



UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
DEPARTMENT OF MATHEMATICS AND
INFORMATICS



Brankica Bratić

APPROXIMATION ALGORITHMS FOR k -NN GRAPH CONSTRUCTION

(APROKSIMATIVNI ALGORITMI ZA GENERISANJE k -NN GRAFA)

-Ph.D. thesis-

Advisor: Vladimir Kurbalija

Novi Sad, 2020

Dedicated to my parents, Mirjana and Zoran Bratić...
Posvećeno mojim roditeljima, Mirjani i Zoranu Bratić...

Contents

Abstract	vii
Izvod (in Serbian)	ix
Preface	xi
1 Introduction	1
1.1 Problems and applications	1
1.2 Contributions	3
2 Background	5
2.1 Preliminaries	5
2.2 k -NN graph	7
2.3 Hubness	9
3 Existing algorithms for k-NN graph construction	13
3.1 Optimization of k -NN graph construction	13
3.1.1 Specialized algorithms	14
3.1.2 Parallelized algorithms	17
3.1.3 Approximation algorithms	19
3.2 <i>NN-Descent</i>	22
3.2.1 <i>NN-Descent</i> as an inspiration for other algorithms	25
3.3 Temporal k nearest neighbor graphs	27
4 <i>NN-Descent</i> on high dimensional data	29
4.1 Influence of hubness on <i>NN-Descent</i>	29
4.2 Influence of initial random graph	33
5 Proposed methods for improving <i>NN-Descent</i>	37
5.1 Hubness approximation using <i>NN-Descent</i>	38

5.2	Hubness-aware variant	40
5.3	Oversized NN list variant	43
5.4	Random walk descent variant	46
5.5	Nearest walk descent variant	48
5.6	Randomized <i>NN-Descent</i> variant	53
5.7	Methods evaluation	55
5.7.1	Experimental setup	55
5.7.2	Results and discussion	56
6	<i>NN-Descent</i>-based approximation algorithms for k-NN graph updates	67
6.1	Naive k -NN graph update algorithm	67
6.2	Online variants of random walk descent and nearest walk descent	69
6.3	Methods evaluation	73
6.3.1	Datasets	74
6.3.2	Simulation design	74
6.3.3	Experimental setup	79
6.3.4	Results and discussion	82
7	Conclusions	93
7.1	Directions for future work	94
A	Console application for algorithms related to k-NN graphs	97
B	Detailed results of the experiments related to k-NN graph update algorithms	109
	Prošireni izvod (in Serbian)	113
	Bibliography	129
	List of Figures	135
	List of Tables	137
	Abbreviations	139
	Kratka biografija (in Serbian)	141

Abstract

Nearest neighbor graphs are modeling proximity relationships between objects. They are widely used in many areas, primarily in machine learning, but also in information retrieval, biology, computer graphics, geographic information systems, etc. The focus of this thesis are k -nearest neighbor graphs (k -NNG), a special class of nearest neighbor graphs. Each node of k -NNG is connected with directed edges to its k nearest neighbors.

A brute-force method for constructing k -NNG entails $O(n^2)$ distance calculations. This thesis addresses the problem of more efficient k -NNG construction, achieved by approximation algorithms. The main challenge of an approximation algorithm for k -NNG construction is to decrease the number of distance calculations, while maximizing the approximation's accuracy.

NN-Descent is one such approximation algorithm for k -NNG construction, which reports excellent results in many cases. However, it does not perform well on high-dimensional data. The first part of this thesis summarizes the problem, and gives explanations for such a behavior.

The second part introduces five new *NN-Descent* variants that aim to improve *NN-Descent* on high-dimensional data. The performance of the proposed algorithms is evaluated with an experimental analysis.

Finally, the third part of this thesis is dedicated to k -NNG update algorithms. Namely, in real world scenarios data often change over time. If data change after k -NNG construction, the graph needs to be updated accordingly. Therefore, in this part of the thesis, two approximation algorithms for k -NNG updates are proposed. They are validated with extensive experiments on time series data.

Graf najbližih suseda modeluje veze između objekata koji su međusobno bliski. Ovi grafovi se koriste u mnogim disciplinama, pre svega u mašinskom učenju, a potom i u pretraživanju informacija, biologiji, računarskoj grafici, geografskim informacionim sistemima, itd. Fokus ove teze je graf k najbližih suseda (k -NN graf), koji predstavlja posebnu klasu grafova najbližih suseda. Svaki čvor k -NN grafa je povezan usmerenim granama sa njegovih k najbližih suseda.

Metod grube sile za generisanje k -NN grafova podrazumeva $O(n^2)$ računanja razdaljina između dve tačke. Ova teza se bavi problemom efikasnijeg generisanja k -NN grafova, korišćenjem aproksimativnih algoritama. Glavni cilj aproksimativnih algoritama za generisanje k -NN grafova jeste smanjivanje ukupnog broja računanja razdaljina između dve tačke, uz održavanje visoke tačnosti krajnje aproksimacije.

NN-Descent je jedan takav aproksimativni algoritam za generisanje k -NN grafova. Iako se pokazao kao veoma dobar u većini slučajeva, ovaj algoritam ne daje dobre rezultate nad visokodimenzionalnim podacima. Unutar prvog dela teze, detaljno je opisana suština problema i objašnjeni su razlozi za njegovo nastajanje.

U drugom delu predstavljeno je pet različitih modifikacija *NN-Descent* algoritma, koje za cilj imaju njegovo poboljšavanje pri radu nad visokodimenzionalnim podacima. Evaluacija ovih algoritama je data kroz eksperimentalnu analizu.

Treći deo teze se bavi algoritmima za ažuriranje k -NN grafova. Naime, podaci se vrlo često menjaju vremenom. Ukoliko se izmene podaci nad kojima je prethodno generisan k -NN graf, potrebno je graf ažurirati u skladu sa izmenama. U okviru ovog dela teze predložena su dva aproksimativna algoritma za ažuriranje k -NN grafova. Ovi algoritmi su evaluirani opširnim eksperimentima nad vremenskim serijama.

Preface

There are many problems which rely on proximity relationships between objects. Many such problems use k -nearest neighbor graph (k -NNG) as an underlying data structure [2, 3, 5, 10, 14, 24, 25, 32, 35, 44, 45, 60]. One defines k -NNG as a directed graph whose vertices are objects themselves. Each vertex is connected to its k nearest neighbors, with respect to a predefined distance function. The simplest way to construct k -NNG is to calculate all objects' pairwise distances, and then to choose k nearest neighbors for each object. This approach entails $\binom{n}{2}$ distance calculations, leading to a quadratic time complexity.

Numerous algorithms for k -NNG construction have been developed in order to decrease the time complexity of the presented brute-force approach. There are three main classes of such algorithms. The first class represents algorithms that introduce certain restrictions with convenient properties, which then allow further optimizations [1, 2, 19, 39, 41, 50, 54, 55]. The second way to optimize k -NNG construction is to parallelize it. Hence, the second class refers to the parallelized algorithms for k -NNG construction [11, 12, 15, 34, 45]. Finally, the third class represents approximation algorithms, which should minimize the computational cost, while maximizing k -NNG approximation correctness [13, 20, 30, 42, 43, 51, 57, 59].

Approximation algorithms for k -NNG construction are the focus of this thesis. *NN-Descent* is one such algorithm that is highly efficient [20]. It is based on the assumption that “a point's neighbor's neighbor is also likely to be the point's neighbor”. The algorithm starts with creation of a random k -NNG, which is then iteratively improved. In each iteration, the algorithm calculates distances between points that share a neighbor, and uses these distances to update the k -NNG approximation. However, *NN-Descent* has a major drawback that k -NNG approximations are accurate only on data of low intrinsic dimensionality.

One research direction of this thesis was to investigate why *NN-Descent* performs poorly on high-dimensional data. We showed that the reason for that behavior is related to a phenomenon called *hubness* [8, 9, 48]. Hubness is an aspect

of the “curse of dimensionality”, which implies existence of k -NNG nodes that have very high in-degrees. High in-degree nodes decrease the probability that two points sharing a neighbor are as well neighbors. Given that exactly this probability is the essential part of *NN-Descent*’s assumption, it is evident that hubness negatively influences *NN-Descent*. We additionally conducted an experimental analysis that confirms such relation between hubness and *NN-Descent* performance.

In order to mitigate the problem, we proposed five different modifications of *NN-Descent* [8, 9]. The first modification is based on the observation that high in-degree nodes have well approximated neighbors, while low in-degree nodes are ones that need more attention. Therefore, the algorithm allocates more resources for approximating low in-degree nodes’ neighbors. The second modification makes use of the fact that *NN-Descent*’s accuracy increases with k . The idea is then to construct k -NNG for a larger k , and reduce it to the wanted k afterwards. The third and fourth modifications provide an easy way to fine-tune the number of comparisons for each data point, which could be used to assign more comparisons to the points that need it. Finally, the fifth approach is based on the fact that the position of a node in the initial random graph influences correctness of its approximated neighbors. For that reason, this approach conducts additional random comparisons for the points that are in wrong initial neighborhoods. All the five proposed modifications of *NN-Descent* are validated on two synthetic and four real high-dimensional datasets.

The second research direction is related to the problem of k -NNG updates. Namely, data very often have a tendency to change over time. Consequently, there is a need for an algorithm that would efficiently update k -NNG after its underlying data change. The simplest way to update k -NNG is to construct a new one from scratch, by using any algorithm for k -NNG construction. However, this approach does not make use of the previous k -NNG—data often change only partially, so the graph could be updated partially as well. One way to perform a partial update of k -NNG is to apply the naive brute-force approach that calculates only necessary distances, updating only the changed part of the graph. However, the naive approach might not be fast enough, especially when many nodes have changed.

In this thesis we propose two *NN-Descent*-based approximation algorithms for k -NNG updates. Both algorithms are performing short walks starting from the nodes affected by the data change. The starting and the ending node of each walk are compared, and the graph is updated accordingly. The proposed algorithms were validated by extensive experiments on time series data. In the experiments,

Preface

we simulated a real world scenario, where time series are getting new values over time.

The thesis is organized as follows. In Chapter 1 the introduction to the problem is given. Chapter 2 introduces the basic definitions and concepts that are used in the thesis. In Chapter 3 an overview of the existing algorithms for k -NNG construction is given. The reasons for *NN-Descent*'s misbehavior on high-dimensional data are elaborated in Chapter 4. In Chapter 5 we introduce the five *NN-Descent* modifications that improve the algorithm on high-dimensional data. Chapter 6 is dedicated to the k -NNG update algorithms—in it we introduce the two new approximation algorithms for k -NNG updates. Finally, the conclusions and future research directions are discussed in Chapter 7.

Acknowledgments

First and foremost, I would like to thank my advisor, Vladimir Kurbalija, for supporting me in this journey, believing in me, and valuing my work, which all helped me to move forward. I would also like to thank Mirjana Ivanović, Miloš Radovanović, Michael Houle and Zoltan Geler, members of the Defense Board, for helping me to improve this thesis. They did not only help me with the thesis, but also in many other ways during my studies. Therefore, I would like to express additional gratitude to Mirjana Ivanović for providing me with many scientific trips and opportunities, and also for giving me many valuable advices. I am also grateful to Miloš Radovanović for introducing me to the topic of this thesis and for many fruitful discussions regarding it. Furthermore, I am very thankful to Michael Houle for inviting me to Tokyo, Japan, and for providing me with two very important months of my professional career. The trip to Japan was an invaluable experience for me; the discussions and experiments we conducted there shaped this thesis, and I am very grateful for that. Finally I would like to thank very much to Zoltan Geler for all the discussions we had regarding this thesis.

