International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

The Influence of Hubness on NN-Descent

Brankica Bratić
Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 3,
21101, Novi Sad, Serbia
brankica.bratic@dmi.uns.ac.rs

Michael E. Houle
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo 101-8430, Japan
meh@nii.ac.jp

Vladimir Kurbalija
Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 3,
21101, Novi Sad, Serbia
kurba@dmi.uns.ac.rs

Vincent Oria
Dept. of Computer Science, New Jersey Inst. of Technology, University Heights
Newark, New Jersey 07102, The United States of America
vincent.oria@njit.edu

Miloš Radovanović
Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 3,
21101, Novi Sad, Serbia
radacha@dmi.uns.ac.rs

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

The K-nearest neighbor graph (K-NNG) is a data structure used by many machine-learning algorithms. Naive computation of the K-NNG has quadratic time complexity, which in many cases is not efficient enough, producing the need for fast and accurate approximation algorithms. NN-Descent is one such algorithm that is highly efficient, but has a major drawback in that K-NNG approximations are accurate only on data of low intrinsic dimensionality. This paper represents an experimental analysis of this behavior, and investigates possible solutions. Experimental results show that there is a link between the performance of NN-Descent and the phenomenon of *hubness*, defined as the tendency of intrinsically high-dimensional data to contain *hubs* – points with large in-degrees in the K-NNG. First, we explain how the presence of the hubness phenomenon causes bad NN-Descent performance. In light of that, we propose four NN-Descent variants to alleviate the observed negative influence of hubs. By evaluating the proposed approaches on several real and synthetic data sets, we conclude that our approaches are more accurate, but often at the cost of higher scan rates.

Keywords: NN-Descent; k-nearest neighbor graph; hubness.

1. Introduction

A K-nearest neighbor graph (K-NNG) is a directed graph whose vertices represent elements upon which some distance function is defined. Vertex v is unidirectionally connected to vertex u only if vertex u is one of the K elements that are nearest to point v with respect to the defined distance function, in which case, vertex u is said to be a *neighbor* of vertex v. K-NNGs are used in many machine-learning and data-mining algorithms,[1] including classification,[2] similarity search,[3] clustering and outlier detection problems,[4–6] manifold learning.[7–9] They are also used in other areas, such as robot motion planning[10] and computer graphics.[11] Fig. 1 shows an example of a simple K-NNG for $K = 2$, whose vertices are five 2-dimensional points. The distance function that was used for the creation of this K-NNG was Euclidean (L_2) distance. As can be seen, each point has exactly two outgoing edges (which is determined by the K value), while the number of incoming edges varies. The figure also depicts the neighborhood of one vertex – this vertex is colored blue, while its neighbors are colored red.

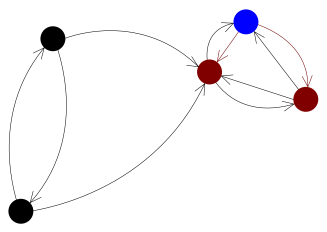

Fig. 1. An example of K-NNG ($K = 2$) of 2-dimensional points with the Euclidean (L_2) distance as an underlying distance function.

Naive brute-force computation of K-NNG entails $\frac{n\cdot(n-1)}{2}$ distance computations, which leads to quadratic time complexity. Therefore, for large scale data sets (with big n values) computation of K-NNG could be very inefficient. For that reason many approximation algorithms were developed (some of them are presented in Section 2). One such algorithm is *NN-Descent*, which efficiently creates K-NNG approximations. However, the algorithm has one major drawback – it does not produce accurate approximations when the dimensionality of a data set is large. As a consequence, high-dimensional data sets like multimedia, medical, DNA data sets, etc.,[12] cannot make use of NN-Descent.

Although several improvements have been proposed for NN-Descent (see Section 2), so far none of them have given an explanation for the variability of its performance, nor have they given a universal solution to the problem. In our earlier work[13] we provided an explanation for this variation in terms of *hubness*, a measure of the variation in nearest neighbor relationships.[14] We also proposed two approaches that improve the quality of NN-Descent approximations for high-dimensional data sets. In this article we extend the previous research by adding two additional approaches and by improving experimental analysis. The main improvement of the experiments are in terms of data sets that we used to validate our findings – we added one additional synthetic data set that is larger (it contains 100,000 instances) and unlike in previous research, we now also use real data sets.

Section 2 overviews previous research in the area. Section 3 will describe the necessary background by providing an explanation of the NN-Descent algorithm and the phenomenon of hubness. In Section 4 we show how a minor modification of the NN-Descent algorithm can produce estimates of hubness. Experimental results are presented in Section 5 that demonstrate the empirical effect of hubness on NN-Descent. Section 6 proposes methods that to some extent overcome the problems of NN-Descent on high-dimensional data. Finally, Section 7 gives an overview of the conclusions and results, and indicates directions for future work.

2. Related Work

Numerous methods for K-NNG construction have been developed in order to decrease the computational cost, but many of them introduce certain restrictions and therefore do not apply to the general case. Chen et al.[15] introduced an approximate divide and conquer algorithm that uses Lanczos bisection[16] for the divide step. The algorithm behaves well on high-dimensional data, but is limited to Euclidean distance. Jang et al.[17] developed an approximate K-nearest neighbor search method based on the Earth Mover's Distance (EMD) for fast multimedia retrieval. In their method the index is built by using M-tree together with dimensionality reduction, while the approximate EMD is used for the index retrieval. A little less restrictive in terms of distance function is an approach by Paredes et al.[18] The paper presents a general methodology for K-NNG construction when the distance function is a metric. For low- and medium-dimensional spaces the algorithm achieves high speedup, while for high-dimensional ones the speedup is lower. A parallel algorithm for K-NNG computation was presented by Connor et al.[19] Their algorithm is based on Morton order or Z-order of points and is most suitable for multi-core machines. Zhang et al.[20] developed an algorithm that creates K-NNG approximation for an arbitrary distance function. The main idea of the algorithm is to group similar points by using locality sensitive hashing (LSH).[21]

Even though many algorithms for fast K-NNG computations were developed, one of the famous and most efficient ones is the one introduced by Dong et al.[22] The algorithm is called NN-Descent and it efficiently produces highly accurate K-NNG approximations independently of the underlying distance function. As reported by the authors, empirical complexity of NN-Descent is around $O(n^{1.14})$. Although the results produced by this algorithm are mostly excellent, there is still one major drawback — NN-Descent often produces inaccurate results when it is used on data of high intrinsic dimensionality.

Park et al.[23] developed a modification of NN-Descent based on a greedy filtering method, with which they obtained improved results for high-dimensional data together with the cosine similarity measure. Houle et al.[24] introduced NNF-Descent (Nearest Neighbor Feature Descent), an NN-Descent-based feature sparsification method optimized for image databases. It uses the Laplacian score in order to identify noisy values, which are then replaced with zeros. Debatty et al.[25] presented a method for constructing a K-NNG for large text data sets. This method applies NN-Descent only to smaller buckets of data, leading to a lower execution time.

3. Background

In this section we will review the NN-descent algorithm (Section 3.1) and the hubness phenomenon (Section 3.2), which, as we will demonstrate later, has a significant impact on NN-Descent.

