

Fast Approximate Similarity Search Based on Degree-Reduced Neighborhood Graphs

Kazuo Aoyama
NTT Communication Science
Laboratories, NTT Corporation
2-4, Hikaridai, Seika-cho,
Soraku-gun, Kyoto, 619-0237,
Japan
aoyama.kazuo@lab.ntt.co.jp

Kazumi Saito
University of Shizuoka
52-1, Yada, Suruga-ku,
Shizuoka, Shizuoka,
422-8526, Japan
k-saito@u-shizuoka-
ken.ac.jp

Hiroshi Sawada
NTT Communication Science
Laboratories, NTT Corporation
2-4, Hikaridai, Seika-cho,
Soraku-gun, Kyoto, 619-0237,
Japan
sawada.hiroshi@lab.ntt.co.jp

Naonori Ueda
NTT Communication Science
Laboratories, NTT Corporation
2-4, Hikaridai, Seika-cho,
Soraku-gun, Kyoto, 619-0237,
Japan
ueda.naonori@lab.ntt.co.jp

ABSTRACT

This paper presents a fast approximate similarity search method for finding the most similar object to a given query object from an object set with a dissimilarity with a success probability exceeding a given value. As a search index, the proposed method utilizes a degree-reduced k -nearest neighbor (k -DR) graph constructed from the object set with the dissimilarity, and explores the k -DR graph along its edges using a greedy search (GS) algorithm starting from multiple initial vertices with parallel processing. In the graph-construction stage, the structural parameter k of the k -DR graph is determined so that the probability with which at least one search trial of those with multiple initial vertices succeeds is more than the given success probability. To estimate the greedy-search success probability, we introduce the concept of a basin in the k -DR graph. The experimental results on a real data set verify the approximation scheme and high search performance of the proposed method and demonstrate that it is superior to E²LSH in terms of the expected search cost.

Categories and Subject Descriptors

I.2 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—*graph and tree search strategies*;
H.3 [Information Storage and Retrieval]: Information Search and Retrieval—*search process*

General Terms

Algorithms, Performance, Theory

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'11, August 21–24, 2011, San Diego, California, USA.
Copyright 2011 ACM 978-1-4503-0813-7/11/08 ...\$10.00.

Keywords

Similarity search, Index structure, Neighborhood graph, Approximate algorithm

1. INTRODUCTION

A similarity search that finds valid objects similar to a given query object has been a fundamental technique for many applications in a variety of fields such as data mining, pattern recognition, computational geometry, computational biology, and statistics [5, 9]. In these fields, some applications require to obtain a solution quickly, even if it is not always exact, rather than an exact solution after a long computation time. An exact similarity search, in particular, is often a time-consuming task when employing a large-scale data set with medium to high *intrinsic* dimensionality estimated based on certain definitions [5, 12, 14]. In such cases, an approximate similarity search is a promising approach to improving search efficiency, although this comes at the expense of degrading the quality of the search result.

Of the many approaches to achieving an approximate similarity search, we focus on a class that gives probabilistic guarantees of the quality of the search results. A representative approach from this class is the locality-sensitive hashing (LSH) scheme [1, 2, 10, 21]. The LSH scheme performs a fast similarity search with a given success probability by reducing the search space. More specifically, the neighborhoods of a query object in an original space are embedded in *buckets* represented by an identical sequence of hash values obtained by a set of hash functions such that the collision probability is much higher for objects that are close to each other than for those that are far apart. A small set of *buckets* with hash values identical to those of the query object corresponds to the search space. The LSH scheme is an efficient technique, but it is only applicable to limited metric spaces such as a Euclidean space with a limited distance measure.

There is a graph-based approach [8, 11, 17, 20] for similarity searches, which is applicable to various types of spaces because it employs a graph representation of a given data set

with a dissimilarity. Experimental results obtained with this approach have been reported for large-scale document data sets with high *intrinsic* dimensionality [3], high-dimensional image data sets [19], and a non-metric space consisting of a Gaussian mixture model set and a Kullback-Leibler divergence as a dissimilarity [4]. However, the graph-based approach has two problems. First, the task of constructing a graph as a search index from a given data set with a dissimilarity incurs a high computational cost. For instance, the computational complexity of constructing a k -nearest neighbor (k -*NN*) graph is $\mathcal{O}(n^2)$ with a brute-force approach, where n denotes the number of objects in the data set. The second major problem is that the accuracy of the search result is unclear because a graph-based approach is heuristic. The former problem of constructing a k -*NN* graph is equivalent to the *all- k -nearest-neighbor* problem, to which considerable research efforts have been devoted [6, 18]. We can utilize their results for constructing the graph index. The latter problem has not really been studied. To improve the availability of the graph-based approach, we tackle the problem of how to control the search accuracy.

This paper presents a fast approximate similarity search method based on a neighborhood-graph index. The proposed method has two main algorithms: one is for constructing a graph as a search index and the other for searching on the graph. The graph-construction algorithm builds a special neighborhood graph, which is called a *degree-reduced k -nearest neighbor (k -*DR*)* graph, from a given data set with a dissimilarity. As the search algorithm, we adopt a greedy search (*GS*) algorithm that efficiently finds the vertex *closest from* a query vertex by exploring the k -*DR* graph along its edges. We determine the graph structural parameter k in the graph-construction stage so that the *GS* algorithm succeeds with a success probability exceeding a given value. To estimate the greedy-search (*GS*) success probability, we introduce the concept of a *basin* in the graph.

The remainder of this paper consists of seven sections. Section 2 formulates the problem we attempt to overcome. Section 3 provides preliminaries regarding graph-based similarity search. Section 4 describes the key concept of a *basin* in a graph, and explains the relationship between the basin and the success probability of a greedy search on the graph. Section 5 details our proposed approximate approach, in particular, a new graph-construction algorithm and a parallel greedy search algorithm. Section 6 describes an extension of the proposed method to a range query. Section 7 shows our experimental settings and reports our results. The final section offers our conclusions.

2. PROBLEM FORMULATION

Let \mathcal{X} be a space consisting of valid objects, where dissimilarity $D(x, y)$, $x, y \in \mathcal{X}$ from object x to another y is defined. We solve an approximate nearest neighbor problem as follows. Given object set $X \subset \mathcal{X}$ and query object $q \in \mathcal{X}$, efficiently find target object $x^*(q)$ *closest from* q , that is,

$$x^*(q) = \arg \min_{x \in X} \{D(q, x)\} \quad (1)$$

with a probability of at least $1 - \delta$, where δ is the failure probability with which the search algorithm fails to find $x^*(q)$. Note that the object whose dissimilarity from q is minimal among all the objects in X is referred to as the object *closest from* q instead of the object *closest to* q .

3. PRELIMINARIES

We deal with a similarity search problem as a graph-search problem. The graph-search problem is to efficiently find the vertex closest from a given query, by exploring a graph along its edges from an initial vertex with a search algorithm. In our problem, the graph is not given in advance but it is constructed from a given object set with a dissimilarity between the objects. This section reviews the graph representations of a given object set X based on a dissimilarity $D(q, x)$, where $q \in \mathcal{X}$ and $x \in X$, followed by a search algorithm on the constructed graph Γ . Henceforth, we use identical symbols for corresponding elements in the original space and the graph unless otherwise stated. For instance, each object $x \in X$ in the original space corresponds to vertex x on Γ constructed from X , and query object q is treated as an isolated vertex q if $q \notin X$.

