Simple Algorithms for Spanners in Weighted Graphs (2003; Baswana, Sen)

Surender Baswana, IIT Kanpur, www.cse.iitk.ac.in/∼sbaswana
Sandeep Sen, IIT Delhi, www.cse.iitd.ernet.in/∼ssen

SYNONYMS and INDEX TERMS: Graph algorithms, Randomized algorithms, Shortest path, Spanner.

1 PROBLEM DEFINITION

A spanner is a *sparse* subgraph of a given undirected graph that preserves approximate distance between each pair of vertices. More precisely, a t-spanner of a graph $G = (V, E)$ is a subgraph $(V, E_S), E_S \subseteq E$ such that, for any pair of vertices, their distance in the subgraph is at most t times their distance in the original graph, where t is called the *stretch factor*. The spanners were defined formally by Peleg and Schäffer [14] though the associated notion was used implicitly by Awerbuch [3] in the context of network synchronizers.

Computing t-spanner of smallest size for a given graph is a well motivated combinatorial problem with many applications. However, computing t-spanner of smallest size for a graph is NP-hard. In fact, for $t > 2$, it is NP-hard [10] even to approximate the smallest size of t-spanner of a graph with ratio $O(2^{(1-\mu)\ln n})$ for any $\mu > 0$. Having realized this fact, researchers have pursued another direction which is quite interesting and useful. Let S_G^t be the size of the sparsest t-spanner of a graph G, and let $\mathcal{S}_n^t$ be the maximum value of S_G^t over all possible graphs on n vertices. Does there exist a polynomial time algorithm which computes, for any weighted graph and parameter t, its t-spanner of size $O(\mathcal{S}_n^t)$? Such an algorithm would be the best one can hope for given the hardness of the original t-spanner problem. Naturally the question arises as to how large can $\mathcal{S}_n^t$ be ? A 43 years old girth lower bound conjecture by Erdös [12] implies that there are graphs on n vertices whose $2k$ as well as $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3$ and 5. Note that a $(2k-1)$-spanner is also a $2k$-spanner and the lower bound on the size is the same for both $2k$-spanner and $(2k-1)$-spanner. So the objective is to design an algorithm that, for any weighted graph on n vertices computes a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Needless to say, one would like to design the fastest algorithm for this problem, and the most ambitious aim would be to achieve the linear time complexity.

2 KEY RESULTS

The key results of this article are two very simple algorithms which compute a $(2k-1)$-spanner of a given weighted graph $G = (V, E)$. Let n and m denote respectively the number of vertices and edges of G. The first algorithm, due to Althöfer et al. [2], is based on a greedy strategy, and runs in $O(mn^{1+1/k})$ time. The second algorithm [6] is based on a very local approach and runs in expected $O(m)$ time. To start with, consider the following simple observation. Suppose there is a subset $E_S \subset E$ that ensures the following proposition for every edge $(x, y) \in E \backslash E_S$.

> $\mathcal{P}_t(x, y)$: the vertices x and y are connected in the subgraph (V, E_S) by a path consisting of at most t edges, and the weight of each edge on this path is not more than that of the edge (x, y).

It follows easily that the sub graph (V, E_S) will be a t-spanner of G. The two algorithms for computing $(2k-1)$-spanner eventually compute such set E_S based on two completely different approaches.

2.1 Algorithm I

This algorithm selects edges for spanner in a greedy fashion, and is similar to Kruskal's algorithm for computing a minimum spanning tree. The edges of the graph are processed in the increasing order of their weights. To begin with, the spanner $E_S = \emptyset$ and the algorithm adds edges to it gradually. The decision as to whether an edge, say (u, v), has to be added (or not) to E_S is made as follows:

If the distance between u and v in the subgraph induced by the current spanner edges E_S is more than $t \cdot weight(u, v)$, then add the edge (u, v) to E_S, otherwise discard the edge.

