Minimax-optimal semi-supervised regression on unknown manifolds: supplementary material

Amit Moscovich
Weizmann Institute of Science

Ariel Jaffe
Weizmann Institute of Science

Boaz Nadler
Weizmann Institute of Science

A Auxiliary lemmas

Consider a graph G constructed from $\{X_1,\dots,X_N\}$ as described in Section 2 of the main text. For any $\mathbf{x}\in\{X_1,\dots,X_N\}$, denote by $X_G^{(\ell)}(\mathbf{x})$ and $X_{\mathcal{M}}^{(\ell)}(\mathbf{x})$ its closest point from $X_1,\dots X_\ell$ points, according to either the graph or the manifold distance,

$$X_G^{(\ell)}(\mathbf{x}) := \underset{X_j\in\{X_1,\dots,X_\ell\}}{\operatorname{argmin}} d_G(X_i,X_j),$$
$$X_{\mathcal{M}}^{(\ell)}(\mathbf{x}) := \underset{X_j\in\{X_1,\dots,X_\ell\}}{\operatorname{argmin}} d_{\mathcal{M}}(X_i,X_j).$$

Lemma A.1. *Let $X_1,\dots,X_\ell \overset{i.i.d.}{\sim} \mu$ and let $\mathbf{x}\in\mathcal{M}$. Assume that $\mu(B_{\mathbf{x}}(r))\ge Qr^d$ for all $r\le R$. Then for any $\lambda>0$,*

$$\int_0^{\lambda^2R^2} \Pr\left[d_{\mathcal{M}}\left(\mathbf{x},X_{\mathcal{M}}^{(\ell)}(\mathbf{x})\right) > \frac{\sqrt{r}}{\lambda}\right]dr \le \frac{2\lambda^2\ell^{-\frac{2}{d}}}{(1-e^{-Q})^2}.$$

Proof. Substituting $t=\sqrt{r}/\lambda$, the integral becomes

$$\int_0^R \Pr\left[d_{\mathcal{M}}\left(\mathbf{x},X_{\mathcal{M}}^{(\ell)}(\mathbf{x})\right)>t\right]2\lambda^2tdt. \tag{A.1}$$

Now, $d_{\mathcal{M}}(\mathbf{x},X_{\mathcal{M}}^{(\ell)}(\mathbf{x}))>t$ if and only if all ℓ samples fall outside the manifold ball $B_{\mathbf{x}}(t)$. Hence,

$$\Pr\left[d_{\mathcal{M}}(\mathbf{x},X_{\mathcal{M}}^{(\ell)}(\mathbf{x}))>t\right]=(1-\mu(B_{\mathbf{x}}(t)))^\ell.$$

Since $\mu(B_{\mathbf{x}}(t))\ge Qt^d$ for all $t\le R$,

$$\Pr\left[d_{\mathcal{M}}(\mathbf{x},X_{\mathcal{M}}^{(\ell)}(\mathbf{x}))>t\right]\le(1-Qt^d)^\ell\le e^{-Qt^d\ell} \tag{A.2}$$

We insert (A.2) into (A.1), extend the domain of integration to $(0,\infty)$ and split it into intervals of length $\ell^{-\frac{1}{d}}$. Each integral is bounded separately using the fact that t is increasing and $e^{-Qt^d\ell}$ is decreasing.

$$\int_0^R t\cdot e^{-Qt^d\ell}dt < \sum_{k=0}^\infty \int_{k\ell^{-\frac{1}{d}}}^{(k+1)\ell^{-\frac{1}{d}}} t\cdot e^{-Qt^d\ell}dt$$
$$\le \sum_{k=0}^\infty \ell^{-\frac{2}{d}}(k+1)\cdot e^{-k^dQ} \le \ell^{-\frac{2}{d}}\sum_{k=0}^\infty (k+1)\cdot e^{-kQ}$$
$$=\ell^{-\frac{2}{d}}\left(\sum_{k=0}^\infty e^{-kQ}\right)^2 = \frac{\ell^{-\frac{2}{d}}}{(1-e^{-Q})^2}.$$