3.1 k -*NN* and k -*DR* Graphs

We construct a degree-reduced undirected k -nearest neighbor (k -*DR*) graph denoted by $\Gamma_{(k)DR}$ from X based on a dissimilarity, and utilize $\Gamma_{(k)DR}$ as an index for a search algorithm. We describe the structure and properties of $\Gamma_{(k)DR}$, compared with an undirected k -nearest neighbor (k -*NN*) graph denoted by $\Gamma_{(k)NN}$ as a baseline. Graph-construction algorithms for these graphs are detailed in Section 5.1.

Let $N_k(x)$ denote a set of k nearest neighbors of x measured with a dissimilarity. $\Gamma_{(k)NN}$ is constructed by the connection of each x and its $N_k(x)$ with undirected edges. Let $V_{(k)NN}(x)$ denote an adjacent vertex set of x , which is expressed by $N_k(x) \cup \{y : x \in N_k(y)\}$. Let $E_{(k)NN}(x)$ denote a set of undirected edges joining x and elements of $V_{(k)NN}(x)$, which is represented as $E_{(k)NN}(x) = \{e\{x, y\} : y \in V_{(k)NN}(x)\}$, where $e\{x, y\}$ is the undirected edge between x and y . Then $\Gamma_{(k)NN} = (X, \cup_{x \in X} E_{(k)NN}(x))$.

$\Gamma_{(k)DR}$ differs from $\Gamma_{(k)NN}$ in not having an edge between x and $y \in N_k(x)$, without which a greedy search (*GS*) algorithm can reach x from y along the existing edges. Then $\Gamma_{(k)DR}$ has a smaller average degree than $\Gamma_{(k)NN}$. Moreover, $\Gamma_{(k)DR}$ has *small-world* properties [22]: homophily, which is the tendency of *like to associate with like* [16] and an extremely small average shortest path length. These properties enable a search algorithm using only local information to find a target vertex on the graph efficiently. In this paper, we use $\Gamma_{(k)DR}$ as an index because it has the foregoing features, which make it suitable for a similarity search. The symbols for $\Gamma_{(k)NN}$ are also used for $\Gamma_{(k)DR}$. We summarize the notations and definitions we have used in “**Object and Dissimilarity**” and “**Graph**” in **Table 1**.

Example 1. Figure 1 shows k -*NN* and k -*DR* graphs for $k = 1, 2$, which are constructed from the ten given vertices depicted as circles. The 1-*NN* graph $\Gamma_{(1)NN}$ and the 1-*DR* graph $\Gamma_{(1)DR}$ are identical as shown in Fig. 1(a). $\Gamma_{(2)NN}$ and $\Gamma_{(2)DR}$ are shown in Fig. 1(b) and (c), respectively. $\Gamma_{(2)DR}$ has a smaller average degree than $\Gamma_{(2)NN}$. These average degrees are proportional to the numbers of edges of $|E_{(2)DR}| = 10$ and $|E_{(2)NN}| = 13$.

3.2 Greedy Search on Graphs

We use a greedy search (*GS*) algorithm to explore graph $\Gamma_{(k)DR}$. Given query vertex q and initial vertex x_0 , the *GS* algorithm first evaluates the dissimilarity from q to x_0 , $D(q, x_0)$, and then evaluates the dissimilarities from q to

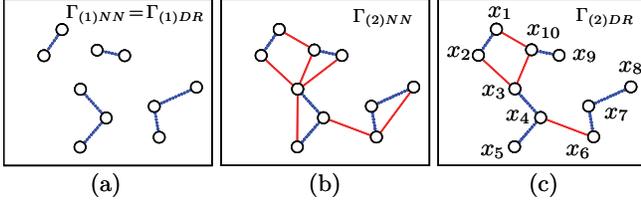


Figure 1: Graphs constructed from 10 given vertices depicted as circles: (a) 1-NN graph or 1-DR graph with 6 undirected edges denoted as dashed lines, (b) 2-NN graph with 7 added edges denoted as solid lines, (c) 2-DR graph with 4 added edges.

Table 1: Notations and Definitions

Notation	Description and Definition
Object and Dissimilarity	
\mathcal{X}	Given space with a dissimilarity
X	Object (vertex) set or a data set; $X \subset \mathcal{X}$
n	The number of objects (vertices) in X ; $ X $
q	Query object (vertex)
$D(x, y)$	Dissimilarity from x to y , $x, y \in \mathcal{X}$
$x^*(q)$	Target object (vertex) that is the vertex closest from q ; $\arg \min_{x \in X} \{D(q, x)\}$
$N_k(x)$	Set of k nearest neighbors of x
Graph	
Γ	Graph
$\Gamma_{(k)NN}$	Undirected k -nearest neighbor graph
$\Gamma_{(k)DR}$	Degree-reduced k -nearest neighbor graph
$V_{(k)NN}(x)$	Adjacent vertex set of x on $\Gamma_{(k)NN}$
$V_{(k)DR}(x)$	Adjacent vertex set of x on $\Gamma_{(k)DR}$
$e\{x, y\}$	Undirected edge between x and y
$E_{(k)NN}(x)$	$\{e\{x, y\} : y \in V_{(k)NN}(x)\}$
$E_{(k)DR}(x)$	$\{e\{x, y\} : y \in V_{(k)DR}(x)\}$
Search	
GS, PGS	Greedy search, Parallel greedy search
δ	Search failure probability
$c(q; \Gamma)$	Search cost for q when given Γ
$C_{[i]}(q; \Gamma)$	Set of vertices whose dissimilarities from q are evaluated in the i th parallel process
$D_{min[i]}(q; \Gamma)$	The minimum dissimilarity from q in the i th parallel process
$R(q)$	Set of objects within dissimilarity r from q
Basin and Search Success Probability	
$Basin(q; \Gamma)$	Set of vertices on given Γ , from which a GS algorithm finds $x^*(q)$
Q', q'	Quasi-query vertex set $Q' \subset X$; $q' \in Q'$
Q	Query vertex set; $q \in Q$
S	Test-vertex set for the Monte Carlo method
T	Target vertex set for Q' or Q
L	The number of search trials for each query q or the number of initial vertices for each q
$p(\text{Success} q; \Gamma)$	GS success probability finding $x^*(q)$ on Γ
$\tilde{p}(\text{Success} q, S; \Gamma)$	GS success probability for q estimated with the Monte Carlo method using S
$\hat{p}(\text{Success} q; L, \Gamma)$	Probability that at least one search trial in L trials for identical q succeeds on Γ
$\langle \hat{p}(\text{Success}; L, \Gamma) \rangle$	Ensemble average of a probability $\hat{p}(\text{Success} q; L, \Gamma)$

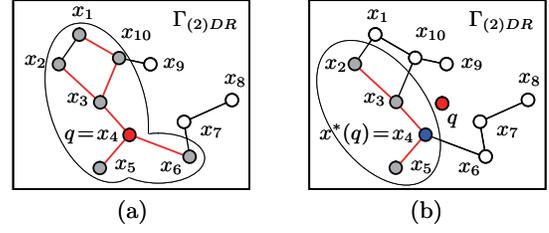


Figure 2: Basins in the 2-DR graph with 10 vertices: (a) Basin of $q = x_4$ denoted as filled circles, (b) Basin of isolated q whose $x^*(q)$ is x_4 .

all the vertices in an adjacent vertex set of x_0 , $V_{(k)DR}(x_0)$. Let us call x_0 an expanded vertex whose adjacent vertices are evaluated in terms of their dissimilarities from q . If $D(q, x_0) > D(q, x)$, where $x = \arg \min_{y \in V_{(k)DR}(x_0)} \{D(q, y)\}$, the GS algorithm sets x at the next expanded vertex and repeats this operation. Conversely, the GS algorithm terminates if $D(q, x_0) \leq D(q, x)$. Suppose that $\Gamma_{(2)DR}$ shown in Fig. 1(c) is provided and x_4 is set as query vertex q . The GS algorithm starting from x_2 first evaluates $D(x_4, x_2)$ and then compares it with the minimum dissimilarity $D(x_4, x_3)$ in $\{D(x_4, x_1), D(x_4, x_3)\}$. Since $D(x_4, x_3) < D(x_4, x_2)$, the GS algorithm sets x_3 as the next expanded vertex and repeats this operation. Finally, the GS algorithm reaches x_4 , and terminates. Note that when Γ and q are given, a set of initial vertices from which q is reachable with the GS algorithm is uniquely specified. This fact leads us to the concept of a basin in a graph, detailed in the next Section.

