CS166 Checkpoint

Aaron Effron, Hassan Fahmy, Magdy Saleh

May 2019

1 Overview

In Approximate Distance Oracle algorithms and data structures, there are four main dimensions along which we may optimize:

1. Stretch: The degree to which we overestimate distance. Stretch is defined as the max over all vertex pairs of $\frac{\text{distance reported by oracle}}{\text{true distance}}$
2. Preprocessing Time: The amount of work required to precompute approximate distances, after which the approximate distance (with a given stretch) between any two points in our graph can be reported in time $O(\text{query time})$
3. Query Time: The amount of time, after preprocessing is complete, required to return the approximate distance between two given vertices
4. Data Structure Space Requirement: The amount of space required to store the data structure that supports the above preprocessing and query time with given stretch

Unsurprisingly, there are tradeoffs involved in the above four dimensions, and we cannot simultaneously achieve optimal stretch, preprocessing time, query time, and data structure space requirement.

2 Thorup and Zwick, Approximate Distance Oracles

We can examine these tradeoffs in the context of the results from Thorup and Zwick's seminal paper, Approximate Distance Oracles. [1] Thorup and Zwick present, for a graph on n vertices with m edges a data structure of size $(kn^{1+1/k})$ which with expected $(kmn^{1/k})$ preprocessing, can return a $2k-1$ approximation to any distance query in time $O(k)$. Previous implementations using $(n^{1+1/k})$ space had a query time of $\omega(n^{1/k})$, and therefore this is an improvement on past work.

Examining these asymptotic complexities, we may observe the following:

- Lower stretch $\rightarrow$ higher preprocessing time. To get a better estimate, we must do more preprocessing.
- Lower stretch $\rightarrow$ lower query time. With a better estimate, we may query in less time
- Lower stretch $\rightarrow$ higher space. To get a better estimate, we need to store more information.

For $k = 1$, we retrieve the APSP solution (All Points Shortest Path), which returns the exact shortest path between all points in $O(n^2)$ space, $O(mn)$ preprocessing, and $O(1)$ query time. Optimal space and preprocessing times are achieved for $k = \lfloor \log(n) \rfloor$, which yields:

- $O(m \log n)$ preprocessing time
- $O(n \log n)$ space
- $O(\log n)$ query time
- $O(\log n)$ stretch

This is optimal in the sense that increasing k does not improve space or preprocessing time.

A naive approach to computing ADO's might be to do preprocessing in $\tilde{O}(mn)$ time, and produce a $O(n^2)$ size data structure holding all inter-vertex distances, in which case any distance query could be answered perfectly in $O(1)$ time. However, this preprocessing time is likely unacceptable, and furthermore, this data structure is far too large for any reasonably sized n.

Interestingly, for Thorup and Zwick's algorithm, for sufficiently dense graphs, the data structure produced is significantly more compact than the network itself.

2.1 Techniques

Thorup and Zwick first discuss a simpler setting, in which we investigate a metric space, and are given an nxn matrix of pairwise distance between the vertices in space.

In preprocessing, a non-increasing sequence of sets $A_0 \supseteq A_1 ... \supset A_{k-1}$, where A_0 contains all vertices, and each subsequent set A_i contains each vertex of A_{i-1} with probability $n^{-1/k}$

For each vertex and each above set, the algorithm computes the smallest distance from an i-center to v, and the nearest possible point to v within each level.

Finally, each vertex has a bunch $B(V)$ calculated, which consists of vertices $w \in A_i - Ai + 1$ that are strictly closer to v than all vertices of A_{i+1}, of expected size at most $kn^{1/k}$. The additional usage of a hash table of size $O(|B(v)|)$ allows $O(1)$ checking whether some vertex $w \; inB(v)$, and distance estimate if so.

The data structure contains, for each vertex $v \in V$

- for $0 \leq i \leq k-1$, store $p_i(v)$ where $\delta(p_i(v), v) = \delta(A_i, v)$
- The two level hash table for $B(v)$, holding $\delta(v, w)$ for each $w \in B(v)$

This leads to a total size of $O(kn+\sum_{v \in V} |B(v)|)$, which simplifies to $O(kn^{1+1/k})$ as the expected size of B(v) for every $v \in V$ is at most $kn^{1/k}$ due to a simple proof exploiting the geometric random quantity $n^{-1/k}$ which governs successive sets A_i.

Distance queries are answered in $O(k)$ time, as distance calculation consists of moving through sets A_i until $p_i(v) \in B(v)$, which can take a maximum of k steps.

The authors prove their stretch bound by showing that at each of our $k-1$ iterations of calculating the distance, the distance only goes up by the true distance at each iteration, based on how we choose and use witnesses.