Many other people helped me along the way. I would like to thank Srđan Škrbić, Žarko Bodroški and Bane Ivošev for giving me resources on our Axiom cluster, making my extensive experiments possible. Many thanks go to all of my colleagues, especially to Doni Pracner, Nataša Sukur, Jovana Ivković and Ivan Pribela, for numerous chats, jokes, and for the pleasant work environment. I am also very happy that, during my bachelor and master studies, I had such a great roommate, Nikolina, who made my university days so nice and joyful; but not only that, together we made our first steps as programmers. I would like to thank

my friends Anna, Bojan and Kristina for visiting many escape rooms with me, for loving board games as much as I do, and for much more; moreover, I would like to express additional gratitude to Bojan for proofreading some mathematical parts of this thesis and for helping me with English-language dilemmas in some parts of the thesis. Furthermore, I would like to thank my cousin Filip, for many nice walks, dinners, a few unforgettable trips, and long enjoyable conversations. There are very few people I can have such conversations with, and he is one of them. Special thanks go to my sister Jelena. Čka, thank you for being who you are, and for approving all of my (sometimes a bit silly) ideas. You have always been a great support to me, and the first person I come to in huge life moments. I am more than grateful for having you in my life. Finally, I would like to thank my Vlada for all the nice moments, advices, talks, trips, for always being there for me and for making everything much nicer and easier.

* * *

Among all the others, my greatest gratitude goes to my dear parents, Mirjana and Zoran Bratić, to whom I dedicate this thesis. Everything I achieved, I achieved thanks to them. They supported my sister and myself in every possible way, giving up many things for our benefit. Thank you for all you have done...

Od svih ljudi, najzahvalnija sam svojim dragim roditeljima, Mirjani i Zoranu Bratić, kojima ujedno i posvećujem ovu tezu. Sve što sam postigla, postigla sam zahvaljujući njima. Uvek su nas podržavali na svaki mogući način, odričući se raznih stvari zarad nas. Hvala vam na svemu što ste nam pružili...

Novi Sad, September 2020

Brankica Bratić

Chapter 1

Introduction

Fast construction of k -NNGs has been a challenge for decades. The motivation for developing these algorithms lies in numerous applications of k -NNGs. In Section 1.1 we will present some of the k -NNG’s applications, providing an insight into the importance of k -NNGs.

Even though many solutions exist for the problem of k -NNG construction, some challenges that arose during the recent years are still unsolved. Namely, the scale of the data has significantly increased—dimensionality of datasets, as well as the number of instances, are nowadays drastically larger. Moreover, data is very often dynamic, having a tendency to change over time. Therefore, the existing algorithms must be adapted in order to meet the current needs. In this thesis we considered the problem of fast, approximate k -NNG construction on high-dimensional data and the problem of fast, approximate k -NNG update. In Section 1.2 we give an overview of the thesis’ contributions.

1.1 Problems and applications

As already noted, k -NNGs have a wide variety of applications. In this section we will present some of them, providing a motivation for the work on k -NNGs.

A problem called *nearest neighbors (NN) search* [33] is one of the most direct applications of k -NNG. Given a value k , an arbitrary set X upon which a distance function $dist$ is defined, a set $S \subset X$ and a query object $q \in X \setminus S$, the goal of NN search problem is to find a set $S' \subseteq S$ such that $|S'| = k$ and $s \in S \setminus S' \wedge s' \in S' \implies dist(s', q) \leq dist(s, q)$, that is, among all the objects in S , the set S' contains k of them that are nearest to q . Many algorithms for this problem were proposed, and some of them are taking the advantage of k -NNG. For example, Hajebi et al. [24] proposed an approximate algorithm called *Graph Nearest Neighbor Search (GNNS)* for $k = 1$. Their algorithm builds k -NNG on set S in the offline phase, and then,

1.1. PROBLEMS AND APPLICATIONS

for a query object q , it performs hill-climbing starting from a randomly chosen node of the prebuilt graph and ending in the node that is highly probable to be q 's nearest neighbor.

Given a set of object labels (categories), *classification* is a machine learning problem of assigning the suitable label to a query object [58]. The label assignment is performed according to the knowledge inferred from the already labeled objects. A simple, yet effective, classification algorithm called *k-NN classifier* is built on top of arbitrary NN search algorithm. Let S' be a set of labeled objects returned by an NN search algorithm for a query object q . Object q is then labeled with the predominant label in S' . If the NN search algorithm uses *k-NNG*, *k-NN classifier* is indirectly using it as well.

Label propagation [56, 61] is another class of machine learning algorithms in which *k-NNGs* can be applied. The goal of such algorithms is to assign labels to previously unlabeled objects—a label propagation algorithm starts with a small set of objects that have labels, and then it propagates them to the unlabeled objects. Unlike classification algorithms which assign a label to a single object, label propagation operates on a batch of objects. One simple iterative label propagation algorithm briefly presented in [5] uses *k-NNG* in the following manner. Under the assumption that there are exactly two labels, the algorithm first transforms the labels into the numbers -1 and 1 (if there are more than two labels, one-hot encoding technique is employed instead), and then initializes the unlabeled objects with label 0 . Afterwards, the algorithm iteratively propagates labels, until the convergence criteria is met. In a single iteration, the algorithm propagates labels to all the nodes by calculating weighted average of its neighboring nodes' labels.

An outlier is an object that differs too much from other objects. Hawkins defined [26] an outlier as follows:

“An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.”

In some cases, outliers are considered as noise objects that erroneously end up in the data, and should therefore be removed in order to boost performance of a machine learning algorithm. In other cases, outliers represent valid but anomalous data, which might be even more interesting than the regular data (for example, network attacks produce anomalous behavior that should be detected). Some algorithms for outlier detection use *k-NNG*. Hautamaki et al. [25] introduced two algorithms for outlier detection, both of them using *k-NNG*. In the first algorithm

a k -NNG's node is declared as an outlier if the size of its R -NN list is at most T , that is, if its in-degree is at most T . In the second algorithm, the graph's nodes are sorted by the average distances in their NN lists. The nodes with large average distances are declared as outliers.

Clustering [58] is the problem of making a partition of an input set S , each set of the partition being called a cluster, such that pairwise distances of objects belonging to the same cluster are minimized, while distances of objects belonging to different clusters are maximized. In some clustering algorithms k -NNG is used. Brito et al. [10] made one such algorithm, where they use the notion of mutual k -NNG. Mutual k -NNG is an undirected graph where an edge exists between two nodes if they both belong to each other's NN lists. Each connected component of this graph is then considered a cluster either if it contains more than one node, or if it contains an outlier. Note that, although this might look like a graph clustering problem, it is actually a regular object clustering problem that only uses k -NNG to achieve its goal.

All the aforementioned applications of k -NNG are related to machine learning. However, k -NNG is used in other disciplines as well. In Internet applications it is often important for nodes to find other nodes that are close to them with respect to a distance based on latency or bandwidth [32]. In information retrieval problems k -NNG is often used as a building block [2, 60]. In sampling-based robot motion planning algorithms, k -NNG can be used to capture the connectivity of the solution space and also to find paths that allow robots to move from one point in the environment to another while satisfying certain constraints (such as avoiding collision with obstacles) [45]. Moreover, k -NNG is used in biology, for example in the study of protein folding [3]. In point-based graphics, k -NNG forms a basic building block in solving many important problems [14, 44]. Geographic information systems use k -NNG very often, usually for finding the set of nearest geographic objects to some query object [35].

1.2 Contributions

The main focus of this thesis are approximation algorithms for k -NNG construction. Most of these algorithms are based either 1) on divide-and-conquer technique [13, 30, 57, 59], or 2) on the assumption that two points that share a neighbor are also likely to be neighbors [20, 51]. Algorithms that are based on divide-and-conquer technique, create small, not necessarily disjoint, subsets, so that similar points are grouped in same subsets. After that, k -NNGs are built on

small subsets, and then merged into the resulting graph. On the other side, algorithms that are based on the mentioned assumption usually iteratively improve k -NNG by comparing points that share a neighbor.

In this thesis we are particularly interested in the algorithm called *NN-Descent* [20]. *NN-Descent* belongs to the second aforementioned class of approximation algorithms. It is very fast and accurate in most of the cases, however, it produces highly inaccurate approximations on high-dimensional data. We showed that the main cause for the bad performance of this algorithm is the hubness phenomenon. Hubs, which are points that appear in neighborhoods of many other points, are being compared much more with other points, while anti-hubs, which are points that do not appear in many neighborhoods, do not get enough comparisons. As a result, hubs end up with correct neighborhoods, while the anti-hubs end up with incorrect neighborhoods. In this thesis, we propose five variants of *NN-Descent*, whose aim is to alleviate the negative influence of hubness.

Many algorithms deal with k -NNG construction, but there are very few algorithms that are designed for updates of an existing k -NNG after its underlying data has changed. In the second part of the thesis, we introduce two approximation algorithms for k -NNG update. Both algorithms are inspired by *NN-Descent*. Performance of the new algorithms is analyzed by extensive experiments on time series data. More precisely, these experiments simulate real world scenarios in which time series are periodically getting new values. Each time some of time series' change, the k -NNG must be updated. The k -NNG updates in these simulations are performed using the two newly proposed algorithms, but also using brute force approach and *NN-Descent*, and it turned out that the new approaches outperformed the rest.

Chapter 2

Background

In this chapter we introduce our notation, basic concepts and definitions related to this thesis.

2.1 Preliminaries

This section covers some general, well-known concepts that we use throughout the thesis.

Graphs are mathematical structures used to model pairwise relations between objects. A graph G is an ordered pair $\langle V, E \rangle$, where V is a set of vertices (nodes) and $E \subseteq \{\{x, y\} | x, y \in V\}$ is a set of edges. Graphs can be represented graphically by indicating each vertex by a point, and each edge by a line joining the points that correspond to the vertices defining the edge. The notion of graph can be further extended to directed graph. A **directed graph** is as well an ordered pair $\langle V, E \rangle$, but with the distinction that the set of edges E is now a set of ordered pairs. More precisely, the edge set in a directed graph is defined as follows: $E \subseteq \{\langle x, y \rangle | x, y \in V\}$. Directed graphs are graphically represented in the same manner as undirected graphs, the only difference being that the edges are now represented with arrows starting from the first vertex of the edge, and ending in the second one.

A **time series** is a series of values linked to a certain moment in time. In general, a time series T is represented as a list of ordered pairs $\langle x_i, t_i \rangle$, where x_i corresponds to a value linked to a time t_i . The values of a time series are usually measurements of some phenomenon taken in different moments in time. Very often, the values $t_i - t_{i-1}$ are the same for any two successive elements in the time series. In that case, the time component can be omitted, meaning that the time series can simply be represented as a list of values given in the chronological order. In this thesis we will use only the representation with omitted time component.

Given a set X , a **distance function** or a **metric** is a function $dist : X \times X \rightarrow \mathbb{R}$ which satisfies following conditions for any $x, y, z \in X$:

1. non-negativity: $dist(x, y) \geq 0$,
2. identity: $dist(x, y) = 0 \iff x = y$,
3. symmetry: $dist(x, y) = dist(y, x)$,
4. triangle inequality: $dist(x, y) \leq dist(x, z) + dist(z, y)$.

A **metric space** is a set together with a metric on that set. One well known class of distance functions are **Minkowski distances**. A Minkowski distance of order p , denoted by L_p , between points $A = (a_1, a_2, \dots, a_d)$ and $B = (b_1, b_2, \dots, b_d) \in \mathbb{R}^d$ is defined by $dist(A, B) = \left(\sum_{i=1}^d |a_i - b_i|^p \right)^{\frac{1}{p}}$. The mostly used Minkowski distances are L_1 distance, also known as **Manhattan distance**, and L_2 distance, also known as **Euclidean distance**.

