Approximate Distance Oracles *

Mikkel Thorup † *Uri Zwick* ‡

Ferbuary 9, 2001
(Revised: March 17, 2004)

Abstract

Let $G = (V, E)$ be an undirected *weighted* graph with $|V| = n$ and $|E| = m$. Let $k \geq 1$ be an integer. We show that $G = (V, E)$ can be preprocessed in $O(kmn^{1/k})$ expected time, constructing a data structure of size $O(kn^{1+1/k})$, such that any subsequent *distance query* can be answered, approximately, in $O(k)$ time. The approximate distance returned is of stretch at most $2k - 1$, i.e., the quotient obtained by dividing the estimated distance by the actual distance lies between 1 and $2k-1$. A 1963 girth conjecture of Erdős, implies that $\Omega(n^{1+1/k})$ space is needed in the worst case for any real stretch strictly smaller than $2k+1$. The space requirement of our algorithm is, therefore, essentially optimal. The most impressive feature of our data structure is its *constant* query time, hence the name "oracle". Previously, data structures that used only $O(n^{1+1/k})$ space had a query time of $\Omega(n^{1/k})$.

Our algorithms are extremely simple and easy to implement efficiently. They also provide faster constructions of sparse *spanners* of weighted graphs, and improved *tree covers* and *distance labelings* of weighted or unweighted graphs.

1 Introduction

Consider the following interesting problem which is, perhaps, the most natural formulation of the classical all-pairs shortest paths problem (APSP). We are given a description of a large network, such as the Internet, or a large road network, such as the US road network[1], with n nodes and m connections. Each connection has a length, or weight, associated with it. Usually $m \ll n^2$. We are to preprocess the network, so that subsequent *distance queries* or *shortest path queries* could be answered quickly, on-line.

This formulation seems to capture more accurately the real nature of the all-pairs shortest paths problem, as in most applications we are not really interested in *all* distances, we just want the ability to retrieve them quickly, if needed. For example, there are probably many pairs of addresses in the US whose distance is of interest to no one. This is precisely the case with "sublinear" algorithms for static metric space problems. The input to such a problem is usually a shortest paths metric. A "sublinear" algorithm attempts to solve such problem while querying only some of the distances. For more details, see Indyk [Ind99].

*A preliminary version of this paper is to appear in the Proceedings of the 33rd ACM Symposium on the Theory of Computing (STOC'01), Crete, Greece, 2001

†AT&T Labs - Research, 180 Park Avenue, Florham Park, NJ 07932, USA. E-mail address: `mthorup@research.att.com`.

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail address: `zwick@post.tau.ac.il`. Work supported in part by THE ISRAEL SCIENCE FOUNDATION founded by The Israel Academy of Sciences and Humanities.

[1]The US road network is a planar network. To get a more interesting non-planar network, assume that the weights attach to the edges represent travel time, and add flight connections between airports.

Using an APSP algorithm, we can do the preprocessing in $\tilde{O}(mn)$ time, and produce a data structure of size $O(n^2)$, an $n \times n$ matrix holding the distances, and perhaps a succinct representation of shortest paths between all pairs of vertices of the graph. Any distance query can then be answered in $O(1)$ time.

There are, however, several serious objections to this solution. First, a preprocessing time of $\tilde{O}(mn)$ may be too long. Second, even if we are willing to wait that long, the $n \times n$ matrix produced may be too large to store efficiently (typically $m \ll n^2$, and then this table is much larger than the network itself).

Here, we explore alternative solutions to this problem. We show that better solutions exist, if the network is *undirected*, and if we are willing to settle for *approximate* distances, instead of exact ones. The approximate distances produced by our algorithms are of a finite *stretch*. An estimate $\hat{\delta}(u, v)$ to the distance $\delta(u, v)$ from u to v is said to be of stretch t if and only if $\delta(u, v) \leq \hat{\delta}(u, v) \leq t \cdot \delta(u, v)$. Stretched distances may be acceptable under some scenarios, while unacceptable in others. Many recent algorithms dealing with finite metric spaces produce only approximate answers, even if exact distances are used. In particular, this is the case with the above mentioned sublinear metric space algorithms of Indyk [Ind99]. Adapting these algorithms to exploit our approximate distance oracles is therefore a straightforward task.

As stated in the abstract, we describe, for any integer $k \geq 1$, a preprocessing algorithm that runs in $O(kmn^{1/k})$ time, producing a data structure of size $O(kn^{1+1/k})$. Note that the preprocessing time is almost linear in the size of the network, if k is a large constant, while the size of the data structure produced is almost linear in the number of nodes. In particular, for dense enough graphs, the data structure produced is much more compact than the network itself. Subsequent queries can then be answered, approximately, in $O(k)$ time. i.e., constant time. The stretch of the approximations returned is at most $2k-1$. This result is summarized in the following theorem.

Theorem 1.1 *Let $G = (V, E)$ be a* weighted *undirected graph with non-negative edge weights with $|V| = n$, $|E| = m$. Let $k \geq 1$ be an integer. Then, the graph G can be preprocessed in $O(kmn^{1/k})$ expected time, producing a data structure of $O(kn^{1+1/k})$ size, such that subsequent distance queries can be answered, approximately, in $O(k)$ time. The stretch of the produced estimates is at most $2k-1$. Paths no longer than the estimates returned can be produced in constant time per edge.*

For $k = 1$, we simply get the APSP solution. When $k = 2$, we get a preprocessing time of $O(mn^{1/2})$, space $O(n^{3/2})$, query time $O(1)$, and stretch at most 3. When $k = \lfloor \log n \rfloor$, we get a preprocessing time of $O(m \log n)$, space $O(n \log n)$, query time $O(\log n)$, and stretch $O(\log n)$. Higher values of k do not improve the space or preprocessing time.

The most interesting feature of our algorithms, we believe, is the fact that for every fixed k we get a *constant* query time, hence the name *distance oracles*.

The space requirements of our oracles are essentially optimal. A 1963 girth conjecture of Erdős, and others, implies that $\Omega(n^{1+1/k})$ *bits* of storage are needed, in the worst case, by any oracle, however slow, that gives estimated distances with stretch strictly less than $2k+1$. This girth conjecture is known to hold for $k = 1, 2, 3, 5$. Thus, in particular, any oracle giving stretch 2.99 answers must use, on some graphs, at least $\Omega(n^2)$ bits of storage, and any oracle giving stretch 4.99 answers must use, on some graphs, at least $\Omega(n^{3/2})$ bits of storage, almost the same amount of storage used by our stretch 3 oracle.

The oracle model of the shortest paths problem was considered before, at least implicitly, by Awerbuch *et al.* [ABCP99], Cohen [Coh99], Dor *et al.* [DHZ00], and by Matoušek [Mat96]. (See discussion in the next section.) Our results significantly improve, however, the previously available results. Most strikingly, using slightly less space, we reduce the query time from $\tilde{O}(kn^{1/k})$ to $O(k)$.

Theorem 1.1 is proved in Section 4. Before that, in Section 3, we present a simplified version of our algorithm for the the special case where the input is the *complete distance matrix* of a finite metric space. This version of the algorithm is faster ($O(n^2)$ time) and particularly suited for *external memory* implementation.

As a byproduct of our oracle construction for graphs, we also get faster algorithms for constructing sparse *spanners* and compact *tree covers* of *weighted* graphs (see Section 4.4), and near-optimal distance labelings of graphs (see Section 3.5).

As mentioned in the abstract, all our algorithms are extremely simple and easy to implement efficiently.

The rest of the paper is organized as follows. In the next section, we compare our results with previously available results. In Section 3 we construct approximate distance oracles for finite metric spaces. The input in this setting is an $n \times n$ matrix giving the distance between any two points in the space. In Section 4 we adapt this construction to work on the shortest paths metric of a given input graph. The input this time is the graph, and not an explicit representation of all the distances in it. Breaking the description of our distance oracles in this way allows us to separate the metric aspects of our constructions from the algorithmic graph techniques needed for efficient implementation. In Section 5 we describe almost matching lower bounds on the space requirements of approximate distance oracles. We also show that essentially no non-trivial distance oracles are possible for *directed* graphs. We end, in Section 6, with some concluding remarks and open problems.

2 Comparison with previous results

A summary of previously published algorithms for computing exact or approximate distances in general *weighted* undirected graphs, cast in our framework, is given in Table 1.

In more detail, the fastest solution for APSP for directed and undirected weighted graphs with non-negative weights from an arbitrary (comparison based) domain is to run a single-source shortest paths (SSSP) algorithm from each node. This takes $O(m + n \log n)$ time using the classical algorithm of Dijkstra [Dij59], implemented using Fibonacci heaps [FT87] (see also Cormen *et al.* [**?**, Chapter 20]). In this paper, we are only interested in undirected graphs, and then an improved running time of $O(m)$ can be obtained when the weights are integer [Tho99] (or floating point [Tho00b]). Consequently, the time bound for APSP is $O(mn)$. The $O(m)$ time bound for SSSP has been incorporated in the other time bounds below, so an $\tilde{O}(\cdot)$ time bound indicates the presence of logarithmic factors not stemming from Dijkstra's algorithm.

For completeness, we note that improved time bounds may be obtained if not all edges are part of shortest paths [KKP93, McG95], or if the graph is dense and all weights are small integers [Zwi02, SZ99].

Zwick [Zwi02] has shown that if a stretch of $1 + \epsilon$, for some fixed $\epsilon > 0$, is allowed, then APSP can be solved in $\tilde{O}(n^\omega)$ time, where $\omega < 2.376$ is the exponent of matrix multiplication. For stretches 2, 7/3, and 3, Cohen and Zwick [CZ01] have shown that APSP can be solved in time $\tilde{O}(m^{1/2}n^{3/2})$, $\tilde{O}(n^{7/3})$, and $\tilde{O}(n^2)$, respectively.