Inserting this bound into (A.1) concludes the proof. □

Lemma A.2. *Consider a fixed point $\mathbf{x}\in\mathcal{M}$ and let $\mathbf{x}^*=\operatorname{argmin}_{\mathbf{x}'\in\{X_1,\dots,X_{n+m}\}}\|\mathbf{x}-\mathbf{x}'\|$ be its Euclidean nearest neighbor from the sample. We denote by A the event that all the inequalities in Eq. (4) hold. Conditioned on this event,*

$$\mathbb{E}\left[d_{\mathcal{M}}^2(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*))\big|A\right]\le c_1(\mathcal{M},\mu,\delta)n^{-\frac{2}{d}} \tag{A.3}$$

Proof. Assume that A holds. By applying Eq. (4) twice, we obtain an upper bound on the manifold distance to the closest labeled point in the graph.

$$d_{\mathcal{M}}(\mathbf{x}^*,X_{\mathcal{M}}^{(n)}(\mathbf{x}^*))\ge\frac{d_G(\mathbf{x}^*,X_{\mathcal{M}}^{(n)}(\mathbf{x}^*))}{1+\delta} \tag{A.4}$$
$$\ge\frac{d_G(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*))}{1+\delta}\ge\frac{1-\delta}{1+\delta}d_{\mathcal{M}}(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*))$$

Now we apply the well-known equality for a non-negative random variable $\mathbb{E}[X]=\int_0^\infty\Pr[X>r]dr$,

$$\mathbb{E}\left[d_{\mathcal{M}}^2(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*))\big|A\right]$$
$$=\int\Pr\left[d_{\mathcal{M}}^2(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*))>r\big|A\right]dr$$
$$=\int_0^{\operatorname{diam}(\mathcal{M})^2}\Pr\left[d_{\mathcal{M}}\left(\mathbf{x}^*,X_G^{(n)}(\mathbf{x}^*)\right)>\sqrt{r}\big|A\right]dr$$
$$\le\int_0^{\operatorname{diam}(\mathcal{M})^2}\Pr\left[d_{\mathcal{M}}\left(\mathbf{x}^*,X_{\mathcal{M}}^{(n)}(\mathbf{x}^*)\right)>\frac{1-\delta}{1+\delta}\sqrt{r}\bigg|A\right]dr$$
$$=\int_0^{\left(\frac{1+\delta}{1-\delta}\right)^2R^2}\Pr[\dots]dr+\int_{\left(\frac{1+\delta}{1-\delta}\right)^2R^2}^{\operatorname{diam}(\mathcal{M})^2}\Pr[\dots]dr.$$

Lemma A.1 with $\lambda = (1+\delta)/(1-\delta)$ gives a bound on the first integral. For the second integral, the probability inside it is monotonically decreasing, and by Eq. (A.2) it is in particular smaller than $e^{-QR^d n}$. Hence,

$$\mathbb{E}\left[d^2_{\mathcal{M}}(\mathbf{x}^*, X_G^{(n)}(\mathbf{x}^*))|A\right] \leq \frac{2(1+\delta)^2}{(1-\delta)^2(1-e^{-Q})^2} n^{-\frac{2}{d}} + \operatorname{diam}(\mathcal{M})^2 \cdot e^{-QR^d n}.$$

As a function of n, the second summand is negligible with respect to the first. Hence Eq. (A.3) follows. □

B Fast computation of transductive geodesic nearest neighbors

In this section we provide more details regarding the efficient computation of geodesic kNN for all vertices in a graph. We begin by proving the correctness of Algorithm 1 and analyzing its runtime. Then, in Section B.3 we present Algorithm 2 by Har-Peled (2016) which has a tighter bound on the asymptotic running time. Finally, in Section B.4 we test the empirical running time of these algorithms on the simulated WiFi data set.