3.1. *NN-Descent*

The main purpose of the NN-Descent algorithm is to create a good approximation of the true K-NNG, and to create it as fast as possible. The basic assumption made by NN-Descent can be summarized as "my neighbor's neighbor is also likely to be my neighbor." The algorithm starts by creating a random K-NNG, and then iteratively improves the neighbor lists by using the current neighborhood relationships to suggest new neighbor candidates. During the execution of NN-Descent, for a data point u, we will refer to the current list of its K nearest neighbors as the *NN list* of u. A reverse neighbor of u is a data point which has u in its own NN list; we will refer to the current list of reverse neighbors of u as its *RNN list*.

NN-Descent is organized into stages, during each of which a single improvement iteration is applied to each of the data points. The algorithm iteration for point u examines its NN and RNN lists to see whether any points in these lists should be added to the NN lists of any of the others. If this is the case, the algorithm makes the necessary changes, and increases the number of total updates. After every stage, the algorithm tests a termination condition based on the number of updates that were applied during the stage. If the number of updates is smaller than a supplied threshold, the algorithm terminates; otherwise, it continues with the next stage. An outline of NN-Descent is shown as Algorithm 1 and the list of NN-Descent parameters is given in Table 1.

An example of NN-Descent execution is given in Table 2. In the example, NN-Descent is ran on a small data set of five 2-dimensional points. The table contains two columns, each representing one NN-Descent iteration. As can be seen, NN-Descent converged after only two iterations. In the first iteration, the algorithm starts with a random NN list and performs local joins from each point. For example,

Algorithm 1: Outline of NN-Descent algorithm.

```
input  : data set $D$, distance function $dist$, neighborhood size $K$
output: $K$-NNG $G$
1  foreach data point $u \in D$ do
2      Initialize $G$ by randomly generating a tentative $K$-NN list for $u$ with an
         assigned distance of $+\infty$;
3  end
4  repeat
5      foreach data point $u \in D$ do
6          Check different pairs of $u$'s neighbors $(v, w)$ in $u$'s $K$-NN and $R$-NN
             (reverse nearest neighbor) lists, and compute $dist(v, w)$;
7          Use $\langle v, dist(v, w)\rangle$ to update $w$'s $K$-NN list, and use $\langle w, dist(v, w)\rangle$ to
             update $v$'s $K$-NN list;
8      end
9  until $G$ converges;
10 Return $G$.
```

Table 1. List of NN-Descent parameters.

Parameter	Description
K	K value of the resulting K-NNG approximation.
conv	Convergence criterion (value between 0 and 1). Algorithm converges when there is less than $conv \cdot K \cdot N$ updates in the current iteration, where N is the number of K-NNG vertices.
ρ	Sampling (value between 0 and 1, not including 0). The algorithm takes $\rho \cdot K$ points from each NN and RNN lists.

point A has points B and D in its NN list, while its RNN list consists of points B and E (because point A is in B's and E's NN lists). Local joins from point A are actually pairwise comparisons of A's direct and reverse neighbors. Since A's neighbors are {B, D, E} (the union of A's NN and RNN lists), the following comparisons (local joins) are performed: (B, D), (B, E) and (D, E). Out of all these local joins, only (D,E) resulted with an update – point D is inserted into the NN list of point E. After local joins from point A, local joins from other data-set points are performed, too, which resulted with two additional updates of NN lists. At the end of the iteration, the total number of updates was 3, which means that algorithm did not converge, so it resumed with the next iteration. In iteration 2, some points of NN lists are flagged with the N symbol, while the others are not. The symbol N marks the points that are newly added to the NN lists, and therefore need to be compared with the rest of the neighborhood. (Old points do not have to be compared with other old points, since this was already done.) This feature is an optimization of the NN-Descent algorithm. An example of this optimization can be seen in the local joins from point D of iteration 2. Instead of making all pairwise comparisons, only the comparisons with new points are performed (in this case, only the comparisons with point E). Finally, since iteration 2 does not lead to any NN list update, the algorithm terminates.

Table 2. An example of NN-Descent execution on a small data set of five 2-dimensional points. Euclidean distance was used as a distance function. In each iteration of the algorithm NN and RNN lists are shown for all data-set points. NN lists are given as: $X \rightarrow Y^{[N]}(D_{XY}), Z^{[N]}(D_{XZ})$, where X, Y and Z are data-set points, X being the point whose NN list is shown, Y and Z being the points from X's NN list, D_{XY} and D_{XZ} are distances between X and Y, and between X and Z, respectively, while the N letter is optional and, if present, it marks a point in NN list that is newly added. RNN lists are represented analogously, except that distance values are omitted (since they are already given in NN lists). In addition, the table contains information about local joins. One local join is represented as: $(X, Y, D_{XY}) - U$, where X and Y are data-set points, D_{XY} is the distance between these two points and U is the count of updates that were introduced by the local join (0 if there where no updates, 1 if only one of the two points changed its NN list and 2 if both points changed their NN lists.).

Data set points
A(1,6); B(2,8); C(6,5); D(4,9); E(8,7)

Iteration 1	Iteration 2
NN lists	**NN lists**
$A \rightarrow B^N(1), D^N(3)$	$A \rightarrow B(1), D(3)$
$B \rightarrow A^N(1), D^N(2)$	$B \rightarrow A(1), D(2)$
$C \rightarrow D^N(2), B^N(4)$	$C \rightarrow D(2), E^N(2)$
$D \rightarrow B^N(2), C^N(2)$	$D \rightarrow B(2), C(2)$
$E \rightarrow B^N(6), A^N(7)$	$E \rightarrow C^N(2), D^N(4)$
RNN lists	**RNN lists**
$A \rightarrow B^N, E^N$	$A \rightarrow B$
$B \rightarrow A^N, C^N, D^N, E^N$	$B \rightarrow A, D$
$C \rightarrow D^N$	$C \rightarrow D, E^N$
$D \rightarrow A^N, B^N, C^N$	$D \rightarrow A, B, C, E^N$
$E \rightarrow$	$E \rightarrow C^N$
Local joins	**Local joins**
From point A: B^N, D^N, E^N	*From point A:* B, D
(B,D,2)-0; (B,E,6)-0; (D,E,4)-1	-
From point B: A^N, C^N, D^N, E^N	*From point B:* A, D
(A,C,5)-0; (A,D,3)-0; (A,E,7)-0; (C,D,2)-0;	-
(C,E,2)-2; (D,E,4)-0	*From point C:* D, E^N
From point C: B^N, D^N	(D,E,4)-0;
(B,D,2)-0;	*From point D:* A, B, C, E^N
From point D: A^N, B^N, C^N	(A,E,7)-0; (B,E,6)-0; (C,E,2)-0;
(A,B,1)-0; (A,C,5)-0; (B,C,4)-0;	*From point E:* C^N, D^N
From point E: A^N, B^N	(C,D,2)-0;
(A,B,1)-0;	

The speed of NN-Descent is strongly influenced by the K value. As K becomes larger, the algorithm becomes slower. To be more precise, the time complexity has a quadratic dependence on K, since during each iteration, points appearing in NN or RNN lists of the same point are evaluated as candidates for each others NN lists.

