Efficient Exact Learning Algorithms for Road Networks and Other Graphs with Bounded Clustering Degrees

Ramtin Afshar
University of California, Irvine, CA, USA

Michael T. Goodrich
University of California, Irvine, CA, USA

Evrim Ozel
University of California, Irvine, CA, USA

Abstract

The completeness of road network data is significant in the quality of various routing services and applications. We introduce an efficient randomized algorithm for exact learning of road networks using simple distance queries, which can find missing roads and improve the quality of routing services. The efficiency of our algorithm depends on a *cluster degree* parameter, $d_{\max}$, which is an upper bound on the degrees of vertex clusters defined during our algorithm. Unfortunately, we leave open the problem of theoretically bounding $d_{\max}$, although we conjecture that $d_{\max}$ is small for road networks and other similar types of graphs. We support this conjecture by experimentally evaluating our algorithm on road network data for the U.S. and 5 European countries of various sizes. This analysis provides experimental evidence that our algorithm issues a quasilinear number of queries in expectation for road networks and similar graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Random network models; Theory of computation → Query learning

Keywords and phrases Road Networks, Exact Learning, Graph Reconstruction, Randomized Algorithms

Digital Object Identifier 10.4230/LIPIcs.SEA.2022.9

Supplementary Material *Software (Source Code)*: `https://github.com/UC-Irvine-Theory/RoadNetworkReconstruction`; archived at `swh:1:dir:337dc5ac6d78bd68031d6f94b24332b935797a67`

1 Introduction

We study the problem of reconstructing an undirected, unweighted and connected graph $G = (V, E)$, by taking as input its set of vertices V and issuing queries to a *distance oracle*, which takes as input a pair of vertices $u, v \in V$ and returns the number of edges on the shortest path between them. The goal is to learn the edges in E by using the results that are returned from these queries. In particular, we are concerned with reconstructing *road networks*, which have been characterized in numerous ways, e.g., see [18, 21, 22, 25]. As a starting point, we can view road networks as undirected, unweighted, and connected graphs with a constant maximum degree, where each vertex corresponds to a road junction or terminus, and each edge corresponds to road segments that connect two vertices. In this paper, we present a randomized incremental algorithm for *exact learning* of road networks, where we assume the existence of a distance oracle that responds to distance queries.

Even though our algorithm only works with unweighted graphs, it is possible to use weighted graphs as input by subdividing each edge, replacing each edge e with $\lceil w(e) \rceil$ edges, where $w(e)$ is the weight of e. Since the average edge weight in road networks is typically small (e.g., as observed in [25]), this will only increase the number of vertices and edges in the

graph by a constant factor that is independent of the size of the graph. This preprocessing step is important for applications of road network reconstruction in routing services, where the completeness of road network data has great importance. For example, machine learning techniques have been utilized in the past to find the missing roads in incomplete road network data [24]. Though our experiments focus on unweighted road networks, we include some experimental results for weighted road networks with subdivided edges as well.

Another application relevant to this work is the use of structured encryption [16] in the context of cloud computing, where a data owner encrypts structured data, such as a graph, stores it in a database managed by a third-party cloud provider, and wishes to query it privately (e.g., using single-pair shortest path queries [26]). In the scenario where an adversary server is able to generate valid queries of its own, it would be able to use a graph reconstruction algorithm to learn the edges in the graph, resulting in a breach of privacy.

A graph reconstruction algorithm A is evaluated based on the number of queries it issues, which we call the *query complexity* of A, following nomenclature from learning theory (e.g., see [2, 3, 17, 20, 35]) and complexity theory (where this is also known as "decision-tree complexity," e.g., see [38, 15]). For instance, Kannan, Mathieu and Zhou [29] present exact learning algorithms for connected, undirected graphs that have bounded degree, including a randomized algorithm that has expected query complexity $O(\Delta^3 n^{3/2} \operatorname{polylog}(n))$, where Δ is the maximum degree of the graph, using distance queries. This bound simplifies to $O(n^{3/2} \operatorname{polylog}(n))$ for graphs with maximum degree $O(\operatorname{polylog}(n))$.

We note that a bound on the maximum degree is necessary for subquadratic exact learning algorithms, as there is a simple $\Omega(n^2)$ lower bound for the query complexity of graphs with unbounded degrees, e.g., see [29]. Likewise, a trivial upper bound for the task of reconstructing a general graph G is $O(n^2)$ distance queries, as one can issue a distance query for every pair of vertices in the graph and return all pairs of vertices that have distance 1 between them as edges. We refer to this as an *exhaustive search* on G.

1.1 Related Prior Results

The problem of reconstructing graphs by issuing queries has been studied extensively, e.g., see [1, 5, 8, 13, 32, 37, 4, 6, 7, 9, 10, 11, 17, 19, 27, 28, 30, 33]. These works differ in terms of their assumptions about the hidden graph (e.g., whether the hidden graph is a tree, a general graph, or something else) or the types of queries that they issue.

In terms of the most relevant prior work, Kannan, Mathieu and Zhou [29] showed how to reconstruct a connected, unweighted graph G using $O(\Delta^3 n^{3/2} \operatorname{polylog}(n))$ distance queries in expectation, where they performed an exhaustive search on the Voronoi cells created by a call to a graph clustering algorithm inspired by Thorup and Zwick [36]. They also raised the open question of whether we can achieve an algorithm that uses $O(n \operatorname{polylog}(n))$ distance queries in expectation for bounded degree graphs. In a recent work [31], Mathieu and Zhou provided a partial answer for that open question by providing an algorithm that uses $O(n \operatorname{polylog}(n))$ distance queries in expectation for random Δ-regular graphs. However, this does not imply an algorithm with an expected query complexity of $O(n \operatorname{polylog}(n))$ distance queries for road networks as they are not necessarily regular. For general graphs of bounded degree, their algorithm uses $O(n^{5/3} \operatorname{polylog}(n))$ distance queries in expectation.

In another work, Afshar, Goodrich, Matias and Osegueda [6] introduced a parallel implementation of the graph clustering technique of Thorup and Zwick [36] and presented a parallel algorithm for reconstructing connected, unweighted graphs using $O(\Delta^2 n^{3/2+\epsilon})$ distance queries in $O(1)$ parallel rounds for constant $0 < \epsilon < 1/2$, with high probability.

1.2 Our Contributions

In this paper, we introduce a randomized incremental algorithm for exact reconstruction of bounded degree graphs and demonstrate through experiments that it has expected empirical query complexity $O(n \operatorname{polylog}(n))$, providing an empirical answer to the open question raised by [29] mentioned above.

The main idea of our algorithm is to cluster the graph into cells by incrementally selecting random vertices as centers. We then issue distance queries between that center and the rest of the graph to decide which vertices should be added to the new cell. We continue this process until the size of each cell is below some threshold value. The final step is to then perform exhaustive searches in each cell.

Our algorithm uses the same overall strategy used in [29], which is based on finding a Voronoi cell decomposition of the graph. However, our algorithm differs in a number of important ways. In [29], the goal of the algorithm is to produce cells such that the size of each cell is $O(\sqrt{n}/\Delta)$. Our algorithm, however, produces cells that have size at most a chosen constant. Since performing exhaustive searches on all of these cells requires only $O(n)$ queries, the query complexity of our algorithm depends mainly on the initial step of constructing the cells and not the exhaustive querying step. Moreover, our algorithm incrementally constructs the cell decomposition by updating it with each newly added center, whereas [29] updates the cell decomposition only after adding multiple centers.

