Simultaneous Nearest Neighbor Search*

Piotr Indyk\textsuperscript{1}, Robert Kleinberg\textsuperscript{2}, Sepideh Mahabadi\textsuperscript{3}, and Yang Yuan\textsuperscript{4}

\textsuperscript{1} MIT, CSAIL, Cambridge, USA  
indyk@mit.edu
\textsuperscript{2} Cornell University, MSR, Ithaca, USA  
rdk@cs.cornell.edu
\textsuperscript{3} MIT, CSAIL, Cambridge, USA  
mahabadi@mit.edu
\textsuperscript{4} Cornell University, Ithaca, USA  
yy528@cornell.edu

Abstract

Motivated by applications in computer vision and databases, we introduce and study the Simultaneous Nearest Neighbor Search (SNN) problem. Given a set of data points, the goal of SNN is to design a data structure that, given a collection of queries, finds a collection of close points that are "compatible" with each other. Formally, we are given \( k \) query points \( Q = q_1, \ldots, q_k \), and a compatibility graph \( G \) with vertices in \( Q \), and the goal is to return data points \( p_1, \ldots, p_k \) that minimize (i) the weighted sum of the distances from \( q_i \) to \( p_i \) and (ii) the weighted sum, over all edges \((i,j)\) in the compatibility graph \( G \), of the distances between \( p_i \) and \( p_j \). The problem has several applications in computer vision and databases, where one wants to return a set of consistent answers to multiple related queries. Furthermore, it generalizes several well-studied computational problems, including Nearest Neighbor Search, Aggregate Nearest Neighbor Search and the 0-extension problem.

In this paper we propose and analyze the following general two-step method for designing efficient data structures for SNN. In the first step, for each query point \( q \), we find its (approximate) nearest neighbor point \( \hat{p} \); this can be done efficiently using existing approximate nearest neighbor structures. In the second step, we solve an off-line optimization problem over sets \( q_1, \ldots, q_k \) and \( \hat{p}_1, \ldots, \hat{p}_k \); this can be done efficiently given that \( k \) is much smaller than \( n \). Even though \( \hat{p}_1, \ldots, \hat{p}_k \) might not constitute the optimal answers to queries \( q_1, \ldots, q_k \), we show that, for the unweighted case, the resulting algorithm satisfies a \( O(\log k/\log \log k) \)-approximation guarantee. Furthermore, we show that the approximation factor can be in fact reduced to a constant for compatibility graphs frequently occurring in practice, e.g., 2D grids, 3D grids or planar graphs.

Finally, we validate our theoretical results by preliminary experiments. In particular, we show that the "empirical approximation factor" provided by the above approach is very close to 1.

1998 ACM Subject Classification F.2.2 Geometrical Problems and Computations

Keywords and phrases Approximate Nearest Neighbor, Metric Labeling, 0-extension, Simultaneous Nearest Neighbor, Group Nearest Neighbor

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.44

* This work was in part supported by NSF grant CCF 1447476 [11] and the Simons Foundation.

© Piotr Indyk, Robert Kleinberg, Sepideh Mahabadi, and Yang Yuan; licensed under Creative Commons License CC-BY.

32nd International Symposium on Computational Geometry (SoCG 2016).
Editors: Sándor Fekete and Anna Lubiw; Article No. 44; pp. 44:1–44:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
Simultaneous Nearest Neighbor Search

1 Introduction

The nearest neighbor search (NN) problem is defined as follows: given a collection $P$ of $n$ points, build a data structure that, given any query point from some set $Q$, reports the data point closest to the query. The problem is of key importance in many applied areas, including computer vision, databases, information retrieval, data mining, machine learning, and signal processing. The nearest neighbor search problem, as well as its approximate variants, have been a subject of extensive studies over the last few decades, see, e.g., [6, 4, 16, 22, 21, 2] and the references therein.

Despite their success, however, the current algorithms suffer from significant theoretical and practical limitations. One of their major drawbacks is their inability to support and exploit structure in query sets that is often present in applications. Specifically, in many applications (notably in computer vision), queries issued to the data structure are not unrelated but instead correspond to samples taken from the same object. For example, queries can correspond to pixels or small patches taken from the same image. To ensure consistency, one needs to impose “compatibility constraints” that ensure that related queries return similar answers. Unfortunately, standard nearest neighbor data structures do not provide a clear way to enforce such constraints, as all queries are processed independently of each other.

To address this issue, we introduce the Simultaneous Nearest Neighbor Search (SNN) problem. Given $k$ simultaneous query points $q_1, q_2, \ldots, q_k$, the goal of a SNN data structure is to find $k$ points (also called labels) $p_1, p_2, \ldots, p_k$ in $P$ such that (i) $p_i$ is close to $q_i$, and (ii) $p_1, \ldots, p_k$ are “compatible”. Formally, the compatibility is defined by a graph $G = (Q, E)$ with $k$ vertices which is given to the data structure, along with the query points $Q = q_1, \ldots, q_k$. Furthermore, we assume that the data set $P$ is a subset of some space $X$ equipped with a distance function $\text{dist}_X$, and that we are given another metric $\text{dist}_Y$ defined over $P \cup Q$. Given the graph $G$ and the queries $q_1, \ldots, q_k$, the goal of the SNN data structure is to return points $p_1, \ldots, p_k$ from $P$ that minimize the following function:

$$
\sum_{i=1}^{k} \kappa_i \text{dist}_Y(p_i, q_i) + \sum_{(i,j) \in E} \lambda_{i,j} \text{dist}_X(p_i, p_j)
$$

(1)

where $\kappa_i$ and $\lambda_{i,j}$ are parameters defined in advance.