As a way to reduce the number of combinations of points considered, the authors introduced sampling.[22] Sampling reduces the number of evaluations by taking a random selection of NN and RNN lists, and then operates only on the points from those lists. Sampling is controlled by a parameter ρ that takes a value from 0 to 1. The algorithm takes $\rho \cdot K$ points both from NN and RNN lists.

The second drawback of NN-Descent is that quality of approximation it produces is highly influenced by the intrinsic dimensionality of the data set, in that the quality of the approximation decreases as the intrinsic dimensionality increases. The reason for this has not been adequately explained. Although low-dimensional embeddings have often been used as a general method of reducing the complexity of data indexing, no solutions have yet been proposed that allow the NN-Descent strategy to work more effectively with high-dimensional data. In this paper we will explain the influence of high dimensionality on NN-Descent, and will also propose four approaches that are designed to overcome this challenge to some extent.

3.2. *Hubness*

Hubness is an aspect of the curse of dimensionality pertaining to nearest neighbors which has come to the attention of the research community only relatively recently.[14] Let $D \subset \mathbb{R}^d$ be a set of data points and let the *k-occurrences* of point $x \in D$ be the number of times x occurs in k-nearest-neighbor lists of other points from D. $N_k(x)$ is then the number of *k-occurrences* of point $x \in D$ — that is, the in-degree of node x in the K-NNG. As the dimensionality of the data increases, the distribution of $N_k(x)$ becomes considerably skewed. As a consequence, some data points, which we will refer to as *hubs*, are included in many more k-nearest-neighbor lists than other points.

In the remainder of the discussion we will refer to the number of k-occurrences of point $x \in D$ as its *hubness value*. If a data set contains many points that are hubs — that is, if the distribution of hubness values is highly skewed — it can be said that the hubness phenomenon is present therein. It has been shown that hubness, as a phenomenon, appears in high-dimensional data as an inherent property of intrinsically high dimensionality, and is not an artifact of finite samples nor a peculiarity of specific data sets.[14] It was shown that hubness influences various data-mining and machine-learning algorithms,[14, 26–29] often in a negative way. Practical computation of (exact) hubness values entails the creation of the K-NNG, from which hubness values can be easily obtained by extracting the in-degrees of nodes.

4. Hubness Estimation Using NN-Descent

Approximate hubness values for each data point could be very easily calculated during NN-Descent, with minimal impact on algorithm performance. At the very beginning, we initialize the hubness values of each data set point to zero. Then, during algorithm execution, we increase the hubness value of a given point by one if that point is added to the NN list of some other point, and analogously, we decrease the hubness value by one if the point is removed from some NN list.

Some of the results that we obtained after this small upgrade of NN-Descent algorithm are illustrated in Fig. 2, which shows the correlation between estimated and exact hubness values of data points. For the purposes of the experimentation in this paper, we created several synthetic data sets drawn from a Gaussian distribution with standard deviation 1 and mean $(0, 0, \ldots, 0)$. The data sets each have $n = 10000$ points, but differ in their dimensionality d. During the analysis we also varied the size of the nearest neighbor lists, represented by the parameter K. We considered the values 10, 20 and 50 for K, and the values 2, 10, 20 and 100 for parameter d. As a distance measure we used the Euclidean distance. In Fig. 2 we show results only for the data set of highest dimensionality ($d = 100$), since the hubness phenomenon is known to strengthen as the dimensionality of the data increases. As can be seen, the correlation between real and estimated hubness values is very high. A shortcoming of NN-Descent is that estimated hubness values are lower than they should be for points with low true hubness values, and greater than they should be for points with high true hubness. The precision improves as K increases, but even for small K values, the algorithm produces strong correlations.

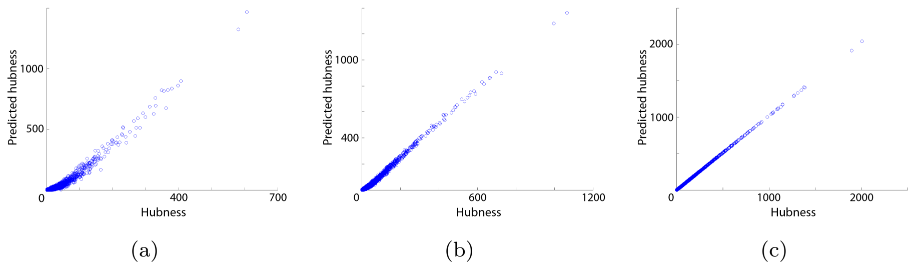

Fig. 2. Correlation between estimated and exact hubness values of data set points. The figures show the results for data sets with dimensionality $d = 100$ and the following choices of neighbor list sizes: (a) $K = 10$; (b) $K = 20$; (c) $K = 50$.

5. Influence of Hubness on NN-Descent

NN-Descent is known to produce large amount of incorrect K nearest neighbors when applied upon high-dimensional data.[22] Since hubness is a phenomenon that appears in high-dimensional data, and at the same time was shown to influence many other data-mining algorithms, in this section we investigate whether it also influences the performance of NN-Descent.

For a given data set point, we will define the hit count to be the number of K-nearest neighbors produced by NN-Descent that are at the same time one of the point's K-nearest neighbors in the exact K-NNG. In other words, the hit count tells us how many correct selections NN-Descent has made for the neighborhood list of a given data point. The first part of our analysis will be to examine the correlation between hit count values and hubness values of data points. In order to do that, we first calculated the exact K-NNGs, as well as hubness values for each data set point. We then ran NN-Descent without sampling and early termination, and calculated hit count values based on the previously calculated K-NNG and NN-Descent graph approximation. We did this for all generated data sets.

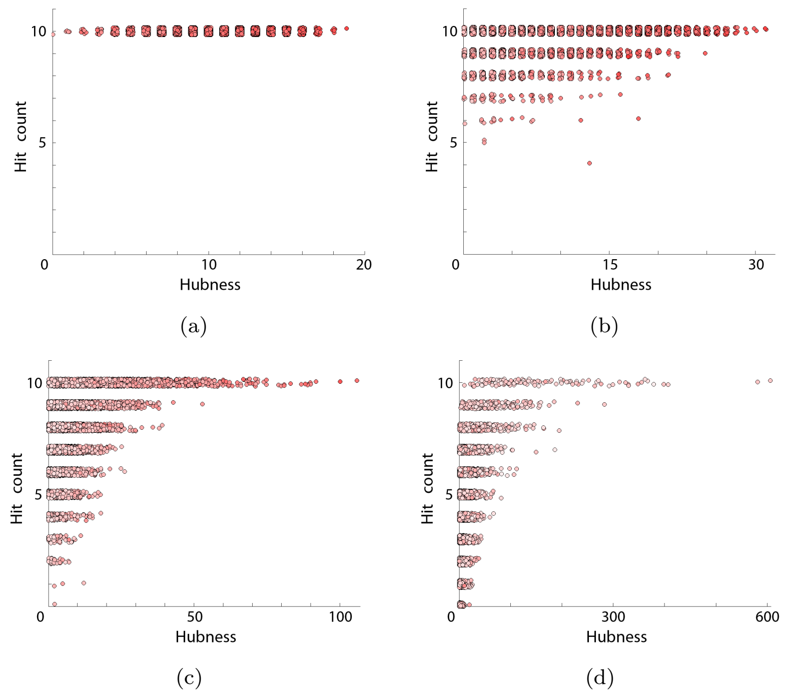

Fig. 3. Hubness and hit count values of data set points. The figures show the results for neighbor list sizes fixed at $K = 10$, and for the following choices of data set dimensionalities: (a) $d = 2$; (b) $d = 10$; (c) $d = 20$; (d) $d = 100$.

