Additive Spanners and Distance Oracles in Quadratic Time

Mathias Bæk Tejs Knudsen[*]

University of Copenhagen, Copenhagen, Denmark
`mathias@tejs.dk`

―――― **Abstract** ――――

Let G be an unweighted, undirected graph. An additive k-spanner of G is a subgraph H that approximates all distances between pairs of nodes up to an additive error of $+k$, that is, it satisfies $d_H(u,v) \leq d_G(u,v) + k$ for all nodes u, v, where d is the shortest path distance. We give a deterministic algorithm that constructs an additive $O(1)$-spanner with $O(n^{4/3})$ edges in $O(n^2)$ time. This should be compared with the randomized Monte Carlo algorithm by Woodruff [ICALP 2010] giving an additive 6-spanner with $O(n^{4/3} \log^3 n)$ edges in expected time $O(n^2 \log^2 n)$.

An (α, β)-approximate distance oracle for G is a data structure that supports the following distance queries between pairs of nodes in G. Given two nodes u, v it can in constant time compute a distance estimate $\tilde{d}$ that satisfies $d \leq \tilde{d} \leq \alpha d + \beta$ where d is the distance between u and v in G. Sommer [ICALP 2016] gave a randomized Monte Carlo $(2,1)$-distance oracle of size $O(n^{5/3} \operatorname{poly} \log n)$ in expected time $O(n^2 \operatorname{poly} \log n)$. As an application of the additive $O(1)$-spanner we improve the construction by Sommer [ICALP 2016] and give a Las Vegas $(2,1)$-distance oracle of size $O(n^{5/3})$ in time $O(n^2)$. This also implies an algorithm that in $O(n^2)$ time gives approximate distance for all pairs of nodes in G improving on the $O(n^2 \log n)$ algorithm by Baswana and Kavitha [SICOMP 2010].

1 Introduction

Let $G = (V, E)$ be an unweighted, undirected graph on n nodes and m edges. A subgraph H of G is an *additive k-spanner* if the following holds for every pair u, v of nodes in G:
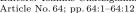
$$d_H(u,v) \leq d_G(u,v) + k \,,$$

where $d_H(u,v)$ and $d_G(u,v)$ is the distance between u and v in H and G respectively. This paper will only consider additive spanners and not multiplicative or mixed spanners, so we will simply say that H is a *k-spanner* when we mean that H is an additive k-spanner.

In this paper we consider algorithms constructing k-spanners, and there are therefore three interesting parameters: The distortion k, the running time of the algorithm, and the size of the spanner created. Elkin and Peleg [19] showed how to construct 2-spanners with $O(n^{3/2})$ edges in $O(n^{5/2})$ time, and Baswana et al [9] gave an algorithm that constructs 6-spanners with $O(n^{4/3})$ edges in $O(n^{2/3}m)$ time.
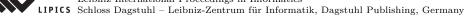

■ **Table 1** A summary of the performance of selected algorithms that creates a k-spanner H from a graph on n nodes. It shows the additive distortion, k, and an upper bound on the number of edges in H as well as the running time of the algorithm that constructs H.

k	Number of Edges	Running Time	Comment	Reference
2	$O\big(n^{3/2}\big)$	$O\big(n^{5/2}\big)$	Deterministic	[19]
2	$O\big(n^{3/2}\log^{1/2} n\big)$	$O\big(n^2\log^2 n\big)$	Deterministic	[18]
2	$O\big(n^{3/2}\big)$	$O\big(n^2\big)$	Deterministic	Theorem 5
6	$O\big(n^{4/3}\big)$	$O\big(n^{2/3}m\big)$	Deterministic	[9]
6	$O\big(n^{4/3}\log^3 n\big)$	$O\big(n^2\log^2 n\big)$	Randomized Monte Carlo	[33]
8	$O\big(n^{4/3}\big)$	$O\big(n^2\big)$	Deterministic	Theorem 9

The running time of these algorithms can be improved if we allow the k-spanners to be larger by a $\operatorname{poly}\log n$ factor. Dor, Halperin and Zwick [18] showed that we can construct 2-spanners with $O\big(n^{3/2}\log^{1/2} n\big)$ edges in $O\big(n^2\log^2 n\big)$ time, and Woodruff [33] gave an algorithm to construct 6-spanners with $O\big(n^{4/3}\log^3 n\big)$ edges in $O\big(n^2\log^2 n\big)$ time. The construction of Woodruff is furthermore randomized Monte Carlo. These results are summarized in Table 1.

These improvements to the running time fit into the following paradigm. For a fixed k the authors find algorithms that produce spanners that are *almost* as small as the best known construction of k-spanners and have *near*-quadratic running time. We reverse this way of looking at the problem. We are now trying to find algorithms that yield k-spanners that are exactly as small as the best known constructions for any $k = O(1)$, i.e. $O(n^{4/3})$, and at the same time we want the algorithm to run as fast as possible. All known algorithms for creating $O(1)$-spanners that have close to optimal size run in time $\Omega(n^2)$. [1] So a natural question is to ask if there exists a $k = O(1)$ and an algorithm that constructs a k-spanner with $O\big(n^{4/3}\big)$ edges in $O\big(n^2\big)$ time. In fact Sommer [28] mentioned at his talk at ICALP 2016 that the main obstacle towards getting a better running time for constructing the distance oracle he presented is the lack of such an algorithm. In his case the distortion $k = O(1)$ is only factored into the running time and not the distortion of oracle. Therefore, it does not matter what k is as long as it is constant.