It follows that $\mathcal{P}_t(x, y)$ would hold for each edge of E missing in E_S, and so at the end, the subgraph (V, E_S) will be a t-spanner. A well known result in elementary graph theory states that a graph with more than $n^{1+1/k}$ edges must have a cycle of length at most $2k$. It follows from the above algorithm that the length of any cycle in the subgraph (V, E_S) has to be at least $t + 1$. Hence for $t = 2k - 1$, the number of edges in the subgraph (V, E_S) will be less than $n^{1+1/k}$. Thus the algorithm I described above computes a $(2k - 1)$-spanner of size $O(n^{1+1/k})$, which is indeed optimal based on the lower bound mentioned earlier.

A simple $O(mn^{1+1/k})$ implementation of algorithm I follows based on Dijkstra's algorithm. Cohen [9], and later Thorup and Zwick [18] designed algorithms for $(2k - 1)$-spanner with an improved running time of $O(kmn^{1+1/k})$. These algorithms relied on several calls to Dijkstra's single-source shortest-path algorithm for distance computation and therefore were far from achieving linear time. On the other hand, since a spanner must approximate all pairs distances in a graph, it appears difficult to compute a spanner by avoiding explicit distance information. Somewhat surprisingly, algorithm II, described in the following section, avoids any sort of distance computation and achieves expected linear time.

2.2 Algorithm II

This algorithm employs a novel clustering based on a very local approach, and establishes the following result for the spanner problem.

> Given a weighted graph $G = (V, E)$, and an integer $k > 1$, a spanner of $(2k - 1)$-stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time.

The algorithm executes in $O(k)$ rounds, and in each round it essentially explores adjacency list of each vertex to prune dispensable edges. As a testimony of its simplicity, we will present the entire algorithm for 3-spanner and its analysis in the following section. The algorithm can be easily adapted in other computational models (parallel, external memory, distributed) with nearly optimal performance (see [6] for more details).

2.2.1 Computing a 3-spanner in linear time

To meet the size constraint of a 3-spanner a vertex, on an average contributes $\sqrt{n}$ edges to the spanner. So the vertices with degree $O(\sqrt{n})$ are easy to handle since all their edges can be selected in the spanner. For vertices with higher degree a clustering (groupings) scheme is employed to tackle this problem which has its basis in *dominating sets*.

To begin with, there is a set of edges E' initialized to E, and empty spanner E_S. The algorithm processes the edges E', moves some of them to the spanner E_S and discards the remaining ones. It does so in the following two phases.

1. *Forming the clusters* :
   A sample $\mathcal{R} \subset V$ is chosen by picking each vertex independently with probability $\frac{1}{\sqrt{n}}$. The clusters will be formed around these sampled vertices. Initially the clusters are $\{\{u\} | u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ is called the *center* of its cluster. Each unsampled vertex $v \in V - \mathcal{R}$ is processed as follows.

   (a) If v is not adjacent to any sampled vertex, then every edge incident on v is moved to E_S.

(b) If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest [1] to v. The edge $(v, \mathcal{N}(v, \mathcal{R}))$ along with every edge that is incident on v with weight less than this edge is moved to E_S. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

As a last step of the first phase, all those edges (u, v) from E' where u and v are not sampled and belong to the same cluster are discarded.

Let V' be the set of vertices corresponding to the endpoints of the edges E' left after the first phase. It follows that each vertex from V' is either a sampled vertex or adjacent to some sampled vertex, and the step 1(b) has partitioned V' into disjoint clusters each centered around some sampled vertex. Also note that, as a consequence of the last step, each edge of the set E' is an inter-cluster edge. The graph (V', E'), and the corresponding clustering of V' is passed onto the following (second) phase.

2. *Joining vertices with their neighboring clusters* :
Each vertex v of graph (V', E') is processed as follows. Let $E'(v, c)$ be the edges from the set E' incident on v from a cluster c. For each cluster c neighboring to v, the least-weight edge from $E'(v, c)$ is moved to E_S and the remaining edges are discarded.