Some of the results can be seen in Fig. 3. Let us now say that for the points of fixed hubness value, hit count values are distributed between $\mathrm{hc}_{\min}$ and $\mathrm{hc}_{\max}$. Results are showing us that both values are directly proportional to the hubness value, with this behavior being much more evident in the case of the $\mathrm{hc}_{\min}$ value. In the case of $\mathrm{hc}_{\max}$, the phenomenon is less evident (Fig. 3(d)). As a consequence, points with extremely high hubness values have $\mathrm{hc}_{\min} \simeq \mathrm{hc}_{\max} \simeq K$, which means that their hit counts usually attain the greatest possible value. For the lowest hubness values, it holds that $\mathrm{hc}_{\min}$ is near zero, while the $\mathrm{hc}_{\max}$ depends on K and d; however, for all tested data sets, this value is K or very nearly K. In other words, some points with extremely low hubness values have a high hit count, and some have hit counts that are very low, which means that the probability of error when determining the K nearest neighbors of point x is inversely proportional to the hubness of x. The described phenomenon is more evident in data sets with lower K values and greater dimensionality d. If K is sufficiently large, and d is sufficiently small, the phenomenon is less evident, as the hubness values of the points are more uniform. In these cases, NN-Descent produces a very good approximation of K-NNG. In further research we will concentrate on settings for which the performance of NN-Descent is poor, as we saw for the case where $d = 100$.

Now that the influence of hubness on NN-Descent has been established, we examine the cause of this phenomenon. As was already described in Section 3.1, the NN-Descent algorithm improves the K-NN graph approximation in each iteration. If the hubness of the data set is high, then in the first few stages hubs will be placed among the K nearest neighbors of a large number of data points. This implies that the NN lists of a majority of points will contain hubs. Points with lower hubness values will quickly be expelled from NN lists of other points. This expulsion of non-hubs from NN lists implies that the neighborhoods of those points, according to the nature of the algorithm, will not be updated as often — in order for the NN list of a given point to be updated, that point must be present in the NN or RNN lists of other data points. If a point is rapidly expelled from NN lists, then it will be updated only from RNN lists of the points that are very unlikely to be their true neighbors, since the initial approximation of the K-NN graph is essentially random. For some points of low hubness value, this produces relatively poor results.

Another issue to consider is that some points with low hubness values may still have high hit count values, while other low-hubness points may have very low hit counts. In order to quantify the extent to which the variation in hit count values depends on hubness, we calculated hit count values and their standard deviations for each data point, over 100 runs of NN-Descent, each time generating the random graph with a different random seed. The results are shown in Fig. 4, where the intensity of the red color represents the mean of hit count values among different runs — points with greater intensity have greater average hit count values. As can be seen, points with low hubness values have hit counts with a relatively high standard deviation, while points with high hubness values have hit count standard deviations that tend toward zero. This implies that hit count values vary significantly across different runs, while the only difference between runs is the initial random graph. All of this leads us to the conclusion that the main factor that affects the hit count of points with low hubness is the initial random K-NNG: if the point is in a high-quality neighborhood in the initial random graph, then NN-Descent is more likely to produce a high hit count. Otherwise, the point will be quickly expelled from NN lists, and will stay trapped in the RNN lists of points that are not its true neighbors, leading to a relatively low probability that the point's NN list will be updated with true neighbors.

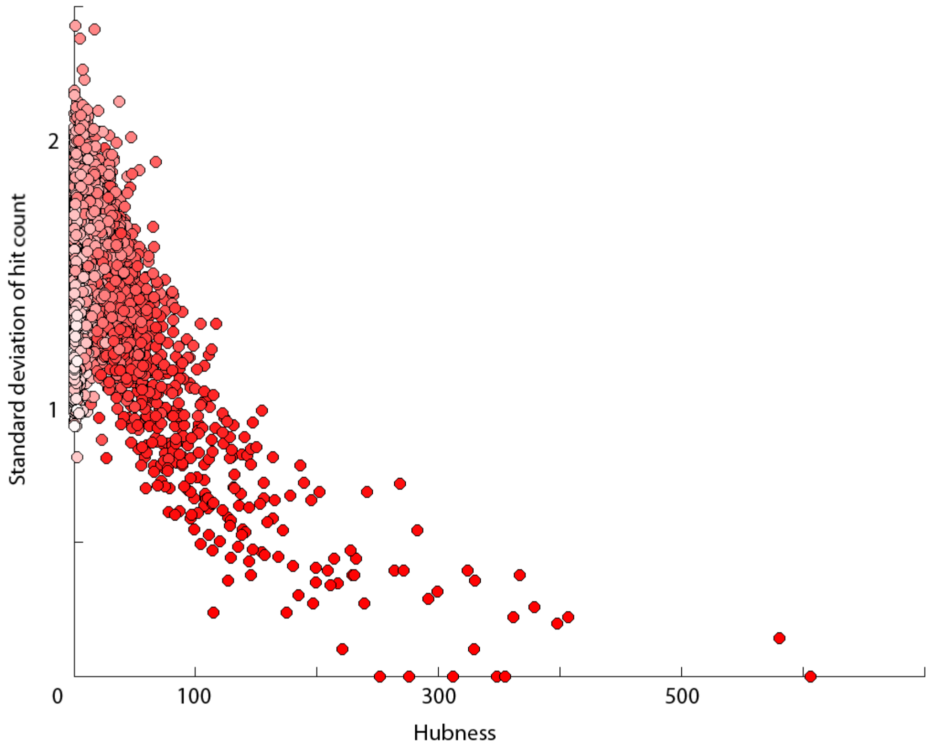

Fig. 4. Results with $K = 10$ and $d = 100$. The y-axis shows standard deviations of hit count values for 100 runs of NN-Descent, while the x-axis shows hubness values of the corresponding data points.

6. Proposed Methods for Improving NN-Descent on High-Dimensional Data

In order to overcome the problem described in Section 5, we implemented four different approaches. The first one is based on the idea that NN lists should not be updated strictly according to the NN and RNN lists of other points. The goal is to integrate the information about hubness values into the choice of points to compare with the current approximation of the NN list. A high hubness value indicates that a certain point already has a reasonably stable NN list, and vice versa — a low hubness value suggests that the point has a greater probability of being assigned an incorrect NN list. The second approach is based on the observation that for greater K values the algorithm tends to be more accurate. If we run the NN-Descent algorithm with a greater K value, and then reduce the resulting K-NNG to the K we actually need, the precision of final graph is expected to be better. The third approach provides an easy way to fine-tune the minimum number of comparisons for each data point.

In that way, a larger number of comparisons could be assigned to those points that actually need it. Finally, the fourth approach is built up on the fact that point's initial position in the random graph influences its hit count value. In order to place point in the right neighborhood, we perform additional random comparisons to the points that need it.