4. BASINS IN GRAPHS

We first define the basin of query vertex $q \in \mathcal{X}$ in Definition 1, which is a key concept for understanding the proposed approximate approach, and then explain the relationship between the basin of q and the success probability of a greedy search (GS) algorithm that finds the target vertex $x^*(q)$ closest from q , starting from an initial vertex chosen uniformly at random from all the vertices in X in graph Γ .

Definition 1. Given graph Γ constructed from object set $X \subset \mathcal{X}$ based on a dissimilarity and query vertex $q \in \mathcal{X}$, a basin of q denoted as $Basin(q; \Gamma)$ is defined by a set of vertices on Γ from which a greedy search (GS) algorithm can find the target vertex $x^*(q)$ closest from q along the edges of Γ . Let $GS(x_0, q; \Gamma)$ denote a function that returns the closest vertex from q exactly when a GS algorithm with initial vertex x_0 on Γ terminates. Then $Basin(q; \Gamma)$ is expressed as

$$Basin(q; \Gamma) = \{x_0 \in X : x^*(q) = GS(x_0, q; \Gamma)\}. \quad (2)$$

Example 2. To provide an intuitive understanding of the basin of a vertex, we show examples in Figs. 2(a) and (b), where each q is given for the 2-DR graph constructed from a set of 10 vertices X . In Fig. 2(a) where $q \in X$ and $q = x^*(q) = x_4$, the basin of $q = x_4$ denoted by $Basin(q; \Gamma_{(2)DR})$ is $\{x_4, x_1, x_2, x_3, x_5, x_6, x_{10}\}$ and the ratio of the number of the basin's elements to $|X| = 10$ is 0.7. In Fig. 2(b) where q is an isolated vertex, i.e., $q \notin X$ and $x^*(q) = x_4$, the basin of q , $Basin(q; \Gamma_{(2)DR})$, is $\{x_4, x_2, x_3, x_5\}$ and the ratio is 0.4. In spite of the identical target vertex $x^*(q) = x_4$ in these two cases, the basins have different numbers of elements. This arises from q 's position, namely whether $q \in X$ or $q \notin X$.

In the latter case, for instance, the *GS* algorithm starting from x_{10} does not select x_3 as the next expanded vertex but selects x_9 because $D(q, x_9) < D(q, x_3)$. Therefore, $x_{10} \notin \text{Basin}(q; \Gamma_{(2)DR})$. Thus the position of q affects its basin.

From *Definition 1*, we note that the success probability of the *GS* algorithm coincides with the ratio of the number of vertices in the basin of q to that of all the vertices $n = |X|$. The *GS* success probability $p(\text{Success}|q; \Gamma)$ is expressed by

$$p(\text{Success}|q; \Gamma) = \frac{1}{n} \sum_{x_0 \in X} I(x_0 \in \text{Basin}(q; \Gamma)) = \frac{|\text{Basin}(q; \Gamma)|}{n}, \quad (3)$$

where $I(\cdot)$ is the indicator function that returns one only if the condition holds, and $|\text{Basin}(q; \Gamma)|$ denotes the number of elements the basin contains, which is called *basin size*.

Suppose that L search trials on Γ for an identical q are performed independently with the *GS* algorithm, and initial vertex $x_{0[i]}$ ($i = 1, 2, \dots, L$) in multiset X_0 chosen uniformly at random from X is employed in a distinct search trial among the L trials. Consider the exact probability that at least one search trial succeeds in finding $x^*(q)$. Then the probability $\hat{p}(\text{Success}|q; L, \Gamma)$ with which at least one of the L search trials for query q succeeds is expressed by

$$\hat{p}(\text{Success}|q; L, \Gamma) = 1 - (1 - p(\text{Success}|q; \Gamma))^L. \quad (4)$$

In the ideal case, the probability $\hat{p}(\text{Success}; L, \Gamma)$ is expressed as

$$\hat{p}(\text{Success}; L, \Gamma) = \sum_{\forall q} \hat{p}(\text{Success}|q; L, \Gamma) \cdot p(q), \quad (5)$$

where $p(q)$ denotes the probability mass function of q . In practice, it is difficult to calculate the probability in Eq. (5) because $p(q)$ is unknown or the calculation of exact $\hat{p}(\text{Success}|q; L, \Gamma)$ requires a high computational cost even if $p(q)$ were given in advance.

Instead of the exact calculation of $\hat{p}(\text{Success}; L, \Gamma)$, we first approximate $p(q)$ with an empirical distribution function of quasi-query vertex $q'_{[j]} \in Q'$ ($j = 1, 2, \dots, |Q'|$), which is chosen from X by simple random sampling. Then $\hat{p}(\text{Success}; L, \Gamma)$ is approximated by

$$\langle \hat{p}(\text{Success}; L, \Gamma) \rangle \simeq \frac{1}{|Q'|} \sum_{q' \in Q'} \hat{p}(\text{Success}|q'; L, \Gamma), \quad (6)$$

where $\langle \star \rangle$ denotes an ensemble average of \star . Moreover, we make practical use of the Monte Carlo method for calculating $\hat{p}(\text{Success}|q'; L, \Gamma)$. The Monte Carlo method uses a set of test vertices chosen from X by simple random sampling, which works as a set of initial vertices for a *GS* algorithm and which is denoted by $S_{[j]}$, consisting of $s_{[j][h]} \in S_{[j]}$ ($h = 1, 2, \dots, |S_{[j]}| \ll n$). Then we can estimate the ensemble average of the probability as

$$\begin{aligned} & \langle \hat{p}(\text{Success}; L, \Gamma) \rangle \\ & \simeq \frac{1}{|Q'|} \sum_{j=1}^{|Q'|} \left\{ 1 - \left(1 - \frac{1}{|S_{[j]}|} \sum_{h=1}^{|S_{[j]}|} I(s_{[j][h]} \in \text{Basin}(q'_{[j]}; \Gamma)) \right)^L \right\} \\ & = \frac{1}{|Q'|} \sum_{q'_{[j]} \in Q'} \left\{ 1 - (1 - \tilde{p}(\text{Success}|q'_{[j]}; \Gamma))^L \right\}, \quad (7) \end{aligned}$$

where $\tilde{p}(\text{Success}|q'_{[j]}; \Gamma)$ denotes the *GS* success probability estimated by the Monte Carlo method using the test-vertex set $S_{[j]}$ corresponding to each $q'_{[j]}$.

5. PROPOSED GRAPH-BASED METHOD

This section details the graph-based approximate similarity search method. This method consists of two main algorithms: one for building an index and the other for searching based on the index. We construct $\Gamma_{(k)DR}$ as the index and use a *GS* algorithm for traversing the $\Gamma_{(k)DR}$ along its edges with parallel processing using multiple initial vertices.