B.1 Proof of correctness

We start with some auxiliary definitions and lemmas regarding shortest paths. Given an undirected and weighted graph G, we denote paths in G by $v_i \to v_j \to \ldots \to v_k$ or simply $v_i \rightsquigarrow v_k$ when the context makes it clear what path we are referring to. The *length* of a path is the sum of its edge weights,

$$w(v_1 \to v_2 \to \ldots v_m) = \sum_{i=1}^{m-1} w(v_i, v_{i+1}).$$

The length of the shortest (geodesic) path between two vertices $v, v' \in V$ is denoted by $d_G(v, v')$. A path $v \rightsquigarrow v'$ is called a *shortest path* if $w(v \rightsquigarrow v') = d_G(v, v')$. The following lemma is the key to the correctness of Algorithms 1 and 2.

Lemma B.1. *Let $v \in V$ be a vertex and let s be its j-th nearest labeled vertex. If $s \rightsquigarrow u \rightsquigarrow v$ is a shortest path then $s \in NLV(u, j)$, where $NLV(u, j)$ is the set of j nearest labeled vertices to u.*

Proof. Assume by contradiction that there are j labeled nodes $s_1, \ldots, s_j$ such that

$$\forall i : d_G(s_i, u) < d_G(s, u).$$

Then

$$\begin{aligned} d_G(s_i, v) &\leq d_G(s_i, u) + d_G(u, v) && \text{(triangle inequality)} \\ &< d_G(s, u) + d_G(u, v) && \text{(by assumption)} \\ &= d_G(s, v). && \text{(B.1)} \end{aligned}$$

where the last equality is derived from the assumption that u is on a shorted path between s and v. Eq. (B.1) implies that the vertices $s_1, \ldots, s_j$ are all closer than s to v, which contradicts the assumption. □

We now continue to the main part of the proof.

Lemma B.2. *For every triplet$(dist, seed, v_0)$ popped from Q there is a path $seed \rightsquigarrow v_0$ of length $dist$.*

Proof. We prove the claim by induction on the elements inserted into Q.

Base of the induction: The first L inserts correspond to the labeled vertices $\{(0, s, s) : s \in \mathcal{L}\}$. For these triplets the claim holds trivially.

Induction step: Any triplet inserted into Q or updated is of the form $(dist + w(v, v_0), seed, v)$ where $(dist, seed, v_0)$ was previously inserted into Q. Hence, by the induction hypothesis there exists a path $seed \rightsquigarrow v_0$ of length $dist$. Since v is a neighbor of v_0 with edge weight $w(v_0, v)$, there exists a path $seed \rightsquigarrow v_0 \to v$ of length $w(seed \rightsquigarrow v_0) + w(v_0 \to v) = dist + w(v_0, v)$. □

Lemma B.3. *The distances popped from Q in the main loop form a monotone non-decreasing sequence.*

Proof. This follows directly from the fact that Q is a minimum priority queue and that the edge weights are non-negative, hence future insertions or updates will have a priority that is higher or equal to that of an existing element in the queue. □

Lemma B.4. *Every time a triplet $(dist, seed, v_0)$ is popped from Q, the following conditions hold*

1. *If $seed \in NLV(v_0, k)$ then $dist = d_G(seed, v_0)$.*
2. *All pairs $(s, v) \in \mathcal{L} \times V$ that satisfy $d_G(s, v) < dist$ and $s \in NLV(v, k)$ are in the visited set.*

Proof. We prove both claims simultaneously by induction on the popped triplets.

Base of the induction: The first L triplets are equal to $\{(0, s, s) : s \in \mathcal{L}\}$. Part 1 holds since $d_G(s, s) = 0$. Part 2 holds because there are no paths shorter than 0.