We perform experiments on several real-world road networks and show, by considering the number of queries performed at each step, that our algorithm has expected empirical query complexity $O(n \operatorname{polylog}(n))$. Moreover, we theoretically analyze our algorithm and prove an upper bound of $O(d_{\max}^2 n \log n)$ expected queries, where $d_{\max}$ is the maximum degree in the dual graph of cells during our algorithm. To characterize $d_{\max}$, we collect data on the maximum cell degrees during our experiments, and find that the value of $d_{\max}$ scales logarithmically with respect to n for road networks. When combined with our theoretical analysis, this results in an alternative way to obtain an empirical upper bound of $O(n \operatorname{polylog}(n))$ expected queries for our algorithm. In addition, we perform experiments to directly compare the number of queries our algorithm issues to the number of queries issued by existing algorithms, and observe empirically that our algorithm issues significantly fewer queries.

Our paper is organized as follows. We provide some preliminaries in Section 2, our algorithm is in Section 3, the results from our analysis are in Section 4, experimental results are in Section 5, comparisons between theoretical/experimental results are in Section 6, and the conclusions are in Section 7.

2 Preliminaries

We reconstruct graphs $G = (V, E)$ that consist of $n = |V|$ vertices and $m = |E|$ edges, and are undirected, unweighted and connected. For a graph $G = (V, E)$, a *cell* is defined as any subset of V. A *cover* of G is a set of cells $\mathcal{C}$ such that $\bigcup_{C \in \mathcal{C}} C = V$ and for each edge $(u, v) \in E$ there exists at least one cell $C \in \mathcal{C}$ such that $u, v \in C$.

For two vertices $u, v \in V$, $\delta(u, v)$ denotes the number of edges on the shortest path between u and v in G. For a subset of vertices $A \subseteq V$, $\delta(v, A) = \min_{a \in A} \delta(v, a)$. For a vertex v and a cell C, the subroutine Distances(v,C) determines $\delta(v, u)$ $\forall u \in C$ by issuing distance queries between v and every vertex in C.

Algorithm 1 Graph Reconstruction.

1: **Function** RECONSTRUCT(V):
2: $E \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$, $W \leftarrow V$
3: Let M denote the maximum size of cells, initially $+\infty$
4: **while** $M > g$ **do** // g is a chosen constant
5: $a \leftarrow$ a random vertex from W
6: $W \leftarrow W \setminus \{a\}$
7: $\mathcal{C} \leftarrow \mathsf{cover}(V, \mathcal{C}, a)$
8: $M \leftarrow \max_{C \in \mathcal{C}} \{|C|\}$
9: **for** $C \in \mathcal{C}$ **do**
10: $E \leftarrow E \cup$ EXHAUSTIVE-QUERY(C)
11: **return** E

3 Algorithm

The first component of our algorithm is RECONSTRUCT(V), which takes as input the set of vertices V of the input graph and returns the reconstructed graph with the correct edge assignments. We start by choosing a constant, g, which is the threshold value for the maximum sized cell in our cover. In a loop, we randomly select an unselected vertex to be the center for the new cell, and call COVER($V, \mathcal{C}, a$) to get a new cover which includes the new cell with center a. We describe how COVER($V, \mathcal{C}, a$) works later in this section.

We keep performing this loop until the maximum sized cell in the cover becomes less than g, in which case we terminate the loop and perform an exhaustive search on each cell of the cover. The function EXHAUSTIVE-QUERY(C) takes as input a cell C and returns all edges between vertices in C by issuing distance queries for each pair of vertices in C. We provide details in Algorithm 1.

The second and main component of our algorithm is COVER($V, \mathcal{C}, a$) (see Algorithm 2), which takes as input the set of vertices V, a set of cells $\mathcal{C}$ and a vertex a, and returns a new cover where a is the center of a new cell N. We define S, which we call the *frontier*, to be the set of cells that we should search in expanding N, and we initialize it with the cells that a belongs to. The only exception is when we first call COVER($V, \mathcal{C}, a$), in which case we initialize S to be $\{V\}$ (see lines 3-6). Then, an arbitrary cell, C, from S is chosen, and we issue distance queries between a and all of the vertices of C.

Using the results from these queries, we determine which vertices in C are close to a, compared to their distances to all the other centers. We define A to be a global variable that stores the set of all centers that were added before the new center a. For a vertex $v \in C$, if $\delta(a, v) \leq \delta(A, v) - 1$, we remove v from all of the cells that contains it (see line 18). If, however, $\delta(a, v) = \delta(A, v)$ or $\delta(a, v) = \delta(A, v) + 1$, we consider v to be on the *boundary* between C and N and so we do not remove v from any cells. In both cases, we add v to the new cell N, and we add any unvisited cells that contain v to S since they might have vertices that are close to a as well.

We say that a cell $C_2 \in \mathcal{C}$ is a *neighbor* of $C_1 \in \mathcal{C}$ if $C_1 \cap C_2 \neq \emptyset$. In other words, two cells are neighbors if there exists a boundary vertex that belongs to both of them. So, each iteration of COVER($V, \mathcal{C}, a$) ends up adding to the frontier all unvisited neighbors of the current cell $C \in S$ that share at least one boundary vertex with C such that this boundary vertex can be added to N according to the closeness definition in line 15. Note that we do

Algorithm 2 COVER$(V, \mathcal{C}, a)$ algorithm for constructing a new cover after adding a cell centered at vertex a.

```
1: N ← {a}   // N is the new cell centered at a
2: L ← ∅   // L is the set of cells that have been visited
3: if C = ∅ then
4:    S ← {V}
5: else
6:    S ← {C ∈ C | a ∈ C}   // S is the set of cells that we should search in expanding N
7: while S ≠ ∅ do
8:    C ← an arbitrary cell from S
9:    S ← S \ {C}
10:   L ← L ∪ C
11:   Distances(a, C)
12:   // A is a global variable denoting the set of all cell centers
13:   If A = ∅, ∀v ∈ V: set δ(A, v) = +∞.
14:   for v ∈ C do
15:      if δ(a, v) ≤ δ(A, v) + 1 then
16:         S ← S ∪ {C′ ∈ C | v ∈ C′ and C′ ∉ L}   // add all of the unvisited cells that
            contain v to the frontier S
17:         if δ(a, v) ≤ δ(A, v) − 1 then
18:            Remove v from all the cells that contain it
19:         N ← N ∪ {v}
20: A ← A ∪ {a}
21: return C ∪ {N}
```

not necessarily add all the neighbors of C to the frontier: if none of the boundary vertices v between C and a neighboring cell N' have distance at most $\delta(A, v) + 1$ to N's center, then it is clear that none of the vertices in N' can be added to N.

4 Correctness and Analysis

▶ **Theorem 1.** *For any undirected, unweighted and connected graph $G = (V, E)$,* RECONSTRUCT(V) *correctly reconstructs E.*

Proof. We use an inductive argument to prove that the union of exhaustive searches performed on the cells created by the algorithm discovers all $(u, v) \in E$.

Initially, there is a single cell containing all of the vertices V, which trivially covers all the edges of E. Now, let A_i represent the first i centers that we add in the algorithm and assume, at every step $2 \leq s \leq i$, that for each edge $(u, v) \in E$ there is a cell with its center in A_s that contains both u and v. We then prove that if we create a new cell N, centered at the $(i + 1)$-th center $a \in A_{i+1}$, the union of the new cells still covers all the edges in E.