We show that it possible to attain this goal by giving an algorithm that constructs 8-spanners deterministically with $O\big(n^{4/3}\big)$ edges in $O\big(n^2\big)$ time. Comparing this with the algorithm by Woodruff [33] this gets rid of the $\log^3 n$ factor on the number of edges and a factor of $\log^2 n$ in the running time. Furthermore, the algorithm is deterministic and not randomized Monte Carlo. The price of these improvements is that the distortion is larger than 6. We note that there are no lower bounds ruling out the possibility of a 4-spanner with $O\big(n^{4/3}\big)$ edges. For the application to the distance oracle by Sommer [28], the distortion is unimportant as long as it is constant. We also show how to construct 2-spanners with $O\big(n^{3/2}\big)$ edges in $O\big(n^2\big)$ time. For a comparison to previous work see Table 1.

Related work. Elkin and Peleg [19] showed that[2] any graph on n nodes has a 2-spanner with $O(n^{3/2})$ edges, Chechik [15] showed that it has a 4-spanner with $O\big(n^{7/5}\log^{1/5} n\big)$ edges, and Baswana et al [9] showed that it has a 6-spanner with $O(n^{4/3})$ edges. These results are

---

[1] For instance the algorithm by Baswana et al [9] gives a 6-spanner with $O\big(n^{4/3}\big)$ edges and is therefore only interesting when $m = \Omega\big(n^{4/3}\big)$, in which case the running time is $\Theta\big(n^{2/3}m\big) = \Omega\big(n^2\big)$.

[2] Aingworth et al [5] earlier showed the same result up to logarithmic factors on the size of the spanner.

■ **Table 2** For a given k an upper bound of $f(n)$ is a proof that any graph on n nodes has a k-spanner with no more than $f(n)$ edges. A lower bound of $g(n)$ is a proof that there exists a graph on n nodes for which any k-spanner must have at least $g(n)$ edges.

k	Upper Bound	Lower Bound	Reference
2 & 3	$O\big(n^{3/2}\big)$	$\Omega\big(n^{3/2}\big)$	[19]/[31]
4 & 5	$O\big(n^{7/5}\log^{1/5}n\big)$	$\Omega\big(n^{4/3}\big)$	[15]/[11]
≥ 6	$O\big(n^{4/3}\big)$	$n^{4/3-o(1)}$	[9]/[1]

complemented by a negative result of Abboud and Bodwin [1]. A consequence of their result is that for any $k = O(1)$ there exists a graph on n nodes such that any k-spanner of this graph has at least $n^{4/3-o(1)}$ edges.

Another negative result comes from Erdős's girth conjecture [20]. It states that for any constant k there exists graphs with n nodes and $\Omega\big(n^{1+1/k}\big)$ edges where the girth is $2k+2$. This conjecture has been proved for $k = 2, 3, 5$ [31, 11]. In particular if the conjecture is true this implies that there exists graphs for which any $(2k-1)$-spanner must have at least $\Omega\big(n^{1+1/k}\big)$ edges. Woodruff [32] proved that whether the conjecture is true or not, there exists a graph on n nodes such that any $(2k-1)$-spanner of the graph has at least $\Omega\big(k^{-1}n^{1+1/k}\big)$ edges.

There are also upper and lower bounds when we allow the distortion k to depend on n, see [14, 13, 15, 22]. In this paper, however, we are only interested in the case where $k = O(1)$. The upper and lower bounds for $k = O(1)$ are summarized in Table 2.

Techniques. Previous algorithms that construct k-spanners in $\tilde{O}\big(n^2\big)$ time all relied on constructing a hitting set for some set of neighbourhoods. In [18] this is done deterministically via a dominating set algorithm, and in [33] this is done via sampling. This approach will inherently come with the cost of a poly $\log n$ factor. Furthermore, in the construction of 6-spanners by Woodruff [33] the number of neighbourhoods that need to be hit is so large that it seems impossible with current techniques to modify the algorithm to be Las Vegas. To avoid this we instead use a clustering approach described in Section 2. The algorithm in Theorem 9 is obtained using this clustering and a careful modification of the path-buying algorithm of [9].

Approximate Distance Oracles and All Pairs Almost Shortest Paths. Given an undirected an unweighted graph G an (α, β)-approximate distance oracle for G is a data structure that supports the following query. Given two nodes u, v it can compute a distance estimate $\tilde{d}$ that satisfies $d \leq \tilde{d} \leq \alpha d + \beta$ where d is the distance between u and v in G. For work on approximate distance oracles see e.g. [2, 3, 4, 6, 7, 8, 10, 12, 16, 17, 23, 24, 26, 27, 29, 30, 34]. Sommer [28] gave a randomized Monte Carlo $(2, 1)$-distance oracle that can be constructed in $O\big(n^2 \operatorname{poly} \log n\big)$ time, has size $O\big(n^{5/3} \operatorname{poly} \log n\big)$ and can answer queries in $O(1)$ time. We improve the construction time and the size to $O\big(n^2\big)$ and $O\big(n^{5/3}\big)$ respectively, and our construction is randomized Las Vegas. As a corollary we can compute an estimate $\tilde{d}(u, v)$ for all pairs of nodes in G satisfying $d_G(u, v) \leq \tilde{d}(u, v) \leq 2d_G(u, v) + 1$ in time $O\big(n^2\big)$. This improves upon the $O\big(n^2 \log n\big)$ algorithm by Baswana and Kavitha [8].