The above formulation captures a wide variety of applications that are not well modeled by traditional NN search. For example, many applications in computer vision involve computing nearest neighbors of pixels or image patches from the same image [14, 8, 5]. In particular, algorithms for tasks such as de-noising (removing noise from an image), restoration (replacing a deleted or occluded part of an image) or super-resolution (enhancing the resolution of an image) involve assigning “labels” to each image patch\(^1\). The labels could correspond to the pixel color, the enhanced image patch, etc. The label assignment should have the property that the labels are similar to the image patches they are assigned to, while at the same time the labels assigned to nearby image patches should be similar to each other. The objective function in Equation 1 directly captures these constraints.

From a theoretical perspective, Simultaneous Nearest Neighbor Search generalizes several well-studied computational problems, notably the Aggregate Nearest Neighbor problem [27, 

\(^1\) This problem has been formalized in the algorithms literature as the metric labeling problem [19]. The problem considered in this paper can thus be viewed as a variant of metric labeling with a very large number of labels.
and the 0-extension problem [18, 10, 9, 3]. The first problem is quite similar to the basic nearest neighbor search problem over a metric \( \text{dist} \), except that the data structure is given \( k \) queries \( q_1, \ldots, q_k \), and the goal is to find a data point \( p \) that minimizes the sum \( \sum_i \text{dist}(q_i, p) \). This objective can be easily simulated in SNN by setting \( \text{dist}_Y = \text{dist} \) and \( \text{dist}_X = L \cdot \text{uniform} \), where \( L \) is a very large number and \( \text{uniform}(p, q) \) is the uniform metric. The 0-extension problem is a combinatorial optimization problem where the goal is to minimize an objective function quite similar to that in Equation 1. The exact definition of 0-extension as well as its connections to SNN are discussed in detail in Section 2.1.

1.1 Our results

In this paper we consider the basic case where \( \text{dist}_X = \text{dist}_Y \) and \( \lambda_{i, j} = \kappa_i = 1 \); we refer to this variant as the unweighted case. Our main contribution is a general reduction that enables us to design and analyze efficient data structures for unweighted SNN. The algorithm (called Independent Nearest Neighbors or INN) consists of two steps. In the first (pruning) step, for each query point \( q_i \) we find its nearest neighbor\(^2\) point \( \hat{p}_i \); this can be done efficiently using existing nearest neighbor search data structures. In the second (optimization) step, we run an appropriate (approximation) algorithm for the SNN problem over sets \( q_1, \ldots, q_k \) and \( \hat{p}_1, \ldots, \hat{p}_k \); this can be done efficiently given that \( k \) is much smaller than \( n \). We show that the resulting algorithm satisfies a \( O(b \log k / \log \log k) \)-approximation guarantee, where \( b \) is the approximation factor of the algorithm used in the second step. This can be further improved to \( O(b \delta) \), if the metric space \( \text{dist} \) admits a \( \delta \)-padding decomposition (see Preliminaries for more detail). The running time incurred by this algorithm is bounded by the cost of \( k \) nearest neighbor search queries in a data set of size \( n \) plus the cost of the approximation algorithm for the 0-extension problem over an input of size \( k \). By plugging in the best nearest neighbor algorithms for \( \text{dist} \) we obtain significant running time savings if \( k \ll n \).

We note that INN is somewhat similar to the belief propagation algorithm for super-resolution described in [14]. Specifically, that algorithm selects 16 closest labels for each \( q_i \), and then chooses one of them by running a belief propagation algorithm that optimizes an objective function similar to Equation 1. However, we note that the algorithm in [14] is heuristic and is not supported by approximation guarantees.

We complement our upper bound by showing that the aforementioned reduction inherently yields super-constant approximation guarantee. Specifically, we show in the full version that, for an appropriate distance function \( \text{dist} \), queries \( q_1, \ldots, q_k \), and a label set \( P \), the best solution to SNN with the label set restricted to \( \hat{p}_1, \ldots, \hat{p}_k \) can be \( \Theta(\sqrt{\log k}) \) times larger than the best solution with label set equal to \( P \). This means that even if the second step problem is solved to optimality, reducing the set of labels from \( P \) to \( \hat{P} \) inherently increases the cost by a super-constant factor.

However, we further show that the aforementioned limitation can be overcome if the compatibility graph \( G \) has pseudoarboricity \( r \) (which means that each edge can be mapped to one of its endpoint vertices such that at most \( r \) edges are mapped to each vertex). Specifically, we show that if \( G \) has pseudoarboricity \( r \), then the gap between the best solution using labels in \( P \), and the best solution using labels in \( \hat{P} \), is at most \( O(r) \). Since many graphs used in practice do in fact satisfy \( r = O(1) \) (e.g., 2D grids, 3D grids or planar graphs), this means that the gap is indeed constant for a wide collection of common compatibility graphs.

\(^2\) Other aggregate functions, such as the maximum, are considered as well.

\(^3\) Our analysis immediately extends to the case where the we compute approximate, not exact, nearest neighbors. For simplicity we focus only on the exact case in the following discussion.
In Appendix A we also present an alternative algorithm for the $r$-pseudoarboricity case. Similarly to INN, the algorithm computes the nearest label to each query $q_i$. However, the distance function used to compute the nearest neighbor involves not only the distance between $q_i$ and a label $p$, but also the distances between the neighbors of $q_i$ in $G$ and $p$. This nearest neighbor operation can be implemented using any data structure for the Aggregate Nearest Neighbor problem [27, 25, 24, 1, 20]. Although this results in a more expensive query time, the labeling computed by this algorithm is final, i.e., there is no need for any additional postprocessing. Furthermore, the pruning gap (and therefore the final approximation ratio) of the algorithm is only $2r + 1$, which is better than our bound for INN.