In Chapter 3 we also mention **cosine distance**, which is complement of cosine similarity. The cosine similarity for two non-zero vectors is defined as the cosine of the angle between them. The cosine distance is then obtained by subtracting the cosine similarity from the value 1. It is important to note that cosine distance is not a metric as it does not satisfy two necessary conditions: the symmetry condition and the triangle inequality condition.

In Chapter 6 **dynamic time warping (DTW)** is used as a distance measure for time series. DTW is a dissimilarity measure that allows non-linear alignment of two time series. That means that DTW introduces comparisons between time series' points that are not identically positioned on the time axis. Let us denote two time series by $A = (a_1, a_2, \dots, a_{d_A}) \in \mathbb{R}^{d_A}$ and $B = (b_1, b_2, \dots, b_{d_B}) \in \mathbb{R}^{d_B}$. DTW calculation starts by creation of a $d_A \times d_B$ matrix D , whose element $D_{i,j}$ is equal to $dist(a_i, b_j)$, where $dist$ is an arbitrary distance function (usually the Euclidean distance). In the next step, a warping path through the matrix D is found. The warping path is simply a list that contains elements of D , which additionally must satisfy the following conditions: 1) its first element is $D_{1,1}$, 2) its last element is D_{d_A, d_B} , 3) if $D_{i,j}$ and $D_{i',j'}$ are elements that are adjacent in the warping path, $D_{i,j}$ appearing first, then

$$(0 \leq i' - i \leq 1) \wedge (0 \leq j' - j \leq 1) \wedge \neg(i = i' \wedge j = j')$$

must hold, 4) it minimizes the sum of its elements. The warping path is usually calculated by using dynamic programming, which leads to the time complexity of $O(d_A \cdot d_B)$. Once the warping path is calculated, the final DTW value is easily

obtained by summing the values of the elements from the warping path. Like the cosine distance, DTW is also not a metric because it does not satisfy the triangle inequality condition.

The **curse of dimensionality** is a term introduced by Bellman [4] that refers to a phenomenon arising in high-dimensional data. This phenomenon is causing bad performance of many algorithms. The problem usually arises as a consequence of the following—when the dimensionality increases, the volume of the space even more rapidly increases, making the data in it extremely sparse. One way to deal with the curse of dimensionality is to create a simpler representation of data, i.e., to reduce data dimensionality. For that purpose many dimensionality reduction techniques were introduced.

Additionally, dimensionality reduction is also used to remove unnecessary data, the goal being to lower data dimensionality d to some $d' < d$ with minimal information loss. One of the challenges here is to determine the optimal value of d' . The optimality of the value d' can be defined in different ways, one of which is the following: d' is optimal if it is minimal value such that no information is lost after dimensionality reduction. In that case, d' is called the **intrinsic dimensionality (ID)** of the data. In machine learning the intrinsic dimensionality for a dataset can be seen as the minimal number of attributes needed to represent the data.

2.2 k -NN graph

A k -nearest neighbor graph (k -NNG) G is a directed graph whose vertices represent objects from an input set S upon which some distance function is defined. Throughout this thesis, we will assume that all pairwise distances within the input set S are unique. A vertex $s \in S$ is unidirectionally connected to a vertex $s' \in S$ only if the vertex s' is one of the k vertices that are nearest to s with respect to the distance function. In that case, vertex s' is said to be a *neighbor* of vertex s , while vertex s is said to be a *reverse neighbor* of vertex s' . The list of all neighbors of some vertex s is also called the *nearest neighbor (NN) list*, and we will denote it by $NN_G(s)$; the list of all reverse neighbors of vertex s is called the *reverse nearest neighbor (R-NN) list*, and we will denote it by $RNN_G(s)$. The elements of NN list of a point s will be represented as ordered pairs $\langle s', d \rangle$, where s' is a neighbor of s , and $d = \text{dist}(s, s')$ is the distance between s and s' .

Figure 2.1 shows an example of a simple k -NNG for $k = 2$, whose vertices are five 2-dimensional points in the Euclidean plane. As can be seen, each point has exactly two outgoing edges (which is determined by the value k), while the number

of incoming edges varies. The figure also depicts the direct and reverse neighborhoods of one vertex—this vertex is colored blue, while its nearest neighbors (2.1a) and reverse neighbors (2.1b) are colored red.

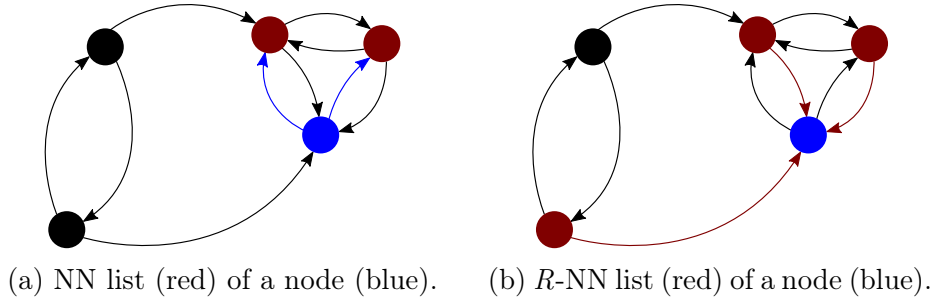


Figure 2.1: An example of k -NNG ($k = 2$) on a set of 2-dimensional points in Euclidean plane.

As already said, a k -NNG is defined by an input set S and a distance function defined on S . However, it is also possible to construct a k -NNG with a distance measure that is not a proper distance function. For example, two such distance measures, cosine distance and DTW, were presented in Section 2.1, and will be later used for k -NNG construction in Chapter 3 and Chapter 6. Generally speaking, even though some distance measures do not satisfy some of the metric conditions, they are still often used in similar contexts as proper distance functions.

In order to reduce the computational cost of k -NNG construction, many approximation algorithms were developed (see Section 3.1). We will finalize this section with an overview of the measures that we will use to assess the quality of a k -NNG approximation. Let \tilde{G} be an approximation of k -NNG G , both G and \tilde{G} being defined on the same node set S of n elements, and let $s, s' \in S$. We will refer to a node $s' \in NN_{\tilde{G}}(s)$ as a *true neighbor of s* if and only if $s' \in NN_G(s)$, that is, if and only if s' is in s 's NN list in both G and \tilde{G} . We will denote the number of s 's true neighbors in \tilde{G} by $tn_{\tilde{G},s}$. The *recall of a node s* in \tilde{G} , denoted by $recall_{\tilde{G},s}$, is defined by (2.1). The *recall of a k -NNG approximation \tilde{G}* , denoted by $recall_{\tilde{G}}$, is defined by (2.2).

$$recall_{\tilde{G},s} = \frac{tn_{\tilde{G},s}}{k} \tag{2.1}$$

$$recall_{\tilde{G}} = \frac{\sum_{s \in S} recall_{\tilde{G},s}}{n} \tag{2.2}$$

There are in total $\frac{n \cdot (n-1)}{2}$ distance computations during the naive k -NNG construction. A k -NNG approximation algorithm should decrease this number. The

CHAPTER 2. BACKGROUND

larger decrease of the number of distance computations, the better is the approximation algorithm. Therefore, we introduce the $\text{scanrate}_{\tilde{G}}$ measure which assesses an algorithm in terms of the number of distance computations. This measure is given in (2.3), where $\text{dists}_{\tilde{G}}$ is the number of distance computations employed during the construction of the k -NNG approximation \tilde{G} . Small $\text{scanrate}_{\tilde{G}}$ values are better.

$$\text{scanrate}_{\tilde{G}} = \frac{\text{dists}_{\tilde{G}}}{\frac{n \cdot (n-1)}{2}} \quad (2.3)$$

For comparing two approximation algorithms by taking into account both $\text{recall}_{\tilde{G}}$ and $\text{scanrate}_{\tilde{G}}$ at the same time, we use harmonic mean, which is suitable for averaging ratios. However, before calculating harmonic mean, we first have to synchronize the two values. One problem is the range— $\text{recall}_{\tilde{G}}$ is a value in the range $[0, 1]$, while $\text{scanrate}_{\tilde{G}}$ is a value in the range $[0, \infty)$. Scan rate that is higher than 1 indicates that the algorithm recalculates distances that it has already calculated. Even though this is not optimal, it could happen in certain cases. Therefore, when calculating the harmonic mean, we limit $\text{scanrate}_{\tilde{G}}$ to be at most 1, meaning that each scan rate higher than 1 will be considered as the scan rate of exactly 1. Second, $\text{recall}_{\tilde{G}}$ values are better as they increase, while for the scan rate opposite holds— $\text{scanrate}_{\tilde{G}}$ values are better as they decrease. In order to synchronize the two measures, we introduce *scan gain*, $\text{scangain}_{\tilde{G}}$, that is given by (2.4). Finally, the harmonic mean, $\text{harmonic}_{\tilde{G}}$, is calculated based on the recall and the scan gain, as given by (2.5).

$$\text{scangain}_{\tilde{G}} = 1 - \min\{1, \text{scanrate}_{\tilde{G}}\} \quad (2.4)$$

$$\text{harmonic}_{\tilde{G}} = \begin{cases} 0, & \text{if } \text{recall}_{\tilde{G}} = 0 \text{ or } \text{scangain}_{\tilde{G}} = 0; \\ \frac{2}{\frac{1}{\text{recall}_{\tilde{G}}} + \frac{1}{\text{scangain}_{\tilde{G}}}}, & \text{otherwise.} \end{cases} \quad (2.5)$$

2.3 Hubness

Hubness is an aspect of the curse of dimensionality pertaining to nearest neighbors, which has come to the attention of the machine learning research community only

relatively recently [48]. In this section we will define the hubness phenomenon and point out the issues related to it.

Let us define the function $h_G(s) = |RNN_G(s)|$, where s is a node in a k -NN graph G . We will refer to $h_G(s)$ as the *hubness value* of the node s , which in other words is the size of s 's R -NN list. Additionally, we define the *normalized hubness value* as $\hat{h}_G(s) = \frac{h_G(s)}{k}$. Let now $S_d \subset \mathbb{R}^d$ be a set of n random points and let G_d be a k -NNG constructed on S_d for some fixed k . As the dimensionality d increases, the distribution of the value of h_{G_d} becomes considerably skewed. Consequently, some nodes, which we will refer to as *hubs*, are included in many more NN lists than other points. More formally, a hub is a node s in the k -NN graph G for which $h_G(s) \gg k$. If a dataset contains hubs, that is, if the distribution of hubness values is highly skewed, it can be said that the *hubness phenomenon* is present therein.

In Figure 2.2 we show the aforementioned dependence of hubness values on data dimensionality. For this purpose, four synthetic datasets S_d of dimensionalities $d \in \{10, 20, 50, 100\}$ were created. All datasets were drawn from the uniform distribution with standard deviation 1 and mean $(0, 0, \dots, 0)$, and all of them have 100,000 instances. The hubness values presented in the figure are extracted from k -NNGs, with $k = 5$ and L_2 distance, that are constructed on the four datasets. The figure contains four histograms of normalized hubness values, one for each dataset. The hubness values are shown on the x axis (note that the x axis is logarithmic), while the percentages of points that have corresponding hubness values are shown on the y axis.