5.1 Graph-Construction Algorithm

We begin by clarifying the construction of a simple k -*NN* graph $\Gamma_{(k)NN}$. A parameter k of $\Gamma_{(k)NN}$, which is provided in advance, indicates the number of out-edges for each vertex, i.e., an out-degree. Given object set $X \subset \mathcal{X}$, k and rank list $List(X, k_{max})$, $k_{max} > k$, where the k_{max} nearest neighbors of each object in X are listed in ascending order of dissimilarity, we construct $\Gamma_{(k)NN} = (X, \cup_{x \in X} E_{(k)NN}(x))$ by connecting each vertex x and $y \in N_k(x)$ with undirected edges $E_{(k)NN}(x)$ with reference to the $List(X, k_{max})$. We adopt an incremental procedure on k , that is, we construct $\Gamma_{(1)NN}$ for $k' = 1$, then construct $\Gamma_{(2)NN}$ by incrementing k' by one, and we repeat this procedure until $k' = k$.

Next, let us consider the construction of $\Gamma_{(k)DR}$ on the assumption that the parameter k is provided in advance. $\Gamma_{(k)DR}$ is constructed with the incremental procedure on k in the same way as $\Gamma_{(k)NN}$. However, unlike the construction algorithm for $\Gamma_{(k)NN}$, that for $\Gamma_{(k)DR}$ generates an undirected edge between x and its k' th nearest neighbor vertex y only if the *GS* algorithm starting from y cannot find x along the already existing edges on $\Gamma_{(k')DR}$. This operation reduces the number of edges of each vertex, i.e., the degree, compared with that in $\Gamma_{(k')NN}$, maintaining a path through which x is reachable from y with the *GS* algorithm.

Finally, consider the construction of $\Gamma_{(k)DR}$ on the assumption that the parameter k is *not* provided in advance so that $\Gamma_{(k)DR}$ enables the *GS* algorithm to find the nearest neighbor of query vertex q with a given success probability. Given X , $List(X, k_{max})$, failure probability δ , a set Q' of quasi-queries $q_{[j]} \in \mathcal{X}$, $j = 1, 2, \dots, |Q'|$, a target vertex set of the $x^*(q_{[j]}) \in X$ denoted by T , a test-vertex set $S_{[j]} = S \subset X$ for estimation of a basin size, and the number of initial vertices L for each query for a *GS* algorithm, $\Gamma_{(k)DR}$ is constructed by **Algorithm 1**. A novel procedure in this algorithm is the judgment of whether it increments k' , ($k' \leq k$), shown on **Line 2** in **Algorithm 1**. Once the algorithm constructs $\Gamma_{(k')DR}$ at the k' th stage, it compares the given search success probability $1 - \delta$ with $\langle \hat{p}(\text{Success}; L, \Gamma_{(k')DR}) \rangle$. The $\langle \hat{p}(\text{Success}; L, \Gamma_{(k')DR}) \rangle$ monotonically increases with k' because the basin size $|\text{Basin}(q'_{[j]}; S; \Gamma_{(k')DR})|$ monotonically increases. Note that we appropriately replace the variables in Eqs. (3) and (7) with the above-mentioned variables, for instance, Γ and q are replaced with $\Gamma_{(k')DR}$ and $q'_{[j]}$, respectively. In addition, S independent of j is introduced instead of $S_{[j]}$. Using the monotonic increase of $\langle \hat{p}(\text{Success}; L, \Gamma_{(k')DR}) \rangle$ with k' , the algorithm terminates only precisely when the following inequality holds.

$$1 - \delta < \frac{1}{|Q'|} \sum_{q'_{[j]} \in Q'} \left\{ 1 - (1 - \tilde{p}(q'_{[j]}; \Gamma_{(k')DR}))^L \right\} \quad (8) \\ = \langle \hat{p}(\text{Success}; L, \Gamma_{(k')DR}) \rangle$$

This algorithm is characterized by two procedures: the termination condition in inequality (8) shown on **Line 2** in **Algorithm 1** and the degree-reduction procedure shown

on **Lines 6–8**, where x and y are mutually added to the set of their adjacent vertices, only if the *greedy-reachability checking (GRC)* algorithm shown in **Algorithm 2** returns **FALSE**. The proposed graph-construction algorithm completes $\Gamma_{(k)DR}$ by controlling k so that the ensemble average of the probability is greater than $1-\delta$ that at least one of the L search trials succeeds.

Algorithm 1 *Graph-Construction Algorithm*
 $GC(X, List(X, k_{max}), \delta, Q', T, S, L)$

Input: $X, List(X, k_{max}), \delta, Q', T, S, L$

Output: $\Gamma_{(k)DR}$

1. Define $\Gamma_{(0)DR} = (X, \emptyset)$ at $k' = 0$, and Set $k' \leftarrow 0$
 2. **while** $1 - \delta \geq \langle \hat{p}(Success; L, \Gamma_{(k')DR}) \rangle$ **do**
 3. Set $k' \leftarrow k' + 1$
Set $\Gamma_{(k')DR} \leftarrow \Gamma_{(k'-1)DR}$
 4. **for all** $x \in X$ **do**
 5. Set $\{y\} \leftarrow N_{k'}(x) \setminus N_{(k'-1)}(x)$
 6. **if** $GRC(x, y; \Gamma_{(k')DR}) = \mathbf{FALSE}$ **then**
 7. Set $V_{(k')DR}(x) \leftarrow V_{(k')DR}(x) \cup \{y\}$
Set $V_{(k')DR}(y) \leftarrow V_{(k')DR}(y) \cup \{x\}$
 8. **end if**
 9. **end for**
 10. **end while**
 11. Set $k \leftarrow k'$
 12. Set $\Gamma_{(k)DR} \leftarrow (X, \cup_{x \in X} E_{(k)DR}(x))$,
where $E_{(k)DR}(x) = \{e\{x, y\} : y \in V_{(k)DR}(x)\}$
 13. **return** $\Gamma_{(k)DR}$
-

Algorithm 2 *Greedy-Reachability Checking Algorithm*
 $GRC(x, y; \Gamma_{(k')DR})$

Input: $x, y \in X, \Gamma_{(k')DR}$

Output: **TRUE** or **FALSE**

1. Set $z \leftarrow \arg \min_{z' \in V_{(k')DR}(y)} \{D(x, z')\}$
 2. **if** $D(x, z) \leq D(x, y)$ **then**
 3. **return TRUE**
 4. **else**
 5. **return FALSE**
 6. **end if**
-

5.2 Search Algorithm on Graphs

The proposed approximate similarity search method employs L search trials for each query for the estimation of the GS success probability. Since these search trials are completely independent of each other, we perform the L search trials with L parallel processes using the simple parallel greedy search (PGS) algorithm shown in **Algorithm 3**. In practice, the L search trials may be partitioned into any combinations of serial and parallel processes.

Given $\Gamma_{(k)DR}$, $q \in \mathcal{X}$, and a set $X_0(q)$ of initial vertices $x_{0[i]}(q)$ ($i = 1, 2, \dots, L$: i denotes the identification number of processes in parallel.), the PGS algorithm returns both the closest vertex z from q among those found by the L search trials, and search cost $c(q)$ defined by the maximum number of dissimilarity calculations among those performed by the L search trials until each trial terminates. In **Algorithm 3**, $D_{min[i]}(q; \Gamma_{(k)DR})$ and $C_{[i]}(q; \Gamma_{(k)DR})$ denote the minimum dissimilarity from q in the search process so far and a set of vertices whose dissimilarity from q are evaluated, respectively. Moreover, $x_{[i]}$ is the vertex closest from q in the adjacent vertices of the currently expanded vertex, and $z_{[i]}$ is the vertex closest from q in the expanded vertices.