Induction step:

Part 1. (If $seed \in NLV(v_0, k)$ then $dist = d_G(seed, v_0)$)

By Lemma B.2, *dist* is the length of an actual path, so it cannot be smaller than the shortest path length $d_G(seed, v_0)$. Assume by contradiction that $d_G(seed, v_0) < dist$. There is some *shortest* path $seed \rightsquigarrow v_p \to v_0$ of length $d_G(seed, v_0)$ (v_p may be equal to *seed*, but not to v_0). Clearly $d_G(seed, v_p) \leq d_G(seed, v_0) < dist$ and by Lemma 3.1, $seed \in NLV(v_p, k)$. Thus by Part 2 of the induction hypothesis $(seed, v_p) \in visited$. This implies that $(seed, v_p)$ was visited in a previous iteration. Note that it follows from $seed \in NLV(v_p, k)$ and from Lemma B.3 that during the visit of $(seed, v_p)$ the condition $length(kNN[v_p]) < k$ was true. Therefore the command dec-or-insert$(Q, d_G(seed, v_p) + w(v_p, v_0), seed, v_0)$ should have been called, but because $seed \rightsquigarrow v_p \to v_0$ is a shortest path, $d_G(seed, v_p) + w(v_p, v_0) = d_G(seed, v_0)$, leading to the conclusion that the triplet $(d_G(seed, v_0), seed, v_0)$ must have been inserted into Q in a previous iteration, which leaves two options:

1. Either $(d_G(seed, v_0), seed, v_0)$ was never popped from Q, in that case it should have been popped from Q in the current iteration instead of $(dist, seed, v_0)$. Contradiction.
2. The triplet $(d_G(seed, v_0), seed, v_0)$ was popped from Q in a previous iteration. However this implies a double visit of $(seed, v_0)$, which is impossible due to the use of the *visited* set.

Part 2. (All pairs $(s, v) \in \mathcal{L} \times V$ that satisfy $d_G(s, v) < dist$ and $s \in NLV(v, k)$ are contained in *visited*)

Let $(s, v) \in \mathcal{L} \times V$ be a pair of vertices that satisfies $d_G(s, v) < dist$ and $s \in NLV(v, k)$. Assume by contradiction that $(s, v) \notin visited$. Let $s \rightsquigarrow v' \to v'' \rightsquigarrow v$ be a shortest path such that $(s, v') \in$ visited but $(s, v'') \notin$ visited. Such v', v'' must exist since $(s, s) \in visited$ and by our assumption $(s, v) \notin visited$. We note that s may be equal to v' and v'' may be equal to v. Since $(s, v') \in visited$, some triplet (d, s, v') was popped from Q in a previous iteration and by Part 1 of the induction hypothesis $d = d_G(s, v')$. By Lemma 3.1 we know that $s \in NLV(v', k)$, therefore during the visit of (s, v') the condition $length(kNN[v']) < k$ was true. Following this, there must have been a call to decrease-key-or-insert$(Q, d_G(s, v') + w(v', v''), s, v'')$. Now,

$$\begin{aligned} & d_G(s, v') + w(v', v'') \\ & = d_G(s, v'') && (s \rightsquigarrow v' \to v'' \text{ is a shortest path}) \\ & \leq d_G(s, v) && (s \rightsquigarrow v'' \text{ is a subpath}) \end{aligned}$$

Hence the pair (s, v'') was previously stored in Q with distance $d_G(s, v'') \leq d_G(s, v) < dist$. Since $(s, v'') \notin$ visited, it should have been present in Q with a smaller distance than (s, v) as key, thus $(d_G(s, v''), s, v'')$ should have been popped instead of $(dist, seed, v_0)$, contradiction. □

Finally, we are ready to state the theorem that Algorithm 1 produces correct output, namely that its output for every vertex $v \in V$ is indeed the set of its k nearest labeled points, as measured by the geodesic graph distance.

