^{k−1} D_i × D_{k−1}
The weight of an edge (u, v) ∈ F is δ̂(u, v)
```

Figure 8: An $\tilde{O}(n^2)$ time algorithm for generating a 6-emulator

Let e_k be the infimum of all numbers for which each graph on n vertices has a k-emulator with $\tilde{O}(n^{1+e_k})$ edges. We have shown that $e_2 \le 1/2$ and $e_4 \le 1/3$. Does $e_k \to 0$ as $k \to \infty$? This remains an intriguing open problem.

We are not able to construct emulators with $o(n^{4/3})$ edges. We can, however, construct 6-emulators with $\tilde{O}(n^{4/3})$ edges in $\tilde{O}(n^2)$ time.

Theorem 6.4 *Let $G = (V, E)$ be an unweighted undirected graph of n vertices. A 6-emulator of G with $\tilde{O}(n^{4/3})$ edges can be constructed in $\tilde{O}(n^2)$ time.*

Proof: The required 6-emulator is generated by algorithm **emul**$_6$ given in Figure 8. Algorithm **emul**$_6$ is similar to algorithm **emul**$_2$. Note that s_k, the last degree threshold in **emul**$_6$, is about $n^{1/3}$.

The complexity analysis of **emul**$_6$ is almost identical to the complexity analysis of **emul**$_2$ and it is omitted. It is also easy to check that $|F| = \tilde{O}(n^{4/3})$. This follows from the fact that $|E_k| = O(ns_{k-1}) = O(n^{4/3})$, and that $|D_i| \cdot |D_{k-1}| = \tilde{O}((n/s_{k-1})^2) = \tilde{O}(n^{4/3})$, for every $1 \le i \le k-1$, and from the fact that $k = O(\log n)$.

All that remains is to show that $H = (V, F)$ is indeed a 6-emulator of G. We have to show that for every $u, v \in V$ we have $\delta_G(u,v) \le \delta_H(u,v) \le \delta_G(u,v) + 6$. The fact that $\delta_G(u,v) \le \delta_H(u,v)$ for every $u, v \in V$ is obvious.

Case 1: There is a shortest path between u and v in G all whose edges are contained in E_k.

In this case, $\delta_H(u,v) = \delta_G(u,v)$.

Case 2: Every shortest path between u and v in G contains edges that are not contained in E_k.

Consider a shortest path p between u and v in G. The path must pass through at least two vertices from V_{k-1}. Let w_1 and w_3 be the first and last such vertices on the path (the vertex w_1 may be u and the vertex w_3 may be v). Let w_2 be a vertex with the maximum degree on the path (the vertex w_2 may be one of w_1 and w_3). Let $1 \le i < k$ be such that $w_2 \in V_i \setminus V_{i-1}$. Let $w_1', w_3' \in D_{k-1}$ be neighbors of w_1 and w_3 such that $(w_1, w_1') \in E^*$ and $(w_3, w_3') \in E^*$. Let $w_2' \in D_i$ be a neighbor of w_2 such that $(w_2, w_2') \in E^*$.

As w_1 and w_3 are the first and last vertices from V_{k-1} on the path, and as $(w_1, w_1'), (w_3, w_3') \in E^*$, we get that $\delta_H(u, w_1') \le \delta_G(u, w_1) + 1$ and $\delta_H(w_3', v) \le \delta_G(w_3, v) + 1$. As w_2 is a vertex with maximum degree on the shortest path from u to v, the shortest path p and the edges $(w_1, w_1'), (w_2, w_2'), (w_3, w_3')$ are contained in the graph $G_i = (V, E_i \cup E^*)$. We get, therefore, that $\delta_G(w_1, w_2) = \delta_{G_i}(w_1, w_2)$ and $\delta_G(w_2, w_3) = \delta_{G_i}(w_2, w_3)$ and thus $\delta_{G_i}(w_1', w_2') \le \delta_G(w_1, w_2) + 2$ and $\delta_{G_i}(w_2', w_3') \le \delta_G(w_2, w_3) + 2$. For every $x \in D_i$ and $y \in D_k$, we have added to F an edge (x, y) whose weight is at most $\delta_{G_i}(x, y)$. We get therefore that $\delta_H(w_1', w_2') \le \delta_G(w_1, w_2) + 2$ and $\delta_H(w_2', w_3') \le \delta_G(w_2, w_3) + 2$. Combining these bounds we get

$$\delta_H(u,v) \;\le\; \delta_H(u, w_1') + \delta_H(w_1', w_2') + \delta_H(w_2', w_3') + \delta_H(w_3', v)$$

$$\le\; (\delta_G(u, w_1) + 1) + (\delta_G(w_1, w_2) + 2) + (\delta_G(w_2, w_3) + 2) + (\delta_G(w_3, v) + 1) \;=\; \delta_G(u, v) + 6\ .$$

This completes the proof of the theorem. □

It is easy to see that k-emulators are Steiner $(k+1)$-spanners. It follows easily from the arguments of Althöfer *et al.* [ADD+93] and the constructions of Wenger [Wen91] that there are unweighted undirected graphs on n vertices for which every Steiner 3-spanner, and therefore any 2-emulator, must have $\tilde{\Omega}(n^{3/2})$ edges, and there are graphs for which every Steiner 5-spanner, and therefore any 4-emulator, must have $\tilde{\Omega}(n^{4/3})$ edges (where $\tilde{\Omega}(f) = \Omega(f\,/\,\mathrm{polylog}\, n)$).