Finally, we validate our theoretical results by preliminary experiments comparing our SNN data structure with an alternative (less efficient) algorithm that solves the same optimization problem using the full label set $P$. In our experiments we apply both algorithms to an image denoising task and measure their performance using the objective function (1). In particular, we show that the “empirical gap” incurred by the above approach, i.e., the ratio of objective function values observed in our experiments, is very close to 1.

1.2 Our techniques

We start by pointing out that SNN can be reduced to 0-extension in a “black-box” manner. Unfortunately, this reduction yields an SNN algorithm whose running time depends on the size of labels $n$, which could be very large; essentially this approach defeats the goal of having a data structure solving the problem. The INN algorithm overcomes this issue by reducing the number of labels from $n$ to $k$. However the pruning step can increase the cost of the best solution. The ratio between the optimum cost after pruning to the optimum cost before pruning is called the pruning gap.

To bound the pruning gap, we again resort to existing 0-extension algorithms, albeit in a “grey box” manner. Specifically, we observe that many algorithms, such as those in [9, 3, 10, 23], proceed by first creating a label assignment in an “extended” metric space (using a LP relaxation of 0-extension), and then apply a rounding algorithm to find an actual solution. The key observation is that the correctness of the rounding step does not rely on the fact that the initial label assignment is optimal, but instead it works for any label assignment. We use this fact to translate the known upper bounds for the integrality gap of linear programming relaxations of 0-extension into upper bounds for the pruning gap. On the flip side, we show a lower bound (which will appear in the full version) for the pruning gap by mimicking the arguments used in [9] to lower bound the integrality gap of a 0-extension relaxation.

To overcome the lower bound, we consider the case where the compatibility graph $G$ has pseudoarboricity $r$. Many graphs used in applications, such as 2D grids, 3D grids or planar graphs, have pseudoarboricity $r$ for some constant $r$. We show that for such graphs the pruning gap is only $O(r)$. The proof proceeds by directly assigning labels in $P$ to the nodes in $Q$ and bounding the resulting cost increase. It is worth noting that the “grey box” approach outlined in the preceding paragraph, combined with Theorem 11 of [9], yields an $O(r^3)$ pruning gap for the class of $K_{r,r}$-minor-free graphs, whose pseudoarboricity is $O(1)$. Our $O(r)$ pruning gap not only improves this $O(r^3)$ bound in a quantitative sense, but it also applies to a much broader class of graphs. For example, three-dimensional grid graphs have pseudoarboricity 6, but the class of three-dimensional grid graphs includes graphs with $K_{r,r}$ minors for every positive integer $r$.

Finally, we validate our theoretical results by experiments. We focus on a simple denoising scenario where $X$ is the pixel color space, i.e., the discrete three-dimensional space...
Each pixel in this space is parametrized by the intensity of the red, green and blue colors. We use the Euclidean norm to measure the distance between two pixels.

We also let $P = X$. We consider three test images: a cartoon with an MIT logo and two natural images. For each image we add some noise and then solve the SNN problems for both the full color space $P$ and the pruned color space $\hat{P}$. Note that since $P = X$, the set of pruned labels $\hat{P}$ simply contains all pixels present in the image.

Unfortunately, we cannot solve the problems optimally, since the best known exact algorithm takes exponential time. Instead, we run the same approximation algorithm on both instances and compare the solutions. We find that the values of the objective function for the solutions obtained using pruned labels and the full label space are equal up to a small multiplicative factor. This suggests that the empirical value of the pruning gap is very small, at least for the simple data sets that we considered.

2 Definitions and Preliminaries

We define the Unweighted Simultaneous Nearest Neighbor problem as follows. Let $(X, \text{dist})$ be a metric space and let $P \subseteq X$ be a set of $n$ points from the space.

**Definition 1.** In the Unweighted Simultaneous Nearest Neighbor problem, the goal is to build a data structure over a given point set $P$ that supports the following operation. Given a set of $k$ points $Q = \{q_1, \cdots, q_k\}$ in the metric space $X$, along with a graph $G = (Q, E)$ of $k$ nodes, the goal is to report $k$ (not necessarily unique) points from the database $p_1, \cdots, p_k \in P$ which minimize the following cost function:

$$\sum_{i=1}^{k} \text{dist}(p_i, q_i) + \sum_{(q_i,q_j) \in E} \text{dist}(p_i, p_j)$$

We refer to the first term in sum as the nearest neighbor (NN) cost, and to the second sum as the pairwise (PW) cost. We denote the cost of the optimal assignment from the point set $P$ by $\text{Cost}(Q, G, P)$.

In the rest of this paper, simultaneous nearest neighbor (SNN) refers to the unweighted version of the problem (unless stated otherwise). Next, we define the pseudoarboricity of a graph and $r$-sparse graphs.

**Definition 2.** Pseudoarboricity of a graph $G$ is defined to be the minimum number $r$, such that the edges of the graph can be oriented to form a directed graph with out-degree at most $r$. In this paper, we call such graphs as $r$-sparse.

Note that given an $r$-sparse graph, one can map the edges to one of its endpoint vertices such that there are at most $r$ edges mapped to each vertex. The doubling dimension of a metric space is defined as follows.

**Definition 3.** The doubling dimension of a metric space $(X, \text{dist})$ is defined to be the smallest $\delta$ such that every ball in $X$ can be covered by $2^\delta$ balls of half the radius.

It is known that the doubling dimension of any finite metric space is $O(\log |X|)$. We then define padding decompositions.