Theorem B.1. *Let $G = (V, E, w)$ be a graph with non-negative weights. For every vertex $v \in V$ let $\mathcal{L}_v$ denote the set of labeled vertices in the connected component of v and let $\ell_v = \min\{k, |\mathcal{L}_v|\}$. Then*

1. *Algorithm 1 stops after a finite number of steps.*
2. *Once stopped, for every $v \in V$ the output list $kNN[v]$ is of the form $kNN[v] = [(d_G(s_1, v), s_1), \ldots, (d_G(s_\ell, v), s_{\ell_v})]$ where $s_1, \ldots, s_{\ell_v}$ are the nearest labeled vertices, sorted by their distance to v.*

Proof. Part 1 follows trivially from the fact that every pair $(seed, v_0)$ can be popped at most once and part 2 follows from Lemma B.4 and an induction on the popped triplets (d, s, v). □

B.2 Transductive runtime analysis of Algorithm 1

Lemma B.1 explains why Algorithm 1 can stop exploring the neighbors of vertices whose k nearest labeled neighbors were found. As Theorem B.2 shows, this can lead to dramatic runtime savings.

Theorem B.2. *Given a graph $G = (V, E)$ with n labeled vertices, the runtime of Algorithm 1 is bounded by $O(k|E| + N_p \log |V|)$ where N_p is the total number of pop-minimum operations, which satisfies $N_p \leq \min\{n|V|, k|E|\}$.*

Proof. Recall that in a priority queue based on a Fibonacci heap all operations cost $O(1)$ amortized time except pop-minimum which costs $\log |Q|$. The runtime is dominated by the cost of all pop-minimum operations, plus the total cost of traversing the neighbors of the examined vertices. The latter takes $O(k|E|)$ time. Denote the total number of pop-minimum operations by N_p. We derive two different bounds on N_p. Every time a $(seed, v)$ pair is popped from Q, it is added to the *visited* set, which prevents future insertions of that pair into Q. Hence, each pair $(seed, v) \in \mathcal{L} \times V$ may be popped at most once from Q, which implies that $N_p \leq n|V|$. In addition, N_p is bounded by the number of insertions into Q. First, there are n insertions during the initialization phase. Then, for each vertex $v_0 \in V$, the "if $length(kNN[v_0]) < k$" clause can hold true at most k times for that vertex. Each time, the neighbors of v_0 are examined and up to $deg(v_0)$ neighbors are inserted into Q. This yields the second bound, $N_p \leq n + k|E| = O(k|E|)$. □

B.3 Saving unneeded extractions

Algorithm 1 keeps a priority queue with pairs $(seed, v) \in \mathcal{L} \times V$. However, once a vertex v has been visited from k different seed vertices, we no longer need to process it further. Thus any later pop operations which involve v are a waste of CPU cycles. Conceptually, once v is visited for the k^{th} time, we would like to purge the queue of all pairs (s, v), but this is expensive since every pop operation costs $\log|Q|$. Instead we use an idea by Har-Peled (2016), which we detail in Algorithm 2. For every vertex $v \in V$ we keep a separate priority queue Q_v, which will be disabled once v is visited k times. A global priority queue Q is maintained that keeps the lowest element of each of the local queues Q_v. Now popping an element involves popping some v from Q and then popping Q_v. This is followed by an insert of the new minimum element from Q_v into Q.

Theorem B.3. *Given a graph $G = (V, E)$ with n labeled vertices, the runtime of Algorithm 2 is bounded by $O(k|E| + kV \log|V|)$*

Proof. The runtime of Algorithm 2 is bounded by $O(k|E| + N_p \log|V|)$. Let $v \in V$ be a vertex. Pairs of the form $(seed, v)$ may be popped at most k times. hence $N_p \leq k|V|$. □

An immediate corollary of Theorem B.3 is that for a graph $G = (V, E)$ of bounded degree d, the runtime of Algorithm 2 is $O(k|V|\log|V|)$. In contrast, the runtime of the naïve approach based on multiple Dijkstra runs is $O(n|V|\log|V|)$. Comparing these two formulas, we see that major speedups are obtained in typical cases where $n \gg k$. As we illustrate empirically in the following Section, these speedups are very large in practice, even on graphs of moderate size.