In Section 6.1 experimental setup will be described, Section 6.2 contains information about the performance of the original NN-Descent algorithm, while Sections 6.3, 6.4, 6.5 and 6.6 give insights into the new approaches and their performance.

6.1. *Experimental setup*

In order to validate proposed approaches against the original NN-Descent algorithm, we ran experiments on high-dimensional synthetic and real data sets. Table 3 summarizes properties of all data sets that were used in the experiments.

Table 3. Data sets that are used in the experiments.

Name	Type	Instances	Dimensionality
i10000d100	Synthetic	10000	100
i100000d100	Synthetic	100000	100
BCI5	Real	31216	96
Google-23	Real	6686	1937
ISOLET	Real	7797	617
MNIST	Real	70000	784

Data sets **i10000d100** and **i100000d100** were created for the purpose of this research. Their instances have 100 dimensions, each being a value generated by uniformly choosing random real number from range $[-1, 1]$. **BCI5**[30] is a brain-computer interface data set of brain signal recordings taken while the subject contemplated some action. Data set **Google-23**[31] consists of 6686 faces extracted from web images of 23 celebrities. For each face, 13 points of interest were detected, each of which was represented by a 149-dimensional vector. Concatenating these 13 vectors into a single descriptor yielded a 1937-dimensional data point for each face image. **ISOLET**[32] from the UCI repository[33] is a data set of spoken letters containing 26 classes of 150 instances each (3 instances are missing in the data set), with each class referring to a letter of the alphabet. The total 617 features include spectral coefficients, contour features and sonorant features. Finally, the **MNIST**[34] data set has 70000 images of handwritten digits. The 784 pixel values of each image were treated as its image features.

We used three measures to express the effectiveness of algorithms: *recall*, *scan rate* and *execution time*. Recall can be expressed as the sum of all hit count values divided by the product of data-set size and K (and, in this case, recall is actually equivalent to precision and accuracy). Its definition is given in Equation 1, D being a data set (set of points), hc being a function that returns the hit count value of a given point and n being the data set size.

$$recall = \frac{\sum_{X \in D} hc(X)}{n \cdot K} \tag{1}$$

There are $\frac{n \cdot (n-1)}{2}$ calls of the distance function during naive K-NNG computation, while approximation algorithms tend to lower this number. The scan rate value was introduced for comparison of approximation algorithms in terms of the number of distance function calls. It represents the ratio between number of calls to the distance function and $\frac{n \cdot (n-1)}{2}$. Scan rate is given in Equation 2, *dists* value being the number of calls to the distance function, and n being data-set size.

$$scanrate = \frac{dists}{\frac{n \cdot (n-1)}{2}} \tag{2}$$

Finally, execution time represents time in seconds needed for an algorithm to execute. All the algorithms that were ran upon a single data set were executed on the same machine, which made it possible to compare execution times of different algorithms.

The experiments were run on Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz with 96GB of RAM, but only 2GB being available per core. Each experiment was ran 5 times meaning that all presented values are the averages of 5 different runs.

6.2. *NN-Descent performance*

In order to be able to compare the results of the original NN-Descent with the results of the proposed improvements, we show the performance of the original algorithm in Table 4. Let us present a brief overview of the results generated by the original NN-Descent algorithm.

The results of the original algorithm, as shown in Table 4, depend strongly on the size of NN lists (K), on the size of the data set and on the intrinsic dimensionality of the data set. Higher K values cause an increase of recall values, and scan rates as well. Both are the consequence of the fact that higher K values mean more local joins (by the nature of the algorithm), and if there are more local joins, the recall is more likely to be higher and the scan rate is higher by definition.

For a larger data set (i.e. for data set with a higher number of instances), the outcome of NN-Descent is such that both recall and scan rate values decrease. This phenomenon can be observed in Table 4 by comparing the results of the data sets i10000d100 and i100000d100. These two data sets have exactly the same characteristics except that data set i10000d100 has 10,000 instances, while data set i100000d100 has 100,000 instances. The explanation of this NN-Descent's behavior is connected with the phenomenon from the Fig. 4. As it can be seen in the figure, the position of the point in the initial random K-NNG is very important for the

Table 4. Recall, scan rate and execution time values for the **NN-Descent** algorithm with different K values (5, 10, 20), $conv = 0.01$ and $\rho = 1$.

	i10000d100			i100000d100			BCI5		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.09	0.36	0.73	0.02	0.1	0.36	0.57	0.97	0.99
Time	11.65	49.18	175.23	154.43	631.62	2730.91	61.56	116.8	359.86
Scan rate	0.03	0.13	0.48	0	0.01	0.06	0.01	0.04	0.13

	Google-23			ISOLET			MNIST		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.63	0.92	0.98	0.83	0.98	1	0.74	0.96	0.99
Time	26.14	54.83	147.84	13.07	33.55	104.5	276.9	458.89	1379.05
Scan rate	0.05	0.14	0.43	0.04	0.12	0.37	0.01	0.02	0.06

point's final hit count value – if the point is in the better neighborhood in the initial graph, its hit count value is more likely to be higher. The probability that the point is placed in the good neighborhood in the initial random K-NNG decreases as the data set size increases. Therefore, the final recall of NN-Descent on larger data set decreases. Scan rate value also decreases due to the fact that denominator of the scan rate equation increases quadratically on data-set size, while the numerator does not increase in the same manner because convergence of NN-Descent does not have quadratic dependency on data-set size.

Finally, data set's intrinsic dimensionality highly influences NN-Descent as was already described in Section 5 and validated by the presented results. Synthetic data sets (i10000d100 and i100000d100) have very high intrinsic dimensionality because all the values are independently generated. The Table 4 shows that these data sets have the worst recall values.

6.3. *Hubness-aware variant*

We now show how hubness values can be used to help guide the choice of candidate points for inclusion in the NN list of a given data point. After implementing the simple hubness estimator described in Section 4, up-to-date hubness estimates become available at any given iteration of the algorithm execution. After each stage, these values become more accurate; they are reasonably precise even after the very first few stages.

Let us now describe in detail how this strategy works. In each iteration, we check if a given point has hubs in its NN list. Let h be a threshold on the number of hub points to be considered from the NN list of the given point. The idea is to replace h points of high hubness value in the current NN list with h new points chosen at random. The intuition behind this modification of NN-Descent is to diminish the impact of hubs on the update of NN lists. By adding one random point in place of each hub, we attempt to increase the probability of undiscovered neighbor points to associate themselves with the given point.