Algorithm 3 *Parallel Greedy Search Algorithm*
 $PGS(q, X_0(q); \Gamma_{(k)DR})$

Input: $\Gamma_{(k)DR}$, $q \in \mathcal{X}$,

$X_0 = (x_{0[1]}, x_{0[2]}, \dots, x_{0[L]}), x_{0[i]} \in X$

Output: (z, c)

1. **for** each initial vertex $x_{0[i]}(q)$ **in parallel do**
 2. Set $z_{[i]} \leftarrow x_{0[i]}(q)$, $x_{[i]} \leftarrow x_{0[i]}(q)$,
 $D_{min[i]} \leftarrow D(q, z_{[i]})$, $C_{[i]} \leftarrow \{z_{[i]}\}$
 3. **while** $z_{[i]} = x_{[i]}$ **do**
 4. Set $x_{[i]} \leftarrow \arg \min_{y_{[i]} \in V_{(k)DR}(z_{[i]})} \{D(q, y_{[i]})\}$,
 $C_{[i]} \leftarrow C_{[i]} \cup V_{(k)DR}(z_{[i]})$
 5. **if** $D(q, x_{[i]}) < D(q, z_{[i]})$ **then**
 6. Set $D_{min[i]} \leftarrow D(q, x_{[i]})$, $z_{[i]} \leftarrow x_{[i]}$
 7. **end if**
 8. **end while**
 9. **end for**
 10. Set $z \leftarrow \arg \min_{i=1, \dots, L} \{D_{min[i]}\}$,
 $c \leftarrow \arg \max_{i=1, \dots, L} \{|C_{[i]}\}|$
 11. **return** (z, c)
-

6. EXTENSIONS

The proposed method can be easily extended to one for a range query to be answered by E^2 LSH. This problem is formulated as follows.

Problem. Suppose that the same space \mathcal{X} as that described in Section 2 is given. Given object set $x \in X \subset \mathcal{X}$, query object $q \in \mathcal{X}$, and a dissimilarity from q denoted by r , report a set of objects $R(q)$ within r from q with greater than or equal to a given *recall*. $R(q)$ and the *recall* for q denoted by $Recall(q)$ are defined as follows.

$$R(q) = \{x : D(q, x) < r\}. \quad (9)$$

$$Recall(q) = |\tilde{R}(q) \cap R(q)| / |R(q)|, \quad (10)$$

where $\tilde{R}(q)$ denotes a set of reported objects.

The PGS algorithm in **Algorithm 3** is modified so that the search algorithm collects the vertices within a given dissimilarity (or radius) r from q . We assume that $|R(q)| < |Basin(q; \Gamma_{(k)DR})|$. This assumption guarantees that the GS algorithm for q always finds target vertex $x^*(q)$ once it reaches a vertex within the radius $R(q)$. It is also experimentally shown in Section 7.2 that this assumption is reasonable since the expected basin size in $\Gamma_{(k)DR}$ reaches over 6,000 when $|X| = 60,000$, even if $k = 10$, while we rarely need such a large number of vertices as an answer. The modified search algorithm differs from a simple GS algorithm in its use of stack storing vertices within $R(q)$. The search algorithm pushes the vertices, which are evaluated in terms of their dissimilarities from q but are not expanded, into the stack. The stack plays a similar role to the *open list* in the A^* or the Frontier search algorithm [7, 13]. However, the proposed search algorithm pops a vertex from the stack and employs it as the next expanded vertex only after a greedy search part in the search algorithm terminates. This procedure is repeated until the stack is empty. The modified PGS algorithm for range queries is detailed in **Appendix A**.

7. EXPERIMENTS

This section shows our experimental settings and results. We first describe data sets and then show the properties of basins in $\Gamma_{(k)DR}$. Finally, we demonstrate the search performance of the proposed similarity search method, compared with E^2 LSH.

Table 2: Average normalized basin size on $\Gamma_{(k)DR}$; $(1/|Y|) \sum_{y \in Y} (|Basin(y; \Gamma_{(k)DR})|/n)$.

Position of vertex y $y \in Y \subset X$	k				
	10	20	40	60	100
$y \in Y = Q', (Q' \cap X = \emptyset)$	0.12	0.26	0.39	0.45	0.50
$y \in Y = X$	0.23	0.47	0.69	0.78	0.86

7.1 Data sets

To demonstrate the effectiveness and efficiency of the proposed approximate similarity search method, we prepared an image data set, the MNIST database of handwritten digits [15]. We used 60,000, 4,850, and 4,841 images from the MNIST database for object set X , query set Q , and quasi-query set Q' , which were disjoint from each other, respectively. In particular, quasi-query set Q' was used for graph construction and query set Q was used to evaluate search performance. For evaluation only, the two target-vertex sets that correspond to the quasi-query set and the query set were prepared in advance. A feature vector extracted from the image was represented as a point on the unit hypersphere in a 784-dimensional Euclidean space. A dissimilarity, which is a distance in this case, between a pair of points was measured by the Euclidean distance.

7.2 Basin properties

We analyzed the properties of basins in $\Gamma_{(k)DR}$ constructed from the object set X to verify that the basin size of $\Gamma_{(k)DR}$ monotonically increases with k , i.e., GS success probability $p(Success|q; \Gamma_{(k)DR})$ monotonically increases with k , and the Monte Carlo method works effectively, maintaining the accuracy of search results.

First, we calculated the exact basin sizes in $\Gamma_{(k)DR}$ s with various k values, $k = 10, 20, 40, 60, 100$, for vertex y in two distinct vertex sets Y , i.e., $Y = Q'$ ($Y \cap X = \emptyset$) and $Y = X$. Note that y is an isolated vertex that is not connected to $\Gamma_{(k)DR}$ when $Y = Q'$. Table 2 shows the average basin sizes in $\Gamma_{(k)DR}$ s, which are normalized by $n = |X|$. The average basin size increased with k for both $Y = Q'$ and $Y = X$. In particular, the average normalized basin size in $\Gamma_{(100)DR}$ at $k = 100$ reaches a very large value of 0.50, which corresponds to half of all the vertices in X , even if y does not exist in $\Gamma_{(100)DR}$, i.e., $Y = Q'$, although $\Gamma_{(100)DR}$ is constructed to guarantee that each vertex is reachable only from the vertices within its 100th nearest neighbor by the GS algorithm. In addition to this property of the average normalized basin size, we experimentally confirmed that basin-size distributions for $\Gamma_{(k)DR}$ shifted from a small to a large basin-size region with an increase in k . These properties allow us to utilize inequality (8) as the termination condition in the graph-construction algorithm.