Consider an edge $(e_1, e_2) \in E$. Let x be the last center among the first $i + 1$ centers such that $x = \text{argmin}_{a \in A_{i+1}}\{\min(\delta(a, e_1), \delta(a, e_2))\}$. In other words, x is the last center that is closest to either endpoint of the edge (e_1, e_2). If $\delta(x, e_1) \neq \delta(x, e_2)$, we denote the endpoint that is closer to x as v, and denote the other endpoint as u. Otherwise, we denote the endpoints arbitrarily as u and v. So, we have $\min(\delta(x, e_1), \delta(x, e_2)) = \delta(x, v) = \delta(A_{i+1}, v) \leq \delta(A_{i+1}, u)$. We prove that both u and v belong to the cell centered at x, after the $(i + 1)$-th iteration.

First, we prove that we add both v and u to the cell at x. Let $(s_1, s_2, \ldots, s_m)$ denote the ordered vertices on a shortest path from x to v, where $s_1 = x$ and $s_m = v$. Using the inductive hypothesis for each $2 \leq j \leq m$, there exists a cell that contains both s_{j-1} and s_j right before adding center x. Now, consider the smallest j such that s_j is not added to the cell at x during the loop at line 14. Since s_{j-1} is added to the cell at x, and since s_{j-1} and s_j are connected, then by the inductive hypothesis there is a cell C that contains both s_{j-1} and s_j. Therefore, when we add s_{j-1} to the cell at x, we add C to the set of cells that we should explore in expanding the cell centered at x (see line 16). On the other hand, since s_j is on the shortest path from x to v and $\delta(x, v) = \delta(A_{i+1}, v)$, then $\delta(x, s_j) = \delta(A_{i+1}, s_j)$. Therefore, s_j will be added to the cell at x when exploring cell C. Using this inductive approach, all vertices on the shortest path from x to v will be added to the cell at x. Finally, since $\delta(x, u) \leq \delta(x, v) + 1 = \delta(A_{i+1}, v) + 1 \leq \delta(A_{i+1}, u) + 1$, and since u and v also have a common cell, we add u to the cell at x.

Next, we prove that if we add v and u to the cell centered at x, no other cells that we create later on in the first $(i+1)$-th steps removes v or u from the cell at x. Note that for removing a node from cell x, the condition at line 17 must hold. Since $\delta(x, v) = \delta(A_{i+1}, v)$, there will be no center b among the first $(i+1)$ centers such that $\delta(b, v) \leq \delta(A_{i+1}, v) - 1$, meaning that v will stay in cell at x. On the other hand, we remove u from x only if for a center b: $\delta(b, u) \leq \delta(x, u) - 1$. If $\delta(b, u) \leq \delta(x, u) - 1$, and given the fact that $\delta(x, u) \leq \delta(x, v) + 1$ and $\delta(x, v) \leq \delta(x, u)$, then $\delta(b, u) \leq \delta(x, v) \leq \delta(x, u)$. However, we assumed that x is the last center, among the first $i + 1$ centers, that is closest to either of the endpoints u and v. Therefore, u will also stay in the cell at x. ◀

▶ **Theorem 2.** *The expected query complexity of* RECONSTRUCT(V) *is* $O(d_{\max}^2 n \log n)$*, where* $d_{\max}$ *is the maximum cell degree over all steps.*

Proof. We use a backwards analysis [34] to derive an expression for the expected query complexity of the algorithm. We assume i centers have already been added, and analyze the expected number of queries we issue at step i.

We observe that our algorithm only issues distance queries for cells in the set S. Moreover, the only cells we add to S are the ones that contain vertices that get added to the ith cell. This means that all cells in S will become neighbors of the ith cell at the end of step i. So the number of distance queries issued at step i is the sum of the sizes of each cell that gets added to S, which is at most the sum of the sizes of the ith cell and its neighbors at the end of step i. Denoting the set of cells at the end of step i as $\mathcal{C}_i$, and the set of cells neighboring any cell C as $N(C)$, we have that the expected number of queries issued at the ith step is

$$\leq \sum_{C \in \mathcal{C}_i} \frac{1}{i} (|C| + \sum_{C' \in N(C)} |C'|)$$
$$= \sum_{C \in \mathcal{C}_i} \frac{(d(C)+1)|C|}{i},$$

by observing that each cell size $|C|$ is summed $d(C) + 1$ times, where $d(C)$ denotes the *degree* of cell C, i.e. the number of neighboring cells it has. To bound this summation, we express each cell size as the sum of boundary and non-boundary vertices. We have

$$\sum_{C \in \mathcal{C}_i} \frac{(d(C)+1)|C|}{i} = \sum_{C \in \mathcal{C}_i} \frac{(d(C)+1)(|C|_{NB} + |C|_B)}{i}$$
$$\leq \frac{(d_{\max}+1)}{i} \left((\sum_{C \in \mathcal{C}_i} |C|_{NB}) + (\sum_{C \in \mathcal{C}_i} |C|_B) \right)$$

where $d_{\max}$ is the maximum degree of any cell during any step, $|\cdot|_B$ denotes the number of boundary vertices, and $|\cdot|_{NB}$ denotes the number of non-boundary vertices. We use the fact that $\sum_{C\in\mathcal{C}_i}|C|_{NB} \leq n$, and observe that $\sum_{C\in\mathcal{C}_i}|C|_B \leq (d_{\max}+1)\cdot n$ as each boundary vertex can belong to at most $d_{\max}+1$ cells, and thus can only be counted that many times at most in the summation. So, we have

$$\frac{(d_{\max}+1)}{i}\left((\sum_{C\in\mathcal{C}_i}|C|_{NB})+(\sum_{C\in\mathcal{C}_i}|C|_B)\right) < \frac{n(d_{\max}+2)^2}{i}.$$

The expected number of queries when all steps are considered is

$$< \sum_{i=1}^{\#\text{steps}} \frac{n(d_{\max}+2)^2}{i} = n(d_{\max}+2)^2 \sum_{i=1}^{\#\text{steps}} \frac{1}{i},$$

which is $O(d_{\max}^2 n \log n)$. To finish our analysis, we also need to consider the number of queries issued during the exhaustive searches in each cell. Since the total number of cells is $O(n)$, and each cell is of size at most a constant g, the exhaustive querying part has query complexity $O(n)$. ◀

5 Experimental results

5.1 Implementation and Datasets

We implemented our algorithm in C++, and simulated the distance query oracle by performing BFS in each iteration to compute distances between nodes while keeping track of how many distance queries would be necessary to find these distances. We selected the value of g to be 50. We include experimental results for road networks from 50 U.S. states and Washington, D.C. obtained from the formatted TIGER/Line dataset available from the 9th DIMACS Implementation Challenge website[1] and road networks from Belgium, the U.K. (limited to the road network of Great Britain), Italy, Luxembourg, and the Netherlands obtained from formatted OpenStreetMaps data available from the 10th DIMACS Implementation Challenge [12]. For all of the datasets, only the largest connected component is considered. In Section 6, we discuss how the upper bounds derived from these experiments compare to our theoretical upper bound.

5.2 Batch Length

We define the *batch length* of a step to be the number of distance queries issued at that step. To find the relation between batch length and the step number, based on the theoretical upper bound we derived in Section 4, we fit the function $\textsc{Batch-Length}(step) = a + b\frac{n}{\text{step}}$ to the data points in our results, where a and b are the fitting parameters. Data points for the batch lengths of some of our datasets are provided in Figure 1. We list our results for all of the datasets in Table 1, which includes the best-fit parameters in columns a and b, and the maximum number of cells visited at any step in column M.