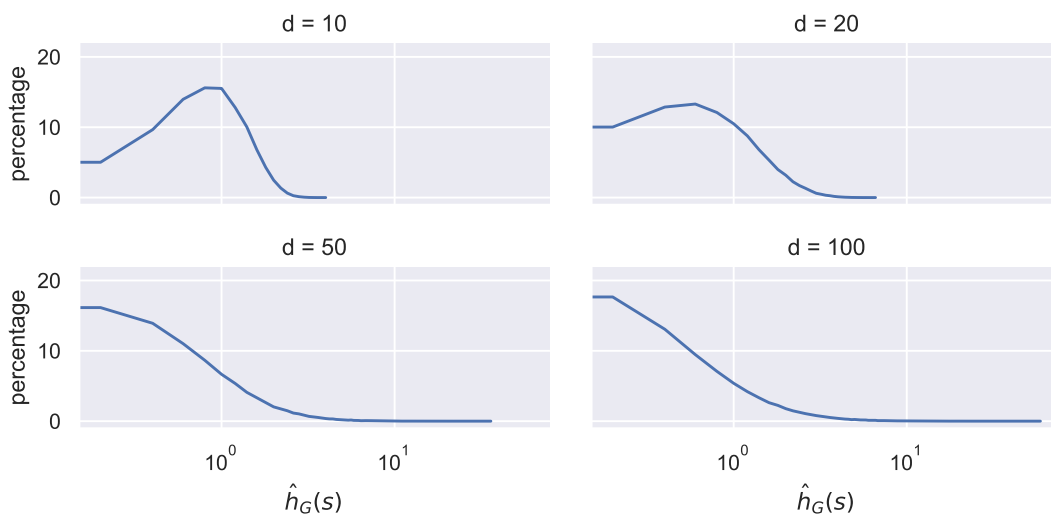


Figure 2.2: Dependence of hubness values (extracted from k -NNGs with $k = 5$ and L_2 distance) on data dimensionality (d).

CHAPTER 2. BACKGROUND

It has been shown that hubness, as a phenomenon, appears in intrinsically high-dimensional machine learning data as an inherent property of high intrinsic dimensionality itself, and is not an artifact of finite samples nor a peculiarity of specific datasets [48]. However, one should have in mind that high dimensionality of a dataset does not necessarily imply high intrinsic dimensionality, and therefore does not necessarily imply the presence of hubness phenomenon—only datasets with high *intrinsic* dimensionality are affected by this phenomenon. It was shown that hubness influences various data-mining and machine-learning algorithms [46–49, 52], often in a negative way.

Chapter 3

Existing algorithms for k -NN graph construction

This chapter gives an overview of the existing algorithms for k -NNG construction. In Subsection 3.1 we will present different classes of such algorithms, while Subsection 3.2 will be dedicated to a single k -NNG approximation algorithm, called *NN-Descent*, which is the most relevant for this thesis. Finally, in Subsection 3.3 we will present the analyses and algorithms regarding k -NNGs that change over time.

3.1 Optimization of k -NN graph construction

The naive brute-force computation of k -NNG entails $\binom{n}{2}$ distance computations, which leads to quadratic time complexity. Numerous methods for k -NNG construction have been developed in order to decrease the computational cost. An optimization of k -NNG construction can be performed in different ways. The first way would be to introduce certain restrictions to the problem (for example to fix the distance function that will be used for k -NNG construction, to define properties of underlying objects upon which the graph is built, etc.) that would then allow further optimizations. In the following text we will call this class of algorithms *specialized algorithms*, and they will be presented in Subsection 3.1.1.

The second way to optimize k -NNG construction is to parallelize it. The parallelization can take place on a single machine, by using multiple cores of a single processor, or it can make use of larger clusters that contain multiple machines. We will refer to this class of algorithms as *parallelized algorithms*. Parallelized algorithms will be presented in Subsection 3.1.2.

Finally, the third way to optimize the problem is to build k -NNG approximation instead of building the exact k -NNG. Approximation algorithms are generally used

3.1. OPTIMIZATION OF k -NN GRAPH CONSTRUCTION

when speed is more important than accuracy. An approximation algorithm should minimize the computational cost, while maximizing the approximation correctness. We will refer to this class of algorithms as *approximation algorithms*. This class of algorithms is most relevant to this thesis, and Subsection 3.1.3 will give an overview of the existing approximation algorithms for k -NNG construction.

3.1.1 Specialized algorithms

Specialized algorithms for k -NNG construction are the ones that introduce certain restrictions and therefore do not apply to the general case. This subsection gives an overview of the existing algorithms from this class.

Vaidya [54] proposed an $O(n \log n)$ algorithm for 1-NNG construction for a set S of n points in a d -dimensional space and an L_p -metric. The algorithm can easily be modified to support the more general k -NNG construction problem, and in this case the complexity is $O(kn \log n)$. The outline of the 1-NNG construction algorithm is as follows. The algorithm maintains a collection B of disjoint closed cubes which contain all the n points. At the beginning B contains a single box which is the smallest cube containing all the points in S , and at the end, every box in B is degenerate, consisting of a single point in S . For each box in B , the information about its neighboring boxes is kept. For each point p in a box, every nearest neighbor is located in the box itself or in some of the box's neighboring boxes. In each step, the algorithm splits a box of the greatest volume into 2^d smaller boxes. The split is defined by d mutually orthogonal hyperplanes passing through the center of the box, each being parallel to one axis. After the split, the boxes that do not contain points from S are discarded. Each preserved box is shrunk to the minimal box that contains all points from S that are inside it. These boxes are then put in the collection B , and their neighboring boxes are updated accordingly. At the end of the algorithm, when each box contains only one point from S , the corresponding neighboring boxes contain the points' nearest neighbors.

One famous class of algorithms for k -NNG construction uses the Voronoi diagram to build k -NNG. Given a metric space X and a finite, nonempty set S of n points in the space X , the Voronoi region associated with $s \in S$, denoted by $\mathcal{V}(s)$, is the set of all points in X whose distance to s is not greater than their distance to all $s' \in S, s' \neq s$. Therefore the Voronoi diagram partitions the space into Voronoi regions defined by all $s \in S$. We would like to point out that building Voronoi diagram in spaces of dimensions higher than 2 is impractical, hence the

CHAPTER 3. EXISTING ALGORITHMS FOR k -NN GRAPH CONSTRUCTION

algorithms that use Voronoi diagram for k -NNG construction can be used only in two dimensional spaces.

Shamos et al. [50] used the Voronoi diagram to solve the 1-NNG construction problem in the Euclidean plane. After building the Voronoi diagram in $O(n \log n)$ (which is asymptotically optimal), the nearest neighbor of each point $s \in S$ is found by simply examining all $s' \in S$ for which $\mathcal{V}(s)$ and $\mathcal{V}(s')$ share an edge. The point s' that is closest to s is then s 's nearest neighbor. The complexity of finding the nearest neighbor for each point upon a Voronoi diagram is $O(n)$, which together with the algorithm for building the Voronoi diagram leads to the overall complexity of $O(n \log n)$.

The Voronoi diagram can be further generalized. Instead of associating a Voronoi region with a single point $s \in S$, it can be associated with a set of points $S' \subset S$, in which case a Voronoi region $\mathcal{V}(S')$ contains all the points in X whose $|S'|$ nearest points from S are exactly the points in S' . The Voronoi diagram of o -order is then the Voronoi diagram whose Voronoi regions are defined by sets of cardinality o . The problem of k -NNG construction can then be solved by constructing the $(k + 1)$ -order Voronoi diagram. Each point's k nearest neighbors are then easily determined by taking the points lying in the same Voronoi region. Lee [39] introduced an algorithm for the construction of the $(k + 1)$ -order Voronoi diagram of n points in the Euclidean plane in $O(k^2 n \log n)$, and Aggarwal et al. [1] later improved it to $O(k^2 n + n \log n)$.

Similar work has been done by Dickerson et al. [19]. The authors used the Delaunay triangulation as a tool for k -NNG construction. The Delaunay triangulation for a set S in a d -dimensional metric space X , denoted by $DT(S)$, is the triangulation such that no point in S is inside the circum-hypersphere of any d -simplex in $DT(S)$ (it can be proved that, modulo some mild assumptions about the set S , such a triangulation always exists and it is unique; in particular, it can be thought of as the dual graph of the Voronoi diagram created on the same set S). In their algorithm, Dickerson et al. are constructing k -NNG on the set of points S by building the Delaunay triangulation in the preprocessing phase, and then using it to construct the final k -NNG. The Delaunay triangulation is built by using the algorithm introduced by Bern et al. [6] In this algorithm a new "nicer" set of points $S' \supset S, |S'| = O(|S|)$ is defined, and then the Delaunay triangulation is built on it in $O(n \log n)$. After the preprocessing phase, for each point $s \in S$ a breadth-first search is performed on the built Delaunay triangulation to find k nearest neighbors of s in S . The overall time complexity of this algorithm is $O(kn \log n)$.

3.1. OPTIMIZATION OF K -NN GRAPH CONSTRUCTION

Virmajoki et al. [55] developed a divide-and-conquer algorithm for k -NNG construction in a d -dimensional metric space. In the “divide” step the set of points S is split into two approximately equal sized subsets S_1 and S_2 by using the *principal component analysis* (PCA). Namely, the principal axis of the points in S is calculated, and a $(d - 1)$ -dimensional hyperplane that is perpendicular to the principal axis is selected in such a way that approximately half of the points belong to one side of the space, and the rest to the other side. The problem is then solved recursively for S_1 and S_2 . The recursion stops when a set smaller than c_k is reached, and then a brute-force k -NNG construction is performed instead. After the solutions for S_1 and S_2 are obtained, a third subset S_3 is created. This subset contains all the points that are closer to the dividing hyperplane than to their nearest neighbor in the corresponding subset (S_1 or S_2). The algorithm is then recursively applied to the S_3 , too. Finally, the results of all the three subproblems are merged into the resulting k -NNG. The time complexity of this approach is $O(d^2 n^{1.58} \log n)$.

Paredes et al. [41] proposed two variants of an algorithm for k -NNG construction in general metric spaces. Both variants have a similar outline. An index data structure for fast nearest neighbors queries is created. During the creation of the index, each calculated distance is simultaneously used to update another data structure that stores current best nearest neighbors for each point. After the index is created, the current best neighbors lists have to be updated so that they correspond to the exact neighborhoods. In order to achieve that, the index data structure is used to obtain a list of potential neighbors for each point. Each such list is guaranteed to contain the point’s real neighbors, but can contain non-neighboring points as well. In order to avoid unnecessary distance computations, these lists are additionally pruned by applying various implications of the triangle inequality property of the distance function. Finally the distances between a query point and all the points from the pruned list are computed, and the neighbors set of the query point is updated accordingly. The authors reported empirical complexities ranged from $O(n^{1.10})$ for 4-dimensional space to $O(n^{1.96})$ for 24-dimensional space.

Anastasiu et al. [2] proposed an algorithm called *L2Knnng* for k -NNG construction with cosine distance. The basic assumption of the algorithm is that the set S contains sparse high-dimensional vectors. The algorithm operates in two stages. In the first stage an approximation of k -NNG is efficiently built relying on two ideas: 1) high-weight features count heavily toward the (dis)similarity of two vectors [43], 2) a vector’s neighbor’s neighbor is also likely to be the vector’s neighbor [20]. In the second stage, the algorithm iterates through all the vectors from S , one by

CHAPTER 3. EXISTING ALGORITHMS FOR K -NN GRAPH CONSTRUCTION

one, in each step updating the current vector's NN list considering only the vectors that were already processed, and vice versa—the already processed vectors' NN lists are updated with the current vector. During this stage, various pruning techniques are used to reduce the number of distance computations. The authors verified their solution by conducting experiments on real-world datasets. They compared the results of their approach with the results of a few approximation algorithms and a few exact algorithms, and thereby they concluded that for the chosen datasets L2Knnng outperforms other algorithms.

3.1.2 Parallelized algorithms

A standard tool for optimizing an algorithm in terms of execution time is parallelization. This section gives an overview of parallel algorithms for k -NNG construction.