Next, we show the effectiveness of the Monte Carlo method in comparison with the approach using all the vertices in X as the test vertices that work as initial vertices for the GS algorithm in the graph-construction stage. We calculated the ensemble average of $\hat{p}(Success; L, \Gamma_{(k)DR})$ for the following various settings. For each quasi-query $q'_{[j]} \in Q'$ ($|Q'| = 4,841$), we prepared three test-vertex sets $S_{[j]}$ with a different number of elements: the first is $|X| = n$ denoted by ‘‘All,’’ the second is 160, and the last is 40. The size of each test-vertex set is denoted by $|\bar{S}|$ ($|\bar{S}| = |S_{[j]}|$). In addition, $\Gamma_{(k)DR}$ s with various k values ($10 \leq k \leq 100$) were employed,

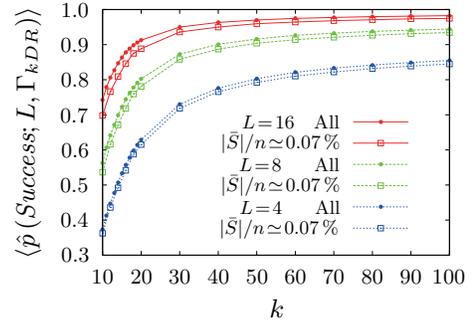


Figure 3: Comparison of two types of ensemble averages of the probability where at least one of L search trials succeeds for $\Gamma_{(k)DR}$. One type was calculated using all the vertices in X and the other was estimated by the Monte Carlo method using 40 test vertices corresponding to less than 0.07 % of $n = |X|$.

Table 3: Values of k of $\Gamma_{(k)DR}$ s constructed by using test-vertex sets with three distinct sizes when $L = 16$.

Basin size calculation method	$1 - \delta$			
	0.95	0.90	0.85	0.80
Exact, $\bar{S} = X$ (All)	31	19	15	12
Monte Carlo, $ \bar{S} = 160$ (0.27 %)	33	20	15	13
Monte Carlo, $ \bar{S} = 40$ (0.07 %)	40	22	17	14

and three different L values, $L = 4, 8, 16$, were adopted. Figure 3 shows comparison results for ensemble averages of the probabilities with which at least one of the L search trials succeeds. Surprisingly, the ensemble average, shown by the lines labeled $|\bar{S}|/n \simeq 0.07\%$, closely coincided with those of the exact average for any k and L , even though the number of test vertices was only 40 corresponding to less than 0.07% of all the vertices in X .

Given $\Gamma_{(k)DR}$, the value of $\hat{p}(Success; L, \Gamma_{(k)DR})$ estimated by the Monte Carlo method using the small set was always smaller than that exactly calculated using X as \bar{S} , $\bar{S} = X$. This finding implies that a k value of $\Gamma_{(k)DR}$ obtained with the graph-construction algorithm that uses the Monte Carlo method for the estimation of $\hat{p}(Success; L, \Gamma_{(k)DR})$ was larger than or equal to that obtained with the graph-construction algorithm using the exactly calculated $\hat{p}(Success; L, \Gamma_{(k)DR})$, as shown in Table 3. Then the GS algorithm on the $\Gamma_{(k)DR}$ constructed based on the termination condition shown as inequality (8) using $\hat{p}(Success; L, \Gamma_{(k)DR})$ estimated by the Monte Carlo method succeeds with more than the given success probability. Consequently, we can adopt the Monte Carlo method to reduce the computational cost of estimating the basin size.

7.3 Graph construction

This section confirms that tuple (k, L) is feasible for the given failure probability δ , that is, $k \ll n$ and L is an appropriate value for the degree of parallelism in a practical search task. The computational complexity of graph construction is also discussed.

We have a degree of the freedom in the selection of k and L to achieve an identical search success probability $1 - \delta$ although L is fixed at a certain value in Section 5.1. When δ and the estimated basin sizes are given and either k or L is fixed, the lower bound on the remaining parameter is derived

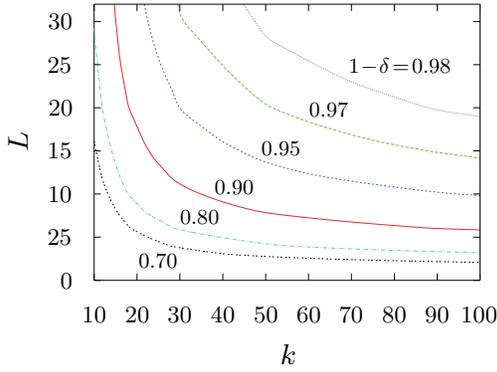


Figure 4: Contour plot of search success probability $1-\delta$ versus k and L , evaluated with basin sizes estimated by the Monte Carlo method using 40 test vertices.

from inequality (8). We employed vertex set X with 60,000 vertices, quasi-query set Q' with 4,841 vertices, target-vertex set T where a target vertex for each quasi-query was calculated in advance, and test-vertex set S with 40 vertices chosen from X by simple random sampling. The basin size for each quasi-query was estimated as a GS success probability by the Monte Carlo method using test-vertex set S . Figure 4 shows feasible solutions on tuple (k, L) for the given $1-\delta$, which were obtained by applying the estimated basin sizes to inequality (8). When the search success probability $1-\delta$ was given, L decreased monotonically with k . In contrast, when a large L was given, $1-\delta$ increased with k , because generally the basin size of a vertex increases with k , i.e., the GS success probability increases with k .

Next, we discuss the computational cost evaluated by the number of dissimilarity calculations (or distance calculations for the MNIST database). The proposed graph-construction algorithm needs $List(X, k_{max})$, where $k_{max} \ll n$, e.g., $k_{max} = 200$, is determined in advance. The computational cost for making the $List(X, k_{max})$ is $\mathcal{O}(n^2)$ with the brute-force approach. Moreover, the space complexity is $\mathcal{O}(n)$. Although the graph-construction algorithm in **Algorithm 1** seems to perform time-consuming basin-size calculations every time k' is updated, the Monte Carlo method successfully reduces the computational cost, while maintaining the accuracy of the search results as shown in Fig. 3 and Table 3. The basin-size calculations do not have any effect on the computational complexity.

7.4 Search performance

We evaluated the search performance with the expected search cost. Given $\Gamma_{(k)DR}$, a set of query vertices $q \in Q \subset \mathcal{X}$, and the number of search trials L per query, the expected search cost $E[c(\Gamma_{(k)DR})]$ is defined by

$$E[c(\Gamma_{(k)DR})] = \frac{1}{|Q|} \sum_{q \in Q} \max_{i=1, \dots, L} \{ |C_{[i]}(q; \Gamma_{(k)DR})| \}, \quad (11)$$

where $|C_{[i]}(q; \Gamma_{(k)DR})|$ denotes the search cost of the i th search trial for q on $\Gamma_{(k)DR}$. Figure 5 shows the expected search costs that will be incurred in achieving the given success probability $1-\delta$. For the various success probabilities, we employed $\Gamma_{(k)DR}$ constructed with L and the corresponding k shown in Section 7.3.

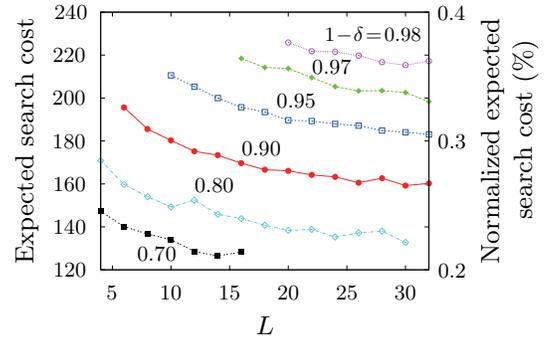


Figure 5: Expected search cost against L . Levels for achieving success probabilities of $1-\delta=0.98, 0.97, 0.95, 0.90, 0.80, 0.70$ are given in the graph-construction stage. Although k is omitted, it is uniquely determined from δ and L .