We can see that parameter b does not exceed 2, and that parameter a is close to 0 for all of the datasets. This suggests that a constant or logarithmic factor of $\frac{n}{\text{step}}$ could be an upper bound for the batch size at any step, which leads us to predict an upper bound of $\log n \cdot \frac{n}{\text{step}}$ which we show in Figure 1. We report the percentage of steps that fall below this upper bound for each dataset in Table 1, column U.

[1] http://www.diag.uniroma1.it/~challenge9/data/tiger/

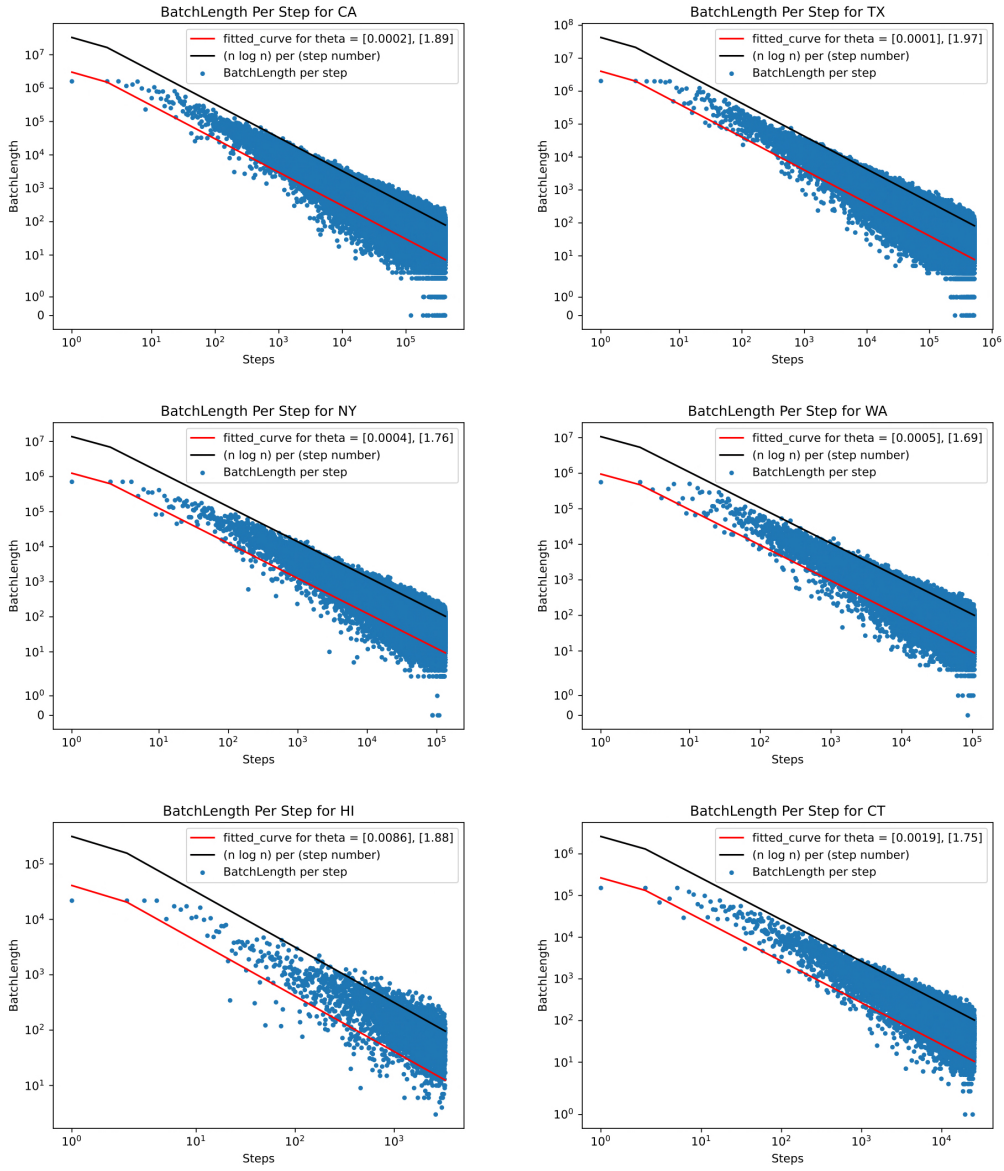


Figure 1 Batch lengths for select datasets of varying sizes.

Table 1 Batch length results for all datasets. Columns a, b and U were rounded to 4, 2 and 2 decimal places respectively.
a, b: best-fit parameters for Batch-Length$(step\#) = a + b\frac{n}{step\#}$
U: percentage of batch lengths that are below the upper bound of $\frac{n}{step\#} \log n$.

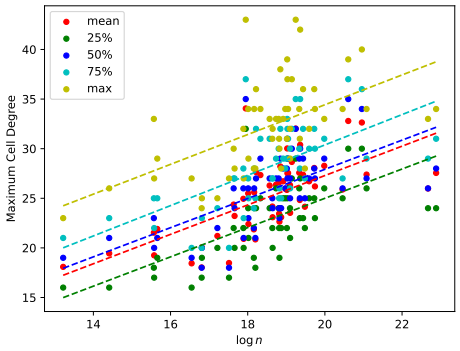
Dataset	n	a	b	U	Dataset	n	a	b	U
AK	48 560	0.0036	1.69	96%	ND	203 583	0.0015	1.76	92%
AL	561 459	0.0005	1.91	98%	NE	304 335	0.0011	1.99	93%
AR	478 024	0.0005	1.85	98%	NH	115 055	0.0023	1.91	97%
AZ	533 008	0.0005	1.95	95%	NJ	329 404	0.0010	1.90	94%
CA	1 595 577	0.0002	1.89	96%	NM	456 896	0.0006	1.88	97%
CO	436 084	0.0006	1.91	96%	NV	253 012	0.0009	1.85	94%
CT	152 036	0.0019	1.75	94%	NY	708 520	0.0004	1.76	97%
DC	9522	0.0321	1.89	70%	OH	672 527	0.0005	1.93	95%
DE	48 812	0.0053	1.77	91%	OK	535 032	0.0006	1.87	96%
FL	1 036 647	0.0003	1.86	97%	OR	529 702	0.0005	1.90	98%
GA	731 954	0.0004	1.97	97%	PA	866 352	0.0004	1.83	97%
HI	21 774	0.0086	1.88	90%	RI	51 642	0.0047	1.88	92%
IA	388 487	0.0008	1.92	93%	SC	460 763	0.0005	1.81	97%
ID	265 552	0.0010	1.81	97%	SD	206 998	0.0014	1.85	94%
IL	790 439	0.0004	1.89	96%	TN	578 981	0.0004	1.70	98%
IN	495 581	0.0007	1.88	94%	TX	2 037 156	0.0001	1.97	97%
KS	471 066	0.0007	1.87	93%	UT	242 432	0.0010	1.95	96%
KY	463 542	0.0006	1.90	98%	VA	620 680	0.0004	1.92	98%
LA	408 161	0.0006	1.84	97%	VT	95 672	0.0022	1.95	98%
MA	294 345	0.0009	1.98	95%	WA	560 336	0.0005	1.69	97%
MD	264 378	0.0010	1.86	95%	WI	514 687	0.0006	1.89	96%
ME	187 315	0.0013	1.81	99%	WV	292 557	0.0006	1.89	99%
MI	661 718	0.0005	1.74	95%	WY	243 545	0.0010	1.92	97%
MN	541 166	0.0006	1.76	95%	BE	1 441 295	0.0002	2.00	99%
MO	668 322	0.0004	1.89	97%	GB	7 733 822	0	1.81	100%
MS	409 994	0.0007	1.93	98%	IT	6 686 493	0	1.81	100%
MT	300 809	0.0007	1.80	98%	LU	114 599	0.0019	1.96	99%
NC	876 954	0.0003	1.72	99%	NL	2 216 688	0.0001	1.86	99%