What remains to be clarified is the precise mechanism by which these h points are determined. Rather than simply choosing the h points with highest hubness values, in order to allow for the possibility of other improvements, we let the hubness value of a given point to determine the probability (between 0 and 1) of the point being included among the h points to be replaced. To generate a valid probability, we employed a linear transformation of the raw hubness values into the interval $[0, 1]$. For the purpose of the transformation, we introduce values $h_{\min}$ and $h_{\max}$ representing the maximum and minimum hubness values for which the linear transformation is applied; if the hubness value is greater than $h_{\max}$ or smaller than $h_{\min}$, the probabilities assigned would be 1 or 0, respectively. The probability that a point is selected for discard is shown in Equation 3, where x denotes the given data point and h_x is the current hubness value of point x.

$$\Pr[\mathit{discard}(x)] = \begin{cases} 0, & \text{if } h_x < h_{\min} \\ 1, & \text{if } h_x > h_{\max} \\ \frac{h_x - h_{\min}}{h_{\max} - h_{\min}}, & \text{otherwise.} \end{cases} \tag{3}$$

Experimental results for this approach are presented in Table 5, where the algorithm was executed for the choices $h_{\min} = 2K$ and $h_{\max} = 20K$. The experiments confirm the effectiveness of the choice of values for $h_{\min}$ and $h_{\max}$. For data sets with lower intrinsic dimensionality, such as MNIST, this method performs the same as NN-Descent. On the other hand, when there are hubs in data set, this method improves recall values. The improvements are usually followed by slightly higher scan-rate values. For some cases, scan rate value even stays the same, while recall increases. For example, this happened for data set i10000d100 and $K = 5$, where recall increased from 0.09 to 0.15 while the scan rate did not change.

To summarize, this method achieved higher recall values than NN-descent, at a small cost in terms of scan rate.

Table 5. Recall, scan rate and execution time values for the **Hubness-aware NN-Descent variant** with different K values (5, 10, 20), $\mathit{conv} = 0.01$, $\rho = 1$, $h_{min} = 2K$ and $h_{max} = 10K$.

	i10000d100			i100000d100			BCI5		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.15	0.46	0.79	0.02	0.12	0.42	0.61	0.97	0.99
Time	28.08	79.49	209.29	344.35	1218.57	3740.08	55.08	107.21	365.8
Scan rate	0.03	0.15	0.52	0	0.02	0.07	0.01	0.04	0.13

	Google-23			ISOLET			MNIST		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.71	0.93	0.98	0.85	0.98	1	0.75	0.96	0.99
Time	27.97	55.27	155.9	12.6	28.93	90.16	305.85	517.22	1434.68
Scan rate	0.05	0.16	0.47	0.05	0.13	0.38	0.01	0.02	0.06

6.4. *Oversized NN list variant*

The second NN-Descent variant considered involved the execution of NN-Descent with some larger choice of neighborhood size $K' > K$, followed by a truncation of the NN lists to the target size K. We analyzed this approach with $K' = 20$ and sampling rate $\rho = 0.05 \cdot K$. By combining these two parameters, the upper bound value of local joins in the Oversized NN list variant is the same as in the original NN-Descent for $\rho = 1$. Namely, even though the neighborhood size is increased to 20, the upper bound of local joins from each data set point is $\rho \cdot K' = 0.05 \cdot K \cdot 20 = K$.

The experimental results obtained for this setting are given in Table 6, from which we can observe a trade-off between scan rate and recall. This approach achieves really high increases of recall values, but at an evident cost of increased scan-rate values. However, the obtained scan rates are still much smaller than 1, which makes this approach much faster than naive K-NNG computation.

Table 6. Recall, scan rate and execution time values for the **Oversized NN list variant** with different K values (5, 10, 20), $conv = 0.01$, $\rho = 0.05 \cdot K$ and $K' = 20$.

	i10000d100			i100000d100			BCI5		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.43	0.52	0.73	0.13	0.18	0.36	0.89	0.99	0.99
Time	52.01	62.44	175.24	690.67	826.91	2730.93	156.73	207.4	359.85
Scan rate	0.18	0.27	0.48	0.02	0.03	0.06	0.04	0.06	0.13

	Google-23			ISOLET			MNIST		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.8	0.94	0.98	0.94	0.99	1	0.91	0.98	0.99
Time	49.66	75.9	147.82	29.46	47.18	104.48	547.98	722.92	1379.05
Scan rate	0.14	0.23	0.43	0.1	0.18	0.37	0.02	0.03	0.06

The higher recall values of this method are a direct consequence of increased neighborhood size. On the other hand, the reason for the increased scan rate is not that straightforward. As already said, our choices of K' and ρ values are such that the upper bound of local joins stays the same as in the NN-Descent algorithm. But, even though the number of local joins is in both cases bounded by K, the actual number of local joins in NN-Descent is smaller than in the Oversized NN list variant. This is a consequence of sampling. Sampling implies that in a single iteration no more than $\rho \cdot K$ neighbors are used in a local join. Those neighbors are then marked with a flag in order to avoid performing already completed local joins in future iterations. Therefore, for $\rho = 1$, the upper bound of local joins is exactly K, which means that all new neighbors are immediately participating in local joins and none of them are left for future iterations. As the algorithm converges, the number of new neighbors decreases, which leads to a smaller number of local joins. Contrary to that, when the ρ value is smaller than 1, the following iterations' local joins contain not only newly added neighbors, but also the old neighbors which did not get a chance to participate in local joins in previous iterations. For that reason, the overall number of local joins does not necessarily decrease as the algorithm starts to converge. This leads to the higher scan rate.

Additionally, the experimental results show no difference between NN-Descent and Oversized NN list variant for $K = 20$. In this case, the parameters of the Oversized NN list variant are $K' = 20$ and $\rho = 1$, which implies no increase of neighborhood and no sampling, making it equivalent to the original NN-Descent.

6.5. ***RW-Descent***

The main aim of this NN-Descent variant is to provide an easy way to fine-tune the number of comparisons in which a particular point will take part. In this way one could use different balancing strategies: for example, equal number of comparisons could be assigned to each data set point, or antihubs could be assigned more comparisons than other points.

The algorithm, Random Walk Descent (RW-Descent), employs a random walk strategy for determining candidates for improvement. Like NN-Descent, RW-Descent constructs the initial similarity graph by means of random selection. Thereafter, the algorithm iterates either a predetermined number of times, or until a convergence criterion is satisfied. In each iteration, each data point is allocated a number of comparisons according to some weighting strategy: instead of a top-down approach in which the pivot point determines which neighboring points will be mutually compared, we introduce a bottom-up approach in which each point determines the number of comparisons that it will initiate. After allocating a number of comparisons c to a given data point u, the points to be compared with are selected as the stopping points of c short random walks that start from u. The random walks are applied upon the current K-NNG's underlying simple graph; that is, each successive point in the random walk is chosen from the set of its predecessor's direct and reverse nearest neighbors. For the case where the random walks are all limited to length 2, the candidate stopping points would be neighbors of u's own neighbors, and thus RW-Descent would perform essentially as NN-Descent. If the termination criterion is chosen to be convergence, the parameter ρ from range $[0, 1)$ determines when the algorithm terminates. If in the current iteration less than $\rho \cdot c$ point's random walks resulted with an update of NN list, that means that the point converged. When all points converge, the algorithm terminates. For a pseudocode description of RW-Descent, see Algorithm 2.

For the purpose of evaluation, we used the simplest possible balancing approach, in which the same number of comparisons is assigned to each data point; we will refer to this number as c. The algorithm was run for $c = 8K$. The results are shown in Table 7.