The number of edges added to the spanner E_S during the algorithm described above can be bounded as follows. Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly independently with probability $\frac{1}{\sqrt{n}}$. It thus follows from elementary probability that for each vertex $v \in V$, the expected number of incident edges with weight less than that of $(v, \mathcal{N}(v, \mathcal{R}))$ is at most $\sqrt{n}$. Thus the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is at most $\sqrt{n}$. The number of edges added to the spanner in the second phase is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected number of edges added to the spanner in the second phase is at most $n^{3/2}$. Hence the expected size of the spanner E_S at the end of the algorithm described above is at most $2n^{3/2}$. The algorithm is repeated if the size of the spanner exceeds $3n^{3/2}$. It follows using Markov's inequality that the expected number of such repetitions will be $O(1)$.

We now establish that E_S is a 3-spanner. Note that for every edge $(u, v) \notin E_S$, the vertices u, v belong to some cluster in the first phase. There are two cases now.

Case 1 : *(u and v belong to same cluster)*
Let u and v belong to the cluster centered at $x \in \mathcal{R}$. It follows from the first phase of the algorithm that there is a 2-edge path $u - x - v$ in the spanner with each edge not heavier than the edge (u, v). (This provides a justification for discarding all intra-cluster edges at the end of first phase).

Case 2 : *(u and v belong to different clusters)*
Clearly the edge (u, v) was removed from E' during phase 2, and suppose it was removed while processing the vertex u. Let v belong to the cluster centered at $x \in \mathcal{R}$

In the beginning of the second phase let $(u, v') \in E'$ be the least weight edge among all the edges incident on u from the vertices of the cluster centered at x. So it must be that $weight(u, v') \leq weight(u, v)$. The processing of vertex u during the second phase of our algorithm ensures that the edge (u, v') gets added to E_S. Hence there is a path $\Pi_{uv} = u - v' - x - v$ between u and v in the spanner E_S, and its weight can be bounded as $weight(\Pi_{uv}) = weight(u, v') + weight(v', x) + weight(x, v)$. Since (v', x) and (v, x) were chosen in the first phase, it follows that $weight(v', x) \leq weight(u, v')$ and $weight(x, v) \leq weight(u, v)$. It follows that the spanner (V, E_S) has stretch 3. Moreover both phases of the algorithm can be executed in $O(m)$ time using elementary data structures and bucket sorting.

The algorithm for computing a $(2k - 1)$-spanner executes k iterations where each iteration is similar to the first phase of the 3-spanner algorithm. For details and formal proofs, the reader may refer to [6].

[1] Ties can be broken arbitrarily. However, it helps conceptually to assume that all weights are distinct

2.3 Other related work

The notion of a spanner has been generalized in the past by many researchers.
Additive spanners : A t-spanner as defined above approximates pairwise distances with multiplicative error, and can be called a multiplicative spanner. In an analogous manner, one can define spanners that approximate pairwise distances with additive error. Such a spanner is called an additive spanner and the corresponding error is called *surplus*. Aingworth et al. [1] presented the first additive spanner of size $O(n^{3/2} \log n)$ with surplus 2. Baswana et al. [7] presented a construction of $O(n^{4/3})$ size additive spanner with surplus 6. It is a major open problem if there exists any sparser additive spanner.
(α, β)-*spanner* : Elkin and Peleg [11] introduced the notion of (α, β)-spanner for unweighted graphs, which can be viewed as a hybrid of multiplicative and additive spanners. An (α, β)-spanner is a subgraph such that the distance between any pair of vertices $u, v \in V$ in this subgraph is bounded by $\alpha\delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v in the original graph. Elkin and Peleg showed that an $(1 + \epsilon, \beta)$-spanner of size $O(\beta n^{1+\delta})$, for arbitrarily small $\epsilon, \delta > 0$, can be computed at the expense of sufficiently large surplus β. Recently Thorup and Zwick [19] introduced a spanner where the additive error is sublinear in terms of the *distance* being approximated.

Other interesting variants of spanner include *distance preserver* proposed by Bollobás et al. [8] and *Light-weight* spanner proposed by Awerbuch et al. [4]. A subgraph is said to be a d-preserver if it preserves exact distances for each pair of vertices which are separated by distance at least d. A light-weight spanner tries to minimize the number of edges as well as the total edge weight. A *lightness* parameter is defined for a subgraph as the ratio of total weight of all its edges and the weight of the minimum spanning tree of the graph. Awerbuch et al. [4] showed that for any weighted graph and integer $k > 1$, there exists a polynomially constructible $O(k)$-spanner with $O(k\rho n^{1+1/k})$ edges and $O(k\rho n^{1/k})$ lightness, where $\rho = \log(Diameter)$.