The above $\tilde{O}(n^2)$ time bound is clearly near-optimal, if we insist on producing a complete table of distances. It is interesting to note, however, that stretch 3 is also the smallest stretch for which one can hope to produce distance oracles that use $o(n^2)$ space for all graphs. Indeed, Dor *et al.* [DHZ00] describe a stretch 3 oracle that uses only $\tilde{O}(m^{1/3}n + n^2/m^{1/3})$ space, which is always $\tilde{O}(n^{5/3})$, has a preprocessing time of $\tilde{O}(m^{2/3}n)$, and $O(1)$ query time. The preprocessing time of this algorithm is $o(n^2)$ if $m = o(n^{3/2})$. The ideas of Dor *et al.* [DHZ00] do not extend, however, to larger stretches.

Our new stretch 3 oracle (Theorem 1.1 with $k = 2$) has a preprocessing time of $O(mn^{1/2})$, *optimal* space $O(n^{3/2})$, and $O(1)$ query time. The new preprocessing time is faster than all the other preprocessing times when $m = O(n^{3/2})$, in which case it achieves a preprocessing time of $O(n^2)$.

Stretch	Query time	Space	Preproc. time	Reference	Space lower-bound
1	$O(1)$	$O(n^2)$	$O(mn)$	[Tho99]	$\Omega(m)$ bits
	$O(m)$	$O(m)$	0	[Tho99]	
$1+\epsilon$	$O(1)$	$O(n^2)$	$\tilde{O}(n^\omega)$	[Zwi02]	
2	$O(1)$	$O(n^2)$	$\tilde{O}(m^{1/2}n^{3/2})$	[CZ01]	
7/3	$O(1)$	$O(n^2)$	$\tilde{O}(n^{7/3})$	[CZ01]	
3	$O(1)$	$O(n^2)$	$\tilde{O}(n^2)$	[CZ01]	$\Omega(\min\{m, n^{3/2}\})$ bits [Rei58]
	$O(1)$	$\tilde{O}(m^{1/3}n + n^2/m^{1/3})$	$\tilde{O}(m^{2/3}n)$	[DHZ00]	
	$\boldsymbol{O(1)}$	$\boldsymbol{O(n^{3/2})}$	$\boldsymbol{O(mn^{1/2})}$	**This paper**	
$2k-1$	$O(n^{1+1/k})$	$O(n^{1+1/k})$	$O(mn^{1+1/k})$	[ADD+93]	$\Omega(\min\{m, n^{1+1/k}\})$ bits Conjecture [Erd64]
	$\tilde{O}(kn^{1/k})$	$\tilde{O}(kn^{1+1/k})$	$\tilde{O}(mn^{1/k})$	[Mat96]	
	$\boldsymbol{O(k)}$	$\boldsymbol{O(kn^{1+1/k})}$	$\boldsymbol{O(kmn^{1/k})}$	**This paper**	
$2k+\epsilon$	$\tilde{O}(kn^{1/k})$	$\tilde{O}(kn^{1+1/k})$	$\tilde{O}(kmn^{1/k})$	[Coh99]	
$64k$	$\tilde{O}(kn^{1/k})$	$\tilde{O}(kn^{1+1/k})$	$\tilde{O}(kmn^{1/k})$	[ABCP99]	$\Omega(\min\{m, n^{1+1/32k}\})$ bits Conjecture [Erd64]

Table 1: All available exact and approximate distance oracles for weighted undirected graphs

For general stretch, Awerbuch *et al.* [ABCP99] gave, for every integer $k \geq 1$, a stretch $64k$ oracle with space $\tilde{O}(kn^{1+1/k})$, expected preprocessing time $\tilde{O}(mn^{1/k})$, and $\tilde{O}(kn^{1/k})$ query time. Cohen [Coh99] significantly improved this result, reducing the stretch to $2k+\epsilon$ while leaving the other parameters unchanged. The stretch was further improved to $2k-1$, using entirely different techniques, by Matoušek [Mat96].

The main result of this paper is that we improve the query time time from $\tilde{O}(kn^{1/k})$ to $O(k)$. We also get slightly improved space and preprocessing time. More precisely, we present a stretch $2k-1$ oracle with space $O(kn^{1+1/k})$, expected preprocessing time $O(mn^{1/k})$, and $O(k)$ query time.

Most work on distance oracles is closely related to spanners. A *t-spanner* of a weighted undirected graph G is a subgraph H of G such that the distances in H are stretch t estimates of the distances in G (see Peleg and Schäffer [PS89]). Clearly, a stretch t oracle, like ours, capable of producing paths witnessing the estimated distances, must explicitly or implicitly contain a t-spanner. Hence, t-spanners provide a clean mathematical view of compact distance oracles. Indeed, all of the above mentioned results providing $o(n^2)$ space bounds [ABCP99],[Coh99],[DHZ00] can be viewed as producing spanners.

The sizes of spanners are closely related to the *girth* of a graph, which is the size of its smallest simple cycle. Clearly, the girth of a graph is at least $t+2$ if and only if no proper subgraph of it is a t-spanner. A classical result in extremal graph theory (see discussion and references in Section 5) states that an n-vertex graph with at least $n^{1+1/k}$ edges is of girth at most $2k$. As pointed out by Althöfer *et al.* [ADD+93], this implies that every *weighted* undirected graph on n vertices has a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges. Such a spanner can be constructed using an algorithm similar to Kruskal's algorithm (see [Kru56] or Cormen *et al.* [CLRS01, Chapter 23]) for constructing minimum spanning trees: Building the spanner from scratch, consider the edges of the graph in a non-decreasing order of weight, adding each edge to the spanner if its endpoints are not already connected, in the spanner, by a path using at most $2k-1$ edges. At any stage, the spanner is a $(2k-1)$-spanner of the edges already considered, and its unweighted girth is at least $2k+1$, so it has only $O(n^{1+1/k})$ edges. The fastest implementation of this algorithm that we are aware of runs in $O(mn^{1+1/k})$ time. Our preprocessing algorithm constructs a similar spanner much faster. The expected construction time is now $O(kmn^{1/k})$, instead of $O(mn^{1+1/k})$. Very recently, Baswana and Sen [BS03] found a randomized linear time algorithm for constructing such spanners, but their construction does not support fast distance queries.

It is conjectured by many (e.g., Erdős [Erd64], equation 7 on p. 33, Bondy and Simonovits [BS74], remark 1 on p. 98 and Bollobás [Bol78], item 13 on p. 164), that there are n-vertex graphs with $\Omega(n^{1+1/k})$ edges that are of girth $2k+2$. This conjecture is proved for $k = 1, 2, 3, 5$ (see Section 5). Since these graphs have no proper t-spanners, for $t < 2k+1$, the conjecture would imply that the above mentioned upper bounds are best possible. The conjecture also implies that $\Omega(n^{1+1/k})$ bits are needed, in the worst case, by any oracle giving estimates of stretch smaller than $2k+1$, even if it *not* required to construct appropriate paths. This lower-bound is proved by Matoušek [Mat96], but for completeness, we repeat the simple argument in Section 5.

Interestingly, the previous best distance oracle of Matoušek [Mat96] (see also Chapter 15 of [Mat02]) does not provide a spanner. Generalizing a technique of Bourgain [Bou85], Matoušek provides a randomized low-distortion embedding of a metric into L_∞^d. Using $d = O(kn^{1/k}\log n)$ dimensions, w.h.p., his distances become shorter by at most a factor $2k-1$ (so multiplying them by $2k-1$, he gets stretch $2k-1$). When applied to a graph, he only needs $O(m)$ time to compute a given coordinate for all vertices, and his distance estimate is just the maximal difference over all coordinates. Thus with space $O(nd) = \tilde{O}(kn^{1+1/k})$, preprocessing time $O(md) = \tilde{O}(kmn^{1/k})$, and query time $O(d) = \tilde{O}(kn^{1/k})$, he gets stretch $2k-1$ with high probability. For comparison, we note that embeddings into Euclidian space or tree metrics (see Bourgain [Bou85], Linial *at al.* [LLR95] and Bartal [Bar99], Fakcharoenphol *at al.* [FRT03]), have stretch at least $\Omega(\log n)$.

Some distance oracles were constructed for special classes of graphs. Exact linear-space distance oracles for graphs of small treewidth were obtained by Chaudhuri and Zaroliagis [CZ00]. For planar graphs, or graphs with bounded genus, stretch $(1+\varepsilon)$ distance oracles are constructed in near-linear time and space by Thorup [Tho01] and Klein [Kle02], providing distance estimates in constant time. For Euclidean graphs, the same bounds were obtained by Gudmundsson *et al.* [GLNS02].

Finally, we mention that there has been some work on approximating distances, in *unweighted* undirected graphs, with additive rather than multiplicative errors (see Aingworth *et al.* [ACIM99] and Dor *et al.* [DHZ00]), and on approximating distances with multiplicative *and* additive errors (see Elkin and Peleg [EP01] and Elkin [Elk01]).

Techniques Our construction technique is most closely related to the techniques employed by Awerbuch *et al.* [ABCP99] and Cohen [Coh99]. A common feature of these previously used techniques is the construction of a family of *balls* with the property that each vertex is contained in at most of $\tilde{O}(kn^{1/k})$ balls. The returned distance between two vertices is then the smallest diameter of a ball containing them both. To find this ball, they inspect each of the $\tilde{O}(kn^{1/k})$ balls containing the first vertex, and check, in constant time per ball, whether it also contains the second. Though conceptually simple, the use of balls leads to several technical complications. One of them, for example, is an added logarithmic factor paid for the construction of balls with exponentially increasing diameters. The main drawback of this approach, however, is the lack of a quick way of finding the smallest diameter ball containing two given vertices.

In our construction, we relax the rigid notion of balls with limited diameter, and use instead collections of induced trees that form a *tree cover* of the graph. Each vertex is contained in only in a small number of trees, and for any pair of vertices, there is a tree in the cover containing a small-stretch path between them. Furthermore, we can identify the appropriate tree in constant time.

3 Approximate distance oracles for metric spaces

We begin by presenting approximate distance oracles for general metric spaces.