5.3 Maximum Cell Degree

To combine our experimental results with our theoretical upper bound, we collected data on the maximum cell degree at each step. We combine the step-wise data in each dataset using different measures: mean, max, and the 1st, 2nd and 3rd quartiles to see how the data is spread. Then for each dataset, we represent the value corresponding to each measure as a point. Based on our intuition, we fit the function $a \log n + b$ for each measure. We list our results in Figure 2, which includes the best-fit parameters for each measure. We can see that $a < 2$, and b is a small constant for each measure. The datasets with the largest maximum cell degrees turned out to be VA, NV and OH, with values of 43, 43 and 42 respectively. It is clear from the figure that a small constant multiple of $\log n$ would be enough to produce an upper bound that covers all of the data points, suggesting that the maximum cell degree might have an upper bound of $O(\log n)$.

5.3.1 Road Networks with Subdivided Edges

We provide experimental results in Table 2 for the maximum cell degrees of the weighted road networks of the District of Columbia and the state of Hawaii, which we transform into unweighted graphs by replacing each edge e with $\lceil w(e) \rceil$ edges, where $w(e)$ is the weight of e. The size of each road network increased by factors of approximately 192 and 167 respectively, while the maximum cell degree values ended up decreasing for both road networks. This indicates that our algorithm can also perform efficiently on weighted road networks.



(a) Best-fit lines for the function $a + b \log n$.

Measure	a	b
mean	-2.34	1.48
1st quartile	-4.52	1.48
2nd quartile	-1.49	1.47
3rd quartile	-0.2	1.53
max	4.4	1.50

(b) Parameters for best-fit lines. Columns a and b were rounded to 2 decimal places.

Figure 2 Results from combining step-wise maximum cell degrees.

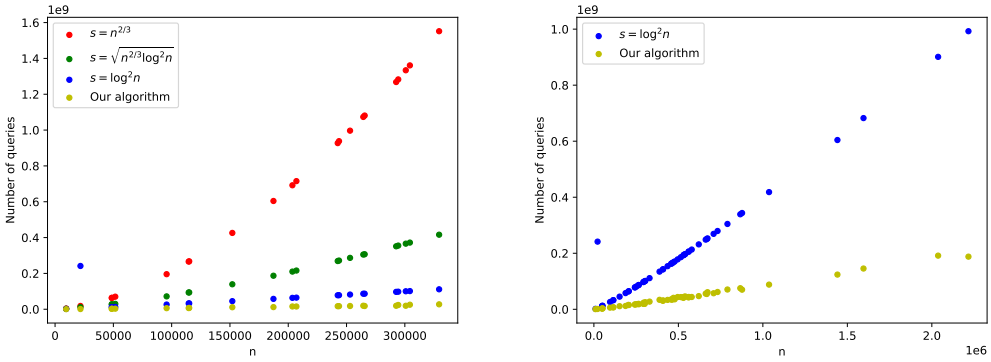


Figure 3 Number of queries issued by our algorithm compared to [31].

5.4 Comparisons with Existing Algorithms

We directly compare the number of queries issued by our algorithm to the algorithm introduced in [31], which takes as input a parameter s that affects the query complexity. The authors prove their query complexity bounds for Δ-regular graphs and bounded graphs with the value of s being set to $\log^2 n$ and $n^{2/3}$ respectively. We use these values for s in our experiments, and we also try setting s to be the geometric mean of these values. We summarize the results of our experiments on road networks in Figure 3.

We then compare our results to the number of queries issued by the algorithm in [29]. Without performing any experiments, it can be observed that this algorithm will issue significantly more queries than our algorithm: the first iteration of Estimated-Centers (Algorithm 2 in [29]) will issue at least $\Omega(n \cdot \Delta\sqrt{n} \cdot \log n \cdot \log\log n)$ distance queries.

Table 2 Maximum cell degrees for weighted road networks compared to their unweighted versions.

	Unweighted		Weighted	
	$\lvert V\rvert$	$d_{\max}$	$\lvert V\rvert$	$d_{\max}$
DC	9522	23	1 826 049	12
HI	21 774	26	3 643 818	11

6 Comparison of Theoretical/Experimental Results and Future Work

The theoretical upper bound we derived in Section 4 contains a $d_{\max}^2$ term, which can be $O(n^2)$ in the worst case. However, our experiments show that the maximum cell degree is actually low for road networks throughout the algorithm. From our results in Section 5.3, we can see that $O(\log n)$ would be a suitable upper bound for $d_{\max}$. In this case, the expected query complexity of our algorithm would be $O(n \log^3 n)$. The experimental results for batch length support this observation, as the upper bound for the batch length at any step number amounted to be $\frac{n \log n}{step\#}$, from which we get an expected query complexity of $O(n \operatorname{polylog}(n))$ as well. Future work might involve trying to find if there exists a better theoretical upper bound on the query complexity of our algorithm. This might require making some additional simplifying assumptions about the graphs being used as input.

We would like to point out a connection between our results and the graph-theoretical Delaunay triangulation of road networks.

6.1 Delaunay Triangulations and $d_{\max}$

We can consider the cells (resp. covers) that are constructed during our algorithm as a redefinition of graph-theoretical Voronoi cells (resp. diagrams) (e.g., see [23]). Similarly, we can consider the dual graph connecting neighboring cells in the cover as being a form of a graph-theoretical Delaunay triangulation of G. There exists prior work on bounding the expected maximum degree of the Delaunay triangulation of a set of points selected randomly from the Euclidean plane. In [14], the authors consider the Delaunay triangulation of a Poisson point process limited to the portion of the triangulation within a cube of d dimensions. They show that the expected maximum degree of this triangulation is $\Theta(\log n / \log \log n)$. Having such a bound for the expected maximum degree for our redefinition of the graph-theoretical Delaunay triangulation might allow us to prove a theoretical quasilinear bound for the query complexity of our algorithm, so another interesting direction for future work can be to adapt this result for random point sets in Euclidean d-space to our setting, where the point set is selected randomly from the vertex set of a road network.

7 Conclusions

We introduced an efficient exact reconstruction algorithm for road networks and showed through experiments on several real-world road networks that our algorithm has an expected empirical query complexity that is quasilinear. As mentioned in Section 6, an important direction for future work can be to derive a theoretical upper bound for our algorithm that matches our experimental results.

References

1 Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph reconstruction with a betweenness oracle. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.5.

2 Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. The query complexity of finding a hidden permutation. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 1–11, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40273-9_1.

3 Ramtin Afshar, Amihood Amir, Michael T. Goodrich, and Pedro Matias. Adaptive exact learning in a mixed-up world: Dealing with periodicity, errors and jumbled-index queries in string reconstruction. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 155–174. Springer, 2020. doi:10.1007/978-3-030-59212-7_12.

4 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing binary trees in parallel. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 491–492. ACM, 2020. doi:10.1145/3350755.3400229.

5 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Reconstructing biological and digital phylogenetic trees in parallel. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 3:1–3:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.3.

6 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Parallel network mapping algorithms. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 410–413. ACM, 2021. doi:10.1145/3409964.3461822.