**Definition 4.** A metric space $(X, \text{dist})$ is $\delta$-padded decomposable if for every $r$, there is a randomized partitioning of $X$ into clusters $C = \{C_i\}$ such that, each $C_i$ has diameter at most $r$, and that for every $x_1, x_2 \in X$, the probability that $x_1$ and $x_2$ are in different clusters is at most $\delta \text{dist}(x_1, x_2)/r$. 
Simultaneous Nearest Neighbor Search

It is known that any finite metric with doubling dimension $\delta$ admits an $O(\delta)$-padding decomposition [15].

2.1 0-Extension Problem

The 0-extension problem, first defined by Karzanov [18] is closely related to the Simultaneous Nearest Neighbor problem. In the 0-extension problem, the input is a graph $G(V, E)$ with a weight function $w(e)$, and a set of terminals $T \subseteq V$ with a metric $d$ defined on $T$. The goal is to find a mapping from the vertices to the terminals $f : V \rightarrow T$ such that each terminal is mapped to itself and that the following cost function is minimized:

$$\sum_{(u,v) \in E} w(u,v) \cdot d(f(u), f(v))$$

It can be seen that this is a special case of the metric labeling problem [19] and thus a special case of the general version of the SNN problem defined by Equation 1. To see this, it is enough to let $Q = V$ and $P = T$, and let $\kappa_i = \infty$ for $q_i \in T$, $\kappa_i = 0$ for $q_i \notin T$, and $\lambda_{i,j} = w(i, j)$ in Equation 1.

Calinescu et al. [9] considered the semimetric relaxation of the LP for the 0-extension problem and gave an $O(\log |T|)$ algorithm using randomized rounding of the LP solution. They also proved an integrality ratio of $O(\sqrt{\log |T|})$ for the semimetric LP relaxation. Later Fakcharoenphol et al. [10] improved the upper-bound to $O(\log |T| / \log \log |T|)$, and Lee and Naor [23] proved that if the metric $d$ admits a $\delta$-padded decomposition, then there is an $O(\delta)$-approximation algorithm for the 0-extension problem. For the finite metric spaces, this gives an $O(\delta)$ algorithm where $\delta$ is the doubling dimension of the metric space. Furthermore, the same results can be achieved using another metric relaxation (earth-mover relaxation), see [3]. Later Karloff et al. [17] proved that there is no polynomial time algorithm for 0-extension problem with approximation factor $O((\log n)^{1/4-\epsilon})$ unless $NP \subseteq DTIME(n^{poly(log n)})$.

SNN can be reduced to 0-extension in a “black-box” manner via the following lemma whose proof will appear in the full version.

▶ Lemma 5. Any $b$-approximate algorithm for the 0-extension problem yields an $O(b)$-approximate algorithm for the SNN problem.

By plugging in the known 0-extension algorithms cited earlier we obtain the following:

▶ Corollary 6. There exists an $O(\log n / \log \log n)$ approximation algorithm for the SNN problem with running time $n^{O(1)}$, where $n$ is the size of the label set.

▶ Corollary 7. If the metric space $(X, \text{dist})$ is $\delta$-padded decomposable, then there exists an $O(\delta)$ approximation algorithm for the SNN problem with running time $n^{O(1)}$. For finite metric spaces $X$, $\delta$ could represent the doubling dimension of the metric space (or equivalently the doubling dimension of $P \cup Q$).

Unfortunately, this reduction yields a SNN algorithm with running time depending on the size of labels $n$, which could be very large. In the next section we show how to improve the running time by reducing the labels set size from $n$ to $k$. However, unlike the reduction in this section, our new reduction will no longer be “black-box”. Instead, its analysis will use particular properties of the 0-extension algorithms. Fortunately those properties are satisfied by the known approximation algorithms for this problem.
we have
\[ \alpha \]
We define the pruning gap of the INN algorithm, where
\[ b \]
would show that for the sparse graphs, the pruning gap remains constant. Finally, in the section we define the
\[ \alpha \]
we call
\[ \Omega(\sqrt{\log k}) \] for the set
\[ \hat{p} \]
for the set
\[ P \]
and if the metric
\[ G \]
finite metric spaces
\[ \Omega(\sqrt{\log k}) \] and as large as
\[ r \] is the pseudoarboricity of the graph
\[ Q,G,\hat{P} \]
We define the pruning gap of the INN algorithm, \( \alpha \), as the largest value of \( \alpha(Q,G,\hat{P}) \) over all instances.

First, in Section 3.1, by proving a reduction from algorithms for rounding the LP solution of the 0-extension problem, we show that for arbitrary graphs \( G \), we have
\[ \alpha = \Theta(\log k / \log \log k) \]
and if the metric \( (X, \text{dist}) \) is \( \delta \)-padded decomposable, we have
\[ \alpha = \Theta(\delta) \] (for example, for finite metric spaces \( X, \delta \) can represent the doubling dimension of the metric space). Then, in Section 3.2, we prove that \( \alpha = \Theta(r) \) where \( r \) is the pseudoarboricity of the graph \( G \). This would show that for the sparse graphs, the pruning gap remains constant. Finally, in the full version, we present a lower bound showing that the pruning gap could be as large as
\[ \Omega(\sqrt{\log k}) \] as large as
\[ \Theta(r) \] for \( r \leq \sqrt{\log k} \). Therefore, we get the following theorem.

\[ \text{Theorem 9.} \] The following bounds hold for the pruning gap of the INN algorithm. First we have
\[ \alpha = \Theta(\log k / \log \log k) \], and if metric \( (X, \text{dist}) \) is \( \delta \)-padded decomposable, we have
\[ \alpha = \Theta(\delta) \]. Second, \( \alpha = \Theta(r) \) where \( r \) is the pseudoarboricity of the graph \( G \). Finally, we have that
\[ \alpha = \Omega(\sqrt{\log k}) \] and \( \alpha = \Omega(r) \) for \( r \leq \sqrt{\log k} \).