Theorem 3.1 *Let (V, δ) be a finite metric space represented as an $n \times n$ distance matrix. Let $k \geq 1$ be an integer. The metric space (V, δ) can be preprocessed in $O(n^2)$ expected time, producing a data structure of $O(kn^{1+1/k})$ size, such that subsequent distance queries can be answered, approximately, in $O(k)$ time. The stretch of the produced estimates is at most $2k-1$.*

Our preprocessing and query answering algorithms are given, respectively, in Figure 1 and Figure 2. Both are extremely simple and easy to implement. The algorithms and their implementation details are discussed in more detail in the next two subsections. The following two subsections are then devoted to the analysis of the algorithms, showing that they satisfy the requirements of Theorem 3.1. In Section 3.5 we show that our approximate distance oracles also produce, as a byproduct, almost optimal *distance labels*. In Section 3.6 we show that our randomized preprocessing algorithm may be derandomized with only a small loss of efficiency. Finally, in the last two subsections of this section we consider more practical issues. In Section 3.7 we mention some hash functions that may be used by our algorithms. In Section 3.8 we note that our preprocessing algorithm is an efficient *external memory* algorithm.

3.1 Preprocessing a finite metric space

A description of the preprocessing algorithm $\mathbf{prepro}_k(V, \delta)$ is given in Figure 1. The missing implementation details are explained below. The algorithm receives an $n \times n$ matrix representing a finite metric $\delta(u, v)$ on a set V containing n points referred to as *vertices*. (In the next section, we consider the case in which the input to the preprocessing algorithm is not an explicit $n \times n$ matrix that describes the metric on V, but rather a weighted undirected graph $G = (V, E)$ that induces a shortest paths metric on V.)

The preprocessing algorithm starts by constructing a non-increasing sequence of sets $A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1}$ by a process of repeated sampling. The sequence begins with $A_0 = V$. Each set A_i, where $1 \leq i < k$, is then obtained by taking, roughly, an $n^{-1/k}$ fraction of the elements of A_{i-1}. More precisely, each element of A_{i-1} is placed in A_i, independently, with probability $n^{-1/k}$. Finally, $A_k = \emptyset$. The expected size of A_i, for $0 \leq i \leq k$, is clearly $n^{1-i/k}$. Sometimes we will refer to the vertices of A_i as *i-centers*.

In the following, for simplicity, we assume that $A_{k-1} \neq \emptyset$. This is the case with extremely high probability, and if not, we can just rerun the algorithm. Now, for each vertex v and index $i = 0, ..., k-1$, the algorithm computes $\delta(A_i, v)$, the smallest distance from an i-center to v. It also lets $p_i(v) \in A_i$ be nearest possible to v, that is, $\delta(p_i(v), v) = \delta(A_i, v)$. Note that $A_0 = V$, so $\delta(A_0, v) = 0$, and $p_0(v) = v$, for every $v \in V$. Next, the algorithm sets $\delta(A_k, v) \leftarrow \delta(\emptyset, v) = \infty$, with $p_k(v)$ undefined.

Finally, for each vertex $v \in V$, the algorithm computes a *bunch* $B(v) \subseteq V$ as follows. A vertex $w \in A_i - A_{i+1}$ is put in the bunch $B(v)$ if and only if $\delta(w, v) < \delta(A_{i+1}, v)$, i.e., if w is *strictly* closer to v than all the vertices of A_{i+1}. Note that as $\delta(A_k, v) = \infty$, we get that $A_{k-1} \subseteq B(v)$, for every $v \in V$. We show in Section 3.3, that the expected size of the bunch $B(v)$, for every $v \in V$, is at most $kn^{1/k}$.

```
algorithm prepro_k(V, δ)

A_0 ← V ; A_k ← ∅
for i ← 1 to k − 1
    let A_i contain each element of A_{i−1},
    independently, with probability n^{−1/k}.
for every v ∈ V
    for i ← 0 to k − 1,
      let δ(A_i, v) ← min{ δ(w, v) | w ∈ A_i },
      and p_i(v) ∈ A_i be nearest to v, i.e. δ(p_i(v), v) = δ(A_i, v).
    δ(A_k, v) ← ∞
    let B(v) ← ∪_{i=0}^{k−1} { w ∈ A_i − A_{i+1} | δ(w, v) < δ(A_{i+1}, v) }.
```

Figure 1: Preprocessing a finite metric space

```
algorithm dist_k(u, v)

w ← u ; i ← 0
while w ∉ B(v)
    i ← i + 1
    (u, v) ← (v, u)
    w ← p_i(u)
return δ(w, u) + δ(w, v)
```

Figure 2: Answering a distance query

A schematic description of the construction of the bunch $B(v)$, for some $v \in V$, is given in Figure 3. It is assumed there that $k = 3$. The black vertices are the vertices of A_2, the grey ones are the vertices of $A_1 - A_2$, while the white ones are those of $A_0 - A_1$. The bunch $B(v)$ is composed of the the vertices pointed to by an arrow from v. Also shown in the figure are $p_1(v)$, the vertex of A_1 closest to v, and $p_2(v)$, the vertex of A_2 closest to v.

This almost completes the operation of the preprocessing algorithm. For each bunch $B(v)$, the preprocessing algorithm constructs a *hash table* (see Carter and Wegman [CW79], or [CLRS01, Chapter 11]) of size $O(|B(v)|)$ that holds, for every $w \in B(v)$, the distance $\delta(w, v)$. Using this hash table it can be checked for every $w \in V$, in expected $O(1)$ time, whether $w \in B(v)$ and if so what is $\delta(w, v)$. Alternatively, it constructs a *2-level hash table* (see Fredman, Komlos and Szemeredi [FKS84]), again of size $O(|B(v)|)$, using which it is possible to check whether $w \in B(v)$, and return $\delta(w, v)$ if so, in $O(1)$ worst case time.

The data structure constructed by the preprocessing algorithm stores for each vertex $v \in V$,

- for $0 \le i \le k - 1$, the witness $p_i(v)$ and the corresponding distance $\delta(p_i(v), v) = \delta(A_i, v)$.
- the (2-level) hash table for the bunch $B(v)$, holding $\delta(v, w)$, for every $w \in B(v)$.

The total size of the data structure is $O(kn + \sum_{v \in V} |B(v)|)$. In Section 3.3, we show that the expected

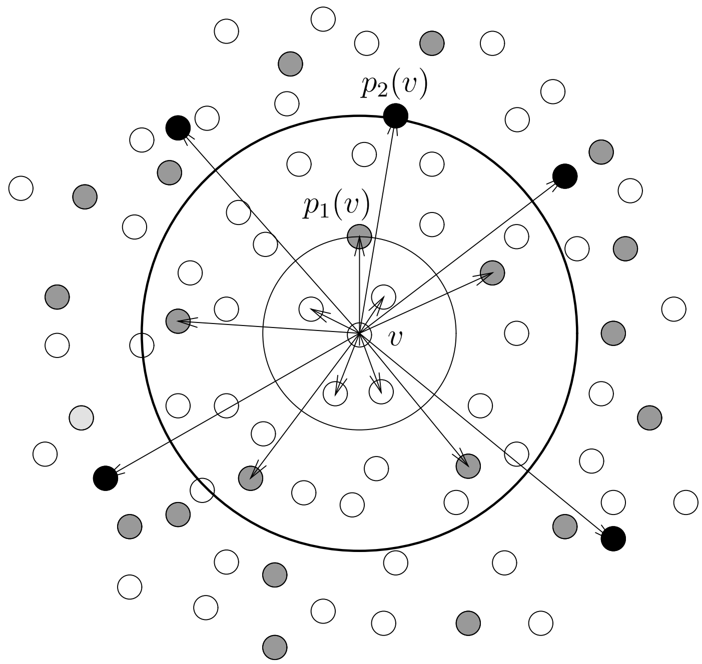

Figure 3: The bunches constructed by the preprocessing algorithm.

size of $B(v)$, for every $v \in V$, is at most $kn^{1/k}$. The total expected size of the data structure is therefore $O(kn^{1+1/k})$. The time complexity of $\mathbf{prepro}_k(V, \delta)$ is clearly $O(n^2)$.

3.2 Answering a distance query

A description of the very simple query answering algorithm $\mathbf{dist}_k(u, v)$ is given in Figure 2. It uses only four variables: u and v, the two vertices whose distance is to be estimated, a third vertex w and an index i. The algorithm repeatedly swaps u and v. This clearly does not affect their distance. Initially, $w = u = p_0(u)$ and $i = 0$. If $w \in B(v)$, a condition checked by accessing the (2-level) hash table of $B(v)$, we are done. Otherwise, the algorithm increments i, swaps u and v, and lets $w \leftarrow p_i(u) \in A_i$. It continues in this way until $w \in B(v)$. This condition is guaranteed to hold when $i = k - 1$, if not before, as then $w \in A_{k-1}$ and $A_{k-1} \subseteq B(v)$ for every $v \in V$.

When $w \in B(v)$, the algorithm returns $\delta(w, u) + \delta(w, v)$ as an upper bound on $\delta(u, v)$. The distance $\delta(w, u) = \delta(p_i(u), u)$ is read directly from the data structure constructed during the preprocessing stage. The distance $\delta(w, v) = \delta(v, w)$ is returned by the (2-level) hash table of $B(v)$ together with the answer to the query $w \in B(v)$.

The complexity of $\mathbf{dist}_k(u, v)$ is clearly $O(k)$. The most time consuming operations are the at most k accesses to the hash tables to test whether $w \in B(v)$, returning $\delta(w, v)$ if so. The stretch of the estimate produced by $\mathbf{dist}_k(u, v)$ is analyzed in Section 3.4.

3.3 Analysis of the preprocessing algorithm

We have shown already, in Section 3.1, that the running time of the preprocessing algorithm is $O(n^2)$ and that the size of the data structure produced by it is $O(kn + \sum_{v \in V} |B(v)|)$. All that remains, therefore, is to analyze the expected sizes of the bunches $B(v)$, for $v \in V$.