Preliminaries. For a graph G and two nodes u, v we denote the distance from u to v in G by $d_G(u, v)$. All graphs considered in this paper are unweighted, and unless otherwise specified they are undirected as well. For an undirected graph G and a node u the neighbourhood of u is the set of nodes adjacent to u and is denoted by $\Gamma_G(u)$.

Overview. In Section 2 we introduce the clustering we use when constructing the spanners. In Section 3 we show how to create an 8-spanner with $O(n^{4/3})$ edges in $O(n^2)$ time and thereby prove Theorem 9. In Section 4 we provide the details on how to give an improved $(2, 1)$-distance oracle.

2   Clustering

Our construction of additive spanners uses clustering techniques, and we present our clustering framework below. Let $G = (V, E)$ be a graph with n vertices and m edges. We let t be a parameter that can depend on our needs. For a sequence $u_1, \ldots, u_\ell$ of nodes we define the clusters $C_i, i \in \{1, \ldots, \ell\}$ by

$$C_i = (\Gamma_G(u_i) \cup \{u_i\}) \setminus (C_1 \cup \ldots \cup C_{i-1}) .$$

Furthermore we also define graphs $G_0, G_1, \ldots, G_\ell$ in the following way. We let $G_0 = G$, and for $i > 0$ we let G_i be the subgraph of G defined in the following way. The nodes of G_i are the same as the nodes of G. An edge (u, v) from G is contained in G_i unless both endpoints u and v are are contained in $C_1 \cup \ldots \cup C_i$. From each node u_i we let T_i be a BFS tree in G_{i-1} rooted at u_i.

▶ **Definition 1.** A sequence $u_1, \ldots, u_\ell$ is called a *t-clustering* if the following requirements are satisfied.
- The node u_i maximizes the size of $(\Gamma_G(u_i) \cup \{u_i\}) \setminus (C_1 \cup \ldots \cup C_{i-1})$.
- Every cluster C_i contains at least t nodes.
- For every node v we have $|(\Gamma_G(v) \cup \{v\}) \setminus (C_1 \cup \ldots \cup C_\ell)| < t$.

We say that a node v is *clustered* if $v \in C_1 \cup \ldots \cup C_\ell$ and *unclustered* otherwise. We note that since every cluster C_i contains at least t nodes and the clusters are disjoint we have $\ell \leq \frac{n}{t}$.

▶ **Lemma 2.** *Let $u_1, \ldots, u_\ell$ be a t-clustering of a graph $G = (V, E)$. For every $i = 1, 2, \ldots, \ell$ the number of edges in G_{i-1} is at most $n |C_i|$. The number of edges in G_ℓ is less than nt.*

Proof. The set of edges in G_{i-1} can be written as

$$\{\{u, v\} \mid u \in V, v \in (\Gamma_G(u)) \setminus (C_1 \cup \ldots \cup C_{i-1})\} .$$

Therefore, the number of edges in G_{i-1} is bounded by

$$\sum_{v \in V} |(\Gamma_G(v)) \setminus (C_1 \cup \ldots \cup C_{i-1})| . \tag{1}$$

Since every term in (1) is bounded by $|C_i|$, we conclude that the number of edges in G_i is at most $n |C_i|$.

In the same manner we see that the number of edges in G_ℓ is bounded by the sum $\sum_{v \in V} |(\Gamma_G(v)) \setminus (C_1 \cup \ldots \cup C_\ell)|$, which is clearly less than nt. ◀

▶ **Lemma 3.** *Let $u_1, \ldots, u_\ell$ be a t-clustering of $G = (V, E)$ and let $u, v \in V$ be a pair of nodes. Assume that some shortest path from u to v in G is not contained in G_ℓ from Lemma 2. Then there exists an index $i \in \{1, 2, \ldots, \ell\}$ such that*

$$d_{T_i}(u_i, u) + d_{T_i}(u_i, v) \leq d_G(u, v) + 2 .$$

Proof. Consider a shortest path p from u to v that is not contained in G_ℓ and let w be a clustered node on p such that $w \in C_i$. We choose w such that i is smallest possible. By choosing i smallest possible p is contained in G_{i-1}. Furthermore since the distance from w to u_i is at most 1 we see that

$$d_{G_{i-1}}(u_i, u) + d_{G_{i-1}}(u_i, v) \leq d_{G_{i-1}}(w, u) + d_{G_{i-1}}(w, v) + 2 = d_G(u, v) + 2 \,.$$

Since T_i is a is shortest path tree in G_{i-1} the conclusion follows. ◀

▶ **Lemma 4.** *Given a graph G and a parameter $t > 0$ we can construct a t-clustering $u_1, \ldots, u_\ell$, the corresponding BFS trees $T_1, \ldots, T_\ell$ and G_ℓ in $O(n^2)$ time.*

Proof. The algorithm will work by finding the nodes $u_1, \ldots, u_\ell$ consecutively, i.e. first u_1, then u_2 and so on. The algorithm will maintain a graph G'. In the beginning of the algorithm we have $G' = G_0$, and after we add u_i we will alter G' such that $G' = G_i$. The total cost of altering all G' will be $O(m) = O(n^2)$.