Callahan et al. [11, 12] proposed a parallel algorithm for k -NNG construction that works in $O(\log n)$ time when run on $O(n)$ processors for a constant k . The algorithm is based on so called *well-separated pair decomposition* which is a structure consisting of a binary tree and a *well-separated realization* (which is also a concept introduced by the authors). The intuition behind well-separated pair decomposition is to make geometrically separated "clusters" of points, and to do that for different granularity levels in terms of clusters size. The authors introduced an algorithm that builds well-separated pair decomposition in $O(\log n)$ time with $O(n)$ processors on a CREW PRAM. Upon a well-separated pair decomposition, by using its various geometric properties, for each point s , a set of $O(1)$ candidates that may have s in their k nearest neighbors is computed in $O(\log n)$ time with $O(n/\log n)$ processors. After this step, k nearest neighbors for each point are computed in $O(\log n)$ time with $O(n)$ processors.

Plaku et al. [45] developed a distributed algorithm for k -NNG construction. The algorithm works under the assumption that p available processors do not share memory. The initial set of points S is partitioned into p subsets S_1, S_2, \dots, S_p , each being assigned to a different processor. Each processor p_i then computes an index structure for NN queries for S_i (the index structure algorithm can be arbitrarily chosen). After this step, each processor queries its index structure to obtain NN lists of points from its set, and to obtain NN lists of points from all the other processors as well. The first task does not require any communication between the processors, but the second one does—a processor has to get the points whose NN lists it should compute, and afterwards the computed NN lists have to be sent

3.1. OPTIMIZATION OF K -NN GRAPH CONSTRUCTION

back to the corresponding processors. In order to decrease communication, authors suggested a pruning technique. Namely, each processor clusterizes its points set. The information about the clusters' bounds are sent to other processors. Before asking a processor p_i for a point's NN list, a processor p_j should check whether the point can find its neighbors within p_i 's points by looking at p_i 's clusters' bounds. Finally, after a point's NN lists are obtained from all the processors, they are merged into the resulting NN list. By the experimental analysis authors reported a speedup that is nearly linear in the number of processors.

A parallel algorithm, most suitable for multi-core machines, for k -NNG construction upon points in d -dimensional space for $d \leq 3$, was presented by Connor et al. [15]. The algorithm is based on Z-order (Morton-order) which is a space-filling curve that preserves locality. The Z-value of a point is calculated by interleaving the binary representations of its coordinates' values. The algorithm starts by sorting the points from S by their Z-value. Then the initial k -NNG approximation is created by comparing each point with a certain amount of points that surround it in the sorted array (sliding window method). Finally, by using quad tree, each point's neighborhood is refined in an efficient way. The quad tree can be induced from the sorted array by defining each tree's node with a range of points in the array, which is possible due to geometric properties of Z-order. Each point then recursively goes from the root of the tree to its child nodes, stopping when reaching a node that contains small amount of points, in which case the querying point is compared to all of them, improving its NN list. There are also other early termination criteria defined by geometric properties of the tree. Parallelization is then implemented as follows. A parallel distribution sort is used for the sorting phase. The sorted array is then split into p chunks, p being the number of available processors, with each processor computing the initial approximate NN lists for one chunk. The k -NNG approximation is refined to exact k -NNG by letting each processor perform the recursive step of the algorithm for the points in their chunks.

Komarov et al. [34] parallelized the brute-force k -NNG construction on graphic processing unit (GPU). As a first step, the algorithm efficiently calculates all pairwise distances. The problem of calculating the distances is formulated as a matrix multiplication problem. The authors supported Euclidian, cosine and Pearson distance functions. After that, k nearest neighbors of each point are selected by conducting a GPU-based multi-select algorithm based on quicksort.

3.1.3 Approximation algorithms

Many k -NNG approximation algorithms are based either on divide-and-conquer technique, or on the idea of the famous algorithm *NN-Descent*, which will later be introduced. Examples of both types of k -NNG approximation algorithms will be presented in this section.

Chen et al. [13] introduced an approximation divide-and-conquer algorithm that uses Lanczos spectral bisection [7, 31, 36, 53] for the divide step. Let $\hat{S} = [\hat{s}_1, \dots, \hat{s}_n] \in \mathbb{R}^{d \times n}$ be a data matrix, where each column \hat{s}_i represents a centered data point $s_i \in S$. Lanczos spectral bisection first computes of the largest singular triplet $\langle \sigma, u, v \rangle$ of \hat{S} by using Lanczos algorithm [36]. Then, the input set S is split into two sets $S_+ = \{s_i \mid v_i \geq 0\}$ and $S_- = \{s_i \mid v_i < 0\}$, where v_i is the i^{th} entry of the right singular vector v . The algorithm by Chen et al. is designed for d -dimensional Euclidean space, and has time complexity of $O(dn^t)$, where $t \in (1, 2)$ depends on the algorithm's parameter α . The termination criteria for division step is defined in terms of a set size—when a set is small enough it is not being further divided. When such a small set is reached, brute-force k -NNG construction is performed upon it. The conquer step then trivially merges all these small k -NNGs, making a final k -NNG approximation. In order to achieve high k -NNG approximation's accuracy, some points should be present in multiple small termination sets. For this purpose, the authors proposed two variants of the divide step. In the first variant, called *overlap method*, a set is divided into two overlapping subsets. In the second variant, called *glue method*, a set is divided into two disjoint subsets, but then another subset, called *gluing subset*, is created. The gluing subset contains the points that originate from both disjoint subsets. The authors introduced a parameter α that determines the subsets intersection size for the overlap method and the gluing subset size for the gluing method. The parameter α is expressed as a portion of the size of a set that is divided. Experiments show that for a highly accurate approximation, a small value of α is usually sufficient, which leads to a small exponent in the time complexity.

One of the famous and most efficient approximation algorithms is the one introduced by Dong et al. [20]. 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, the empirical complexity of *NN-Descent* for datasets of relatively low intrinsic dimensionality is around $O(n^{1.14})$. The basic assumption of the algorithm is that two points that share a

3.1. OPTIMIZATION OF K -NN GRAPH CONSTRUCTION

neighbor are also likely to be neighbors. Since *NN-Descent* is highly relevant for this thesis, it is thoroughly explained in Section 3.2.

Jones et al. [30] proposed an iterative randomized approximation algorithm for k -NNG construction (RANN) on a set S of points in d -dimensional Euclidean space. Each iteration of the algorithm independently generates its own approximate k -NNG. The k -NNGs of individual iterations are merged into the resulting k -NNG approximation. Each iteration consists of two main steps. In the first step, a random space rotation is applied, meaning that all data points are rotated in the same manner. The second step utilizes divide-and-conquer technique—in the divide phase the input set S is split into the smaller subsets, while in the conquer phase k -NNGs are calculated on those subsets and then merged into a single k -NNG. The divide phase is recursive—the current set is being divided as long as its cardinality is greater than k . In the recursion depth i of the divide phase, the current set C is split into similarly sized sets $C_+ = \{c \mid c \in C \wedge c \geq m\}$ and $C_- = \{c \mid c \in C \wedge c < m\}$, where m is the median of points' values on dimension $i \bmod d$. The number of the algorithm's iterations is user-defined. At the end of the algorithm, after all the iterations are completed, a technique called *supercharging* is applied. Supercharging is essentially based on the same idea as in *NN-Descent*. Namely, in order to improve neighborhoods, each point is compared with all its $O(k^2)$ neighbors' neighbors. The overall execution time of the algorithm is proportional to $Tn(d \log d + k(d + \log k) \log n) + nk^2(d + \log k)$, T being the number of iterations. The accuracy of the resulting k -NNG approximation depends mostly on d and T (higher d values have negative, while higher T values have positive influence on k -NNG approximation accuracy).

Wang et al. [57] presented another divide-and-conquer algorithm that is actually a mixture of all the three above presented algorithms. Namely, like in [13], Lanczos spectral bisection is used for divide step. Similarly to [30], divide-and-conquer algorithm is run multiple times in order to achieve higher approximation accuracy. The difference between multiple runs lies in the way sets are split in divide steps. Namely, Lanczos algorithm is not fed with the whole set, but with a randomly chosen set's sample. The results from different runs are merged in the same manner as in the algorithm by Jones et al. [30]. Finally, *NN-Descent's* idea is used at the end, for the *neighborhood propagation*. In the neighborhood propagation phase points are compared with neighbors' neighbors, but unlike in [30], the process is recursively continued for the neighbors that are newly added during the propagation. Under the assumption that the number of different runs and the

CHAPTER 3. EXISTING ALGORITHMS FOR K -NN GRAPH CONSTRUCTION

number of visited neighbors during the propagation phase are small, the authors reported time complexity of $O(dn \log n)$.

Another similar divide-and-conquer algorithm was presented by Zhang et al. [59]. Their algorithm is based on locality sensitive hashing (LSH) [23], and therefore can be applied with any distance function for which a locality sensitive hash function is designed. The overall time complexity of the algorithm is $O(l(d + \log n)n)$, where l is usually a small number. The outline of the algorithm is the same as for the above algorithms: divide-and-conquer phase is run multiple times, exact k -NNGs are constructed on small sets of points and then merged, the neighborhood propagation is performed as the final step of the algorithm. The main difference between this and above presented algorithms lies in the divide step. In this algorithm, the current set is divided into multiple subsets—two points are placed in the same subset if their hash value is the same. Since the hash function is designed to preserve locality, similar points are likely to be placed in the same subset. However, the hash function might not distribute the points uniformly over the subsets. In order to solve this problem, authors project points' hash codes onto a random direction $w \in \mathbb{R}^m$, where m is the number of distinct hash values. The points are then sorted by their projection values, and the equal-sized ranges from this sorted order are assigned to each subset.

Park et al. [42, 43] introduced an algorithm called *greedy filtering* which constructs approximate k -NNG by using cosine distance on a set S of d -dimensional vectors. The vectors in S are normalized by L_2 -norm such that each vector's sum of its squared values is equal to 1. The authors represented each vector by a list of pairs (d_i, r_j) , where d_i is dimension and r_j is vector's value for that dimension. The list is decreasingly sorted by values (i.e., r components). Relying on the fact that higher values contribute more for cosine distance, the authors introduced the notion called vector prefix. A prefix of length m is a new vector that contains the first m elements of the original vector. The algorithm starts by creating a list $L[d_i]$ for each dimension d_i . $L[d_i]$ contains the vectors which have d_i in their prefixes. Vectors' prefixes are determined in the following way. Let p be a vector's prefix, and D_p a set of all dimensions that appear in p . The vector's prefix p is such that its length is minimal and $\sum_{d_i \in D_p} |L[d_i]| \geq \mu$, for some predefined value μ . When $L[d_i]$ is filled for all dimensions d_i , pairwise distances are calculated between vectors from each dimension's list, which results with a k -NNG approximation. The authors pointed out a problem with this approach, which occurs when some of the dimensions' lists are very large, in which case the algorithm slows down. In order to alleviate the problem, the authors introduced *fast greedy filtering* in which the

distances are calculated between prefixes, not between the whole vectors. Besides that, in fast greedy filtering not all the pairwise distances from dimensions’ lists are calculated. Instead, authors used a technique based on so called *inverted index*.

Sieranoja et al. [51] combined the algorithm developed by Connor et al. [15] with *NN-Descent* [20], creating a new algorithm that constructs k -NNG approximation for Minkowski distance functions. The initial k -NNG approximation is created in a similar manner as in the algorithm by Connor et al. [15] (see Subsection 3.1.2). However, instead of applying the process only once, it is repeated multiple times for different Z -order projections. Besides that, the authors pointed out one potential problem with this approach—time and space complexity of Z -values calculations are linearly dependent on d , which is a problem for high d values. In order to solve this problem, authors introduced dimensionality reduction step which is performed before calculating Z -values. After each Z -order projection, the current k -NNG approximation is updated with sliding window method, which is then followed by one *NN-Descent* iteration. In order to reduce complexity, not all the neighbors are used for local joins in *NN-Descent*, but instead only k_{ndes} of them. The authors validated their method with various experiments, which showed that the method performs well even on high-dimensional data.