B.4 Empirical runtime comparison

We compare the runtime of Algorithms 1 and 2 to the naïve method of running Dijkstra's algorithm from each of the labeled points. To make the comparison meaningful we implemented all of these algorithms in Python, using a similar programming style and the same heap data structure. The running times were measured for the simulated WiFi signals data set. Figure B.2 shows the relative speedup (measured in sec-

Algorithm 2 Geodesic k nearest labeled neighbors with faster priority queue handling

Input: An undirected weighted graph $G = (V, E, w)$ and a set of labeled vertices $\mathcal{L} \subseteq V$.
Output: For every $v \in V$ a list $kNN[v]$ with the k nearest labeled vertices to v and their distances.

```
$Q \leftarrow$PriorityQueue()
for $v \in V$ do
    $Q_v \leftarrow$PriorityQueue()
    kNN[$v$] $\leftarrow$ Empty-List()
    $S_v \leftarrow \phi$
    if $v \in \mathcal{L}$ then
        insert($Q$, $v$, priority $= 0$)
        insert($Q_v$, $v$, priority $= 0$)

while $Q \neq \phi$ do
    ($v_0$, dist) $\leftarrow$ pop-minimum($Q$)
    (seed, dist) $\leftarrow$ pop-minimum($Q_{v_0}$)
    $S_{v_0} \leftarrow S_{v_0} \cup$ {seed}
    append (seed, dist) to kNN[$v_0$]
    if length(kNN[$v_0$]) $< k$ and $Q_{v_0} \neq \phi$ then
        (newseed, newdist) $\leftarrow$ minimum($Q_{v_0}$)
        insert($Q$, $v_0$, priority $=$ newdist)
    for all $v \in$ neighbors($v_0$) do
        if length(kNN[$v$]) $< k$ and seed $\notin S_v$ then
            decrease-or-insert($Q_v$, seed,
                        priority $=$ dist $+w(v_0, v)$)
            decrease-or-insert($Q$, $v$,
                        priority $=$ dist $+w(v_0, v)$)
```

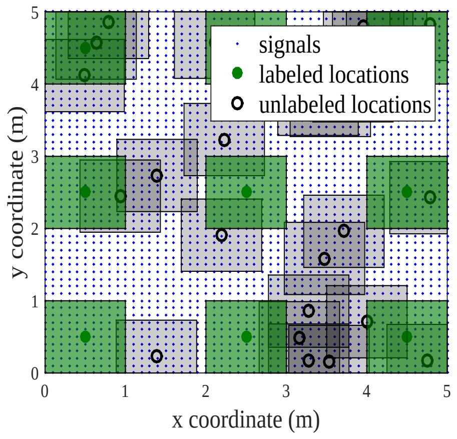

Figure B.1: Schematic of the labeled and unlabeled point generation for the simulated data set. Labeled locations (green circles) were placed on a $4m$ grid, whereas the unlabeled locations (grey circles) were placed at random. The signature of a location is computed by Eq. (C.1) using all the signals in a 1m square neighborhood of the location.