We find u_i by looking at all nodes in $G' = G_{i-1}$ and count the number of neighbours not in $C_1 \cup \ldots \cup C_{i-1}$. Since G_{i-1} has at most $n|C_i|$ edges this takes $O(n|C_i|)$ time. Then the algorithm finds a BFS tree from u_i in G_{i-1} in $O(n|C_i|)$ time. Hence the total time used by the algorithm is:

$$O\left(m + \sum_{i=1}^{\ell} n|C_i|\right) = O(n^2) \,.$$

◀

3 Constructing $O(1)$-Spanners

In this section we present our construction of an 8-spanner with $O(n^{4/3})$ edges in $O(n^2)$ time. As a warmup we show how we can use the clustering from Section 2 to give a 2-spanner with $O(n^{3/2})$ edges in $O(n^2)$ time.

▶ **Theorem 5.** *There exists an algorithm that given a graph G with n nodes constructs a 2-spanner of G with $\leq 2n^{3/2}$ edges in $O(n^2)$ time.*

Proof. Let $t = \sqrt{n}$ and construct a t-clustering $u_1, \ldots, u_\ell$ with Lemma 4. Let $H = T_1 \cup \ldots \cup T_\ell \cup G_\ell$. The number of edges in H is at most $n\ell + nt \leq 2n\sqrt{n}$ by Lemma 2 and the fact that $\ell \leq \frac{n}{t}$.

Now we just need to prove that H is a 2-spanner. Let u, v be arbitrary nodes and let p be a shortest path from u to v in G. We wish to prove that

$$d_H(u, v) \leq d_G(u, v) + 2 \,. \tag{2}$$

If p is contained in G_ℓ then (2) is obviously true. Otherwise there exists an index i such that $d_{T_i}(u, v) \leq d_G(u, v) + 2$ by Lemma 3, and (2) is true since $T_i \subset H$. ◀

Next we turn to showing how to create an 8-spanner H with $O(n^{4/3})$ edges in $O(n^2)$ time. The idea is the following. We start by creating a t-clustering $u_1, \ldots, u_\ell$ with $t = n^{1/3}$ and $\ell \leq n^{2/3}$. Using the BFS trees $T_1, \ldots, T_\ell$ along with Lemma 3 we can then get an additive 2-approximation of $d_G(u_i, u_j)$ for all pairs of indices i, j, which we will call $\delta_{i,j}$. The calculation of the BFS trees in $O(n^2)$ time relies on an idea similar to one in [5]. The BFS trees also gives us a path from u_i to u_j that is at most 2 longer than the shortest path. If we add all these shortest paths to our spanner along with G_ℓ and the neighbours in C_i of each

u_i we will get a 6-spanner. Unfortunately, adding a path could require adding up to $\Omega(\ell)$ edges, and since there are ℓ^2 pairs we can only guarantee that the spanner has $O(\ell^3)$ edges, which is $O(n^2)$ if $\ell \approx n^{2/3}$. (We only need to add edges on the path that are not already in G_ℓ) Instead we use an argument similar to the path-buying argument from [9] and the construction from [21]. We add the path from u_i to u_j unless we can guarantee that there is an additive 2-approximation of this path in the spanner already. We do this by maintaining an upper bound $\Delta_{i,j}$ on the distance from u_i to u_j in the spanner H. We then argue that if we add a path with k edges not already in the spanner, then there are $\Omega(k)$ pairs $u_{i'}, u_{j'}$ for which the upper bound $\Delta_{i',j'}$ is improved. Then, this will imply that at most $O(\ell^2)$ edges are added giving an upper bound of $O(n^{4/3})$ on the number of edges in H.

After this informal discussion of the construction we turn to the details. The algorithm is given a graph $G = (V, E)$ with n nodes and m edges, and will return a spanner $H = (V, F)$. Initially $F = \emptyset$ and we will add edges to H so that H becomes a 8-spanner of G. The algorithm starts by creating a t-clustering $u_1, \ldots, u_\ell$ with $t = n^{1/3}$ using Lemma 4 in $O(n^2)$ time. Since $\ell \leq \frac{n}{t}$ we have $\ell \leq n^{2/3}$. Then we add edges from u_i to all nodes in $C_i \setminus \{u_i\}$ to H for all $i \in \{1, 2, \ldots, \ell\}$. We add at most n edges this way. Then we add all edges from G_ℓ to H. This adds at most $nt = n^{4/3}$ edges to H.