3.2 *NN-Descent*

NN-Descent [20] is a fast approximation algorithm for k -NNG construction. The basic assumption made by *NN-Descent* can be summarized as “a point’s neighbor’s neighbor is also likely to be the point’s neighbor”. The algorithm can be used with any distance function. Moreover, the algorithm can even be used with functions that do not satisfy some of the metric conditions, but in that case the accuracy of the algorithm depends on the extent to which the function aligns with the aforementioned assumption. In the following text a detailed overview of the algorithm will be given, after which, in Subsection 3.2.1, we will review a literature on *NN-Descent*’s modifications.

The algorithm starts with creation of a random k -NNG \tilde{G} , which is then iteratively improved. Inside a single improvement iteration, for each point $s \in S$ its NN and R -NN lists are examined to determine whether any points in these lists should be added to NN lists of any of the others. More precisely, distance between each two $v, w \in NN_{\tilde{G}}(s) \cup RNN_{\tilde{G}}(s)$ is calculated. According to the calculated distance, v ’s NN list is updated with the point w if needed, and vice versa, w ’s NN list is updated with v if needed, which is a process called *local join*. After

CHAPTER 3. EXISTING ALGORITHMS FOR K -NN GRAPH CONSTRUCTION

each iteration, another one is triggered only if a termination condition is not met. The authors defined two variants of termination condition. One way is to simply terminate after a fixed number of iterations (in further text *fixed iterations*). Another way is to terminate when the algorithm converges (in further text *convergence*). The convergence of the algorithm is defined in terms of number of NN lists updates that occurred in the last executed iteration. Namely, if the number of NN lists updates is below a predefined threshold, the algorithm terminates, i.e., the termination condition is in that case met.

In order to avoid unnecessary comparisons, a boolean flag called *new* is stored for each point's neighbor. In the initial random graph the flag has value *true* for all the neighbors in k -NNG. Afterwards, in the i^{th} iteration the flag has value *true* only if the neighbor is added in the corresponding NN list during the $(i - 1)^{\text{th}}$ iteration. Local join between two points is then performed only if at least one of them has value *true* for the flag *new*. Namely, if for the both points holds that *new* = *false*, that means that neither point is added in the NN list in the previous iteration, implying that the distance between them has already been calculated.

Algorithm 1: Function that returns r random elements from a set X .

```
function Sample( $X, r$ )
| return  $r$  uniformly random elements from  $X$ .
end
```

Algorithm 2: Function that returns a random k -NN graph.

```
function RandKNN( $S, k, dist$ )
| // Note: Function Sample is defined in Algorithms 1.
1 |  $\tilde{G} \leftarrow$  empty graph;
2 | foreach data point  $s \in S$  do
3 | |  $R \leftarrow$  Sample( $S \setminus \{s\}, k$ );
4 | | foreach data point  $v \in R$  do
5 | | |  $d \leftarrow$  dist( $s, v$ );
6 | | | Use  $\langle v, d \rangle$  to update  $NN_{\tilde{G}}(s)$ , and use  $\langle s, d \rangle$  to update  $NN_{\tilde{G}}(v)$ ;
7 | | end
8 | end
9 | return  $\tilde{G}$ .
end
```

The speed of *NN-Descent* is strongly influenced by the k value. As k becomes larger, the algorithm becomes slower. More precisely, the time complexity has a

quadratic dependence on k , since during each iteration, points appearing in NN or R -NN lists of the same point are evaluated as candidates for each other's NN lists. As a way to reduce the number of combinations of points considered, the authors introduced *sampling*. Sampling reduces the number of evaluations by taking a random selection of points from NN and R -NN lists, and then operating only on these points. 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 R -NN lists.

Algorithm 3: Outline of *NN-Descent* algorithm.

input : set of points S , distance function $dist$, neighborhood size k ,
sampling ρ
output: k -NNG approximation \tilde{G}
// Note: Functions *Sample* and *RandKNNNG* are defined in Algorithms 1 and 2,
respectively.

- 1 $\tilde{G} \leftarrow \text{RandKNNNG}(S, k, dist)$;
- 2 **repeat**
- 3 **foreach** data point $s \in S$ **do**
- 4 $R \leftarrow \text{Sample}(NN_{\tilde{G}}(s), \rho \cdot k) \cup \text{Sample}(RNN_{\tilde{G}}(s), \rho \cdot k)$;
- 5 **foreach** two points $v, w \in R$ **do**
- 6 $d \leftarrow \text{dist}(v, w)$;
- 7 Use $\langle v, d \rangle$ to update $NN_{\tilde{G}}(w)$, and use $\langle w, d \rangle$ to update $NN_{\tilde{G}}(v)$;
- 8 **end**
- 9 **end**
- 10 **until** *Termination condition is met*
- 11 **return** \tilde{G} .

Table 3.1: List of parameters of *NN-Descent*.

Parameter	Description
k	Neighborhood size (size of NN lists).
$dist$	Distance function.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (value between 0 and 1). Algorithm converges when there is less than $conv \cdot k \cdot n$ updates in the most recent iteration, where n is the number of k -NNG vertices. Note: present only for <i>convergence</i> termination condition.
ρ	Sampling (value between 0 and 1, not including 0). Only $\rho \cdot k$ points from NN and R -NN lists are considered for local joins.

CHAPTER 3. EXISTING ALGORITHMS FOR K -NN GRAPH CONSTRUCTION

An outline of *NN-Descent* is shown as Algorithm 3¹ and the list of *NN-Descent* parameters is given in Table 3.1.

The main drawback of *NN-Descent* is that the quality of approximation it produces is highly influenced by the intrinsic dimensionality of the input set, in that the quality of the approximation decreases as the intrinsic dimensionality increases. In Chapter 4 the influence of high dimensionality on *NN-Descent* will be further explained. Later, in Chapter 5, we will propose new approaches that are designed to overcome this challenge to some extent.

The second drawback of the algorithm is that the speedup over the brute-force k -NNG construction and accuracy are highly influenced by the cardinality of input set (n) and k value. Namely, as n increases, speedup increases and accuracy decreases; as k increases, speedup decreases and accuracy increases. As a conclusion, *NN-Descent* is not suitable for small n values or large k values—if input set is sufficiently small or k value is sufficiently large, brute-force algorithm becomes faster than *NN-Descent*. Similarly, *NN-Descent* is not suitable for very large n and small k values, because in that case accuracy of the algorithm can drop to the point where it becomes unusable.

3.2.1 *NN-Descent* as an inspiration for other algorithms

As a part of their algorithm for exact k -NNG construction with cosine distance, Anastasiu et al. [2] are building a k -NNG approximation which then they improve to the exact k -NNG (see Subsection 3.1.1). The approximation is built on two ideas, one of which is based on *NN-Descent*. Just like in *NN-Descent*, the approximation is iteratively improved by conducting comparisons of points that share a neighbor. However, in this algorithm not all the points that share a neighbor are mutually compared. In each iteration, each point traverses its neighborhood in increasing order of distances and the neighborhood of each visited neighbor is again traversed in the same manner. The neighbors' neighbors are then picked greedily until the predefined number of them is reached. These picked points are then used for comparisons. The algorithm also introduces certain corner cases for which a neighbors' neighbor must not be picked. This modification of *NN-Descent* is compared with the original algorithm and the experiments showed an increase of recall and execution time.

¹Algorithm 3 does not contain aforementioned optimization which implies avoiding unnecessary comparisons by using the *new* flag. The optimization is omitted from the algorithm with a goal of preserving the algorithm's readability.

Houle et al. [28] introduced NNF-Descent (Nearest Neighbor Feature Descent), an *NN-Descent*-based feature sparsification method optimized for image databases. The goal of this algorithm is to improve semantic quality of the resulting k -NNG. Semantic quality of a graph is high if the neighboring points are semantically similar. The authors introduced Local Laplacian Score (LLS) that is based on the Laplacian Score (LS). Both LS and LLS are measures of features qualities, difference being that LS is a global measure calculated upon all points, while LLS is a local measure calculated upon a single point. NNF-Descent identifies noisy features' values by using LLS, and then replaces them with zeros, under the assumption that the dataset points are normalized before the execution of the algorithm. The first step of the algorithm is creation of a k -NNG approximation by using *NN-Descent*. After that, the algorithm iteratively improves semantic quality of the graph. In each iteration LLS values for all the point's features are computed. The values of a point's top z ranked features are replaced with 0. After that, the distances to this point are recalculated, and the neighborhood propagation is performed in order to update k -NNG. The algorithm is verified with respect to the accuracy of k -NNG and to the image labeling task. The experiments showed that NNF-Descent outperforms its competitors in most cases.

Debatty et al. [18] presented a method for constructing k -NNG approximation for large text datasets. This method applies *NN-Descent* only to smaller buckets of data, leading to a lower execution time. The buckets are created with respect to a custom Context Triggered Piecewise Hashing (CTPH), which assigns the same hash to similar strings. The strings with the same hash are then placed in the same bucket. This approach has one major drawback—if one string is put into only one bucket, each bucket will correspond to a disconnected component in the final k -NNG approximation. In order to solve this problem, the authors introduced *stages*. Instead of creating hashes of the length that is aligned with the number of desired buckets, longer hashes are created. These longer hashes are then split into parts of appropriate lengths, each part being mapped to a single bucket. In that way, a single string is being assigned to multiple buckets. The experiments showed that this approach speeds up *NN-Descent* preserving the reasonable accuracy.

Sieranoja et al. [51] also used ideas of *NN-Descent* in their algorithm. Since their algorithm was already explained in Subsection 3.1.3, here we will only point out how it differs from the basic *NN-Descent*. Unlike *NN-Descent*, this algorithm works only with Minkowski distances. Additionally, this algorithm does not start with a random graph, but instead with a more accurate k -NNG approximation that is created by making use of Z-order. Finally, in order to reduce complexity,

CHAPTER 3. EXISTING ALGORITHMS FOR k -NN GRAPH CONSTRUCTION

the algorithm does not use whole neighborhoods for local joins, but instead only a portion of it, which is similar to the sampling technique.

Some approximation algorithms use a concept called neighborhood propagation (see Subsection 3.1.3), which is based on *NN-Descent*'s idea. Neighborhood propagation is usually used as an algorithm's final step, the purpose of which is to make the final approximation more accurate.

3.3 Temporal k nearest neighbor graphs

In various real world problems data change over time. Consequently, a k -NNG built on such temporal data must be updated as data change. In this section we present analyses and algorithms that are related to temporal k -NNGs. In Chapter 6 we further analyse these graphs, and propose two approximation algorithms for k -NNG update after a subset of its nodes has changed.

In their research, Lathia et al. [37] analyzed how k -NNG changes as the underlying data change. For this purpose, authors used MovieLens datasets² that contain users' movie ratings. The nodes of k -NNGs were users, while the distances among them were calculated in four different ways, obtaining four different k -NNGs. What changed over time were the users' ratings—during the time users rated movies, therefore the number of ratings per user monotonically increases over time. In the analysis, one interesting phenomenon emerged. Unexpectedly, users' neighbors did not converge during the time. The reason why it was unexpected is that the distance function is computed on more information as ratings are added, and hence should be more refined. However, some of the distance functions did not show this behavior. Another interesting phenomenon was pointed out in this analysis—the average number of unique neighbors per node in the whole time span was nearly $2k$.