With the described approach we obtained higher recalls, again at a cost of scan-rate increase. The advantage of RW-Descent is that the trade-off between recall and execution time can easily be managed by adjusting the total number of comparisons

Algorithm 2: Outline of RW-Descent algorithm

```
input  : data set $D$, distance function $dist$, neighborhood size $K$
output: $K$-NNG $G$
1  foreach data point $u \in D$ do
2      Initialize $G$ by randomly generating a tentative $K$-NN list for $u$ with an
         assigned distance of $+\infty$;
3  end
4  repeat
5      $G' \leftarrow Reverse(G)$
6      foreach data point $u \in D$ do
7          Determine the number of comparisons $c$ to be allocated to point $u$;
8          for $i = 1$ to $c$ do
9              $w \leftarrow u$;
10             for $j = 1$ to $2$ do
11                 $w \leftarrow Sample(G[w] \cup G'[w], 1)$;
12             end
13             Use $\langle v, dist(u, w)\rangle$ to update $w$'s $K$-NN list, and use
                 $\langle w, dist(u, w)\rangle$ to update $u$'s $K$-NN list;
14         end
15     end
16 until Termination criteria is satisfied;
17 Return $G$.
18 Function Reverse($G$):
19     $G' = G$
20     Change the direction of each edge in graph $G'$
21     return $G'$
22 Function Sample($set$, $count$):
23     return $count$ elements from the $set$
```

to be performed. Furthermore, RW-Descent has the potential for even better performance through the exploration of different balancing strategies. In our evaluation, we used the simplest possible balancing strategy, in which an equal number of comparisons was allocated to each of the data points. One interesting direction for future research would be the development of more finely balanced comparison allocation policies, in which higher numbers of comparisons would be allocated to those points that actually need it. Besides that, RW-Descent enables improvements of NN lists for subsets of K-NNG's nodes. This feature of the algorithm can be useful in cases when only a part of an already existing K-NNG changes.

Table 7. Recall, scan rate and execution time values for **RW-Descent** with different K values (5, 10, 20), $conv = 0.001$ and $c = 8K$.

	i10000d100			i100000d100			BCI5		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.11	0.41	0.75	0.02	0.12	0.38	0.76	0.98	0.99
Time	36.05	102.21	294.05	2712.42	4715.23	9028.18	275.58	290.9	492.67
Scan rate	0.09	0.25	0.56	0.01	0.03	0.09	0.02	0.04	0.07

	Google-23			ISOLET			MNIST		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.82	0.95	0.99	0.91	0.99	1	0.81	0.97	0.99
Time	37.25	69.25	155.99	23.75	42.02	95.93	1386.41	1546.38	2098.12
Scan rate	0.1	0.17	0.32	0.07	0.12	0.21	0.01	0.02	0.03

6.6. *Randomized NN-Descent variant*

The Randomized NN-Descent variant takes into account the phenomenon illustrated in Fig 4. Knowing that the position in the initial K-NNG has an important role in the computation of the final hit count value, our idea was to give each point additional chances to find its real neighborhood. We achieve this by conducting r additional random comparisons after each iteration for selected data points. The criterion for selection is defined in the following manner. At the start of the algorithm, all points need random comparisons. After each iteration of the main loop, if for some point none of the r comparisons resulted in an update of the point's NN list, that point is considered to already be in the correct neighborhood, and no more random comparisons are applied to it until the end of algorithm execution.

Table 8. Recall, scan rate and execution time values for **Randomized NN-Descent** with different K values (5, 10, 20), $conv = 0.01$, $\rho = 1$, and $r = n/500$.

	i10000d100			i100000d100			BCI5		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.21	0.43	0.75	0.15	0.27	0.47	0.82	0.98	0.99
Time	23.25	55.53	183.03	2086.7	3539.59	7021.89	158.23	243.9	549.41
Scan rate	0.14	0.22	0.54	0.13	0.16	0.2	0.09	0.11	0.18

	Google-23			ISOLET			MNIST		
	K=5	K=10	K=20	K=5	K=10	K=20	K=5	K=10	K=20
Recall	0.78	0.93	0.98	0.88	0.98	1	0.82	0.97	0.99
Time	35.82	66.18	164.55	25.63	40.45	100.98	1371.13	1603.69	2553.8
Scan rate	0.11	0.2	0.48	0.11	0.18	0.41	0.09	0.09	0.12

The results of the experiments are presented in Table 8. The r parameter in the experiments is directly proportional to the data-set size. This is because the size of the data set determines the easiness of finding the right neighborhood – if the data set is larger, the probability of finding right neighborhood decreases, thus more random comparisons need to be performed, and vice versa. The experiments showed that this approach also improves recall. However, just like the previous approaches, the Randomized NN-Descent variant also introduces higher scan rates.

6.7. *Discussion*

The overall comparison of all methods is shown in Fig. 5. As can be seen, the Oversized NN lists variant achieves highest recall values, which are most evident for lower K values. At the same time this method has in most of the cases the highest scan rate values. Nevertheless, if high recall is needed, this method should be considered.

The power of RW-Descent lies in its flexibility. This method could be used when only a subset of K-NNG's nodes changes. Instead of calculating K-NNG approximation from the scratch, RW-Descent can be used only on changed nodes. The obtained results are likely to be accurate, since this approach behaves even better than NN-Descent, as can be seen in Fig. 5.

The rest of the approaches achieve similar improvements over original NN-Descent. The improvements are most evident for smaller K values, while for $K = 20$ NN-Descent already performs well enough.

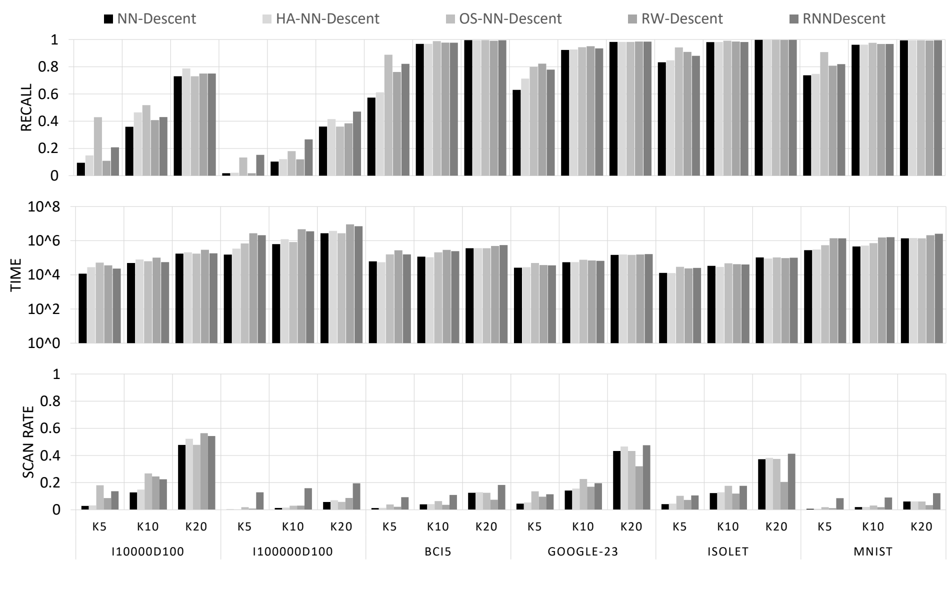

Fig. 5. Recall, execution time and scan rate values of the following methods: NN-Descent, Hubness aware NN-Descent variant, Oversized NN lists variant, RW-Descent and Randomized NN-Descent variant.

