Brief Announcement: A Centralized Local Algorithm for the Sparse Spanning Graph Problem*†

Christoph Lenzen[1] and Reut Levi[2]

1 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
clenzen@mpi-inf.mpg.de

2 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
rlevi@mpi-inf.mpg.de

Abstract

Constructing a sparse *spanning subgraph* is a fundamental primitive in graph theory. In this paper, we study this problem in the Centralized Local model, where the goal is to decide whether an edge is part of the spanning subgraph by examining only a small part of the input; yet, answers must be globally consistent and independent of prior queries.

Unfortunately, maximally sparse spanning subgraphs, i.e., spanning trees, cannot be constructed efficiently in this model. Therefore, we settle for a spanning subgraph containing at most $(1+\epsilon)n$ edges (where n is the number of vertices and ϵ is a given approximation/sparsity parameter). We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$,[1] where Δ is the maximum degree of the input graph. Our algorithm is the first to do so on arbitrary bounded degree graphs. Moreover, we achieve the additional property that our algorithm outputs a *spanner,* i.e., distances are approximately preserved. With high probability, for each deleted edge there is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops in the output that connects its endpoints.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases local, spanning graph, sparse

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.57

1 Introduction

When operating on very large graphs, it is often impractical or infeasible to (i) hold the entire graph in the local memory of a processing unit, (ii) run linear-time (or even slower) algorithms, or even (iii) have only a single processing unit perform computations sequentially. These constraints inspired the Centralized Local model [9], which essentially views the input as being stored in a (likely distributed) database that provides query access to external processing units. To minimize the coordination overhead of such a system, it is furthermore required that there is no shared memory or communication between the querying processes, except for shared randomness provided alongside the access to the input. The idea is now to run sublinear-time algorithms that extract useful global properties of the graph and/or to examine the input graph locally upon demand by applications.

* A full version of the paper is available at http://arxiv.org/abs/1703.05418.
† See [2] for the full version of this paper.
[1] $\tilde{O}$-notation hides polylogarithmic factors in n.

Studying graphs in this model leads to the need for query access to a variety of graph-theoretical structures like, e.g., independent or dominating sets. In such a case, it is crucial that locally evaluating whether a node participates in such a set is *consistent* with the same evaluation for other nodes. This is a non-trivial task, as local decisions can only be coordinated implicitly via the structure of the input (which is to be examined as little as possible) and the shared randomness. Nonetheless, this budding field brought forth a number of elegant algorithms solving, e.g., maximal independent set, hypergraph coloring, k-CNF, and approximate maximum matching (see survey [6] and references therein).

In this work, we consider another very basic graph structure: sparse spanning subgraphs. Here, the task is to select a subset of the edges of the (connected) input graph so that the output is still connected, but has only few edges. By "few" we mean that, for some input parameter $\varepsilon > 0$, the number of selected edges is at most $(1 + \varepsilon)n$, where n denotes the number of nodes. One may see this as a relaxed version of the problem of outputting a spanning tree of the graph, which can not be obtained in sub-linear number of queries.

▶ **Definition 1** ([5])**.** An algorithm $\mathcal{A}$ is a *Local Sparse Spanning Graph (LSSG) algorithm* if, given $n, \Delta \geq 1$, a parameter $\epsilon \geq 0$, and query access to the incidence list representation of a connected graph $G = (V, E)$ over n vertices and of degree at most Δ, it provides oracle access to a subgraph $G' = (V, E')$ of G such that: (1) G' is connected. (2) $|E'| \leq (1 + \epsilon) \cdot n$ with high probability (w.h.p.),[2] where E' is determined by G and the internal randomness of $\mathcal{A}$. By "providing oracle access to G'" we mean that on input $\{u, v\} \in E$, $\mathcal{A}$ returns whether $\{u, v\} \in E'$, and for any sequence of edges, $\mathcal{A}$ answers consistently w.r.t. the same G'.

We are interested in LLSG algorithms that, for each given edge, perform as few queries as possible to G. Observe that Item (2) implies that the answers of an LLSG algorithm to queries cannot depend on previously asked queries. We note that relaxing from requiring a tree as output makes it possible to ask for additional guarantees. Instead of merely preserving connectivity, it becomes possible to maintain *distances* up to small factors. Such subgraphs are known as (sparse, multiplicative) *spanners* [7, 8].