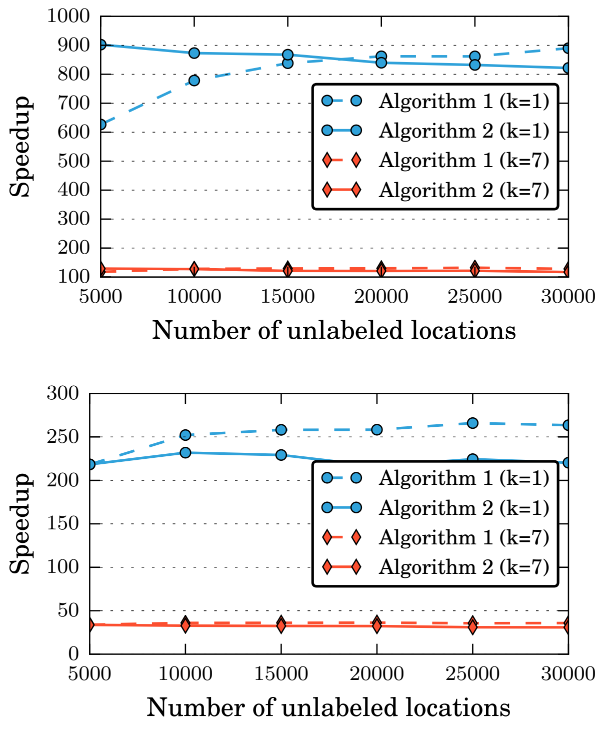

Figure B.2: Relative speedup computing the geodesic 1NN and 7NN of all points in a graph using Algorithms 1 and 2. The speedup of both algorithms is compared to the naïve approach of running Dijkstra's algorithm from each labeled point. Top panel: 1600 labeled locations are placed on a 2m square grid. Bottom panel: 400 labeled locations placed on a 4m grid.

onds) of both algorithms compared to multiple Dijkstra runs. Interestingly, while Algorithm 2 has better asymptotic guarantees than Algorithm 1, both show similar speedups on this particular data set.

Table 1: Runtime of Geodesic 7-NN vs. time to compute Laplacian eigenvectors

#unlabeled	G7NN	Laplacian	Graph build
1000	2.3s	7.6s	9s
10000	7s	195s	76s
100000	56s	114min	66min

Table 1 compares the runtime of the geodesic 7NN method to the runtime of computing the top eigenvectors of the Laplacian matrix, using the simulated indoor localization data set with labeled locations every $2m$. The number of eigenvectors was chosen to be 320, which is equal to 20% of the number of labeled points. Computing the geodesic nearest neighbors by using Algorithm 1 is several orders of magnitude faster than computing the eigenvectors. This is despite the

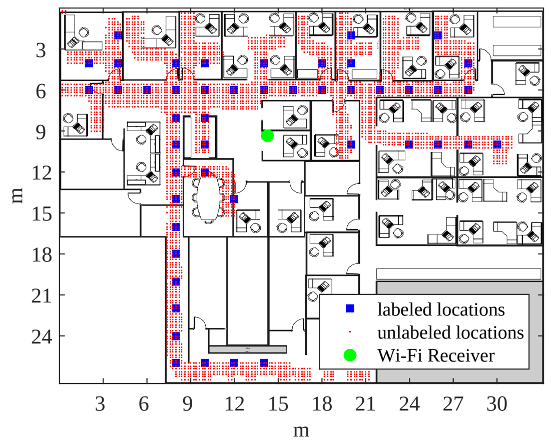

Figure B.3: Schematic of the mapped areas in the real data set

fact that the eigenvector computation is performed using the highly optimized Intel® Math Kernel Library whereas the geodesic nearest neighbor computation uses a simple Python implementation. We expect an efficient implementation of geodesic kNN to be at least 10 times faster. For completeness, the third column shows the graph construction time, using the naïve $O(n^2)$ algorithm.

C Indoor localization details

C.1 Dataset description

Simulated data: This data consists of 802.11 Wi-Fi signals in an artificial yet realistic environment generated by Kupershtein et al. (2013) using a 3D radio wave propagation software. The environment is an $80m \times 80m$ floor. In its center is a Wi-Fi router with $p = 6$ antennas. See Figure 1. At various locations $(x, y) \in \mathbb{R}^2$, $N = 8$ consecutive samples of a Wi-Fi packet's (constant) preamble are recorded, at equally spaced time intervals of $50\mu s$. The samples are stored in a complex-valued vector $s_{x,y} \in \mathbb{C}^{pN}$ which we refer to as the *signal* received from location (x, y). The simulated signals were generated on a dense $0.1m$ grid covering the entire area of the floor.