7 Stretched Paths and Distances

In this section we describe algorithms for finding stretched paths in *weighted* graphs. We use the following result which is part of the folklore.

Lemma 7.1 (Truncated Dijkstra) *Let $G = (V, E)$ be a weighted graph on n vertices. Suppose that the adjacency lists of the vertices of G are sorted according to weight. Let $v \in V$ be a vertex of G and let $1 \le s \le n$. Shortest paths from v to s vertices closest to v can be found in $\tilde{O}(s^2)$ time.*

The set of s vertices returned by the truncated Dijkstra algorithm running from v is not uniquely defined, as there may be many vertices at the same distance from v. All that we require is that if S is the set of vertices returned by the algorithm then for every $u \in S$ and $w \in V \setminus S$ we have $\delta(v, u) \le \delta(v, w)$.

Theorem 7.2 *Let $G = (V, E)$ be a weighted undirected graph with n vertices and m edges. We can preprocess the graph in $\tilde{O}(m^{2/3}n)$ time so that given any two vertices $u, v \in V$, we can in $O(1)$ time output an estimated distance $\hat{\delta}(u, v)$ satisfying $\delta(u, v) \le \hat{\delta}(u, v) \le 3\cdot\delta(u, v)$.*

Proof: Let s be a parameter to be chosen later. We run the truncated Dijkstra algorithm from every vertex $v \in V$ and find a set $N(v)$ of s vertices closest to v. The time required for finding these sets is $\tilde{O}(ns^2)$. Next, we find a set D of size $d = \tilde{O}(n/s)$ so that for every $v \in V$, there is $u \in D$ such that $u \in N(v)$. Such a set can be found in $O(ns)$ time. For every vertex $v \in V$, we keep a pointer to a vertex $u = P(v)$ such that $u \in D \cap N(v)$. We now run the full Dijkstra algorithm from all the vertices of D. The time required is $\tilde{O}(nm/s)$. We keep an $d \times n$ matrix with the distances from the vertices of D to all the other vertices of the graph. The time used so far is $\tilde{O}(ns^2 + nm/s)$. This is minimized by taking $s = m^{1/3}$. The total time is then $\tilde{O}(m^{2/3}n)$.

Given a pair of vertices u and v, we first check whether $v \in N(u)$. If so, we output the exact distance $\delta(u,v)$ computed during the truncated Dijkstra from u. Otherwise, we let $w = P(u) \in D \cap N(u)$ and we output the estimated distance $\hat{\delta}(u,v) = \delta(u,w) + \delta(w,v)$. The distance $\delta(u,w)$ was found during the truncated Dijkstra from u. The distance $\delta(w,v)$ was found during the full Dijkstra from w.