We give each node $u \in V$ a color $c(u) \in \{0, 1, 2, \ldots, \ell\}$. If u is unclustered then u has color $c(u) = 0$. Otherwise $c(u) = i$ where i is the unique index such that $u \in C_i$. For each pair of indices $i, j \in \{1, 2, \ldots, \ell\}$ we define $\delta_{i,j}$ by:

$$\delta_{i,j} = \min_{k \in \{1,2,\ldots,\ell\}} \{d_{T_k}(u_k, u_i) + d_{T_k}(u_k, u_j)\} . \tag{3}$$

We first note that for a choice of i, j we can calculate the right hand side of (3) in $O(\ell)$ time since we are taking the minimum over ℓ different values. So in $O(\ell^3)$ time the algorithm calculates $\delta_{i,j}$ for all pairs of indices i, j. Since $\ell \leq n^{2/3}$ this is within the $O(n^2)$ time bound. As a consequence of Lemma 3 we get that $\delta_{i,j}$ is a good approximation of $d_G(u_i, u_j)$, more precisely:

$$d_G(u_i, u_j) \leq \delta_{i,j} \leq d_G(u_i, u_j) + 2 . \tag{4}$$

We now define T_i' to be the tree obtained from T_i by contracting each edge in G_ℓ. Since an edge is contained in G_ℓ iff at least one of its endpoints is unclustered we can construct T_i' from T_i in $O(n)$ time. The algorithm does so for all $i \in \{1, 2, \ldots, \ell\}$ in $O(n\ell) = O(n^{5/3})$ time. We note that the shortest path between two nodes u, v in T_i' contains exactly the edges on the shortest path between u, v in T_i excluding the edges that are contained in G_ℓ.

The algorithm initializes $\Delta_{i,j} = \infty$ for all pairs of indices i, j with $i \neq j$ and let $\Delta_{i,i} = 0$ for all i. We will maintain that $\Delta_{i,j}$ is an upper bound on $d_H(u_i, u_j)$ throughout the algorithm. Now the algorithm goes through all pairs u_i, u_j and adds a path of length at most 2 longer than a shortest path between the nodes if needed. Specifically, we do the following:

Let L be an upper bound on the number of nodes of the path p from u_i to u_j in T_k' on line 6. Then Algorithm 1 can implemented in $O(\ell^3 + \ell^2 L)$ time. Hence we just need to prove that $L = O(\ell)$ in order to conclude that it can be implemented in $O(\ell^3) = O(n^2)$ time. This follows from the fact that p is an almost shortest path and the following reasoning. If p contained $> C\ell$ nodes for some sufficiently large constant C it would contain more than C nodes of the same color. Since nodes of the same color have distance at most 2 in G this would imply that there was a much shorter path from u to v in G contradicting (4) if C was chosen large enough. The details with $C = 5$ are given in the following lemma:

▶ **Lemma 6.** *The path p contains no nodes of color* 0, *and at most* 5 *nodes of each color* $\neq 0$.

Algorithm 1

```
1   For each pair of indices  i, j ∈ {1, 2, ..., ℓ}:
2       For all  k ∈ {1, 2, ..., ℓ}:
3           Set  Δ_{i,j} := min {Δ_{i,j}, Δ_{i,k} + Δ_{k,j}} .
4       If  Δ_{i,j} > δ_{i,j} + 2  do:
5           Find a  k ∈ {1, 2, ..., ℓ}  such that  d_{T_k}(u_k, u_i) + d_{T_k}(u_k, u_j) = δ_{i,j} .
6           Find the path  p  from  u_i  to  u_j  in  T'_k .
7           Add all edges from  p  to  H .
8           Write  p = (w_0, w_1, w_2, ..., w_{s-1}) .
9           For all  x ∈ {0, 1, 2, ..., s - 1}:
10              Set  y := d_{T_k}(u_i, w_x) .
11              Set  Δ_{i,c(w_x)} := min {Δ_{i,c(w_x)}, y + 1}
12              Set  Δ_{c(w_x),j} := min {Δ_{c(w_x),j}, (δ_{i,j} - y) + 1}
```

Proof. Obviously p does not contain a node with color 0, since all its incident edges would be contained in G_ℓ and hence not in T'_k. Now assume for the sake of contradiction that p contains 6 nodes of some color $r \neq 0$. When traversing p from u_i to u_j let α and β be the first and the last node of color r respectively. The distance from α to β when following p must be at least 5 by assumption. On the other hand α and β have distance at most 2 in G. So there exists a path in G from u_i to u_j that is at least 3 edges shorter than p. This contradicts (4). Hence the assumption was wrong and p contains at most 5 nodes of each color $\neq 0$. ◀