Our Contribution. We give the first non-trivial LSSG algorithm in the Centralized Local model that runs on arbitrary graphs. We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$ per edge, w.h.p. Moreover, we guarantee that for each edge that is not selected into the spanner, w.h.p. there is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops consisting of edges that are selected into the spanner; this is referred to as a *stretch* of $O(\log n \cdot (\Delta + \log n)/\epsilon)$.

For simplicity, assume for the moment that Δ and ε are constants. Our algorithm combines the following key ideas. We classify edges as expanding if there are sufficiently many (roughly $n^{1/3}$) nodes within $O(\log n)$ hops of their endpoints. For non-expanding edges, we can efficiently simulate a standard distributed spanner algorithm at small query complexity, as solutions of running time $O(\log n)$ are known (e.g. [1]).

On the node set induced by expanding edges, we can construct a partition into Voronoi cells with respect to roughly $n^{2/3}$ randomly sampled centers. The Voronoi cells are spanned by trees of depth $O(\log n)$, as expanding nodes have their closest center within $O(\log n)$ hops w.h.p. Finding the closest center has query complexity $\tilde{O}(n^{1/3})$.

We refine the partition into Voronoi cells further into *clusters* of $\tilde{O}(n^{1/3})$ nodes. We simply let a node be a singleton cluster if its subtree in the spanning tree of its cell contains more than $\tilde{O}(n^{1/3})$ nodes. This construction has query complexity $\tilde{O}(n^{2/3})$ for constructing a complete cluster, yet ensures that there are $\tilde{O}(n^{2/3})$ clusters in total due to the low depth of the trees; moreover, each cluster is completely contained in some Voronoi cell.

[2] That is, with probability at least $1 - 1/n^c$ for an arbitrary constant $c > 0$ that is chosen upfront.

It remains to select few edges to interconnect the Voronoi cells. This is the main challenge, for which the above properties of the partition are crucial. To keep the number of selected edges small in expectation, we mark a subset of expected size $\tilde{\Theta}(n^{1/3})$ of the clusters by marking each Voronoi cell (and thereby its constituent clusters) with probability $n^{-1/3}$. We then try to ensure that (i) clusters select an edge to each adjacent marked Voronoi cell and (ii) for each marked Voronoi cell adjacent to an adjacent cluster, they select one edge connecting to *some* cluster adjacent to this cell.

The main issue with the previous step is that we cannot afford to construct each adjacent cluster, preventing us from guaranteeing (ii). We circumvent this obstacle by identifying for adjacent clusters in which cell they are and keeping an edge for the purpose of (ii) if it satisfies a certain minimality requirement with respect to the *rank* of the cell used for symmetry breaking purposes. This way, we avoid construction of adjacent clusters, instead needing to determine the rank of their Voronoi cells only. This way, we maintain query complexity $\tilde{O}(n^{2/3})$. However, this now entails an inductive argument for ensuring connectivity, which also affects stretch. By choosing Voronoi cell ranks uniformly at random, we obtain a total bound of $O(\log^2 n)$ on the stretch of our scheme.

Related work. The problem of finding a sparse spanning subgraph in the Centralized Local model was first studied in [5], where the authors show a lower bound of $\Omega(\sqrt{n})$ queries for constant ϵ and Δ.They also present an upper bound with nearly tight query complexity for graphs that have very good expansion properties. However, for general (bounded degree) graphs their algorithm might query the entire graph for completing a single call to the oracle. They also provide an efficient algorithm for minor-free graphs that was later improved in [4]. A characterization of the query complexity of the problem in terms of expansion properties of the input graph was presented in [3].

References

1 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 652–669, 2017.

2 Christoph Lenzen and Reut Levi. A local algorithm for the sparse spanning graph problem. *CoRR*, abs/1703.05418, 2017. URL: `http://arxiv.org/abs/1703.05418`.

3 R. Levi, G. Moshkovitz, D. Ron, R. Rubinfeld, and A. Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, pages n/a–n/a, 2016. `doi:10.1002/rsa.20652`.

4 R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms*, 11(3):24:1–24:13, 2015.

5 R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. In *Proceedings of the Eighteenth International Workshop on Randomization and Computation (RANDOM)*, pages 826–842, 2014.

6 Reut Levi and Moti Medina. A (centralized) local guide. *Bulletin of EATCS*, 2(122), 2017.

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

8 D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:229–243, 1989.

9 R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of The Second Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.