2.2 ADO's for Graphs

In a more realistic situation, we would not explicitly have the metric $\delta(u, v)$. In this caes, the preprocessing is slightly more involved, creating "clusters" (essentially inverses of bunches) using a modified version of Dijkstra's algorithm which only relaxes edges if they satisfy a certain distance constraint. The clusters are then used to generate bunches, which can be used as before, and the algorithm follows accordingly.

Graph preprocessing now takes time $O(kmn^{1/k})$ time, and outputs, for every $w \in V$, the shortest paths tree $T(w)$ that spans the cluster $C(w)$.

Interestingly, the collection of shortest paths trees that this graph preprocessing algorithm produces form a (2k-1) spanner of the original graph, which has expected size $O(kn^{1+1/k})$ and can be constructed in $O(kmn^{1/k})$ time.

These trees therefore can similarly be used to form a tree cover, where stretch is similar.

2.3 Compacting

In order to store information more compactly, the authors employ a "labeling" scheme by which a node has all the information needed to calculate approximate distances. This involves, for each vertex, $p_i(v)$ and $\delta(A_i, v)$, as well as the 2-level hash table for bunch information. For a graph with diameter Δ, this produces a labeling scheme with $O(n^{1/k} \log n \log(n\Delta))$ bit labels. This work therefore can also be used for improved graph distance labeling.

3 Related Topics

Approximate Distance Oracles share a close relation to spanners and the idea of girth, as discussed below:

3.1 Spanners

There has been a long history of data structures that approximate shortest distances between nodes in a graph. Most of these rely on the idea that you can easily grow a spanning tree of the graph and use that as a heuristic for approximating distance from these spanning trees.

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.

A stretch t oracle therefore can directly produce a $t-spanner$ by producing paths containing the estimated distances, and $t-spanners$ thus enjoy the same space bounds that approximate distance oracles do.

By virtue of this the preprocessing algorithm for this data structure provides a faster way for constructing spanners. This is made possible by relying on Erdo's girth cojecture (discussed in section 3.2) to show that every weighted undirected map on n vertices has a $2k-1$ spanner with $n^{1+1/k}$ edges. Using an algorithm like Kruskal's would find such a spanner in time $mn^{1+1/k}$ while this algorithm could generate such a spanner in time $kmn^{1/k}$ instead reducing the runtime by a factor of n.

3.2 Girth

The girth of a graph is the size of its smallest simple cycle, which is at least $t+2$ iff no proper sub-graph of it is a $t-spanner$. Erdo's girth conjecture popped up repeatedly in discussion of ADOs. It states that a given graph with n vertices and at least $n^{1+1/k}$ edges has a girth no less than $2k+1$. Intuitively the girth of the graph or a sub-graph controls the factor of the spanners. Given that t-spanners are basically a clean mathematical view of t-ADOs we can see how girth plays into the size and accuracy of the data-structure.

Intuitively this relation carries over to other distance approximation algorithms because most of them such as HNSW also have the same common underlying idea that they perform the approximations by clumping the points or vertices that are closer to each other in the graph together. In fact the psudocode for the preprocessing algorithm is a little similar to that of the HNSW greedy search in proximity neighborhood graphs.

4 Improved Preprocessing Time

Wulff-Nilsen improves Thorup and Zwick's preprocessing bound from $O(kmn^{1/k})$ to $O(\sqrt{k}m + kn^{1+c/\sqrt{k}})$ while maintaining the same query time of $O(k)$, data structure space of $kn^{1+1/k}$, and stretch $2k-1$. This is better except for in sparse graphs or for small k, in which the original Thorup and Zwick preprocessing algorithm can be used. If Erdós' Girth conjecture is true, these stretch, space, query, and preprocessing bounds are optimal when $m = \omega(n^{1+c/\sqrt{k}})$ and $k = O(1)$. [2] [3]

This improved preprocessing bound is used through random vertex sampling, and the construction of a "restricted oracle" which only has to consider a subset of the graph which, along with a spanner of small stretch constructed in linear time using the algorithm of Basana and Sen (CITE) allows for $(2k-1)$ stretch and lower preprocessing. For smaller k, one may simply use Dijkstra to get exact distances between all pairs of sampled vertices in the spanner, which can be employed along with distances from vertices to nearest sampled vertex for far apart vertices to get $(2k-1)$ distances even for vertices far apart.

The restricted oracle is used for larger k, which only considers vertex pairs in which both have been sampled. A smaller stretch is needed in this oracle as we are no longer computing exact distance, and the authors derive that the $(2k-1)$ stretch bound will hold for $c \geq \frac{9}{\sqrt{k}} + 9\sqrt{\frac{1}{k} + \frac{4}{9}}$

5 Improved Query Time

Wulff-Nielsen improves from $O(k)$ query time to $O(\log k)$ query time without increasing space, preprocessing time, or stretch.