7 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Mapping networks via parallel kth-hop traceroute queries. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.4.

8 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM J. Discret. Math.*, 18(4):697–712, 2005. doi:10.1137/S0895480103431071.

9 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004. doi:10.1137/S0097539702420139.

10 Dana Angluin and Jiang Chen. Learning a hidden hypergraph. *J. Mach. Learn. Res.*, 7:2215–2236, 2006. URL: http://jmlr.org/papers/v7/angluin06a.html.

11 Dana Angluin and Jiang Chen. Learning a hidden graph using o(logn) queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, 2008. doi:10.1016/j.jcss.2007.06.006.

12 David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013. doi:10.1090/conm/588.

13 Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE J. Sel. Areas Commun.*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.

14 Marshall W. Bern, David Eppstein, and F. Frances Yao. The expected extremes in a delaunay triangulation. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, volume 510 of *Lecture Notes in Computer Science*, pages 674–685. Springer, 1991. doi:10.1007/3-540-54233-7_173.

15 Anna Bernasconi, Carsten Damm, and Igor Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Information and Computation*, 168(2):113–124, 2001. doi:10.1006/inco.2000.3017.

16 Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010. doi:10.1007/978-3-642-17373-8_33.

17 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artificial Intelligence*, 174(9):551–569, 2010. doi:10.1016/j.artint.2010.02.003.

18 Padraig Corcoran, Musfira Jilani, Peter Mooney, and Michela Bertolotto. Inferring semantics from geometry: the case of street networks. In Jie Bao, Christian Sengstock, Mohammed Eunus Ali, Yan Huang, Michael Gertz, Matthias Renz, and Jagan Sankaranarayanan, editors, *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA, November 3-6, 2015*, pages 42:1–42:10. ACM, 2015. doi:10.1145/2820783.2820822.

19 Luca Dall'Asta, J. Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theor. Comput. Sci.*, 355(1):6–24, 2006. doi:10.1016/j.tcs.2005.12.009.

20 Shahar Dobzinski and Jan Vondrak. From query complexity to computational complexity. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1107–1116, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214076.

21 David Eppstein and Michael T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. *CoRR*, abs/0808.3694, 2008. arXiv:0808.3694.

22 David Eppstein and Siddharth Gupta. Crossing patterns in nonplanar road networks. In Erik G. Hoel, Shawn D. Newsam, Siva Ravada, Roberto Tamassia, and Goce Trajcevski, editors, *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, pages 40:1–40:9. ACM, 2017. doi:10.1145/3139958.3139999.

23 Martin Erwig. The graph voronoi diagram with applications. *Networks*, 36(3):156–163, 2000. doi:10.1002/1097-0037(200010)36:3<156::AID-NET2>3.0.CO;2-L.

24 Stefan Funke, Robin Schirrmeister, and Sabine Storandt. Automatic extrapolation of missing road network data in openstreetmap. In Ioannis Katakis, François Schnitzler, Thomas Liebig, Dimitrios Gunopulos, Katharina Morik, Gennady L. Andrienko, and Shie Mannor, editors, *Proceedings of the 2nd International Workshop on Mining Urban Data co-located with 32nd International Conference on Machine Learning (ICML 2015), Lille, France, July 11th, 2015*, volume 1392 of *CEUR Workshop Proceedings*, pages 27–35. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1392/paper-04.pdf.

25 Stefan Funke and Sabine Storandt. Provable efficiency of contraction hierarchies with randomized preprocessing. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, volume 9472 of *Lecture Notes in Computer Science*, pages 479–490. Springer, 2015. doi:10.1007/978-3-662-48971-0_41.

26 Esha Ghosh, Seny Kamara, and Roberto Tamassia. Efficient graph encryption scheme for shortest path queries. In Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung, editors, *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, pages 516–525. ACM, 2021. doi:10.1145/3433210.3453099.

27 Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discret. Appl. Math.*, 88(1-3):147–165, 1998. doi:10.1016/S0166-218X(98)00070-5.

28 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi:10.1007/s004530010033.

29 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4), August 2018. doi:10.1145/3199606.

30 Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 444–453. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644179.

31 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 68:1–68:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.68.

32 Lev Reyzin and Nikhil Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett.*, 101(3):98–100, 2007. doi:10.1016/j.ipl.2006.08.013.

33 Guozhen Rong, Wenjun Li, Yongjie Yang, and Jianxin Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theor. Comput. Sci.*, 859:48–56, 2021. doi:10.1016/j.tcs.2021.01.006.

34 Raimund Seidel. Backwards analysis of randomized geometric algorithms. In János Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 37–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993. doi:10.1007/978-3-642-58043-7_3.

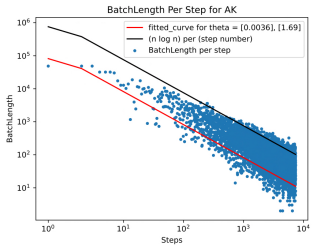
35 G. Tardos. Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A? *Combinatorica*, 9(4):385–392, December 1989. doi:10.1007/BF02125350.

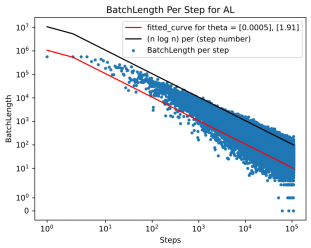
36 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

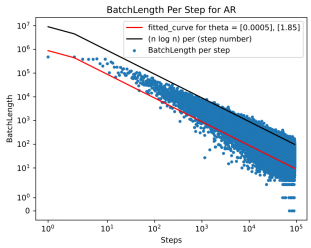
37 M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977. doi:10.1016/0022-5193(77)90351-4.

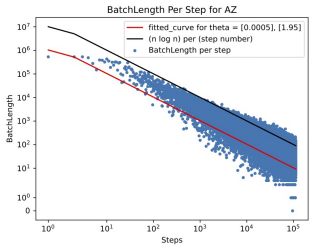
38 Andrew Chi-Chih Yao. Decision tree complexity and Betti numbers. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 615–624, New York, NY, USA, 1994. ACM. doi:10.1145/195058.195414.