Lemma 3.2 *For every $v \in V$, the expected size of the bunch $B(v)$ is at most $kn^{1/k}$.*

Proof: Let $v \in V$. We prove the lemma by showing, for every $0 \le i \le k-1$, that the expected size of $B(v) \cap A_i$ is at most $n^{1/k}$. For $i = k-1$, the statement is trivial as $E[|A_{k-1}|] = n^{1/k}$. Assume, therefore, that $i < k-1$.

We show that the expected size of $B(v) \cap A_i$, for $i < k-1$, is *stochastically dominated* by a geometric random variable with parameter $p = n^{-1/k}$. Let $w_1, w_2, \ldots, w_\ell$ be the elements of A_i, arranged in a non-decreasing order of distance from v. If $w_j \in B(v)$, then $\delta(w_j, v) < \delta(A_{i+1}, v)$, and thus $w_1, w_2, \ldots, w_{j-1} \notin A_{i+1}$. Note that $p = \Pr[w \in A_{i+1} \mid w \in A_i] = n^{-1/k}$, for $i < k-1$. Thus $\Pr[w_j \in B(v)] \le (1-p)^{j-1}$ and the expected size of $B(v) \cap A_i$ is at most

$$\sum_{j=1}^{\ell} \Pr[w_j \in B(v)] \;\le\; \sum_{j=1}^{\ell} (1-p)^{j-1} \;<\; p^{-1} \;=\; n^{1/k} \,.$$

This completes the proof of the lemma. □

As described, the preprocessing algorithm has, therefore, a worst case running time of $O(n^2)$ and the data structure produced has an expected size of $O(kn^{1+1/k})$. We can get a data structure of size $O(kn^{1+1/k})$ by rerunning the algorithm until the data structure produced is small enough. By Markov's inequality, the expected number of repetitions required is constant, so the expected running time of this version of the algorithm is still $O(n^2)$, as stated in Theorem 3.1.

3.4 Analysis of the query answering algorithm

We next obtain an upper bound of $2k-1$ on the stretch of the estimated distance returned by $\mathbf{dist}_k(u, v)$.

Lemma 3.3 $\mathbf{dist}_k(u, v) \le (2k-1)\delta(u, v)$.

Proof: Clearly, the swapping of u and v does not change their distance $\Delta = \delta(u, v)$. Before the while loop starts, $w = u$ so $\delta(w, u) = 0$. We want to show that each iteration increases $\delta(w, u)$ by at most Δ. Since $A_{k-1} \subseteq B(v)$, there are at most $k-1$ iterations, so we will then end up with $\delta(w, u) \le (k-1)\Delta$. Now, $\delta(w, v) \le \delta(w, u) + \delta(u, v) \le (k-1)\Delta + \Delta \le k\Delta$, so the estimated distance returned is at most $(2k-1)\Delta$.

All that remains, therefore, is to show that $\delta(w, u)$ increases, in each iteration, by at most $\Delta = \delta(u, v)$. Let u_i, v_i and w_i be the values of the variables u, v and w assigned with a given value of i. Then v_0 and u_0 are the original values of u and v, and then $w_0 = u_0$, so $\delta(w_0, u_0) = 0$.

We want to show that $\delta(w_i, u_i) \le \delta(w_{i-1}, u_{i-1}) + \Delta$ if the i-th iteration passes the test of the while-loop. Then $w_{i-1} \notin B(v_{i-1})$, so $\delta(w_{i-1}, v_{i-1}) \ge \delta(A_i, v_{i-1}) = \delta(p_i(v_{i-1}), v_{i-1})$. However, $v_{i-1} = u_i$ and $w_i = p_i(u_i)$, so we get

$$\delta(w_i, u_i) \;=\; \delta(p_i(u_i), u_i) \;=\; \delta(p_i(v_{i-1}), v_{i-1}) \;\le\; \delta(w_{i-1}, v_{i-1}) \;\le\; \delta(w_{i-1}, u_{i-1}) + \Delta \,,$$

as required. □

This completes the proof of Theorem 3.1.

3.5 Distance labels

Let $G = (V, E)$ be a weighted undirected graph on n-vertices with integer edge weights. Let Δ be the diameter of G, and let $k \geq 1$ be an integer. Peleg [Pel00b] describes a way of assigning each vertex $v \in V$ of the graph $G = (V, E)$ an $O(kn^{1/k} \log n \log \Delta)$-bit label, denoted *label*(v), such that for any $u, v \in V$, a stretch $8k$ estimate of the distance $\delta(u, v)$ may be obtained just by looking at *label*(u) and *label*(v). Computing this estimate, in Peleg's scheme, may take $\Omega(n^{1/k})$ time.

We obtain the following improvement to Peleg's result:

Theorem 3.4 *Let (V, δ) be a metric space on n points with integral distances with diameter Δ. Let $1 \leq k \leq \log n$ be an integer. Then, it is possible to assign to each point $v \in V$ an $O(n^{1/k} \log^{1-1/k} n \log(n\Delta))$-bit label, denoted* label(v)*, such given* label(u) *and* label(v)*, for any two points $u, v \in V$, it is possible to compute, in $O(k)$ time, an approximation to the distance $\delta(u, v)$ with a stretch of at most $2k - 1$.*

As would follow from the results of Section 5, this result is essentially optimal. Lower bounds on the size of labels in various kinds of labeling schemes are also obtained by Gavoille *et al.* [GPPR01].

In our labeling scheme, *label*(v), for each $v \in V$, is composed of the the witnesses $p_i(v)$ and the distances $\delta(A_i, v)$, for $1 \leq i < k$, as well as the (2-level) hash table that holds, for every $w \in B(v)$, the distance $\delta(w, v)$. It is easy check that all the information needed by the query answering algorithm $\mathbf{dist}_k(u, v)$ is contained in *label*(u) or in *label*(v). Thus, a stretch $2k - 1$ estimate of the distance $\delta(u, v)$ may be obtained in $O(k)$ time just by examining *label*(u) and *label*(v).

It follows from Lemma 3.2 that the *expected* size of *label*(v), for any $v \in V$, is $O(kn^{1/k})$ *words*, where each word holds either a name of a vertex or a distance. As there are n vertices in the graph, and as the diameter of the graph is Δ, each word contains at most $\log(n\Delta)$ bits. We are interested, here, however, in the *maximum* size of a label, not its expected size.

It is not difficult to show, using arguments similar to arguments used below, that with high probability, the size of every bunch $B(v)$, for $v \in V$, is $O(n^{1/k} \log n)$. This yields, therefore, a distance labeling scheme with $O(n^{1/k} \log n \log(n\Delta))$-bit labels. A factor of about $\log^{1/k} n$ may be gained by slightly changing the sampling probability used by the preprocessing algorithm:

Lemma 3.5 *If the sampling probability used by* $\mathbf{prepro}_k(V, \delta)$ *is changed from $n^{-1/k}$ to $(n/\ln n)^{-1/k}$, then with high probability, the size of every bunch $B(v)$, for $v \in V$, is $O(n^{1/k} \log^{1-1/k} n)$.*

Proof: Let $p = (n/\ln n)^{-1/k}$ be the sampling probability. As in the proof of Lemma 3.2 that the size of bunch $B(v)$, for $v \in V$, is stochastically dominated by the sum $\sum_{i=0}^{k-1} X_i$ of k random variables, where X_{k-1} is binomially distributed with parameters n and p^{k-1}, and X_i, for $0 \leq i \leq k - 2$, is a geometric random variable with parameter p. (Here X_i, for $0 \leq i \leq k-1$, bounds the size of $B(v) \cap A_i$.) Furthermore, these k random variables are *independent.*

Chernoff's bounds (see Motwani and Raghavan [MR95, Chapter 4]) say that if X is a binomial random variable with $E[X] = \mu$, then

$$\begin{aligned} \Pr[\, X > (1+\delta)\mu \,] &< \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right]^{\mu} , \\ \Pr[\, X < (1-\delta)\mu \,] &< e^{-\mu\delta^2/2} . \end{aligned}$$

$s \leftarrow n^{1/k}(\ln n+1)^{1-1/k}$

$A_0 \leftarrow V$; $A_k \leftarrow \emptyset$

for $i \leftarrow 1$ to $k-1$

  for every $v \in V$,

  let $N_i(v)$ contain the s vertices of A_{i-1} closest to v.

  let A_i be a subset of A_{i-1} of size at most $\frac{|A_{i-1}|}{s}(\ln n+1)$
  that hits $N_i(v)$, for every $v \in V$.

Figure 4: Deterministic construction of the samples.

It is easy to see that $E[X_{k-1}] = np^{k-1} = n^{1/k}\ln^{1-1/k} n$. Using the first Chernoff inequality, with $\delta = 3$ and $\mu = n^{1/k}\ln^{1-1/k} n$, we get that

$$\Pr[\, X_{k-1} > 4n^{1/k}\ln^{1-1/k} n\,] \;<\; (e^3/4^4)^{n^{1/k}\ln^{1-1/k} n} \;<\; e^{-2n^{1/k}\ln^{1-1/k} n} \;<\; e^{-2\ln n} = n^{-2}\,.$$

Next, a moment reflection shows that, as $X_0,\ldots,X_{k-2}$ are independent geometric random variables, for every integer s we have $\Pr[\sum_{i=0}^{k-2} X_i > s = \Pr[\, B(s,p) < k\,]$, where $B(s,p)$ is a binomial random variable with parameters s and p. Thus,

$$\Pr[\sum_{i=0}^{k-2} X_i > 16n^{1/k}\ln^{1-1/k} n\,] \;=\; \Pr[\, B(16n^{1/k}\ln^{1-1/k} n, (n/\ln n)^{-1/k}) < k\,]\,.$$

Note that $\mu = E[\, B(16n^{1/k}\ln^{1-1/k} n, (n/\ln n)^{-1/k})\,] = 16\ln n$. As $k \le \log n$, we get that $k < \mu/2$, and using the second Chernoff inequality, with $\delta = 1/2$, we get that

$$\Pr[\sum_{i=0}^{k-2} X_i > 16n^{1/k}\ln^{1-1/k} n] \;<\; e^{-\mu/8} \;=\; n^{-2}\,.$$

The probability that at least one bunch $B(v)$ is of size greater than $20n^{1/k}\ln^{1-1/k} n$ is at most $n(n^{-2} + n^{-2}) = 2/n \ll 1$, for n large enough. □

This completes the proof of Theorem 3.4. As mentioned, this result is essentially optimal, as would follow from the results of Section 5. The integrality assumption can be easily removed if we are willing to settle for stretch $2k-1+\epsilon$, for some arbitrarily small $\epsilon > 0$. Each label is then of size $O(n^{1/k}\log^{1-1/k} n \log(n/\epsilon))$-bits.