Real data: This data consists of actual 802.11 signals, recorded by a Wi-Fi router with $p = 6$ antennas placed approximately in the middle of a $27m \times 33m$ office, see Figure B.3. The transmitter was a tablet connected to the router via Wi-Fi. The signal vector of each location (x, y) was sampled $N = 8$ times from every antenna. The transmitter locations were entered manually by the operator. For both the labeled and unlabeled locations, we first generated a square grid covering the area of the office, and then kept only the locations that contained received signals. For the labeled points, we repeated the experiments with several grid sizes ranging from $1.5m$ to $3m$. The location of the WiFi router is marked by the green circle.

C.2 Feature extraction and distance metric for the SSP method

The Signal Subspace Projection method is based on the assumption that signals originating from nearby locations are high dimensional vectors contained in a low dimensional subspace. Hence, signals originating from nearby locations are contained in a low dimensional subspace. The subspace around each location is used as its signature. Specifically, the signature P_ℓ of a location $\ell = (x, y) \in \mathbb{R}^2$ is computed as follows. First, the covariance matrix of the signals in the proximity of ℓ is computed, using all the signals in the dataset that are inside a $1m$ square around ℓ (see Figure B.1),

$$R_\ell := \sum_{\ell' \in \mathbb{R}^2 : \|\ell - \ell'\|_\infty < 0.5m} s_{\ell'} s_{\ell'}^* \tag{C.1}$$

where $s_{\ell'}^*$ denotes the Hermitian transpose of the column vector $s_{\ell'} \in \mathbb{C}^{pN}$. Next, we compute the n_{pc} leading eigenvectors of R_ℓ, forming a matrix $V_\ell \in \mathbb{C}^{pN \times n_{pc}}$. The SSP signature is the projection matrix onto the space spanned by these eigenvectors, $P_\ell := V_\ell V_\ell^*$. In our experiments, we picked $n_{pc} = 10$, though other choices in the range $\{8, \dots, 12\}$ gave results that are almost as good. The distance between pairs of locations is defined as the Frobenius norm of the difference of their projection matrices, $d_{i,j} := \|P_{\ell_\mathrm{i}} - P_{\ell_\mathrm{j}}\|_F$. gave us results that are on par with the best methods of Jaffe and Wax (2014).

For the simulated dataset, to mimic how a real-world semi-supervised localization system might work, we generated the labeled locations on a regular square grid, whereas the unlabeled locations were randomly distributed over the entire area of the floor (see Figure B.1). For the real dataset, due to physical constraints, labeled and unlabeled points were not placed on a regular grid. Then, we created a symmetric kNN graph by connecting every point x_i with its k_G closest neighbors, with corresponding weights given by $w_{i,j} = 1 + \epsilon d_{i,j}$. Here $\epsilon > 0$ is a small constant which gives preference to paths with smaller $d_{i,j}$. Experimentally using the symmetric kNN graph construction with $k = 4$ gave slightly better results than choosing $k \in \{3, 5, 6\}$ for both the geodesic kNN regressor and for the Laplacian eigenvector regressor.

References

Har-Peled, S. (2016). Computing the k Nearest-Neighbors for all Vertices via Dijkstra. *ArXiv e-prints*.

Jaffe, A. and Wax, M. (2014). Single-Site Localization via Maximum Discrimination Multipath Fingerprinting. *IEEE Transactions on Signal Processing*, 62(7):1718–1728.

Kupershtein, E., Wax, M., and Cohen, I. (2013). Single-site emitter localization via multipath fingerprinting. *IEEE Transactions on Signal Processing*, 61(1):10–21.