Note that the above theorem results in an \( O(b \cdot \alpha) \) time algorithm for the SNN problem where \( b \) is the approximation factor of the algorithm used to solve the metric labeling problem for the set \( \hat{P} \), as noted in line 4 of the INN algorithm. For example in a general graph \( b \) would be \( O(\log k / \log \log k) \) that is added on top of \( O(\alpha) \) approximation of the pruning step.

3.1 Bounding the pruning gap using 0-extension

In this section we show upper bounds for the pruning gap \( (\alpha) \) of the INN algorithm. The proofs use specific properties of existing algorithms for the 0-extension problem.

\[ \text{Definition 10.} \] We say an algorithm \( A \) for the 0-extension problem is a \( \beta \)-natural rounding algorithm if, given a graph \( G = (V, E) \), a set of terminals \( T \subseteq V \), a metric space \( (X, d_X) \), and a mapping \( \mu : V \rightarrow X \), it outputs another mapping \( \nu : V \rightarrow X \) with the following properties:
∀t ∈ T: ν(t) = µ(t)
∀v ∈ V: ∃t ∈ T s.t. ν(v) = µ(t)
Cost(ν) ≤ βCost(µ), i.e., \( \sum_{(u,v) \in E} d_X(ν(u), ν(v)) \) ≤ \( \beta \cdot \sum_{(u,v) \in E} d_X(µ(u), µ(v)) \)

Many previous algorithms for the 0-extension problem, such as [9, 3, 10, 23], first create the mapping \( µ \) using some LP relaxation of 0-extension (such as semimetric relaxation or earth-mover relaxation), and then apply a \( β \)-natural rounding algorithm for the 0-extension to find the mapping \( ν \) which yields the solution to the 0-extension problem. Below we give a formal connection between guarantees of these rounding algorithms, and the quality of the output of the INN algorithm (the pruning gap of INN).

Lemma 11. Let \( A \) be a \( β \)-natural rounding algorithm for the 0-extension problem. Then we can infer that the pruning gap of the INN algorithm is \( O(β) \), that is, \( α = O(β) \).

Proof. Fix any SNN instance \((Q, G_S, P)\), where \( G_S = (Q, E_{PW}) \), and its corresponding INN invocation.

We construct the inputs to the algorithm \( A \) from the INN instance as follows. Let the metric space of \( A \) be the same as \((X, dist)\) defined in the SNN instance. Also, let \( V \) be a set of \( 2k \) vertices corresponding to \( \hat{P} \cup P^* \) with \( T \) corresponding to \( \hat{P} \). Here \( P^* = \{p^*_1, \cdots, p^*_k\} \) is the set of the optimal solutions of SNN, and \( \hat{P} \) is the set of nearest neighbors as defined by INN. The mapping \( µ \) simply maps each vertex from \( V = \hat{P} \cup P^* \) to itself in the metric \( X \) defined in SNN. Moreover, the graph \( G = (V, E) \) is defined such that \( E = \{(\hat{p}_i, p^*_i) | 1 ≤ i ≤ k\} \cup \{(p^*_i, p^*_j) | (q_i, q_j) \in E_{PW}\} \).

First we claim the following (note that \( Cost(µ) \) is defined in Definition 10, and that by definition \( Cost(Q, G_S, P) = Cost(Q, G_S, P^*) \))

\[
Cost(µ) \leq 2Cost(Q, G_S, P^*) = 2Cost(Q, G_S, P)
\]

We know that \( Cost(Q, G_S, P^*) \) can be split into NN cost and PW cost. We can also split \( Cost(µ) \) into NN cost (corresponding to edge set \( \{(\hat{p}_i, p^*_i) | 1 ≤ i ≤ k\} \)) and PW cost (corresponding to edge set \( \{(p^*_i, p^*_j) | (q_i, q_j) \in E_{PW}\} \)). By definition we know the PW costs of \( Cost(Q, G_S, P) \) and \( Cost(µ) \) are equal. For NN cost, by triangle inequality, we know \( dist(\hat{p}_i, p^*_i) ≤ dist(\hat{p}_i, q_i) + dist(q_i, p^*_i) ≤ 2 \cdot dist(q_i, p^*_i) \). Here we use the fact that \( \hat{p}_i \) is the nearest database point of \( q_i \). Thus, the claim follows.

We then apply algorithm \( A \) to get the mapping \( ν \). By the assumption on \( A \), we know that \( Cost(ν) ≤ βCost(µ) \). Given the mapping \( ν \) by the algorithm \( A \), consider the assignment in the SNN instance where each query \( q_i \) is mapped to \( ν(p^*_i) \), and note that since \( ν(p^*_i) \in T \), this would map all points \( q_i \) to points in \( \hat{P} \). Thus, by definition, we have that

\[
Cost(Q, G_S, \hat{P}) ≤ \sum_{i=1}^{k} dist(q_i, ν(p^*_i)) + \sum_{(q_i, q_j) \in E_{PW}} dist(ν(p^*_i), ν(p^*_j)) ≤ \sum_{i=1}^{k} dist(q_i, \hat{p}_i) + \sum_{i=1}^{k} dist(\hat{p}_i, ν(p^*_i)) + \sum_{(q_i, q_j) \in E_{PW}} dist(ν(p^*_i), ν(p^*_j)) ≤ \sum_{i=1}^{k} dist(q_i, \hat{p}_i) + Cost(ν) ≤ Cost(Q, G_S, P) + βCost(µ) ≤ (2β + 1)Cost(Q, G_S, P)
\]

where we have used the triangle inequality. Therefore, we have that the pruning gap \( α \) of the INN algorithm is \( O(β) \), as claimed.
Algorithm 2 \( r \)-Sparse Graph Assignment Algorithm