3.6 Derandomization

The preprocessing algorithm **prepro**(V,δ) given in Section 3.1 is randomized. In it not difficult, however, to derandomize it, with only a small loss in efficiency. Randomization is only used by **prepro**(V,δ) in the selection of the samples $A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k$, and in the construction of the (2-level) hash tables.

A deterministic way of constructing a sequence of samples with all the desired properties is given in Figure 4. The sets $A_0, A_1, \ldots$ are constructed one by one. The set A_0 is simply V. Suppose A_{i-1}, for some $1 \le i < k$ was already constructed. The algorithm lets $N_i(v)$, for every $v \in V$, be the set of the $n^{1/k}\ln^{1-1/k} n$ vertices of A_{i-1} that are closest to v. Ties are broken arbitrarily. Then, the algorithm chooses a set A_i of size at most $n^{1-i/k}(\ln n+1)^{i/k}$ that *hits* all the neighborhoods $N_i(v)$, for $v \in V$. To construct the set A_i, the algorithm relies on the following well known lemma, which is a slight modification of Theorem 2.2 of Alon and Spencer [AS92, p. 6]:

Lemma 3.6 *Let $N_1, \ldots, N_n \subseteq U$ be a collection of sets with $|U| = u$ and $|N_i| \geq s$, for $1 \leq i \leq n$. Then, a set A of size at most $\frac{u}{s}(\ln \frac{ns}{u} + 1) \leq \frac{u}{s}(\ln n + 1)$ such that $N_i \cap A \neq \emptyset$, for $1 \leq i \leq n$, can be found, deterministically, in $O(u + \sum_{i=1}^{n} |N_i|)$ time.*

The set A, whose existence is claimed in Lemma 3.6, is obtained by repeatedly adding to A elements of U that hit as many unhit sets as possible, until only $\frac{u}{s}$ sets are unhit. The construction of A is then completed by adding an element from each one of the unhit sets. For more details, see Alon and Spencer [AS92, p. 6]. (Lemma 3.6 is slightly more general than Theorem 2.2 of [AS92] that assumes $u = n$.)

We now claim:

Theorem 3.7 *If the random sampling used by* $\textbf{prepro}_k(V, \delta)$ *is replaced by the deterministic sampling procedure described in Figure 4, then the size of each bunch $B(v)$, for $v \in V$, is at most $kn^{1/k}(\ln n+1)^{1-1/k}$.*

Proof: Let $v \in V$. Note that $B(v) = \cup_{i=0}^{k-1} B_i(v)$, where $B_i(v) = \{\, w \in A_i - A_{i+1} \mid \delta(w, v) < \delta(A_{i+1}, v)\}$, for $1 \leq i \leq k$. We claim that $|B_i(v)| \leq s = n^{1/k}(\ln n + 1)^{1-1/k}$, for $0 \leq i < k - 1$, otherwise $N_i(v) \subseteq B_i(v)$, and by the construction of A_{i+1}, we have $A_{i+1} \cap B_i(v) \neq \emptyset$, a contradiction. Finally, $B_{k-1}(v) = A_{k-1}$, and it is easy to show by induction that $|A_{k-1}| \leq n^{1/k}(\ln n + 1)^{1-1/k}$. □

We have thus lost only a factor of about $\log^{1-1/k} n$ with respect to the expected bunch size of the randomized algorithm, and only a factor of about k with respect to the maximum bunch size of the randomized algorithm with the slightly modified sampling probability.

The preprocessing algorithm $\textbf{prepro}_k(V, E)$ also has to construct a 2-level hash table for each bunch $B(v)$, where $v \in V$. (This step is not explicit in the description of the algorithm given in Figure 1.) The linear time algorithm given by Fredman *et al.* [FKS84] for the construction of such tables is randomized. Their algorithm is derandomized, however, by Alon and Naor [AN96]. To construct a perfect hash table over $q = \tilde{O}(n^{1/k})$ elements from a universe of size n, without assuming that k is constant, they use $O(q \log q \log n) = \tilde{O}(n^{1/k})$ time. Hence constructing the hash table for all $B(v)$ takes $\tilde{O}(n^{1+1/k})$ time, so this derandomization does not affect the overall running time of $O(n^2)$ for the preprocessing algorithm.

3.7 Fast hashing

The query answering algorithm has a *worst case* running time of $O(k)$, when 2-level hash tables are used. The hash functions used in the construction of these 2-level hash tables should be chosen from a *universal family* of hash functions (see Carter and Wegman [CW79]). The evaluation of many standard universal hash functions is quite slow. We mention, therefore, a recently suggested universal family of hash functions whose evaluation is about 10 times faster than the evaluation of hash functions, commonly mentioned in text books, that are based on arithmetic modulo a prime number.

Suppose that each vertex x is represented by a standard 32-bit integer. Let $a > 0$ be a random *odd* 32-bit integer. Define $h_a(x) = (a{*}x) \gg (32-\ell)$, for some $\ell \leq 32$. The above should be understood as C-notation, with $\gg$ denoting right shift and $a{*}x$ denoting multiplication modulo 2^{32}. The result is then an ℓ-bit integer. The family $\{h_a(x) \mid a \text{ odd}\}$ is then a universal family, in the sense that $\Pr[h_a(x) = h_a(y)] \leq 2^{-\ell}$, for every $x \neq y$. The universality of this multiplicative hashing scheme is pointed out by Dietzfelbinger *et al.* [DHKP97]. Its efficiency is demonstrated by Dietzfelbinger and Hüne [DH96] and by Thorup [Tho00a].

3.8 External memory

An interesting application of the above technique is the following: Given a huge $n \times n$ distance matrix that resides in *external memory*, or on *tape*, we can efficiently produce a compressed approximate version of it that could fit in a much smaller, but much faster, *internal memory*. We assume here that it is possible to pick a not too large integer k such that the internal memory can accommodate a data structure of size $O(kn^{1+1/k})$.

A random access to an entry in the original external memory matrix is typically in the order of 10,000 times slower than an internal memory access. (See Vitter [Vit01] for more on external memory issues.) Thus, our simple $O(k)$ time distance query algorithm, working in internal memory, is expected to be significantly faster than a single access to the external memory.

Assuming that the distance matrix is stored in row (or column) order, our preprocessing can take advantage of the fact that one sequential read of the whole external memory matrix is comparatively cheap. First, we generate the sets A_i. As we assume that the internal memory is of size $O(kn^{1+1/k})$, the sets A_i can be easily stored in the internal memory. We then read the rows of the matrix, one by one. Given the row corresponding to a vertex v, we first compute the distances $\delta(A_i, v)$ and the witnesses $p_i(v)$, for $0 \le i \le k-1$, and then construct the bunch $B(v)$. The bunches are just accumulated in internal memory, which we have assumed is large enough to accommodate all of them.

4 Approximate distance oracles for graphs

In the previous section, we assumed that metric $\delta(u, v)$ is given to us explicitly. Here, we consider the more realistic situation in which the metric that we are supposed to process is the shortest paths metric of a weighted undirected graph. The graph, and not the metric, is given to us this time.

We can, of course, begin by solving the APSP problem for the input graph and then use the algorithms of the previous section to preprocess the metric obtained. This solution is wasteful, however, both in terms of running time and in terms of space. It is much more efficient to directly process the graph that induces the metric.

The new preprocessing algorithm is described next, in Section 4.1. A modification to the query answering algorithm that allows it to return paths, and not just approximate distances, is then described in Section 4.2. The analysis of the modified preprocessing algorithm is given in Section 4.3. Finally, in Section 4.4 we show that our preprocessing algorithm is also a very efficient algorithm for constructing sparse spanners and compact tree covers.

4.1 Preprocessing a graph

A description of the preprocessing algorithm $\mathbf{prepro}_k(V, E)$ is given in Figure 5. It receives as input a weighted undirected graph $G = (V, E)$. The preprocessing algorithm is similar to the preprocessing algorithm given in Figure 1. In particular, the sets A_i and the bunches $B(v)$ would be exactly the same. The implementation details, this time, are less trivial, as distances $\delta(u, v)$ have to be computed, instead of just being read from an input matrix. This is why we introduce the new sets $C(\cdot)$ before computing bunches $B(\cdot)$.

The algorithm starts again by constructing the samples $A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k$, where $A_0 = V$ and $A_k = \emptyset$. As before, elements of A_i will be referred to as *i-centers*.

The operation of the algorithm is then composed of k iterations. The i-th iteration starts by computing the distances $\delta(A_i, v)$, for every $v \in V$, where $\delta(A_i, v) = \min\{\, \delta(w, v) \mid w \in A_i\}$. This is done by adding to $G = (V, E)$ a new source vertex s, and edges (s, w) of weight 0, for every $w \in A_i$, and by computing the distances from the new source s to all the other vertices of the graph. The distances are found in $O(m)$ time by running the single-source shortest paths algorithm of Thorup [Tho00b]. It is easy to check

algorithm $\mathbf{prepro}_k(V, E)$

$A_0 \leftarrow V$; $A_k \leftarrow \emptyset$

for $i \leftarrow 1$ to $k-1$

  let A_i contain each element of A_{i-1},
  independently, with probability $n^{-1/k}$.