Wulff-Nielsen uses the same bunches employed by Thorup and Zwick and their original algorithm, but are able to use binary search to cut query time down from $O(k)$ to $O(\log k)$. At a high level, this is done using a series of subsequences of vertices, where each is at most 1/2 the size of the last. Binary search is used to identify the susequences (which takes time $O(\log k)$), and the final identified subsequence has length $O \log k)$ [4]

6 Constant Query Time

Chechik produces an oracle with constant query time by using the idea of "legitimate pairs". Chechik uses a combination of the Thorup and Zwick distance oracle along with the Mendel and Naor oracles.

In querying, Chek-Ind is used frequently, and Find-Legitimate is able to find a leigitimate pair in constant time. Given this legitimate pair, we get a 2k-1 estimate of the true distance. This paper is honestly fairly hard to parse, but we believe the high-level idea is that through some clever bookkeeping and additional preprocessing, we can drive the query time down to $O(1)$.

This improvement in query time comes at the cost of preprocessing, as the Mendel-Naor distance oracle is constructed in $(n^{2+1/k \log n}$ time. Overall, preprocessing time is

$$O(kmn^{1/k} + kn^{1+1/k} \log n + mn1/(ck) \log^3 n),$$

though the data structure retains the same size of $O(kn^{1+1/k})$. [5]

7 Proposed “Interesting Component”

We find that between the previously presented versions of ADO data structures there are clear tradeoffs between accuracy, preprocessing efficiency, space efficiency and query efficiency. We propose to answer the following questions: For real life applications, which use-case is most suitable for which of the above ADO version?

We aim to address this by implementing all four presented versions of ADO’s and running them on various use cases that involve varying degrees of constraints to benchmark their performance. The main objectives behind choosing to implement these data structures are:

- Provide an easy to use open source implementation of these data structures for the larger community.
- Consider the practical limitations of these data structures and how effective they are.

The first objective is motivated by the fact that we could not find any existing open source implementations of approximate distance oracles. We think that these present very useful tools for quick analysis of large graphs.

The second objective is that similiar to what we saw in class with the Fischer-Heun RMQ structure, a good worst case run-time performances might not translate well into practical implementations and so by implementing these data structures we hope to uncover which algorithms practically perform better.

8 Initial results

We implemented the original ADO from [1] in python using the networkx package (`https://github.com/magdyksaleh/approxDistOracles`) for graph processing and ran it on the following randomly generated unweighted Erdos-Reyni graph with 200 nodes and and a 30% edge probability.

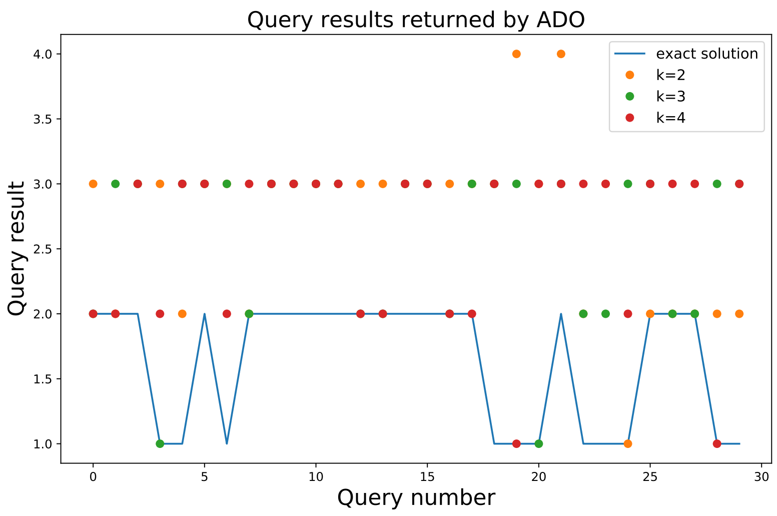

Figure 1: Query values returned by running 30 random queries on nodes in the graph. Each query was run using APSP solution and ADOs with $k = 2, 3$ and 4.

From Figure 1 we find that a large fraction of the queries are correct or overestimate the distance by 1 edge. We also note that the largest error occurs for $k = 2$ at query number 20, and the error is 3, which is within the guarenteed tolerance. Furthermore we compared our implementation to the APSP solution for when $k = 1$ and the results matched as expected.

References

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

[2] Paul Erdos. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3(2):113–116, 1965.

[3] Christian Wulff-Nilsen. Approximate Distance Oracles with Improved Preprocessing Time. 9 2011.

[4] Christian Wulff-Nilsen. Approximate Distance Oracles with Improved Query Time. In *Encyclopedia of Algorithms*, pages 1–4. 2015.

[5] Shiri Chechik. Approximate Distance Oracle with Constant Query Time. Technical report, 2013.