In addition to the above work on the generalization of spanners, a lot of work has also been done on computing spanners for special classes of graphs, e.g., chordal graphs, unweighted graphs, and Euclidean graphs. For chordal graphs, Peleg and Schäffer [14] designed an algorithm that computes a 2-spanner of size $O(n^{3/2})$, and a 3-spanner of size $O(n \log n)$. For unweighted graphs Halperin and Zwick [13] gave an $O(m)$ time algorithm for this problem. Salowe [17] presented an algorithm for computing a $(1+\epsilon)$-spanner of a d-dimensional complete Euclidean graph in $O(n \log n + \frac{n}{\epsilon^d})$ time. However, none of the algorithms for these special classes of graphs seem to extend to general weighted undirected graphs.

3 APPLICATIONS

Spanners are quite useful in various applications in the area of distributed systems and communication networks. In these applications, spanners appear as the underlying graph structure. In order to build compact routing tables [16], many existing routing schemes use the edges of a sparse spanner for routing messages. In distributed systems, spanners play an important role in designing *synchronizers*. Awerbuch [3], and Peleg and Ullman [15] showed that the quality of a spanner (in terms of stretch factor and the number of spanner edges) is very closely related to the time and communication complexity of any synchronizer for the network. The spanners have also been used implicitly in a number of algorithms for computing all pairs approximate shortest paths [5, 9, 18]. For a number of other applications, please refer to the papers [2, 3, 14, 16].

4 OPEN PROBLEMS

The running time as well as the size of the $(2k - 1)$-spanner computed by the algorithm described above are away from their respective worst case lower bounds by a factor of k. For any constant value of k, both these parameters are optimal. However, for the extreme value of k, that is, for $k = \log n$, there is deviation by a factor of $\log n$. Is it possible to get rid of this multiplicative factor of k from the running time of the algorithm and/or the size of the $(2k-1)$-spanner computed ? It seems that a more careful analysis coupled with advanced probabilistic tools might be useful in this direction.

5 EXPERIMENTAL RESULTS

None is reported.

6 DATA SETS

None is reported.

7 URL to CODE

None is reported.

8 CROSS REFERENCES

None is reported. Entry editors please feel free to add some.

9 RECOMMENDED READING

[1] 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.

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

[3] B. Awerbuch. Complexity of network synchronization. *Journal of Association of Computing Machinery*, 32(4):804–823, 1985.

[4] B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light weight spanners. *Tech. Report CS92-22, Weizmann Institute of Science*, 1992.

[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhod covers. *SIAM Journal on Computing*, 28:263–277, 1998.

[6] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures and Algorithms*, 30:532–563, 2007.

[7] S. Baswana, K. Telikepalli, K. Mehlhorn, and S. Pettie. New construction of (α, β)-spanners and purely additive spanners. In *Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 672–681, 2005.

[8] B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preserves and additive spanners. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 414–423, 2003.

[9] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1998.

[10] M. Elkin and D. Peleg. Strong inapproximability of the basic k-spanner problem. In *Proc. of 27th International Colloquim on Automata, Languages and Programming*, pages 636–648, 2000.

[11] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner construction for general graphs. *SIAM Journal of Computing*, 33:608–631, 2004.

[12] 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.

[13] S. Halperin and U. Zwick. Linear time deterministic algorithm for computing spanners for unweighted graphs. *unpublished manuscript*, 1996.

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

[15] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:740–747, 1989.

[16] D. Peleg and E. Upfal. A trade-off between space amd efficiency for routing tables. *Journal of Association of Computing Machinery*, 36(3):510–530, 1989.

[17] J. D. Salowe. Construction of multidimensional spanner graphs, with application to minimum spanning trees. In *ACM Symposium on Computational Geometry*, pages 256–261, 1991.

[18] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of Association of Computing Machinery*, 52:1–24, 2005.

[19] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 802–809, 2006.