7. Conclusion

Poor performance of the NN-Descent algorithm on high-dimensional data has been observed in the past, but until now has lacked a satisfactory explanation. In this paper, we provided an experimental analysis that reveals a connection between performance and the well-studied hubness characteristics of data sets, in that the original formulation of the NN-Descent algorithm does not perform well when the data contains many hubs. In order to address this shortcoming, we introduced four different variants of the original algorithm. Our experimental results show that the new NN-Descent variants achieve better recall at the expense of increased scan rate.

In the future, a more detailed evaluation of the described phenomenon and proposed methods will be conducted. Future work could target the fine tuning of all introduced NN-Descent variants. In particular, with the oversized NN list variant, the main research target should be the improvement of sampling strategies. Conceivably, better performance may be achieved if the points chosen for NN list improvement were selected according to a carefully managed non-uniform strategy. RW-Descent might be significantly improved with a better balancing strategy that would most probably rely on hubness estimates of data set points. One additional research direction is theoretical analysis of RW-Descent that could potentially give more insight into the performance characteristics of NN-Descent.

Acknowledgments

M. E. Houle gratefully acknowledges the financial support of JSPS Kakenhi Kiban (B) Research Grant 18H03296. B. Bratić, V. Kurbalija and M. Radovanović thank the Serbian Ministry of Education, Science and Technological Development for support through project no. OI174023, "Intelligent Techniques and their Integration into Wide-Spectrum Decision Support." V. Oria acknowledges the financial support of NSF Research Grants DGE 1565478 and ICER 1639683.

References

1. B. V. Dasarathy, Data mining tasks and methods: Classification: Nearest-neighbor approaches, in *Handbook of data mining and knowledge discovery* Oxford University Press, Inc.2002, pp. 288–298.
2. T. Cover and P. Hart, Nearest neighbor pattern classification, *IEEE transactions on information theory* **13**(1) (1967) 21–27.
3. K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi and H. Zhang, Fast approximate nearest-neighbor search with k-nearest neighbor graph, in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence* **22**(1)2011, p. 1312.
4. M. Brito, E. Chavez, A. Quiroz and J. Yukich, Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection, *Statistics & Probability Letters* **35**(1) (1997) 33–42.
5. V. Hautamaki, I. Karkkainen and P. Franti, Outlier detection using k-nearest neighbour graph, in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* **3**, IEEE2004, pp. 430–433.

6. M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander, LOF: identifying density-based local outliers, in *ACM sigmod record* **29**(2), ACM2000, pp. 93–104.
7. M. Belkin and P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, *Neural computation* **15**(6) (2003) 1373–1396.
8. S. T. Roweis and L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, *science* **290**(5500) (2000) 2323–2326.
9. L. K. Saul and S. T. Roweis, Think globally, fit locally: unsupervised learning of low dimensional manifolds, *Journal of machine learning research* **4**(Jun) (2003) 119–155.
10. H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation* (MIT press, 2005).
11. J. Sankaranarayanan, H. Samet and A. Varshney, A fast all nearest neighbor algorithm for applications involving large point-clouds, *Computers & Graphics* **31**(2) (2007) 157–174.
12. K.-I. Lin, H. V. Jagadish and C. Faloutsos, The tv-tree: An index structure for high-dimensional data, *The VLDB Journal* **3**(4) (1994) 517–542.
13. B. Bratić, M. E. Houle, V. Kurbalija, V. Oria and M. Radovanović, Nn-descent on high-dimensional data, in *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics* ACM2018, p. 20.
14. M. Radovanović, A. Nanopoulos and M. Ivanović, Hubs in space: Popular nearest neighbors in high-dimensional data, *Journal of Machine Learning Research* **11**(Sep) (2010) 2487–2531.
15. J. Chen, H.-r. Fang and Y. Saad, Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection, *Journal of Machine Learning Research* **10**(Sep) (2009) 1989–2012.
16. C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators* (United States Governm. Press Office Los Angeles, CA, 1950).
17. M.-H. Jang, S.-W. Kim, W.-K. Loh and J.-I. Won, On approximate k-nearest neighbor searches based on the earth mover's distance for efficient content-based multimedia information retrieval, *Computer Science and Information Systems* **16**(2) (2019) 615–638.
18. R. Paredes, E. Chávez, K. Figueroa and G. Navarro, Practical construction of k-nearest neighbor graphs in metric spaces, in *International Workshop on Experimental and Efficient Algorithms* Springer2006, pp. 85–97.
19. M. Connor and P. Kumar, Fast construction of k-nearest neighbor graphs for point clouds, *IEEE transactions on visualization and computer graphics* **16**(4) (2010) 599–608.
20. Y.-M. Zhang, K. Huang, G. Geng and C.-L. Liu, Fast knn graph construction with locality sensitive hashing, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* Springer2013, pp. 660–674.
21. A. Gionis, P. Indyk, R. Motwani *et al.*, Similarity search in high dimensions via hashing, in *Vldb* **99**(6)1999, pp. 518–529.
22. W. Dong, M. Charikar and K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in *Proceedings of the 20th international conference on World wide web* ACM2011, pp. 577–586.
23. Y. Park, S. Park, S.-g. Lee and W. Jung, Scalable k-nearest neighbor graph construction based on greedy filtering, in *Proceedings of the 22nd International Conference on World Wide Web* ACM2013, pp. 227–228.
24. M. E. Houle, X. Ma, V. Oria and J. Sun, Improving the quality of K-NN graphs for image databases through vector sparsification, in *Proceedings of International Con-*

ference on Multimedia Retrieval ACM2014, p. 89.
25. T. Debatty, P. Michiardi, O. Thonnard and W. Mees, Building k-NN graphs from large text data, in *Big Data (Big Data), 2014 IEEE International Conference on* IEEE2014, pp. 573–578.
26. M. Radovanović, A. Nanopoulos and M. Ivanović, Time-series classification in many intrinsic dimensions, in *Proceedings of the 2010 SIAM International Conference on Data Mining* SIAM2010, pp. 677–688.
27. M. Radovanović, A. Nanopoulos and M. Ivanović, On the existence of obstinate results in vector space models, in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval* ACM2010, pp. 186–193.
28. N. Tomašev, M. Radovanović, D. Mladenić and M. Ivanović, The role of hubness in clustering high-dimensional data, in *Pacific-Asia Conference on Knowledge Discovery and Data Mining* Springer2011, pp. 183–195.
29. M. Radovanović, A. Nanopoulos and M. Ivanović, Reverse nearest neighbors in unsupervised distance-based outlier detection, *IEEE transactions on knowledge and data engineering* **27**(5) (2015) 1369–1382.
30. J. R. Millan, On the need for on-line learning in brain-computer interfaces, in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)* **4**, IEEE2004, pp. 2877–2882.
31. M. E. Houle, V. Oria, S. Satoh and J. Sun, Annotation propagation in image databases using similarity graphs, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* **10**(1) (2013) p. 7.
32. M. Fanty and R. Cole, Spoken letter recognition, in *Advances in Neural Information Processing Systems*1991, pp. 220–226.
33. D. Dua and C. Graff, UCI machine learning repository (2017).
34. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11) (1998) 2278–2324.