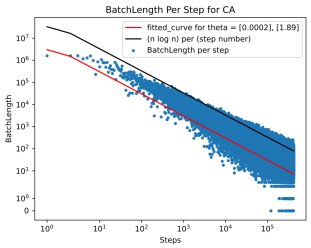
A Batch Length Results for All Datasets

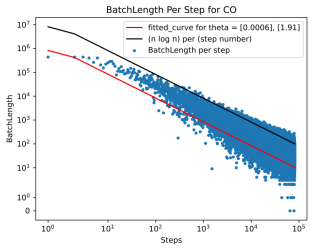


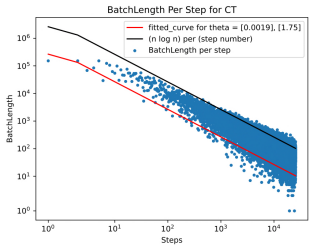


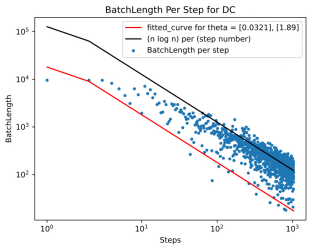


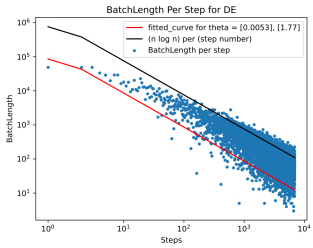


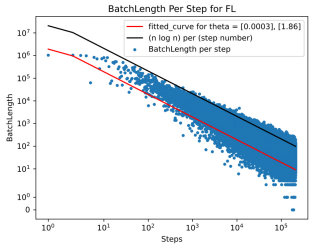


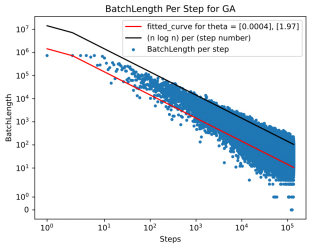


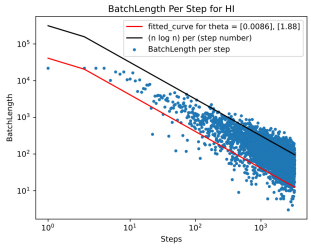


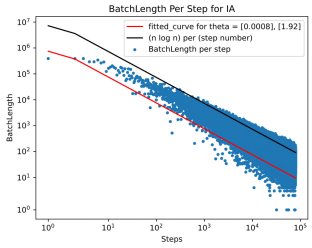


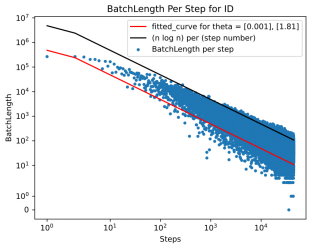


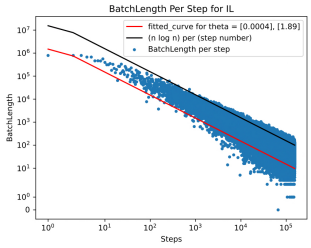


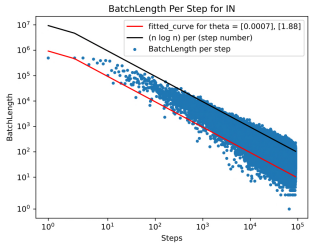


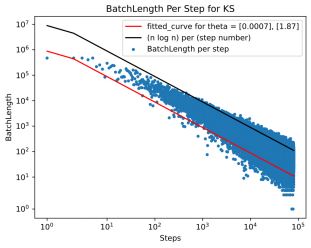


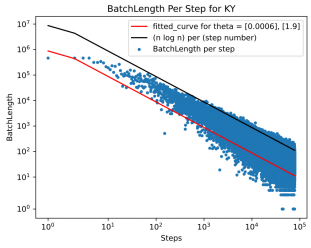


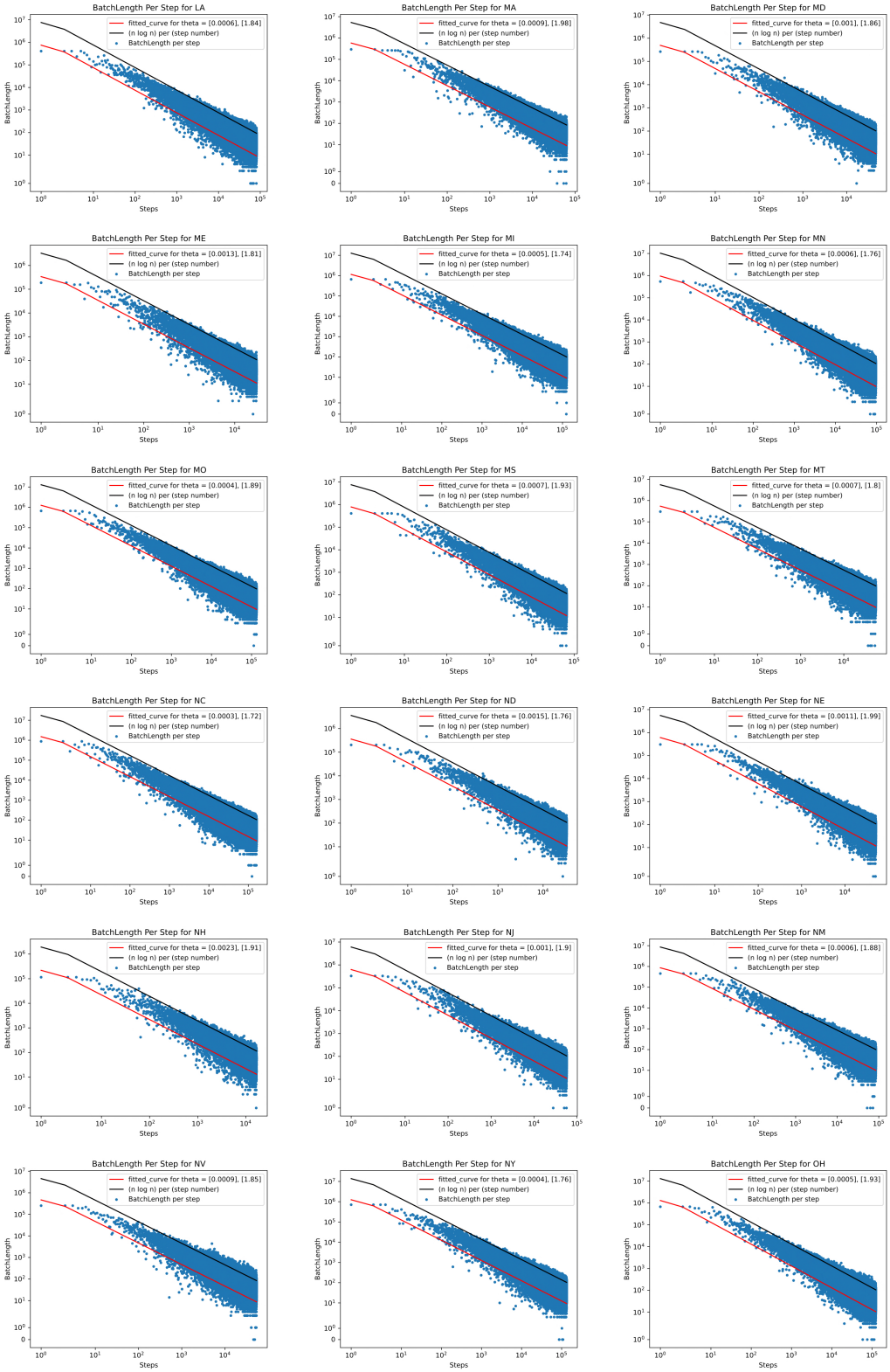


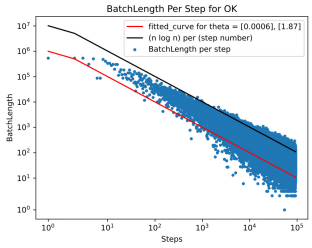


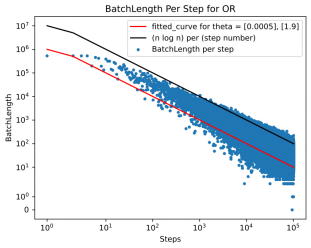


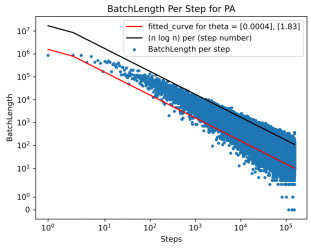


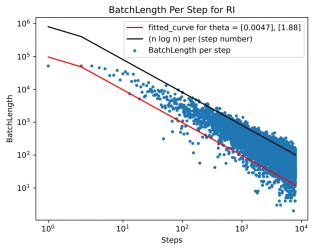


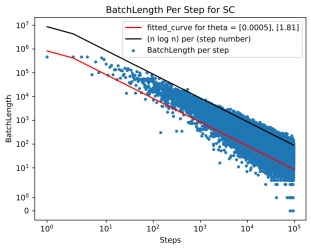

BatchLength Per Step for LA
fitted_curve for theta = [0.0006], [1.84]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MA
fitted_curve for theta = [0.0009], [1.98]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MD
fitted_curve for theta = [0.001], [1.86]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for ME
fitted_curve for theta = [0.0013], [1.81]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MI
fitted_curve for theta = [0.0005], [1.74]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MN
fitted_curve for theta = [0.0006], [1.76]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MO
fitted_curve for theta = [0.0004], [1.89]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MS
fitted_curve for theta = [0.0007], [1.93]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for MT
fitted_curve for theta = [0.0007], [1.8]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NC
fitted_curve for theta = [0.0003], [1.72]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for ND
fitted_curve for theta = [0.0015], [1.76]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NE
fitted_curve for theta = [0.0011], [1.99]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NH
fitted_curve for theta = [0.0023], [1.91]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NJ
fitted_curve for theta = [0.001], [1.9]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NM
fitted_curve for theta = [0.0006], [1.88]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NV
fitted_curve for theta = [0.0009], [1.85]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for NY
fitted_curve for theta = [0.0004], [1.76]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for OH
fitted_curve for theta = [0.0005], [1.93]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

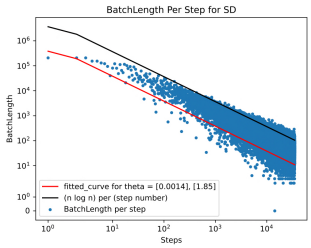

BatchLength Per Step for OK
fitted_curve for theta = [0.0006], [1.87]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

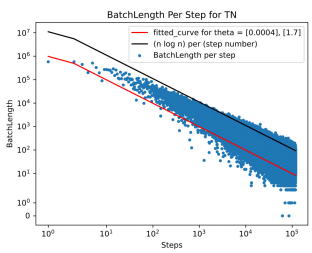