Since there are ℓ different colors $\neq 0$ the path p contains at most 5ℓ nodes and the running time of Algorithm 1 is $O(n^2)$. So now we just need to prove that H is an 8-spanner and that H has at most $O(n^{4/3})$ edges. We start by proving that H is an 8-spanner. Here we will utilize that the $\Delta_{i,j}$ is an upper bound on the distance from u_i to u_j in H. Furthermore, Algorithm 1 guarantees that $\Delta_{i,j} \leq \delta_{i,j} + 2$. Together with (4) this gives that

$$d_H(u_i, u_j) \leq d_G(u_i, u_j) + 4 . \tag{5}$$

▶ **Lemma 7.** *The subgraph H of G is an additive 8-spanner of G.*

Proof. Assume for the sake of contradiction that H is not an additive 8-spanner and let u, v be a pair of nodes with shortest possible distance in G such that:

$$d_H(u, v) > d_G(u, v) + 8 . \tag{6}$$

Say that $d_G(u, v) = D$ and let $p = (w_0, w_1, ..., w_D)$ be a shortest path from u to v in G where $w_0 = u$ and $w_D = v$. Since the pair (u, v) has the smallest possible distance in G such that (6) holds and $d_G(w_1, v) = D - 1$ we have $d_H(w_1, v) \leq (D - 1) + 8$. In particular the edge (u, w_1) is not in H as it would contradict (6). Hence u cannot be unclustered, as all the edges incident to an unclustered node is contained in G_ℓ and therefore H. With the same reasoning we conclude that v is clustered. Let the colors of u and v be i and j respectively. The distances from u and v to u_i and u_j respectively are at most 1. Combining this insight with (5) we get:

$$d_H(u, v) \leq d_H(u_i, u_j) + 2 \leq d_G(u_i, u_j) + 6 \leq d_G(u, v) + 8 .$$

But this contradicts the assumption (6). Hence the assumption was wrong and H is an additive 8-spanner of G. ◀

Lastly, we need to prove that H contains no more than $O(n^{4/3})$ edges. Informally, we argue the following way. Whenever the $s-1$ edges of p are added to H on line 7 of Algorithm 1 there are $\Omega(s)$ different colors on p. For each color r on p we then argue that either $\Delta_{i,r}$ or $\Delta_{r,j}$ are made smaller on line 11 or 12 of Algorithm 1. Lastly, we argue that $\Delta_{i,j}$ can only be updated $O(1)$ times, and since there are $\ell^2 \leq n^{4/3}$ variables $\Delta_{i,j}$ this implies that Algorithm 1 only adds $O(n^{4/3})$ edges to H. This intuition is formalized in Lemma 8 bellow:

▶ **Lemma 8.** *Algorithm 1 adds no more than* $25\ell^2$ *edges to* H.

Proof. Say that the algorithm adds the edges from the path $p = (w_0, w_1, \ldots, w_{s-1})$ on line 7 of Algorithm 1 where $w_0 = u_i, w_{s-1} = u_j$. First we note that since $d_G(u_i, u_j) \geq \delta_{i,j} - 2$ by (4) we have that $d_G(u_i, w_x) \geq y - 2$ for every $x \in \{0, 1, \ldots, s-1\}$, where we consider y to be a function of x defined by $y = d_{T_k}(u_i, w_x)$ as on line 10. Now fix x and let $r = c(w_x)$. Then there is an edge between w_x and u_r and therefore $d_G(u_i, u_r) \geq y - 3$, i.e. $y + 1 \leq d_G(u_i, u_r) + 4$. So if Algorithm 1 decreases $\Delta_{i,r}$ on line 11 we have $\Delta_{i,r} \leq d_G(u_i, u_r) + 4$ after it is decreased. Since $\Delta_{i,r}$ is an upper bound on $d_H(u_i, u_r)$ and therefore also an upper bound on $d_G(u_i, u_r)$ we see that $\Delta_{i,r}$ can be decreased at most 5 times for each choice of i, r. By symmetry we see that we can also decrease $\Delta_{r,j}$ on line 12 at most 5 times. Since there are ℓ^2 pairs of indices the algorithm can change the values of $\Delta_{i,r}$ or $\Delta_{r,j}$ on line 11 and 12 of Algorithm 1 at most $5\ell^2$ times.

Let r be a color on p. After the execution of lines 9-12 we have

$$\Delta_{i,r} + \Delta_{r,j} \leq \delta_{i,j} + 2 \, .$$

Due to the execution of lines 2 and 3 this was not the case before. Hence either $\Delta_{i,r}$ or $\Delta_{r,j}$ were updated. By Lemma 6 there are at least $\frac{s}{5}$ colors on p, so if the algorithm adds A edges in total it makes at least $\frac{A}{5}$ updates of upper bounds $\Delta_{i,r}$ or $\Delta_{r,j}$. Since there can be at most $5\ell^2$ such updates we conclude that $\frac{A}{5} \leq 5\ell^2$ and that Algorithm 1 adds no more than $25\ell^2$ edges. ◀

To summarize, the algorithm presented in this section runs in $O(n^2)$ time and gives an additive 8-spanner with no more than $26n^{4/3} + n = O(n^{4/3})$ edges. We have made no attempt to optimize the constant in the O-notation. Hence we get:

▶ **Theorem 9.** *There exists an algorithm that given a graph* G *with* n *nodes constructs an* 8-*spanner of* G *with* $O(n^{4/3})$ *edges in* $O(n^2)$ *time.*

4   Distance Oracles

In the following we show how to modify the construction by Sommer [28] to obtain a $(2, 1)$-distance oracle of size $O(n^{5/3})$ that can be constructed in expected $O(n^2)$ time.