$\delta(A_k, v) \leftarrow \infty$

for $i \leftarrow k-1$ downto 0

  for every $v \in V$,
    compute $\delta(A_i, v)$ and find $p_i(v) \in A_i$
    such that $\delta(p_i(v), v) = \delta(A_i, v)$.
    if $\delta(A_i, v) = \delta(A_{i+1}, v)$ then $p_i(v) \leftarrow p_{i+1}(v)$ (*)

  for every $w \in A_i - A_{i+1}$,
    grow a shortest path tree $T(w)$ from w
    spanning $C(w) = \{\, v \in V \mid \delta(w, v) < \delta(A_{i+1}, v) \,\}$.

for every $v \in V$,
  let $B(v) \leftarrow \{w \in V \mid v \in C(w)\}$.

Figure 5: Preprocessing a graph

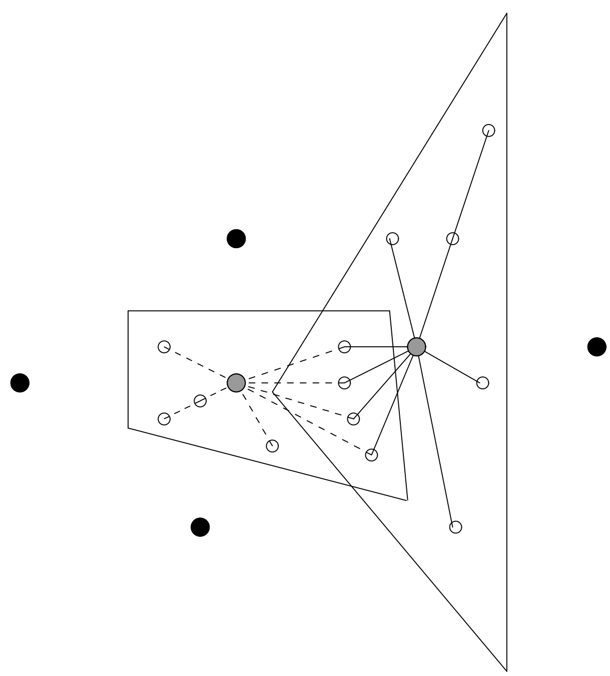

Figure 6: The clusters constructed by the preprocessing algorithm.

that for every $v \in V$, the distance from s to v in the new graph is indeed $\delta(A_i, v)$. Furthermore, the shortest paths tree constructed by the algorithm supplies, for every $v \in V$, a witness $p_i(v) \in A_i$ such that $\delta(p_i(v), v) = \delta(A_i, v)$. Indeed, if v is in the branch of the shortest paths tree that starts with the edge

(s, w), where $w \in A_i$, then $\delta(A_i, v) = \delta(w, v)$ and we can set $p_i(v) \leftarrow w$. All the witnesses are easily found, therefore, in $O(m)$ time.

The minimum distance $\delta(A_i, v)$ may be attained by several vertices of A_i. The preprocessing algorithm of the previous section lets $p_i(v)$ be an arbitrary vertex of A_i satisfying $\delta(p_i(v), v) = \delta(A_i, v)$. The new statement (*) trivially preserves this property. In addition, it also ensures that the following property, that plays a crucial role in the construction of small-stretch paths, as described in Section 4.2, also holds:

Lemma 4.1 *For any $v \in V$ and $0 \leq i \leq k-1$, we have $p_i(v) \in B(v)$.*

Proof: We prove the claim by induction on i from above. The claim trivially holds when $i = k-1$, as then $p_{k-1}(v) \in A_{k-1} \subseteq B(v)$, for every $v \in V$. Suppose therefore that $i < k-1$, and that $p_{i+1}(v) \in B(v)$. If the test of (*) fails, we get $p_i(v) = p_{i+1}(v) \in B(v)$. Otherwise, $\delta(p_i(v), v) = \delta(A_i, v) < \delta(A_{i+1}, v)$, and then $p_i(v) \in B(v)$ by definition of $B(v)$. □

Next, the algorithm constructs a *cluster* $C(w)$ around each i-center $w \in A_i - A_{i+1}$. The cluster $C(w)$ is composed of all the vertices that are closer to w than to any $(i-1)$-center. In other words, $C(w) = \{ v \in V \mid \delta(w, v) < \delta(A_{i+1}, v) \}$. Note that for every $w \in A_{k-1}$ we have $C(w) = V$, as $\delta(A_k, v) = \infty$, for every $v \in V$.

It is easy to see that the bunches of the previous section and the clusters of this section are 'inverses' of each other, in that $w \in B(v)$ if and only if $v \in C(w)$ for any $v, w \in V$. Thus, the bunches constructed by the final loop of the preprocessing algorithm are identical to the bunches that would have been constructed by the preprocessing algorithm of the previous section.

The construction of clusters is reminiscent of the construction of *Voronoi diagrams*. An important difference here, however, is that the cluster $C(w)$ of an i-center $w \in A_i - A_{i+1}$ contains all vertices whose distance to w is smaller than their distance to all $(i+1)$-centers, and not to all i-centers, as the definition of Voronoi diagrams would suggest. In particular, the clusters at a particular iteration are not necessarily disjoint. A schematic description of the clustering construction process is given in Figure 6. The filled vertices there are $(i+1)$-centers. The two large unfilled vertices are i-centers and the two polygons depict the clusters associated with them. (In Figure 6, it is implicitly assumed that the distances between the vertices are Euclidean. This is done for illustration purposes only. Our algorithms work on general weighted graphs.)

Each cluster $C(w)$ is computed by running a slightly modified version of Thorup's SSSP algorithm from w [Tho99]. Since this algorithm is rather complicated, we describe instead a modified version of Dijkstra's classical SSSP algorithm [Dij59] (see also [CLRS01, Chapter 24]). The changes to Thorup's algorithm are very similar.

Modifying Dijkstra's algorithm Dijkstra's algorithm with source w maintains for each vertex v an upper bound $d(v)$ on the distance $\delta(w, v)$. If $d(v)$ has not been assigned yet, it is interpreted as infinite. Initially, we just set $d(w) = 0$, and we have no *visited* vertices. At each iteration, we select an unvisited vertex u with the smallest finite $d(u)$, visit it, and *relax* all its edges, that is, for each incident edge $(u, v) \in E$, we set $d(v) \leftarrow \min\{ d(v), d(u) + \ell(u, v)\}$. We continue in this way until no unvisited vertex v has a finite $d(v)$.

Our simple modification of Dijkstra's algorithm is that we relax the edge (u, v) *only if* $d(u) + \ell(u, v) < \delta(A_{i+1}, v)$. Note that $\delta(A_{i+1}, v)$ was computed in the previous iteration so the test takes constant time.

Lemma 4.2 *The modified Dijkstra's algorithm visits exactly the vertices of $C(w)$, assigning each the correct distance from w.*

Proof: The proof is similar to correctness proof of Dijkstra's original algorithm. Suppose $w \in A_i - A_{i+1}$. The essential new point is the following easily verified claim: if $v \in C(w)$ and v' lies on a shortest path from w to v, then $\delta(w, v') < \delta(A_{i+1}, v')$, so $v' \in C(w)$. By definition, $v \in C(w)$ if and only if $\delta(w, v) < \delta(A_{i+1}, v)$, but then $\delta(w, v') = \delta(w, v) - \delta(v, v') < \delta(A_{i+1}, v) - \delta(v, v') \leq \delta(A_{i+1}, v')$, as desired.

Note that we only relax an edge (u, v) if $d(u) + \ell(u, v) < \delta(A_{i+1}, v)$. Consequently, a vertex $v \notin C(w)$ is never assigned a finite distance, and hence it is never visited.

We now show that all vertices visited are assigned correct distances from w. The proof is by induction. Suppose that $v \in C(w)$ is about to be visited and that all previously visited vertices were assigned the correct distance. Let p be a shortest path from w to v. Let u' be the last visited vertex on this path, and let v' be the next vertex on the path. By the above claim, $\delta(w, v') < \delta(A_{i+1}, v')$. Moreover, by the induction hypothesis, when u' was visited $d(u') = \delta(w, u')$, and then $\delta(w, v') = \delta(w, u') + \ell(u', v') = d(u') + \ell(u', v')$. Hence $d(u') + \ell(u', v') < \delta(A_{i+1}, v')$, so the edge (u', v') was relaxed, setting $d(v') \leftarrow \delta(u, v')$. As v is about to be visited before v', we must have $d(v) \leq d(v') = \delta(w, v') \leq \delta(w, v) \leq d(v)$, so $d(v) = \delta(w, v)$.

Finally, we want to show that all $v \in C(w)$ are visited. Suppose for a contradiction that $v \in C(w)$ is not visited. Let (u', v') be the last edge on a shortest path to v with u' visited. From above, we know that u' got assigned the correct distance, and the same analysis as above implies that (u', v') got relaxed when u' was visited, but then v' will be visited eventually, contradicting the choice of (u', v'). □

Thus, the modified version of Dijkstra's algorithm that we described does construct $C(w)$. It is easy to arrange that it would also produce a shortest path tree $T(w)$ spanning the cluster $C(w)$. This would not affect the running time of the algorithm.

For a simple, yet relatively efficient, implementation of Dijkstra's algorithm, we can just use William's heap [Wil64] (see also [CLRS01, Chapter 6]) to store the finite distances $d(v)$ of the unvisited vertices. We can then both find the v minimizing $d(v)$ and decrease some $d(v)$ in $O(\log n)$ time. The former is done at most $n - 1$ times, and the latter is done at most m times, so the total running time of the unmodified algorithm, from a given source vertex, is $O((m + n) \log n)$. It is easy to see that in the modified version of the algorithm, all the edges relaxed are edges that touch vertices of $C(w)$. Thus, the time spent on the construction of $C(w)$ is $O(|\mathcal{E}(C(w))| \log n)$, where $\mathcal{E}(C(w))$ is the set of edges touching vertices of $C(w)$. The complexity is reduced to $O(|\mathcal{E}(C(w))| + |C(w)| \log n)$ if the more sophisticated Fibonacci heaps of Fredman and Tarjan [FT87] (see also [CLRS01, Chapter 20]) are used.