BatchLength Per Step for OR
fitted_curve for theta = [0.0005], [1.9]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

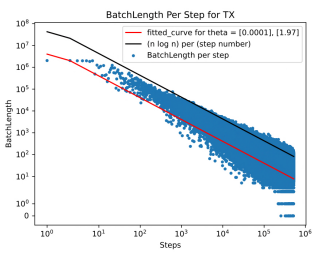

BatchLength Per Step for PA
fitted_curve for theta = [0.0004], [1.83]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

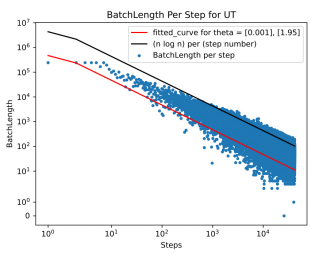

BatchLength Per Step for RI
fitted_curve for theta = [0.0047], [1.88]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

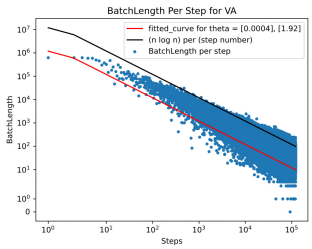

BatchLength Per Step for SC
fitted_curve for theta = [0.0005], [1.81]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

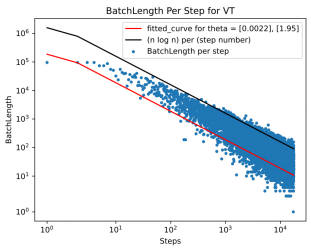

BatchLength Per Step for SD
fitted_curve for theta = [0.0014], [1.85]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

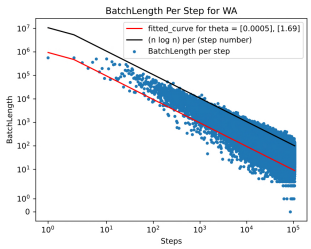

BatchLength Per Step for TN
fitted_curve for theta = [0.0004], [1.7]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

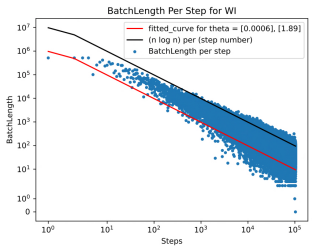

BatchLength Per Step for TX
fitted_curve for theta = [0.0001], [1.97]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

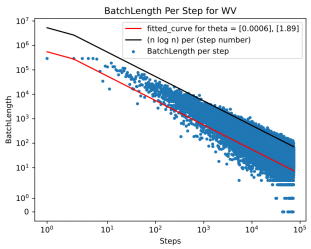

BatchLength Per Step for UT
fitted_curve for theta = [0.001], [1.95]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

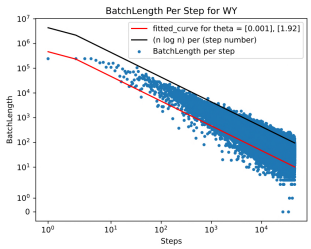

BatchLength Per Step for VA
fitted_curve for theta = [0.0004], [1.92]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

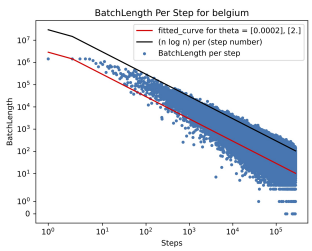

BatchLength Per Step for VT
fitted_curve for theta = [0.0022], [1.95]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

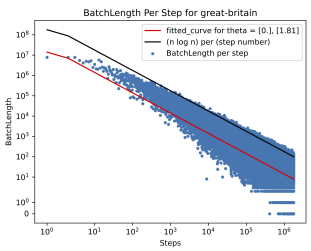

BatchLength Per Step for WA
fitted_curve for theta = [0.0005], [1.69]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

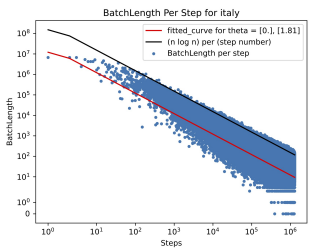

BatchLength Per Step for WI
fitted_curve for theta = [0.0006], [1.89]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps

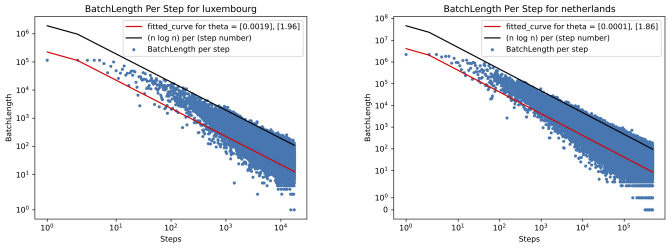

BatchLength Per Step for WV
fitted_curve for theta = [0.0006], [1.89]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps


BatchLength Per Step for WY
fitted_curve for theta = [0.001], [1.92]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps


BatchLength Per Step for belgium
fitted_curve for theta = [0.0002], [2.]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps


BatchLength Per Step for great-britain
fitted_curve for theta = [0.], [1.81]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps


BatchLength Per Step for italy
fitted_curve for theta = [0.], [1.81]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps


BatchLength Per Step for luxembourg
fitted_curve for theta = [0.0019], [1.96]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps
BatchLength Per Step for netherlands
fitted_curve for theta = [0.0001], [1.86]
(n log n) per (step number)
BatchLength per step
BatchLength
Steps