**Input** Query points \( q_1, \ldots, q_k \), Optimal assignment \( p_1^*, \ldots, p_k^* \), Nearest Neighbors \( \hat{p}_1, \ldots, \hat{p}_k \), and the input graph \( G = (Q,E) \)

**Output** An Assignment \( p_1, \ldots, p_k \in \hat{P} \)

1: for \( i = 1 \) to \( k \) do
2: Let \( j_0 = i \) and let \( q_{j_1}, \ldots, q_{j_r} \) be all the neighbors of \( q_i \) in the graph \( G \)
3: \( m \leftarrow \arg \min_{\ell=0}^{i} \text{dist}(p_{j_\ell}^*, p_{j_{\ell+1}}^*) + \text{dist}(p_{j_{\ell+1}}^*, q_{j_{\ell+1}}) \)
4: Assign \( p_i \leftarrow \hat{p}_{j_m} \)
5: end for

Using the previously cited results, and noting that in the above instance \( |V| = O(k) \), we get the following corollaries.

- **Corollary 12.** The INN algorithm has pruning gap \( \alpha = O(\log k / \log \log k) \).

- **Corollary 13.** If the metric space \( (X, \text{dist}) \) admits a \( \delta \)-padding decomposition, then the INN algorithm has pruning gap \( \alpha = O(\delta) \). For finite metric spaces \( (X, \text{dist}) \), \( \delta \) is at most the doubling dimension of the metric space.

### 3.2 Sparse Graphs

In this section, we prove that the INN algorithm performs well on sparse graphs. More specifically, here we prove that when the graph \( G \) is \( r \)-sparse, then \( \alpha(Q,G,P) = O(r) \). To this end, we show that there exists an assignment using the points in \( \hat{P} \) whose cost function is within \( O(r) \) of the optimal solution using the points in the original data set \( P \).

Given a graph \( G \) of pseudoarboricity \( r \), we know that we can map each edge to one of its end points such that the number of edges mapped to each vertex is at most \( r \). For each edge \( e \), we call the vertex that \( e \) is mapped to as the corresponding vertex of \( e \). This would mean that each vertex has a corresponding vertex of at most \( r \) edges.

Let \( p_1^*, \ldots, p_k^* \in P \) denote the optimal solution of SNN. Algorithm 2 shows how to find an assignment \( p_1, \ldots, p_k \in \hat{P} \). We show that the cost of this assignment is within a factor \( O(r) \) from the optimum.

- **Lemma 14.** The assignment defined by Algorithm 2, has \( O(r) \) approximation factor.

**Proof.** For each \( q_i \in Q \), let \( y_i = \text{dist}(p_i^*, q_i) \) and for each edge \( e = (q_i, q_j) \in E \) let \( x_e = \text{dist}(p_i^*, p_j^*) \). Also let \( Y = \sum_{i=1}^{k} y_i \) and \( X = \sum_{e \in E} x_e \). Note that \( Y \) is the NN cost and \( X \) is the PW cost of the optimal assignment and that \( \text{OPT} = \text{Cost}(Q,G,P) = X + Y \). Define the variables \( y'_i, x'_e, Y', X' \) in the same way but for the assignment \( p_1, \ldots, p_k \) produced by the algorithm. That is, for each \( q_i \in Q \), \( y'_i = \text{dist}(p_i, q_i) \), and for each edge \( e = (q_i, q_j) \in E \), \( x'_e = \text{dist}(p_i, p_j) \). Moreover, for a vertex \( q_i \), we define the designated neighbor of \( q_i \) to be \( q_{j_m} \) for the value of \( m \) defined in the line 3 of Algorithm 2 (note that the designated neighbor might be the vertex itself). Fix a vertex \( q_i \) and let \( \hat{p}_c \) be the designated neighbor of \( q_i \). We can bound the value of \( y'_i \) as follows.

\[
y'_i = \text{dist}(q_i, p_i) = \text{dist}(q_i, \hat{p}_c) \\
\leq \text{dist}(q_i, p_i^*) + \text{dist}(p_i^*, \hat{p}_c) + \text{dist}(p_i^*, q_c) + \text{dist}(q_c, \hat{p}_c) \quad \text{(by triangle inequality)} \\
\leq y_i + \text{dist}(p_i^*, \hat{p}_c^*) + 2\text{dist}(p_i^*, q_c) \quad \text{(since } \hat{p}_c \text{ is the nearest neighbor of } q_c) \\
\leq y_i + 2[\text{dist}(p_i^*, \hat{p}_c^*) + \text{dist}(p_i^*, q_c])] \\
\leq 3y_i \quad \text{(by definition of designated neighbor and the value } m \text{ in line 3 of Algorithm 2)}
\]
Thus summing over all vertices, we get that $Y' \leq 3Y$. Now for any fixed edge $e = (q_i, q_s)$ (with $q_i$ being its corresponding vertex), let $q_c$ be the designated neighbor of $q_i$, and $q_z$ be the designated neighbor of $q_s$. Then we bound the value of $x'_e$ as follows.