The same conditional relaxation can be applied to Thorup's SSSP algorithm [Tho99]. We first spend, once and for all, $O(m)$ time on constructing a so-called *component hierachy*. Afterwards, each cluster $C(w)$, for $w \in V$, can be computed in $O(|\mathcal{E}(C(w))|)$ time.

When all the clusters $C(w)$, for $w \in V$ are constructed, they are used to generate the bunches $B(v)$, for $v \in V$. Recall that, by definition, $w \in B(v)$ if and only if $v \in C(w)$. The conversion can clearly be done in $O(\sum_{w \in V} |C(w)|) = O(\sum_{v \in V} |B(v)|)$ time.

Finally, the algorithm constructs (2-level) hash tables for the bunches $B(v)$, for $v \in V$, and outputs the witnesses $p_i(v)$, the distances $\delta(p_i(v), v) = \delta(A_i, v)$, and the hash tables of $B(v)$, for every $1 \leq i \leq k$ and $v \in V$. In addition to that, the preprocessing algorithm also outputs, for every $w \in V$, the shortest paths tree $T(w)$ that spans the cluster $C(w)$.

The sum of the sizes of all the trees $T(w)$, for $w \in W$, is the same as the sum of the sizes of all the clusters, which is also the sum of the sizes of all the bunches. Thus, the size of the data structure produced is, asymptotically, the same as the size of the data structure that would have been produced, had the preprocessing algorithm of the previous section been applied to the shortest paths metric of the graph. Thus, the expected size of the produced data structure is $O(kn^{1+1/k})$. All that remains, therefore, is to analyze the running time of $\mathbf{prepro}_k(V, E)$. This is done in Section 4.3.

As a final remark, we note that instead of constructing a separate hash table for each bunch $B(v)$, for $v \in V$, we can construct a *single* (2-level) hash table of size $O(\sum_{v \in V} |B(v)|)$ that holds $\delta(w, v)$, for every $w, v \in V$ such that $w \in B(v)$. The access time would still be $O(1)$.

4.2 Answering a path query

As all the data structures returned by the metric preprocessing algorithm of Section 3.1 are also returned by the graph preprocessing algorithm of Section 4.1, the query answering algorithm from Figure 2, detailed in Section 3.2, may be used, without any modification, to answer approximate distance queries.

We next describe how to augment the distance query algorithm if it is to return not just an estimated distance $\mathbf{dist}_k(u, v)$ of stretch at most $2k - 1$, but also a path from u to v of length at most $\mathbf{dist}_k(u, v)$.

When the distance query algorithm terminates, $w \in B(v)$ so $v \in C(w)$. Moreover, by Lemma 4.1, $w = p_i(u) \in B(u)$, so we also have $u \in C(w)$. Hence, the path between u and v in $T(w)$, the shortest paths tree of $C(w)$, is of length at most $\delta(w, u) + \delta(w, v)$. To report the edges on this path in constant time per edge, we move in parallel from u and v towards the root w, stopping as soon as we reach, from one of u and v, a vertex w' that was already reached from the other. (This vertex is the *least common ancestor* of u and v in the tree.) We then output the edges on the path from u to w' and, in reversed order, the edges on the path from v to w'.

The above solution constructs the small-stretch path from u to v in *amortized* constant time per edge. Using techniques from [TZ01] it is possible to construct the path in worst case constant time per edge. We do not elaborate on this here.

4.3 Analysis of the graph preprocessing algorithm

As mentioned, the complexity of constructing the cluster $C(w)$ is $O(|\mathcal{E}(C(w))|)$ (or $O((|\mathcal{E}(C(w))| + |C(w)|)\log n)$ if the simple modification of Dijkstra's algorithm is used). Recall that $\mathcal{E}(C(w))$ is the set of edges that touch vertices of $C(w)$. Let $\mathcal{E}(v)$ be the set of edges that touch the vertex v. The total cost of constructing all clusters is asymptotically bounded by

$$\sum_{w \in V} |\mathcal{E}(C(w))| \leq \sum_{w \in V, v \in C(w)} |\mathcal{E}(v)| = \sum_{v \in V, w \in B(v)} |\mathcal{E}(v)| = \sum_{v \in V} (|B(v)| \cdot |\mathcal{E}(v)|)$$

By Lemma 3.2, the expected size of $|B(v)|$ is at most $kn^{1/k}$, for any $v \in V$, so by linearity of expectation, the expected total cost is asymptotically bounded by

$$\sum_{v \in V} (n^{1/k} |\mathcal{E}(v)|) = 2kmn^{1/k} \ .$$

Since all other operations in $\mathbf{prepro}_k(V, E)$ take only $O(km)$ time, its total complexity is $O(kmn^{1/k})$.

As in the last section we note that it is only the expected size of the data structure constructed which is $O(kn^{1+1/k})$. To obtain a data structure of size $O(kn^{1+1/k})$, we may have to run $\mathbf{prepro}_k(V, E)$ several times, but the expected number of repetitions is constant, so the total expected preprocessing time is still $O(kmn^{1/k})$, as specified in Theorem 1.1.

4.4 Sparse spanners and tree covers

As described in Section 4.2, the query answering algorithm may actually find a stretched path between u and v in some tree $T(w)$. We get, therefore, the following interesting corollary:

girth	number of edges	lower-bound references
4	$\Theta(n^2)$	complete bipartite graphs
6	$\Theta(n^{3/2})$	[Rei58],[ERS66],[Bro66],[Wen91]
8	$\Theta(n^{4/3})$	[Tit59],[Ben66],[Wen91]
10	$\Omega(n^{6/5})$, $O(n^{5/4})$	[Tit59],[Ben66],[LU93]
12	$\Theta(n^{6/5})$	[Tit59],[Ben66],[Wen91],[LU93]
14	$\Omega(n^{9/8})$, $O(n^{7/6})$	[LUW95],[LUW96]
16	$\Omega(n^{10/9})$, $O(n^{8/7})$	[WU93],[LUW95]
$4r$, $r \geq 5$	$\Omega(n^{1+\frac{1}{3(r-1)}})$, $O(n^{1+\frac{1}{2r-1}})$	[LUW95],[LUW96]
$4r+2$, $r \geq 4$	$\Omega(n^{1+\frac{1}{3r-1}})$, $O(n^{1+\frac{1}{2r}})$	[LUW95],[LUW96]

Table 2: Best known bounds on the maximum number of edges in an n-vertex graph with a given girth.

Corollary 4.3 *The collection of shortest paths trees $T(w)$, for $w \in V$, constructed by algorithm* $\mathbf{prepro}_k(V, E)$*, forms a $(2k-1)$-spanner of the graph $G = (V, E)$. The expected size of this $(2k-1)$-spanner is $O(kn^{1+1/k})$ and it can be constructed in $O(kmn^{1/k})$ time.*

As mentioned, the fact that every *weighted* graph on n-vertices has a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges is not new. The corollary gives, however, a much faster algorithm for constructing such spanners. The fastest running time known before, for weighted graph, was $O(mn^{1+1/k})$ [ADD+93]. For unweighted graphs, there is linear time algorithm for constructing such spanners (see also Exercise 3 on page 188 of Peleg [Pel00a], attributed to [HZ96]).

Combining Corollary 4.3 with Lemma 3.5 we also get the following corollary:

Corollary 4.4 *The collection of shortest paths trees $T(w)$, for $w \in V$, constructed by algorithm* $\mathbf{prepro}_k(V, E)$*, with the sampling probability changed from $n^{-1/k}$ to $(n/\ln n)^{-1/k}$, forms a* tree cover *of the graph $G = (V, E)$ with the following properties: (i) With high probability, every vertex is contained in only $O(n^{1/k} \log^{1-1/k} n)$ trees. (ii) For every two vertices $u, v \in V$, there is a tree $T(w)$ in this collection that contains a path between u and v that is of stretch at most $2k-1$. Furthermore, the corresponding tree can be identified in $O(k)$ time.*

A deterministic algorithm for constructing such tree covers may be obtained using the technique of Section 3.6. Our tree cover construction improves a construction implicit in Awerbuch and Peleg [AP92] (see also Peleg [Pel00a, Chapter 15]). In [TZ01], we use our tree cover construction, together with other ideas, to obtain *routing schemes* for weighted undirected networks that exhibit an essentially optimal tradeoff between the size of the routing tables used and the stretch of the resulting routes.

5 Space lower bound

A simple argument shows that for any integer k, any graph on n vertices with at least $n^{1+1/k}$ edges contains a cycle of size at most $2k$. (For a proof that $\frac{1}{2}n^{1+1/k}$ edges are in fact enough, see Alon *et al.* [AHL02].)

This result is conjectured by Erdős [Erd64], Bondy and Simonovits [BS74] and Bollobás [Bol78] to be tight. Namely, it is conjectured that for any $k \geq 1$, there are graphs with $\Omega(n^{1+1/k})$ edges and girth greater than $2k$. As any graph contains a bipartite subgraph with at least half the edges, the conjecture actually implies the existence of graphs with $\Omega(n^{1+1/k})$ edges and girth at least $2k+2$. This conjecture was proved, however, only for $k = 1, 2, 3, 5$ (see references below).

Let $m_g(n)$ be the maximal number of edges in an n-vertex graph with girth g. The girth conjecture says that $m_{2k+2}(n) = \Omega(n^{1+1/k})$. Note, as mentioned above, that $m_{2k+2}(n) = \Theta(m_{2k+1}(n))$. The best bounds on $m_g(n)$, for even girth g, are given in Table 2. (Several references are given for each result. This is either because the result was independently discovered by several authors, or because there are several variants of the construction. Some of the references, e.g., Wenger [Wen91], were added as they are more accessible than the older references.) The results for $g = 6$ follow from constructions of finite projective geometries. The constructions of Lazebnik *et al.* [LUW95],[LUW96] slightly improve results obtained by Margulis [Mar88] and the results obtained using the Ramanujan graphs of Lubotzky *et al.* [LPS88].