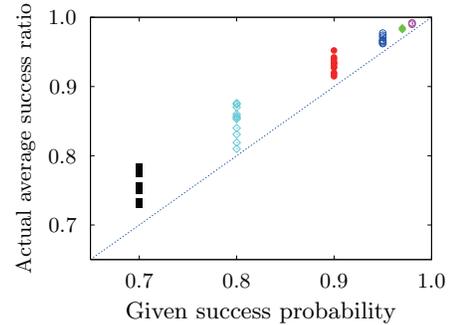


Figure 6: Relationship between given success probability and actual average success ratio obtained by a search using the constructed graph.

The proposed method performed a similarity search at an extremely low cost but with a high search success probability. For instance, when the method employed the $\Gamma_{(22)DR}$ constructed by **Algorithm 1** under the condition of $1-\delta=0.90$ and $L=16$, the expected search cost was only 169.25, corresponding to around 0.28% of n , which is indicated by the point at $L=16$ on line $1-\delta=0.90$ in Fig. 5. Moreover, the actual search success ratio was 0.93, which was higher than the given success probability $1-\delta=0.90$. The expected search cost had a tendency to decrease with L when $\Gamma_{(k)DR}$ s were applied, which were constructed under the condition of an identical search success probability and various L s. This is because the GS algorithm on the $\Gamma_{(k)DR}$ with a smaller k value terminates with a lower search cost independent of search success or failure. Thus, we should construct $\Gamma_{(k)DR}$ using as large an L value as possible.

Figure 6 shows the relationship between a given success probability for the graph construction and an actual average success ratio obtained with an actual search using the constructed graph. The actual average success ratios always exceeded the corresponding given success probabilities.

7.5 Comparison with E²LSH

We compared the proposed method extended for a range query as described in Section 6 with the E²LSH reported in [1], although it is difficult to compare the two methods directly due to the differences in their operating principles. The range query problem for the comparison is to report a

set of points $R(\mathbf{q})$ within r from \mathbf{q} , given a set of points $\mathbf{x} \in X$ in a Euclidean space \mathbb{R}^d with the Euclidean distance, query point $\mathbf{q} \in \mathbb{R}^d$, and distance (or radius) r from \mathbf{q} . We evaluated the *recall* and the search cost defined for each method.

For a fair and simple comparison, we adopted *serial processing* for the proposed method’s search algorithm instead of the *parallel processing* in **Algorithm 4**. The search algorithm executed L search trials for each query in series, where each trial started from the corresponding initial vertex. During the successive L trials, once the distance between a query and a vertex was calculated, it was cached and not calculated twice. The search cost of the proposed method was evaluated by the number of distance calculations.

We briefly review the E²LSH to better understand its search cost. The E²LSH builds an index using a set of hash functions that maps a point into a sequence of integers. The hash function $h_{\mathbf{a},b}(\mathbf{x})$ is expressed by

$$h_{\mathbf{a},b}(\mathbf{x}) = \lfloor (\mathbf{a} \cdot \mathbf{x} + b) / w \rfloor, \quad (12)$$

where b and w are real parameters and \mathbf{a} is a random projection vector whose entries are chosen from a Gaussian distribution. By using \mathcal{L} sets each consisting of κ random projection vectors, a point is represented by distinct \mathcal{L} sequences, i.e., buckets each consisting of κ hash values derived from Eq. (12). To identify \mathcal{L} buckets for a point, we need $\kappa \cdot \mathcal{L}$ inner-product calculations.

In the search stage, the \mathcal{L} buckets for \mathbf{q} are identified by the $\kappa \cdot \mathcal{L}$ inner-product calculations. Then a union of the sets of buckets identical to those used for \mathbf{q} is made. E²LSH answers the range query from the exact distance evaluation of the points in the union of the sets. The parameters κ and \mathcal{L} are determined by a given failure probability δ and the collision probability $p(\mathcal{D}(\mathbf{x}, \mathbf{y}); w)$ of a pair of points (\mathbf{x}, \mathbf{y}) on a random projection vector, where $\mathcal{D}(\mathbf{x}, \mathbf{y})$ denotes a Euclidean distance between \mathbf{x} and \mathbf{y} . For the range query problem, given radius r from \mathbf{q} , the probability p that \mathbf{q} collides with \mathbf{x} within r in at least one bucket in \mathcal{L} is expressed by $1 - (1 - p^\kappa)^\mathcal{L}$. Then κ and \mathcal{L} are set at values such that the following inequality is fulfilled.

$$1 - \delta \leq 1 - (1 - p^\kappa)^\mathcal{L} \quad (13)$$

The search cost of E²LSH was measured by the sum of two costs: one is the product $\kappa \cdot \mathcal{L}$ corresponding to the number of inner-product calculations for identifying the \mathcal{L} buckets for \mathbf{q} and the other is the number of distinct points encountered in the \mathcal{L} buckets for selecting the points within r from \mathbf{q} by the exact distance calculation.

We used object set X with 60,000 points and query set Q with 4,850 points. E²LSH set the parameters r and w at 0.65 and 4.0, respectively, as in [1], and employed tuple (κ, \mathcal{L}) to minimize the expected search cost. In contrast, the proposed method employed L initial vertices per query and the same r as E²LSH, evaluated the $Recall(\mathbf{q})$ for each \mathbf{q} , and obtained the *average recall* by $(1/Q) \sum_{\mathbf{q} \in Q} Recall(\mathbf{q})$, where $Recall(\mathbf{q})$ is expressed by Eq. (14) in **Appendix A**. The expected search costs of the two methods were obtained by $(1/Q) \sum_{\mathbf{q} \in Q} c(\mathbf{q})$, where $c(\mathbf{q})$ denotes the search cost for each \mathbf{q} . Figure 7 shows the expected search costs of both methods on a logarithmic scale against the actual *average recall*. The expected search cost of the proposed method with $(k, L) = (10, 16)$ at an *average recall* of 0.91 was around one-sixth that of E²LSH, and at only 1655.1 corresponding to 2.76% of $|X|$. Moreover, the proposed method with $L = 8$

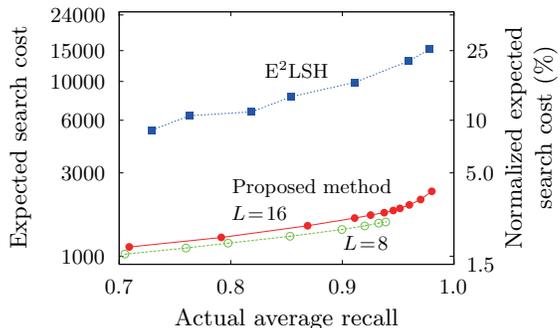


Figure 7: Expected search costs of proposed method and E²LSH on a logarithmic scale against actual average recall. The expected search cost of the proposed method ($L=16$) was around one-sixth that of E²LSH when the actual average recall was 0.91.

reduced the expected search costs from those where $L = 16$. Thus the proposed method efficiently performed a similarity search even for a range query.

8. CONCLUSION

We presented a fast approximate similarity search method that utilized a degree-reduced k -nearest neighbor graph (k -DR graph) with a controlled k value as a search index, and explored the k -DR graph along its edges by using a greedy search (GS) algorithm starting from multiple initial vertices with parallel processing. In the graph-construction stage, we determined the k value so that the GS algorithm on the k -DR graph with parallel processing succeeds with at least a given success probability, by using an ensemble average of the greedy-search success probability estimated based on *basins* in the k -DR graph. Furthermore, we extended the proposed method for a nearest-neighbor query to that for a range query. Compared with E²LSH for the MNIST database, the experimental results demonstrated that the extended method was superior to E²LSH in terms of the expected search cost for achieving a given recall.