Let G be a given graph, and H an 8-spanner of G constructed by Theorem 9. H is constructed in $O(n^2)$ time and has $O(n^{4/3})$ edges. During the construction we use only $O(n^{5/3})$ space.

Let $u_1, u_2, \ldots, u_\ell$ be a $n^{1/3}$-clustering of G. Using Lemma 4 we obtain $T_1, \ldots, T_\ell$ and G_ℓ in $O(n^2)$ time. For each node v we define four *portals* $p_1(v), p_2(v), p_3(v), p_4(v)$. We define $p_1(v) = u_i$, where u_i is chosen such that the distance between v and u_i in T_i is minimized. In case of ties we choose the node u_i with the lowest index i. The node $p_{j+1}(v)$ for $j = 1, 2, 3$ is chosen depending on $p_j(v)$. If $p_j(v) = u_1$ we let $p_{j+1}(v) = u_1$. Otherwise $p_j(v) = u_i$ for some index i. We let $p_{j+1}(v) = u_{i'}$ where $u_{i'}$ is chosen among $u_1, u_2, \ldots, u_{i-1}$ such that the

distance between $u_{i'}$ and v in $T_{i'}$ is minimized. In case of ties we choose the node $u_{i'}$ with the lowest index i'. The portals for all nodes can be found in $O\left(n^{5/3}\right)$ time.

We will use the following lemma by Pǎtraşcu and Roditty [23] to construct a $(2, 1)$-distance oracle for G_ℓ, that uses space $O\left(n^{5/3}\right)$.

▶ **Lemma 10** ([23]). *For any unweighted, undirected graph, there exists a distance oracle of size $O\left(n^{5/3}\right)$ that, given any nodes u and v at distance d, returns a distance of at most $2d + 1$ in constant time. The distance oracle can be constructed in expected time $O\left(mn^{2/3}\right)$.*

In the proof in [23] they only claim a running time of $O\left(mn^{2/3} + n^{7/3}\right)$, however, this can be fixed to give the correct running time of $O\left(mn^{2/3}\right)$ [25]. By [23, Claim 9] it is easy to see how to get a running time of $O\left(mn^{2/3} + n^2\right)$ which suffice for our purposes.

We are now ready to define the distance oracle. For each $i = 1, 2, \ldots, \ell$ we store the distances $d_{T_i}(u_i, v)$ and $d_H(u_i, v)$ for all nodes v. The distances $d_H(u_i, v)$ can be calculated using a BFS in time $O\left(\ell n^{4/3}\right) = O\left(n^2\right)$. For each node v we store its portals $p_j(v), j = 1, 2, 3, 4$. We augment this distance oracle with the Pǎtraşcu-Roditty distance oracle from Lemma 10 for G_ℓ.

We now show how to use the distance oracle to obtain approximate distances for a query u, v. We let $\delta_{PR}(u, v)$ be the approximate distance in G_ℓ returned by the Pǎtraşcu-Roditty distance oracle. We define $\delta_j(u, v)$ in the following way. Let $p_j(u) = u_i$. Then $\delta_j(u, v) = d_{T_i}(u_i, u) + \min\{d_{T_i}(u_i, v), d_H(u_i, v)\}$. The distance returned by the distance oracle is the minimum of $\delta_{PR}(u, v)$, $\delta_j(u, v)$ and $\delta_j(v, u)$ for $j = 1, 2, 3, 4$.

We will now argue that if the the distance between u and v is d, then the distance oracle returns a distance between d and $2d + 1$. The distance returned is obviously at least d, so we just need to show that it is at most $2d + 1$. Consider a shortest path between u and v in G. If there is at most one node on the shortest path which is incident to a node u_i in the clustering then the shortest path is contained in G_ℓ, and therefore:

$$\delta_{PR}(u, v) \le 2d_{G_\ell}(u, v) + 1 = 2d + 1\,.$$

So assume that there exists an edge on the shortest path not in G_ℓ. Let i be the smallest index such that there is an edge (z, t) on the shortest path with $z, t \in C_1 \cup \ldots \cup C_i$. Say that z is closer to u than to v in G. Assume that $z \in C_i$ and $t \in C_{i'}$ for some index $i' \le i$ (the case where $z \in C_{i'}$ and $t \in C_i$ is handled symmetrically). Since the shortest path is contained in G_{i-1} and $G_{i'-1}$ we have that

$$d_{T_i}(u_i, u) + d_{T_{i'}}(u_{i'}, v) \le (d_G(u, z) + 1) + (d_G(t, v) + 1) = d + 1\,,$$

and therefore:

$$\min\{d_{T_i}(u_i, u), d_{T_i}(u_i, v)\} \le \frac{d + 1}{2}\,.$$

Assume that $d_{T_i}(u_i, u) \le \frac{d+1}{2}$. The other case is handled similarly. Say that $p_j(u) = u_{k_j}$ for $j = 1, 2, 3, 4$. First assume that $k_j > i$ for all $j = 1, 2, 3, 4$. Then we conclude that $d_{T_{k_1}}(p_1(u), u) \le d_{T_i}(u_i, u) - 4$. The distance returned by the distance oracle is at most

$$\begin{aligned}
\delta_1(u, v) &\le d_{T_{k_1}}(p_1(u), u) + d_H(p_1(u), v) \\
&\le d_{T_{k_1}}(p_1(u), u) + d_G(p_1(u), v) + 8 \\
&\le 2d_{T_{k_1}}(p_1(u), u) + d_G(u, v) + 8 \\
&\le 2(d_{T_i}(u_i, u) - 4) + d + 8 \le 2d + 1\,.
\end{aligned}$$