Proposition 5.1 *Let k be an integer, and let $t < 2k+1$. Then, any stretch t distance oracle for graphs with n vertices and m edges must use at least* $\min\{m, m_{2k+2}(n)\}$ *bits of storage on at least one input graph.*

Proof: Let O be a stretch t distance oracle for graphs with n vertices and m edges. For any graph H of this size, let O_H be the data structure produced by O by preprocessing H. Let $O_H(v, w)$ be the approximate distance returned by the oracle for the query (v, w). Note that $\delta_H(v, w) \leq O_H(v, w) \leq t\,\delta_H(v, w)$.

Let G be a girth $2k+2$ unweighted graph on n vertices with $m' = \min\{m, m_{2k+2}(n)\}$ edges. (If $m < m_{2k+2}(n)$ we can simply pick an m-edge subgraph of a girth $2k+2$ graph with $m_{2k+2}(n)$ edges.)

Let H be any subgraph of G. Consider any edge (v, w) of G. If (v, w) is in H, then $O_H(v, w) \leq t < 2k+1$. But, if (v, w) is not in H, the shortest path from v to w in H has at least $2k+1$ edges, so $O_H(v, w) \geq 2k+1$. Consequently, all the $2^{m'}$ subgraphs of G have different tables, and hence at least one requires m' bits. □

Proposition 5.1 holds even if the oracle is only required to produce estimated distances, without being required to produce corresponding paths. We point out, however, that there is still a logarithmic gap of $\Theta(k \log n)$ between this lower bound and our upper bound, even if the girth conjecture holds, as our algorithms use $O(kn^{1+1/k})$ *words* while the lower bound is $\Omega(n^{1+1/k})$ *bits.*

Finally, we point out that no space efficient approximate distance oracles are possible for *directed* graphs:

Proposition 5.2 *For any finite stretch distance oracle for* directed *graphs must use at least* $\Omega(n^2)$ *bits of storage on at least one n-vertex graph.*

Proof: Let $V_1 = \{1, 2, \ldots, n/2\}$ and $V_2 = \{n/2+1, n/2+2, \ldots, n\}$. Consider the family of directed graphs on $V_1 \cup V_2$ in which all edges are directed from V_1 to V_2. There are $2^{(n/2)^2}$ such graphs. As in the proof on Proposition 5.1, each such graph must be represented by a different table. Hence, at least one of tables is should contain at least $(n/2)^2$ bits. □

It is not difficult to see that Proposition 5.2 continues to hold even if the input graphs are required to be strongly connected. We let $V_1 = \{1, 2, \ldots, n/3\}$, $V_2 = \{n/3+1, n/3+2, \ldots, 2n/3\}$ and $V_3 = \{2n/3+1, 2n/3+2, \ldots, n\}$, and consider all graphs whose edge set is composed of a non-empty subset of $V_1 \times V_2$ and from all the edges of $(V_2 \times \{2n/3+1\}) \cup \{(i, i+1) \mid 2n/3+1 \leq i \leq n\} \cup (\{n\} \times V_1)$. There are $2^{(n/3)^2} - 1$ such graphs and each one of them requires a distinct table.

6 Concluding remarks and open problems

We presented approximate distance oracles with fast preprocessing times, essentially optimal space requirements, and constant query time. Our construction is extremely simple. It yields, as byproducts, improved algorithms for constructing sparse spanners, more compact tree covers, and more concise distance labelings. Due to their basic nature, we expect our ideas to prove useful in many other contexts.

Some interesting open questions remain. First, our basic preprocessing algorithm is *randomized*. While it was easy to derandomize it when the full distance matrix was available, it is not clear how to do it in $o(mn)$ time in the graph setting. It seems that new ideas would be needed to achieve that.

Our oracles are almost optimal, in all respects, when the parameter k is large. It remains an interesting open problem, however, to reduce the preprocessing times of small stretch oracles. The situation for stretch 3 is especially intriguing. We show here that a stretch 3 oracle with a space requirement of $O(n^{3/2})$ can be constructed in $O(mn^{1/2})$ time. Cohen and Zwick [CZ01] have shown that a stretch 3 oracle that uses $O(n^2)$ space can be constructed in $O(n^2 \log n)$ time. Could these results be combined, i.e., is it possible to construct a stretch 3 oracle that uses only $O(n^{3/2})$ space in $\tilde{O}(n^2)$ time? For unweighted graphs, this problem was recently solved by Baswana and Sen [BS04].

As mentioned in Section 4.4, the results of this paper, combined with some other ingredients, yield essentially optimal routing schemes for weighted undirected networks. More on this can be found in [TZ01].

Acknowledgment

We would like to thank Edith Cohen for her results that inspired this research and for making the cooperation between the authors possible, Felix Lazebnik for his help in compiling Table 2, and Michael Elkin for pointing out the connection between distance oracles and distance labels.

References

[ABCP99] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28:263–277, 1999.

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.

[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.

[AHL02] N. Alon, S. Hoory, and N. Linial. The Moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002.

[AN96] N. Alon and M. Naor. Derandomization, witnesses for boolean matrix multiplication, and construction of perfect hash functions. *Algorithmica*, 16:434–449, 1996.

[AP92] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, 1992.

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

[Bar99] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, Texas*, pages 161–168, 1999.

[Ben66] C.T. Benson. Minimal regular graphs of girth eight and twelve. *Canadian Journal of Mathematics*, 18:1091–1094, 1966.

[Bol78] B. Bollobás. *Extremal graph theory.* Academic Press, 1978.

[Bou85] J. Bourgain. On Libschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[Bro66] W.G. Brown. On graphs that do not contain a Thomsen graph. *Canad. Math. Bull.*, 9:281–285, 1966.

[BS74] J.A. Bondy and M. Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16:97–105, 1974.

[BS03] S. Baswana and S. 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, Eindhoven, The Netherlands*, pages 384–396, 2003.

[BS04] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $O(n^2 \log n)$ time. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Lousiana*, pages 264–273, 2004.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, second edition, 2001.

[Coh99] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sc.*, 18:143–154, 1979.

[CZ00] S. Chaudhuri and C.D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27:212–226, 2000.

[CZ01] E. Cohen and U. Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

[DH96] M. Dietzfelbinger and M. Hüne. A dictionary implementaion based on dynamic perfect hashing, 1996. *DIMACS Implementation Challenge,* unpublished paper.

[DHKP97] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25:19–51, 1997.

[DHZ00] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740–1759, 2000.

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

[Elk01] M.L. Elkin. Computing almost shortest paths. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing, Newport, Rhode Island*, pages 53–62, 2001.

[EP01] M.L. Elkin and D. Peleg. $(1+\epsilon,\beta)$-Spanner constructions for general graphs. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing, Crete, Greece*, pages 173–182, 2001.

[Erd64] P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague, 1964.

[ERS66] P. Erdős, A. Rényi, and V.T. Sós. On a problem of graph theory. *Studia Sci. Math. Hungar.*, 1:215–235, 1966.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31:538–544, 1984.

[FRT03] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, California*, pages 448–455, 2003.

[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.

[GLNS02] J. Gudmundson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles revisited. In *Proceedings of the 13th Annual International Symposium on Algorithms and Computation*, pages 357–368, 2002.

[GPPR01] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, D.C.*, pages 210–219, 2001.

[HZ96] S. Halperin and U. Zwick. Unpublished result, 1996.

[Ind99] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 428–434, 1999.

[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.

[Kle02] P.N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 820–827, 2002.

[Kru56] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[LU93] F. Lazebnik and V.A. Ustimenko. New examples of graphs without small cycles and of large size. *European Journal of Combinatorics*, 14(5):445–460, 1993. Algebraic combinatorics (Vladimir, 1991).

[LUW95] F. Lazebnik, V.A. Ustimenko, and A.J. Woldar. A new series of dense graphs of high girth. *Bulletin of the American Mathematical Society (New Series)*, 32(1):73–79, 1995.

[LUW96] F. Lazebnik, V.A. Ustimenko, and A.J. Woldar. A characterization of the components of the graphs $D(k, q)$. *Discrete Mathematics*, 157(1-3):271–283, 1996.

[Mar88] G.A. Margulis. Explicit group–theoretical construction of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24:39–46, 1988.

[Mat96] J. Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93:333–344, 1996.

[Mat02] J. Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

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

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Pel00a] D. Peleg. *Distributed computing – A locality-sensitive approach*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[Pel00b] D. Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33:167–176, 2000.

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

[Rei58] I. Reiman. Über ein Problem von K. Zarankiewicz. *Acta. Math. Acad. Sci. Hungar.*, 9:269–273, 1958.

[SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, New York*, pages 605–614, 1999.

[Tho99] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.

[Tho00a] M. Thorup. Even strongly universal hashing is pretty fast. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 496–497, 2000.

[Tho00b] M. Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35:189–201, 2000.

[Tho01] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, Las Vegas, Nevada*, pages 242–251, 2001.

[Tit59] J. Tits. Sur la trialité et certains groupes qui s'en déduisent. *Publ. Math. I.H.E.S.*, 2:14–20, 1959.

[TZ01] M. Thorup and U. Zwick. Compact routing schemes. In *Porceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece*, pages 1–10, 2001.

[Vit01] J.S. Vitter. External memory algorithms and data structures: Dealing with MASSIVE DATA. *ACM Computing Surveys*, 33(2):209–271, 2001.

[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.

[Wil64] J.W.J. Williams. Heapsort. *Comm. ACM*, 7(5):347–348, 1964.

[WU93] A.J. Woldar and V.A. Ustimenko. An application of group theory to extremal graph theory. In *Group theory, Proceedings of the Ohio State-Denison Conference*, pages 293–298. World Sci. Publishing, River Edge, NJ, 1993.

[Zwi02] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.