9. ACKNOWLEDGMENTS

This work is partially supported by KAKENHI (20500109) and the FIRST program of JSPS.

10. REFERENCES

- [1] A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. *Nearest-neighbor methods in learning and vision; theory and practice*, chapter 3: Locality-sensitive hashing using stable distributions. The MIT Press, Cambridge, Massachusetts, 2005.
- [2] A. Andoni and P. Indyk. Near-optimal algorithms for approximate nearest neighbor in high dimensions. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 459–468, October 2006.
- [3] K. Aoyama, K. Saito, T. Yamada, and N. Ueda. Fast similarity search in small-world networks. In R. M. et al., editor, *Complex Networks: Int. Workshop on Complex Networks*, pages 185–196. Springer, 2009.
- [4] K. Aoyama, S. Watanabe, H. Sawada, Y. Minami, N. Ueda, and K. Saito. Fast similarity search on a large speech data set with neighborhood graph

indexing. In *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, pages 5358–5361, March 2010.

- [5] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comp. Surveys*, 33(3):273–321, September 2001.
- [6] J. Chen, H.-R. Fang, and Y. Saad. Fast approximate k NN graph construction for high dimensional data via recursive Lanczos bisection. *The Journal of Machine Learning Research*, 10:1989–2012, 2009.
- [7] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, July 1985.
- [8] H. Hacid and T. Yoshida. Neighborhood graphs for indexing and retrieving multi-dimensional data. *J. Intelligent Information Systems*, 34:93–111, 2010.
- [9] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database System*, 28(4):517–580, December 2003.
- [10] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. ACM Symp. Theory of Computing*, pages 604–613, May 1998.
- [11] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80(9):1502–1516, September 1992.
- [12] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proc. ACM Symp. on Theory of Computing*, pages 741–750, May 2002.
- [13] R. E. Korf, W. Zhang, I. T. Thayer, and H. Hohwald. Frontier search. *Journal of the ACM*, 52(5):715–748, September 2005.
- [14] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proc. ACM-SIAM Symp. Discrete Algorithms*, pages 798–807. SIAM/SIGACT, January 2004.
- [15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
- [16] Ö. Şimşek and D. Jensen. Decentralized search in networks using homophily and degree disparity. In *Proc. Int. Joint Conf. Artificial Intelligence*, pages 304–310, July 2005.
- [17] M. T. Orchard. A fast nearest-neighbor search algorithm. In *Proc. Int. Conf. Acoustics, Speech, Signal Process.*, volume 4, pages 2297–2300, 1991.
- [18] P. Ram, D. Lee, W. B. March, and A. G. Gray. Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems 22 (Dec 2009)*. MIT Press, 2010.
- [19] J. Sakagaito and T. Wada. Nearest first traversing graph for simultaneous object tracking and recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 1–7, June 2007.
- [20] T. B. Sebastian and B. B. Kimia. Metric-based shape retrieval in large databases. In *Proc. Int. Conf. Pattern Recognition*, volume 3, pages 291–296, 2002.
- [21] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *Proc. SIGMOD*, pages 563–575, June 2009.
- [22] D. J. Watts and S. H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.

APPENDIX

A. PARALLEL GREEDY SEARCH ALGORITHM FOR RANGE QUERIES

A parallel greedy search algorithm for range queries (*PGSR*) is shown as a pseudo-code in **Algorithm 4**. The symbols in the pseudo-code have the same meaning in **Algorithm 3**. Given $\Gamma_{(k)DR}$, query q , a set of initial vertices $X_0(q)$, and radius r from q , the *PGSR* begins to traverse $\Gamma_{(k)DR}$ from the initial vertex $x_{0[i]}$ with the *GS* algorithm in parallel after initialization on **Lines 2–7**. When the current expanded vertex $z_{[i]}$ is within the radius, the *PGSR* pushes $z_{[i]}$ ’s adjacent vertex $y_{[i]}$, whose dissimilarity from q is less than r , into the stack $W_{[i]}$, and simultaneously adds $y_{[i]}$ to the set $\tilde{R}_{[i]}$, as shown on **Lines 11–15**. The *PGSR* pops a vertex from $W_{[i]}$ and sets it at the next expanded vertex $z_{[i]}$ exactly when the *GS* algorithm terminates, as shown on **Line 21**. The *PGSR* continues to explore $\Gamma_{(k)DR}$, collecting the vertices within the radius until stack $W_{[i]}$ is empty as shown on **Line 22**. Finally, the L sets, $\tilde{R}_{[i]}(q)$, of the vertices within the radius are merged into $\tilde{R}(q)$, and the *recall* of the search result with the *PGSR* is evaluated by

$$\text{Recall}(q) = \frac{|\bigcup_{i=1, \dots, L} \tilde{R}_{[i]}(q; \Gamma_{(k)DR}) \cap R(q)|}{|R(q)|}. \quad (14)$$

Algorithm 4 *Parallel Greedy Search Algorithm for Range Query; PGSR($q, X_0(q); r, \Gamma_{(k)DR}$)*

Input: $\Gamma_{(k)DR}$, r , $q \in \mathcal{X}$,
 $X_0 = (x_{0[1]}, x_{0[2]}, \dots, x_{0[L]}), x_{0[i]} \in X$

Output: $(\tilde{R}(q), c)$

1. **for** each initial vertex $x_{0[i]}$ **in parallel do**
2. Set $z_{[i]} \leftarrow x_{0[i]}$, $x_{[i]} \leftarrow z_{[i]}$,
 $D_{min[i]} \leftarrow D(q, z_{[i]})$, $C_{[i]} \leftarrow \{z_{[i]}\}$
3. **if** $D(q, z_{[i]}) < r$ **then**
4. Set $\tilde{R}_{[i]}(q) \leftarrow \{z_{[i]}\}$, $W_{[i]} \leftarrow \{z_{[i]}\}$
5. **else**
6. Set $\tilde{R}_{[i]}(q) \leftarrow \emptyset$, $W_{[i]} \leftarrow \emptyset$
7. **end if**
8. **repeat**
9. **while** $z_{[i]} = x_{[i]}$ **do**
10. Set $x_{[i]} \leftarrow \arg \min_{y_{[i]} \in V_{(k)DR}(z_{[i]})} \{D(q, y_{[i]})\}$
11. **for all** $y \in V_{(k)DR}(z_{[i]})$ **do**
12. **if** $(D(q, y_{[i]}) < r) \wedge (y_{[i]} \notin C_{[i]})$ **then**
13. Set $\tilde{R}_{[i]}(q) \leftarrow \tilde{R}_{[i]}(q) \cup \{y_{[i]}\}$,
push($W_{[i]}, y_{[i]}$)
14. **end if**
15. **end for**
16. Set $C_{[i]} \leftarrow C_{[i]} \cup V_{(k)DR}(z_{[i]})$
17. **if** $D(q, x_{[i]}) < D(q, z_{[i]})$ **then**
18. Set $D_{min[i]} \leftarrow D(q, x_{[i]})$, $z_{[i]} \leftarrow x_{[i]}$
19. **end if**
20. **end while**
21. Set $z_{[i]} \leftarrow \text{pop}(W_{[i]})$, $x_{[i]} \leftarrow z_{[i]}$
22. **until** $W_{[i]} = \emptyset$
23. **end for**
24. Set $\tilde{R}(q) \leftarrow \bigcup_{i=1, \dots, L} \tilde{R}_{[i]}(q)$,
 $c \leftarrow \arg \max_{i=1, \dots, L} \{C_{[i]}\}$
25. **return** $(\tilde{R}(q), c)$