Now assume that $k_j \leq i$ for some $j \in \{1, 2, 3, 4\}$ and let j be the smallest index such that $k_j \leq i$. By definition we have that $d_{T_{k_j}}(p_j(u), u) \leq d_{T_i}(u_i, u)$. Furthermore the shortest path is contained in G_{k_j-1} and therefore $d_{T_{k_j}}(p_j(u), v) \leq d_{T_{k_j}}(p_j(u), u) + d_G(u, v)$. The distance returned is at most

$$
\begin{aligned}
\delta_j(u, v) &\leq d_{T_{k_j}}(p_j(u), u) + d_{T_{k_j}}(p_j(u), v) \\
&\leq 2d_{T_{k_j}}(p_j(u), u) + d \\
&\leq 2d_{T_i}(u_i, u) + d \leq 2d + 1 \, .
\end{aligned}
$$

We conclude that the distance returned by the distance oracle is always between d and $2d+1$. The result is summarized in Theorem 11.

▶ **Theorem 11.** *For any unweighted, undirected graph, there exists a distance oracle of size $O\big(n^{5/3}\big)$ that, given any nodes u and v at distance d, returns a distance of at most $2d + 1$ in constant time. The distance oracle can be constructed in expected time $O\big(n^2\big)$.*

── **References** ──────────────

1   Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. In *Proc. 48th ACM Symposium on Theory of Computing (STOC)*, pages 351–361, 2016.

2   Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In *Distributed Computing – 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, pages 404–415, 2011. `doi:10.1007/978-3-642-24100-0_39`.

3   Rachit Agarwal. The space-stretch-time tradeoff in distance oracles. In *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 49–60, 2014. `doi:10.1007/978-3-662-44777-2_5`.

4   Rachit Agarwal and Philip Brighten Godfrey. Brief announcement: a simple stretch 2 distance oracle. In *ACM Symposium on Principles of Distributed Computing, PODC'13, Montreal, QC, Canada, July 22-24, 2013*, pages 110–112, 2013. `doi:10.1145/2484239.2484277`.

5   Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. See also SODA'96.

6   Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 609–621, 2008. `doi:10.1007/978-3-540-70575-8_50`.

7   Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in I time. *Theor. Comput. Sci.*, 410(1):84–93, 2009. `doi:10.1016/j.tcs.2008.10.018`.

8   Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010.

9   Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Trans. Algorithms*, 7(1):5, 2010. See also SODA'05.

10 Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $o(n^2)$ time. *ACM Transactions on Algorithms (TALG)*, 2(4):557–577, 2006.

11 Clark T. Benson. Minimal regular graphs of girth eight and twelve. *Canad. J. Math*, 18(1):94, 1966.

12 Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *Workshop on Algorithms and Data Structures*, pages 541–552. Springer, 2007.

13 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 855–872, 2016. `doi:10.1137/1.9781611974331.ch61`.

14 Gregory Bodwin and Virginia Vassilevska Williams. Very sparse additive spanners and emulators. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 377–382, 2015. `doi:10.1145/2688073.2688103`.

15 Shiri Chechik. New additive spanners. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 498–512, 2013. `doi:10.1137/1.9781611973105.36`.

16 Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 654–663. ACM, 2014.

17 Shiri Chechik. Approximate distance oracles with improved bounds. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 1–10. ACM, 2015.

18 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. See also FOCS'96.

19 Michael Elkin and David Peleg. $(1 + \varepsilon, \beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004. See also STOC'01.

20 Paul Erdős. Extremal problems in graph theory. In *"Theory of Graphs and its Applications," Proc. Sympos. Smolenice*. Citeseer, 1964.

21 Mathias Bæk Tejs Knudsen. Additive spanners: A simple construction. In *Proc. 14th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 277–281, 2014.

22 Seth Pettie. Low distortion spanners. *ACM Trans. Algorithms*, 6(1):7:1–7:22, 2009. `doi:10.1145/1644015.1644022`.

23 Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the thorup–zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014.

24 Mihai Pătraşcu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 738–747. IEEE, 2012.

25 Liam Roditty. personal communication.

26 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 2005.

27 Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)*, 46(4):45, 2014.

28 Christian Sommer. All-pairs approximate shortest paths and distance oracle preprocessing. In *LIPIcs – Leibniz International Proceedings in Informatics*, volume 55. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.

29 Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 703–712. IEEE, 2009.

30 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

31 Rephael Wenger. Extremal graphs with no C4's, C6's, or C10's. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991.

32 David P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 389–398, 2006.

33 David P. Woodruff. Additive spanners in nearly quadratic time. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 463–474, 2010.

34 Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 202–208. Society for Industrial and Applied Mathematics, 2012.