Debatty et al. [17] proposed an online approximation algorithm for k -NNG construction. The algorithm inserts the points one by one into a k -NNG. Insertion of a point s is performed by an iterative approach. Each iteration starts by choosing a random point p that is already in the k -NNG. Let d_{\min} be the minimum distance between s and a point from its current NN list. If distance between s and p is greater than a threshold that is linearly dependent on d_{\min} , the current iteration terminates, and the algorithm proceeds with the next iteration. Otherwise, distances between s and p 's neighbors are calculated one by one, until a calculated distance implies an update of s 's NN list or until all p 's neighbors are examined.

²<http://www.grouplens.org/taxonomy/term/14>

3.3. TEMPORAL K NEAREST NEIGHBOR GRAPHS

In the first case, p 's neighbor that was inserted into s 's NN list, takes a role of the point p , and the whole process is repeated. In the second case, when none of the p 's neighbors caused an update of s 's NN list, the iteration terminates, and the algorithm proceeds with the next iteration. The algorithm stops iterating when a predefined number of distance calculations is reached. When s has found its neighborhood, other points should consider s for their own NN lists, as well. This is performed by comparing the point with its extended neighborhood.

Zhao [60] also presented an online approximation algorithm for k -NNG construction. The algorithm supports additions and deletions of k -NNG's nodes, which is exactly the reason why it can be declared as an online algorithm. A k -NNG is built increasingly—at the beginning, k -NNG is empty (i.e., without nodes), and then nodes are added in the graph one by one. An addition of a node s is performed by first calculating the distance between s and a certain number of randomly chosen nodes that are already in k -NNG. The nearest k among them are stored in a list L . After this step, *NN-Descent*'s idea is performed—neighbors of each node that is newly added to L are recursively used for further s 's NN list improvements. The algorithm stops when no new nodes are added in L . Besides this basic approach, the author introduced two additional improvements. The first improvement aims to decrease the number of distance calculations by making use of a phenomenon called *occlusion*. Let a and b be points from NN list of s . Point b is *occluded* by point a if b 's distance to s is larger than a 's distance to s , and b is nearer to a than it is to s . The neighbors of the occluded point are not considered for further improvements of s 's NN list in *NN-Descent* phase of the algorithm, due to an assumption that it would lead to the same local region as the point by which it is occluded. The second improvement introduces neighborhood propagation as a final step of the algorithm.

NN-Descent on high dimensional data

NN-Descent is known to produce many incorrect nearest neighbors when applied on high-dimensional data [20]. Experimental verification of this *NN-Descent*'s behavior is shown in Figure 4.1. For each dataset S_d introduced in Section 2.3, a k -NNG G with $k = 5$ and L_2 distance, and its *NN-Descent* approximation \tilde{G} were constructed. The recall values for each of these approximations are presented in Figure 4.1. As can be seen, *NN-Descent*'s accuracy drastically drops as the dataset dimensionality increases. In Section 4.1 we will explain the reasons for such a behavior. Additionally, in Section 4.2, we will explain how *NN-Descent*'s initial random graph influences points' recalls.

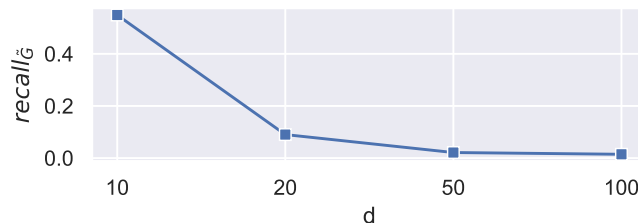


Figure 4.1: Recall values of *NN-Descent* k -NNG approximations ($k = 5$) created on datasets of dimensionalities 10, 20, 50 and 100.

4.1 Influence of hubness on *NN-Descent*

Since hubness is a phenomenon that appears in high-dimensional data, and at the same time was shown to influence many other machine learning algorithms, in this section we investigate whether it also influences the performance of *NN-Descent*.

Let us start by comparing Figures 2.2 and 4.1, which were created on the same datasets. The figures clearly show that an increase of hubness values' skewness is aligned with a decrease of *NN-Descent*'s recall. This high-level alignment is

4.1. INFLUENCE OF HUBNESS ON *NN-DESCENT*

the first indicator that hubness might be the main cause of *NN-Descent*'s bad performance on high-dimensional data.

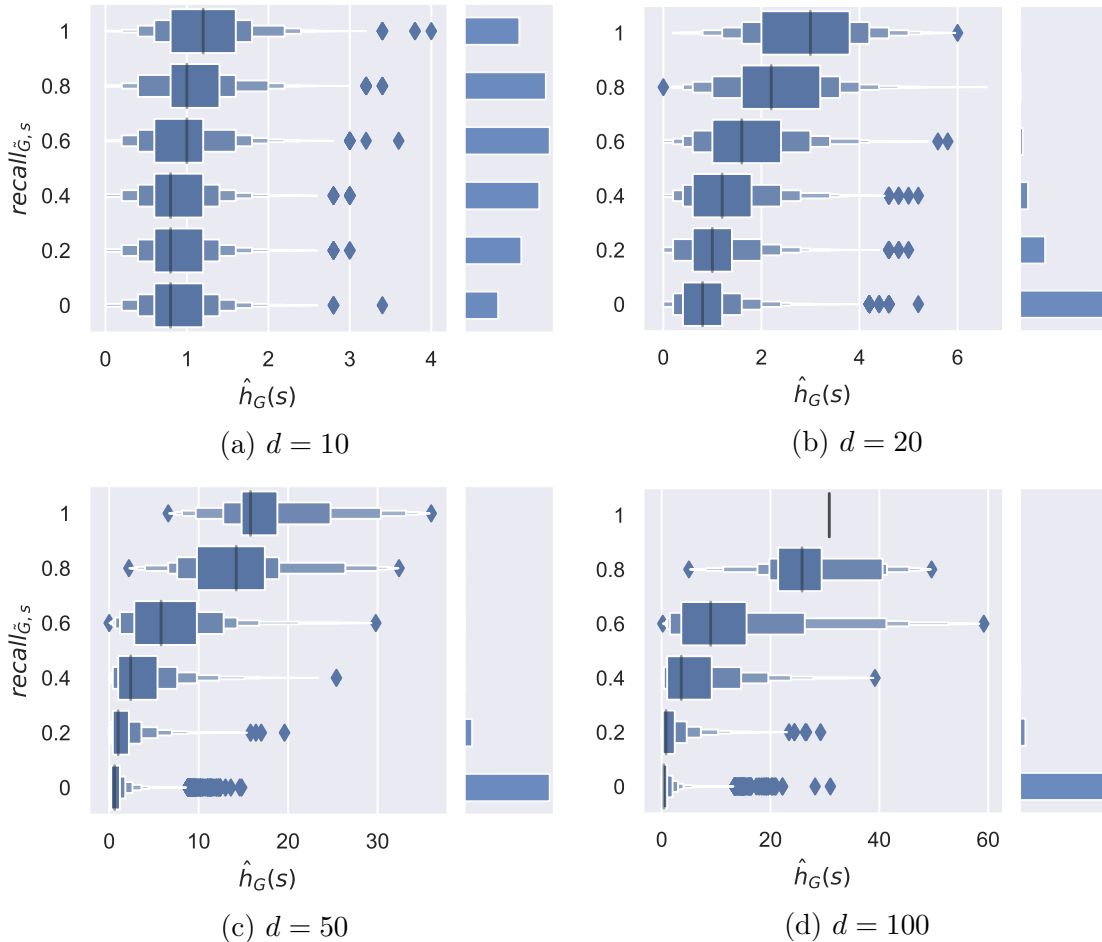


Figure 4.2: Distributions of points' hubness values for different recall values.

The box plots on the left-hand side of each subfigure show the distributions of normalized hubness values for different recall values, while the histograms on the right-hand side of each subfigure show the distribution of dataset points over different recall values.

To go further with the analysis, we investigated the correlation between recall and hubness values of individual dataset points. For this purpose we again used k -NNGs with $k = 5$ and L_2 distance, and their *NN-Descent* approximations, constructed on datasets introduced in Section 2.3. The choice of k is motivated by the fact that *NN-Descent* produces more accurate approximations for larger k values, so our aim was to choose k for which *NN-Descent* produces inaccurate approximations. The reason why the approximations produced by *NN-Descent* are more accurate for larger k is that the dataset points get compared with larger amount of other points since *NN-Descent* performs all pairwise comparisons within

CHAPTER 4. *NN-DESCENT* ON HIGH DIMENSIONAL DATA

individual neighborhoods of size k . Let us denote k -NNG of a dataset S_d by G_d , and its *NN-Descent* approximation by \tilde{G}_d . For each point s in the dataset S_d we extracted its hubness value from G_d and its recall from \tilde{G}_d . The distributions of these hubness values for each recall value are presented with the enhanced box plot [27] in Figure 4.2, while the average recall values for different hubness values are shown in Figure 4.3.

We will first explain Figure 4.2. Before further analysis, let us point out that the recall values on the subfigures' y axes are all the possible recall values for $k = 5$. In this figure, a bad performance of *NN-Descent* came again into focus in the histograms positioned on the right-hand side of each subfigure. Even for $d = 10$ (Subfigure 4.2a) *NN-Descent* is not performing well (which can also be verified in Figure 4.1), since there is non-negligible number of points with small recall values. For larger values of d the problem gets even worse, having that for $d = 20$ (Subfigure 4.2b) majority of the points have recalls that are less or equal to 0.4, while for the cases of $d = 50$ and $d = 100$ (Subfigures 4.2c and 4.2d) majority of points have recalls less than or equal to 0.2.

The distributions of normalized hubness values for different recall values are shown on the left-hand side of each subfigure. An evident phenomenon appeared in these distributions: as the recall values increase, hubness values increase as well. This phenomenon is only slightly visible for $d = 10$, but it becomes more evident for larger values of d . A very important conclusion can be inferred from this phenomenon: in high-dimensional datasets high-recall points are very probable to be hubs, i.e., the probability of error when determining the k nearest neighbors of a point s is inversely proportional to the hubness of s .

Figure 4.3 shows the mean (the solid lines) and the standard deviation (the transparent shades around the lines) of the recall values from the points sharing the same hubness value. The hubness values used in the figure are rounded to the nearest tenth and then normalized. The results for each dataset S_d are presented by a single line. The aforementioned phenomenon can be seen in this figure from a slightly different perspective. The average recall for a given hubness value is high if the hubness value is high itself. The standard deviations are mostly consistent and not very high. Exceptionally, for normalized hubness value of ~ 30 the standard deviation is considerably higher. A deeper insight into the underlying results showed that only two points had normalized hubness of ~ 30 , therefore, this high standard deviation does not depict results of large number of points, meaning that it is not statistically relevant. As a conclusion, once again it can be seen that high recall values are usually achieved by the points with high hubness values.

4.1. INFLUENCE OF HUBNESS ON *NN-DESCENT*

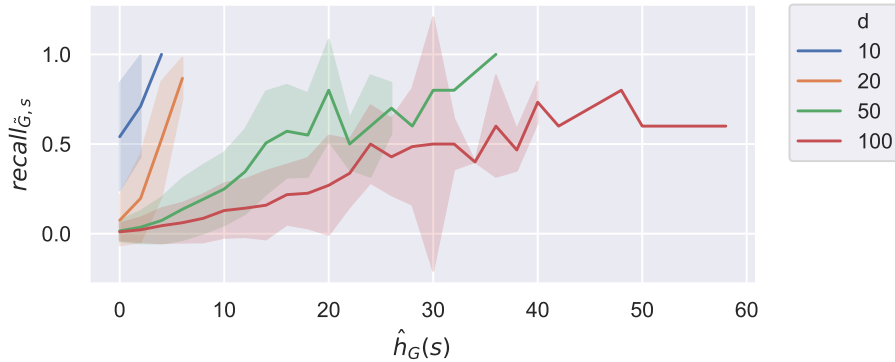


Figure 4.3: Average recall values for different hubness values.