$$
x'_e = \text{dist}(p_i, p_s) = \text{dist}(\hat{p}_c, \hat{p}_z) \quad (\text{by definition of designated neighbor and line 4 of Algorithm 2})
\leq \text{dist}(\hat{p}_c, q_c) + \text{dist}(q_c, p_s^*) + \text{dist}(p_s^*, p_i^*) + \text{dist}(p_i^*, p_s^*)
+ \text{dist}(p_s^*, p_z^*) + \text{dist}(p_z^*, q_s) + \text{dist}(q_s, \hat{p}_z) \quad (\text{by triangle inequality})
\leq 2\text{dist}(q_c, p_s^*) + \text{dist}(p_s^*, p_i^*) + \text{dist}(p_i^*, p_s^*)
+ \text{dist}(p_s^*, p_z^*) + 2\text{dist}(p_z^*, q_s) \quad (\text{since } \hat{p}_c, \hat{p}_z \text{ respectively is a NN of } q_i, q_s \text{ respectively})
\leq 2[\text{dist}(q_c, p_s^*) + \text{dist}(p_s^*, p_i^*) + \text{dist}(p_i^*, p_s^*) + \text{dist}(p_s^*, p_z^*) + \text{dist}(p_z^*, q_s)]
\leq 2y_i + x_e + 2[x_e + y_s]
\leq 4(x_e + y_s)
$$

Hence, summing over all the edges, since each vertex $q_i$ is the corresponding vertex of at most $r$ edges, we get that $X' \leq 4X + 4rY$. Therefore we have the following.

$$
\text{Cost}(Q, G, \hat{P}) \leq X' + Y' \leq 3Y + 4X + 4rY \leq (4r + 3) \cdot \text{Cost}(Q, G, P)
$$

and thus $\alpha(Q, G, P) = O(r)$. ▼

4 Experiments

We consider image denoising as an application of our algorithm. A popular approach to denoising (see e.g. [13]) is to minimize the following objective function:

$$
\sum_{i \in V} \kappa_i d(q_i, p_i) + \sum_{(i, j) \in E} \lambda_{i, j} d(p_i, p_j)
$$

Here $q_i$ is the color of pixel $i$ in the noisy image, and $p_i$ is the color of pixel $i$ in the output. We use the standard 4-connected neighborhood system for the edge set $E$, and use Euclidean distance as the distance function $d(\cdot, \cdot)$. We also set all weights $\kappa_i$ and $\lambda_{i, j}$ to 1.

When the image is in grey scale, this objective function can be optimized approximately and efficiently using message passing algorithm, see e.g. [12]. However, when the image pixels are points in RGB color space, the label set becomes huge ($n = 256^3 = 16,777,216$), and most techniques for metric labeling are not feasible.

Recall that our algorithm proceeds by considering only the nearest neighbor labels of the query points, i.e., only the colors that appeared in the image. In what follows we refer to this reduced set of labels as the image color space, as opposed to the full color space where no pruning is performed.

In order to optimize the objective function efficiently, we use the technique of [13]. We first embed the original (color) metric space into a tree metric (with $O(\log n)$ distortion), and then apply a top-down divide and conquer algorithm on the tree metric, by calling the alpha-beta swap subroutine [7]. We use the random-split kd-tree for both the full color space and the image color space. When constructing the kd-tree, split each interval $[a, b]$ by selecting a random number chosen uniformly at random from the interval $[0.6a + 0.4b, 0.4a + 0.6b]$.

To evaluate the performance of the two algorithms, we use one cartoon image with MIT logo and two images from the Berkeley segmentation dataset [26] which was previously used
Table 1 The empirical values of objective functions for the respective images and algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Avg cost for full color</th>
<th>Avg cost for image color</th>
<th>Empirical pruning gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT</td>
<td>341878 ± 3.1%</td>
<td>340477 ± 1.1%</td>
<td>0.996</td>
</tr>
<tr>
<td>Snow</td>
<td>9338604 ± 4.5%</td>
<td>9564288 ± 6.2%</td>
<td>1.024</td>
</tr>
<tr>
<td>Surf</td>
<td>8304184 ± 6.6%</td>
<td>7588244 ± 5.1%</td>
<td>0.914</td>
</tr>
</tbody>
</table>

in other computer vision papers [13]. We use Matlab imnoise function to create noisy images from the original images. We run each instance 20 times, and compute both the average and the variance of the objective function (the variance is due to the random generating process of kd tree).

The results are presented in Figure 1 and Table 1. In Figure 1, one can see that the images produced by the two algorithms are comparable. The full color version seems to preserve a few more details than the image color version, but it also “hallucinates” non-existing colors to minimize the value of the objective function. The visual quality of the de-noised images can be improved by fine-tuning various parameters of the algorithms. We do not report these results here, as our goal was to compare the values of the objective function produced by the two algorithms, as opposed to developing the state of the art de-noising system.

Note that, as per Table 1, for some images the value of the objective function is sometimes lower for the image color space compared to the full color space. This is because we cannot solve the optimization problem exactly. In particular, using the kd tree to embed the original metric space into a tree metric is an approximate process.

4.1 De-noising with patches

To improve the quality of the de-noised images, we run the experiment for patches of the image, instead of pixels. Moreover, we use Algorithm 3 which implements not only a pruning step, but also computes the solution directly. In this experiment (see Figure 2 for a sample of the results), each patch (a grid of pixels) from the noisy image is a query point, and the dataset consists of available patches which we use as a substitute for a noisy patch.