Clearly $\delta(u,v) \le \hat{\delta}(u,v)$. If $v \in N(u)$, then $\hat{\delta}(u,v) = \delta(u,v)$. If $v \notin N(u)$, then $\delta(u,w) \le \delta(u,v)$ and the estimate $\hat{\delta}(u,v)$ satisfies

$$\hat{\delta}(u,v) \;=\; \delta(u,w) + \delta(w,v) \;\le\; \delta(u,w) + (\delta(w,u) + \delta(u,v)) \;\le\; 2\delta(u,w) + \delta(u,v) \;\le\; 3\delta(u,v)\;,$$

as required. □

Using essentially the same algorithm we can get:

Theorem 7.3 *Every weighted undirected graph $G = (V,E)$ on n vertices has a 3-spanner with $\tilde{O}(n^{3/2})$ edges. Such a 3-spanner can be constructed in $\tilde{O}(mn^{1/2})$ time.*

Proof: Let s be a parameter to be chosen later. Run the truncated Dijkstra algorithm from every vertex and find for every vertex $v \in V$ a set $N(v)$ of s vertices closest to v. Find a set D of size $\tilde{O}(n/s)$ such that for every $v \in V$, there is $u \in D \cap N(v)$. We then run a full Dijkstra from every vertex of D. The 3-spanner will be composed of the shortest paths trees found in all the truncated and full runs of Dijkstra's algorithm. The total number of edges will therefore be $\tilde{O}(ns + n^2/s)$. We choose $s = n^{1/2}$. The number of edges in the 3-spanner is then $\tilde{O}(n^{3/2})$ and the total running time is $\tilde{O}(ns^2 + nm/s) = \tilde{O}(n^2 + mn^{1/2})$. Note however, that if $mn^{1/2} \le n^2$ then $m \le n^{3/2}$ and the original graph is the required 3-spanner. □

Cohen and Zwick [CZ97] extend the techniques presented here and obtain, among other things, an $\tilde{O}(n^2)$ time algorithm for finding stretch 3 distances, and an $\tilde{O}(n^{3/2}m^{1/2})$ time algorithm for finding stretch 2 distances in weighted undirected graphs with n vertices and m edges.

8 Concluding remarks and open problems

We have shown that surplus 2 estimates of all distances in an unweighted undirected graph on n vertices can be computed in $\tilde{O}(n^{7/3})$ time, i.e., faster than the fastest known matrix multiplication algorithm.

Many open problems still remain. We end by mentioning some of them:

1. Is it possible to find surplus 2 estimated distances between all pairs of vertices in a graph on n vertices in $O(n^{7/3-\epsilon})$ time, for some $\epsilon > 0$?

2. Is it possible to find surplus k estimated distances between all pairs of vertices in a graph on n vertices, for some fixed constant $k \ge 2$, in $\tilde{O}(n^2)$ time?

3. Do there exist fixed constants $k \ge 2$ and $\epsilon > 0$ such that every graph on n vertices has a k-emulator with $O(n^{4/3-\epsilon})$ edges?

4. Is it possible to find the *exact* distances between all pairs of vertices in an unweighted *directed* graph on n vertices in $\tilde{O}(M(n))$ time, where $M(n)$ is the time needed to multiply two $n \times n$ matrices?

Acknowledgment

We would like to thank Howard Karloff for many helpful discussions and for comments on an earlier version of this extended abstract, and Edith Cohen and David Peleg for some clarifications regarding their papers.

References

[ABCP93] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *Proceedings of the 34rd Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, pages 638–647, 1993.

[ACIM96] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). To appear in SIAM Journal on Computing. A preliminary version appeared in the Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, Georgia, pages 547–553., 1996.

[ADD+93] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[AGM97] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.

[AGMN92] N. Alon, Z. Galil, O. Margalit, and M. Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania*, pages 417–426, 1992.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[AS92] N. Alon and J.H. Spencer. *The probabilistic method.* Wiley, 1992.

[Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32:804–823, 1985.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[BKM95] J. Basch, S. Khanna, and R. Motwani. On diameter verification and boolean matrix multiplication. Technical Report STAN-CS-95-1544, Department of Computer Science, Stanford University, 1995.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms.* The MIT Press, 1990.

[Coh93] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t (extended abstract). In *Proceedings of the 34rd Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, pages 648–658, 1993.

[Coh94] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montréal, Canada*, pages 16–26, 1994.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[CZ97] E. Cohen and U. Zwick. All-pairs small-stretch paths. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana*, pages 93–102, 1997.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[FM95] T. Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51:261–272, 1995.

[Fre76] M.L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:49–60, 1976.

[FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[GM93] Z. Galil and O. Margalit. Witnesses for boolean matrix multiplication. *Journal of Complexity*, 9:201–221, 1993.

[GM97] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54:243–254, 1997.

[Joh77] D.B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.

[KKP93] D.R. Karger, D. Koller, and S.J. Phillips. Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22:1199–1217, 1993.

[LS93] A.L. Liestman and T.C. Shermer. Additive graph spanners. *Networks*, 23:343–363, 1993.

[McG95] C.C. McGeoch. All-pairs shortest paths and the essential subgraph. *Algorithmica*, 13:426–461, 1995.

[PS89] D. Peleg and A.A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

[Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.

[Tak92] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43:195–199, 1992.

[UY91] J.D. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20:100–125, 1991.

[Wen91] R. Wenger. Extremal graphs with no C^4's, C^6's and C^{10}'s. *Journal of Combinatorial Theory, Series B*, 52:113–116, 1991.