In our experiment, to build the dataset, we take one image from the Berkeley segmentation data set, then add noise to the right half of the image, and try to use the patches from the left half to denoise the right half. Each patch is of size 5 × 5 pixels. We obtain 317 × 236 patches from the left half of the image and use it as the patch database. Then we apply Algorithm 3 to denoise the image. In particular, for each noisy patch \( q_n \) (out of 317 × 237 patches) in the right half of the image, we perform a linear scan to find the closest patch \( p_i \) from the patch database, based on the following cost function:

\[
\text{dist}(q_n, p_i) + \sum_{p_j \in \text{neighbor}(q_n)} \frac{\text{dist}(p_j, p_i)}{5}
\]

where \( \text{dist}(p, q) \) is defined to be the sum of squares of the \( l_2 \) distances between the colors of corresponding pixels in the two patches.

After that, for each noisy patch we retrieve the closest patch from the patch database. Then for each noisy pixel \( x \), we first identify all the noisy patches (there are at most 25 of them) that cover it. The denoised color of this pixel \( x \) is simply the average of all the corresponding pixels in those noisy patches which cover \( x \).

Since the nearest neighbor algorithm is implemented using a linear scan, it takes around 1 hour to denoise one image. One could also apply some more advanced techniques like locality sensitive hashing to find the closest patches with much faster running time.
Figure 1 MIT logo (first column, size 45 × 124), and two images from the Berkeley segmentation dataset [26] (second & third columns, size 321 × 481). The first row shows the original image; the second row shows the noisy image; the third row shows the denoised image using full color space; the fourth row shows the denoised image using image space (our algorithm).

Figure 2 Two images from the Berkeley segmentation dataset [26] (size 321 × 481). The first column shows the original image; the second column shows the half noisy image; the third column shows the de-noised image using our algorithm for the patches.
Acknowledgements. The authors would like to thank Pedro Felzenszwalb for formulating the Simultaneous Nearest Neighbor problem, as well as many helpful discussions about the experimental setup.

References


Simultaneous Nearest Neighbor Search

Motivated by the importance of the \( r \)-sparse graphs in applications, in this section we focus on them and present another algorithm (besides INN) which solves the SNN problem for these graphs. We note that unlike INN, the algorithm presented in this section is not just a pruning step, but it solves the whole SNN problem.

For a graph \( G = (Q, E) \) of pseudoarboricity \( r \), let the mapping function be \( f : E \rightarrow Q \), such that for every \( c = (q_i, q_j) \), \( f(c) = q_i \) or \( f(c) = q_j \), and that for each \( q_i \in Q \), \( |C(q_i)| \leq r \), where \( C(q_i) \) is defined as \( \{ e | f(e) = q_i \} \).

Once we have the mapping function \( f \), we can run Algorithm 3 to get an approximate solution. Although the naive implementation of this algorithm needs \( O(rkn) \) running time, by using the aggregate nearest neighbor algorithm, it can be done much more efficiently. We have the following lemma on the performance of this algorithm.

\[ \textbf{Lemma 15.} \text{ If } G \text{ has pseudoarboricity } r, \text{ the solution of Algorithm 3 gives } 2r + 1 \text{ approximation to the optimal solution.} \]
Algorithm 3 Algorithm for graph with pseudoarboricity \( r \)

**Input** Query points \( q_1, \ldots, q_k \), the input graph \( G = (Q, E) \) with pseudoarboricity \( r \)

**Output** An Assignment \( p_1, \ldots, p_k \in P \)

1: for \( i = 1 \) to \( k \) do
2: Assign \( p_i \leftarrow \min_{p \in P} \text{dist}(q_i, p) + \sum_{j: (q_i, q_j) \in C(q_j)} \frac{\text{dist}(p, q_j)}{r+1} \)
3: end for

**Proof.** Denote the optimal solution as \( P^* = \{p^*_1, \ldots, p^*_k\} \). We know the optimal cost is

\[
\text{Cost}(Q, G, P^*) = \sum_i \text{dist}(q_i, p^*_i) + \sum_{(q_i, q_j) \in E} \text{dist}(p^*_i, p^*_j)
\]

\[
= \sum_i \left( \text{dist}(p^*_i, q_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(p^*_i, p^*_j) \right)
\]

Let \( \text{Sol} \) be the solution reported by Algorithm 3. Then we have

\[
\text{Cost}(\text{Sol}) = \sum_i \left( \text{dist}(q_i, p_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(p_i, p_j) \right)
\]

\[
\leq \sum_i \left( \text{dist}(q_i, p_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(p_i, q_j) + \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(q_j, p_j) \right)
\]

(by triangle inequality)

\[
\leq \sum_i \left( \text{dist}(q_i, p_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(p_i, q_j) \right) + r \sum_j \text{dist}(q_j, p_j)
\]

(by definition of pseudoarboricity)

\[
= (r + 1) \sum_i \text{dist}(q_i, p_i) + \sum_{(q_i, q_j) \in C(q_j)} \text{dist}(p_i, q_j)
\]

\[
\leq (r + 1) \sum_i \left( \text{dist}(q_i, p^*_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \frac{\text{dist}(p^*_i, q_j)}{r+1} \right)
\]

(by the optimality of \( p_i \) in the algorithm)

\[
\leq (r + 1) \sum_i \left( \text{dist}(q_i, p^*_i) + \sum_{j: (q_i, q_j) \in C(q_j)} \frac{\text{dist}(p^*_i, q_j) + \text{dist}(p^*_j, q_j)}{r+1} \right)
\]

(by triangle inequality)

\[
\leq (r + 1) \text{Cost}(Q, G, P^*) + \sum_i \sum_{j: (q_i, q_j) \in C(q_j)} \text{dist}(p^*_j, q_j)
\]

\[
\leq (r + 1) \text{Cost}(Q, G, P^*) + r \sum_j \text{dist}(p^*_j, q_j)
\]

(by definition of pseudoarboricity)

\[
= (2r + 1) \text{Cost}(Q, G, P^*)
\]