The solid lines represent the mean of the recall values, while the transparent shades around the lines represent the standard deviation.

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 3.2, *NN-Descent* algorithm improves the k -NN graph approximation in each iteration. If the hubness of the dataset is high, then in the first few iterations hubs will be placed among the k nearest neighbors of a large number of data points¹. This implies that the NN lists of a majority of points will contain hubs. Points with lower hubness values will quickly be removed from NN lists of other points. This removal 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 R -NN lists of other data points. If a point is removed from NN lists early, then it will be updated only from R -NN lists of the points that are very unlikely to be its true neighbors, since the initial approximation of the k -NNG graph is essentially random. For a majority of points of low hubness value, this produces poor results.

Additionally, the hubness phenomenon refutes the basic *NN-Descent*'s assumption that two points sharing a neighbor are also likely to be neighbors. The reason for this is that the hubness phenomenon implies a significant increase of the hubs' reverse neighborhood size. Therefore, two points that are randomly chosen from such large neighborhood are not very likely to be neighbors. On the other side, the hubness phenomenon decreases reverse neighborhoods of other points that are not hubs. However, two points that are randomly chosen from such neighborhood

¹Experimental verification of this claim can be found in Section 5.1, Figure 5.2. It can be seen there that the hubness phenomenon in k -NNG approximation arose in the second iteration.

are also not very likely to be neighbors regardless of the small neighborhood size. The reason is that low-hubness points do not have accurate NN lists, as already explained, so the shared neighbor of the two chosen points is very unlikely to be their real neighbor.

4.2 Influence of initial random graph

As pointed out in the previous section, points with low recalls are the points with low hubness values. However, there are also certain amount of low-hubness points for which *NN-Descent* produces better approximations, especially for larger k values. In this section we will investigate if the low-hubness points with high recalls have some convenient properties, or they have high recalls out of pure randomness.

Let us first introduce the notion of a “good” *initial neighborhood*, which is a point’s neighborhood in the initial random graph that leads to the point’s high recall value in the final k -NNG approximation. It trivially holds that each point has at least one “good” initial neighborhood—it is the initial neighborhood consisted of the point’s true neighbors. However, the purpose of *NN-Descent* is to direct a point from incorrect neighborhood to the correct one, so it is expected that for a given point more than one “good” initial neighborhood exist. As number of “good” initial neighborhoods is higher, *NN-Descent* is more probable to be more accurate. Therefore, a point with a high number of “good” initial neighborhoods relies less on “luck”, having that its recall is in that case less dependent on the initial random graph.

In order to quantify the extent to which the points’ recall values depend on their position in initial random graph, we calculated recall values and their standard deviations for each data point over 100 runs of *NN-Descent*, each time generating an initial random graph with a different random seed. *NN-Descent*’s parameters were set in the following way: S_{50} dataset was used (see Section 2.3), for k we used values 5, 10 and 20, the distance was L_2 , $conv = 0.01$ and $\rho = 1$. The results are shown in Figure 4.4, where the intensity of the color represents the mean of the recall values among different runs—points with greater intensity have greater average recall values. Let us point out that Figure 4.4, among other things, demonstrates *NN-Descent*’s dependence on k (see Section 3.2)—as k gets higher, the points’ recall values get higher, too.

What can also be seen in Figure 4.4 is that for $k = 10$ and $k = 20$ points with low hubness values have a relatively high standard deviation of recall values, while

4.2. INFLUENCE OF INITIAL RANDOM GRAPH

points with high hubness values have standard deviations that tend toward zero. Having that recall values of low-hubness points vary significantly across different runs, while the only difference between the runs is the initial random graph, the following conclusion can be drawn: these points have sufficiently small number of “good” initial neighborhoods and are therefore more affected by the initial random graph. On the other side, points with the high hubness values have extremely high number of “good” initial neighborhoods—as explained in Section 4.1, wherever they are in the initial random graph, they find their way to their real neighborhoods. Because of that, the standard deviation for hubs is small, since they are constantly having high recall values.

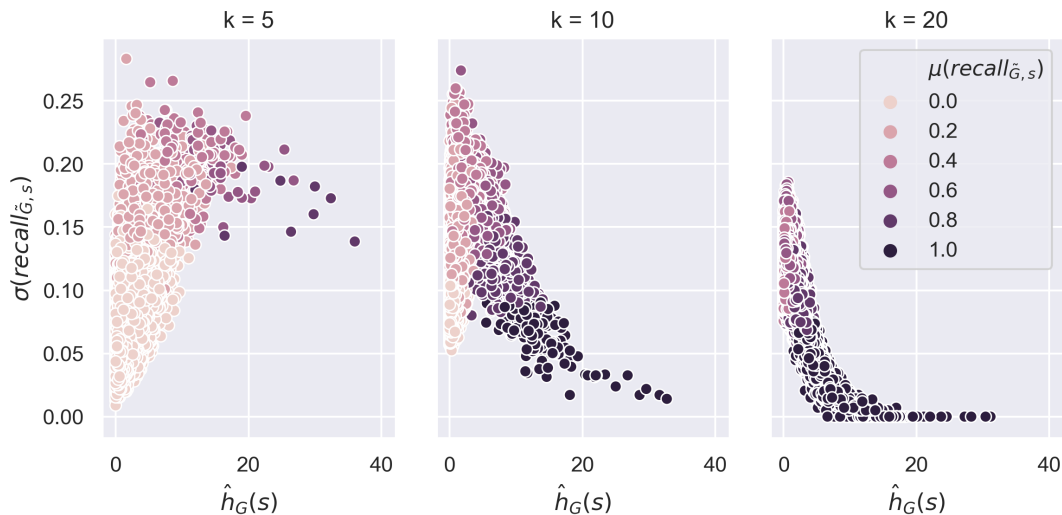


Figure 4.4: Dependence between points’ hubness values and standard deviations of their recall values over 100 runs of *NN-Descent*.

However, for $k = 5$, the aforementioned phenomenon cannot be seen—in this case some low-hubness points do not have high standard deviation, meaning that they do not depend much on the initial random graph. As already said, for lower k values *NN-Descent* produces lower recalls. Precisely, for $k = 5$ none of the low-hubness points reached recall higher than 0.6, while only a small amount of them reached recall higher than 0 (see Figure 4.2c). Consequently, for these points the number of “good” neighborhoods is low, so, by the laws of probability, even more *NN-Descent*’s runs are needed to reach the moment when a point is well positioned in the initial random graph. Thereby, low-hubness points with low standard deviations for $k = 5$ in Figure 4.4, are the ones that did not manage to be placed in the “good” neighborhoods over all 100 runs.

CHAPTER 4. *NN-DESCENT* ON HIGH DIMENSIONAL DATA

The reason behind the low recall value of a point that does not have a “good” initial neighborhood is the following: such point gets quickly removed from NN lists of other points, and stays confined in the R -NN lists of points that are not its true neighbors. Not being present in other points’ NN lists, the point participates in a very few local joins, and on top of that, since R -NN list is inadequate with high probability, these local joins are not improving the point’s NN list. In order to preserve the opportunity to find its nearest neighbors, a point must stay in other points’ NN lists. It is well-known that as a point’s distance to the dataset’s centroid decreases, the average distance from that point to the other dataset points decreases as well [48]. With decreased distances to other dataset points, the probability that a point stays in other points’ NN lists increases. Thereby, low-hubness points that have lower distance to the dataset mean are ones that have a slightly higher number of “good” initial neighborhoods, and hence a slightly higher standard deviation of recalls obtained from multiple *NN-Descent*’s runs.

Figure 4.5 supports the previous statement. In the figure we present the distribution of standard deviations over the points’ distances to the dataset mean. Points presented in the figure are only the ones with $h_G(s) \leq k$. A high color intensity represents a high points density. The settings we used here are exactly the same as the settings we used in Figure 4.4. Finally, it can be seen in the figure that points with lower distances tend to achieve higher standard deviation values.

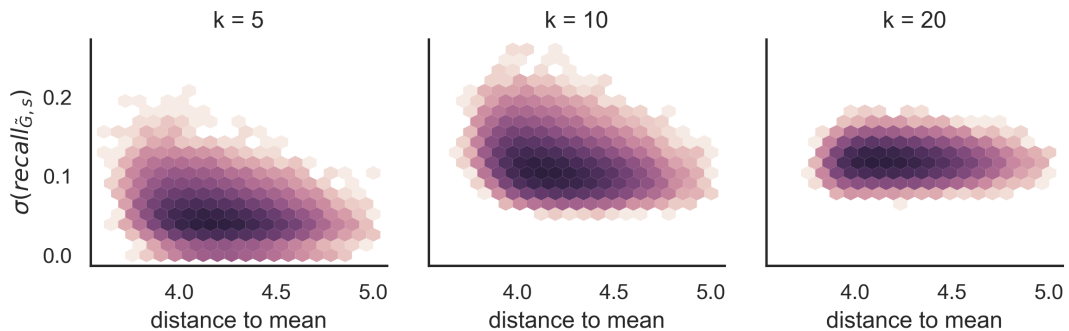


Figure 4.5: Dependence between points’ distance to dataset mean and standard deviations of their recall values over 100 runs of *NN-Descent*.

The color intensity of the hexagons represents points density—low intensity represents low points density, while high intensity represents high density.

4.2. INFLUENCE OF INITIAL RANDOM GRAPH

Chapter 5

Proposed methods for improving *NN-Descent*

As explained in Chapter 4, *NN-Descent* is highly influenced by the hubness phenomenon. In order to mitigate the problem, we implemented and verified five different modifications of *NN-Descent* [8, 9]. The first one is based on the idea that NN lists should not be updated strictly according to NN and *R*-NN lists of other points. The goal is to integrate the information about hubness values into the choice of points that participate in local joins. 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 *NN-Descent* algorithm with a greater k value, and then reduce the resulting k -NNG to the k we actually need, the precision of the final graph is expected to be better. The third and the fourth approaches provide 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 fifth approach is built up on the fact that a point’s position in the initial random graph influences its recall value. In order to place the point in the right neighborhood, we perform additional random comparisons to the points that need it.

In Section 5.1 we introduce a notion of *hubness approximation* that is later used by some of the aforementioned algorithms. In sections 5.2, 5.3, 5.4, 5.5 and 5.6 we introduce all *NN-Descent*’s modifications. Finally, Section 5.7 evaluates all the modifications, comparing them with the original *NN-Descent*.

5.1 Hubness approximation using *NN-Descent*

Let us say that the *approximate hubness value* of a point s is $h_{\tilde{G}}(s)$, where \tilde{G} is an arbitrary k -NNG approximation, that is, the approximate hubness of a point is its hubness value in some k -NNG approximation. Approximate hubness values for each data point could be very easily maintained during an execution of *NN-Descent*, with minimal impact on the algorithm performance. At the very beginning, we initialize the hubness value of each dataset point to zero. Then, during the 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. In this way, approximate hubness values for all points are available in a constant time during the algorithm execution.

Figure 5.1 shows the correlation between real hubness values and approximate hubness values extracted from an *NN-Descent* approximation. In the figure we show results for S_{50} dataset (see Section 2.3)—the hubness phenomenon is present in the chosen dataset. The rest of the settings were as following: distance was L_2 , $conv = 0.01$ and $\rho = 1$. As can be seen, the correlation between real and approximate hubness values is very high. A shortcoming of *NN-Descent* is that approximate hubness values are lower than they should be for points with low real hubness values, and greater than they should be for points with high real hubness. The precision improves as k increases, but even for small k values, the algorithm produces strong correlations. These strong correlations are very useful, since they tend to preserve hubness-defined ordering of the points.

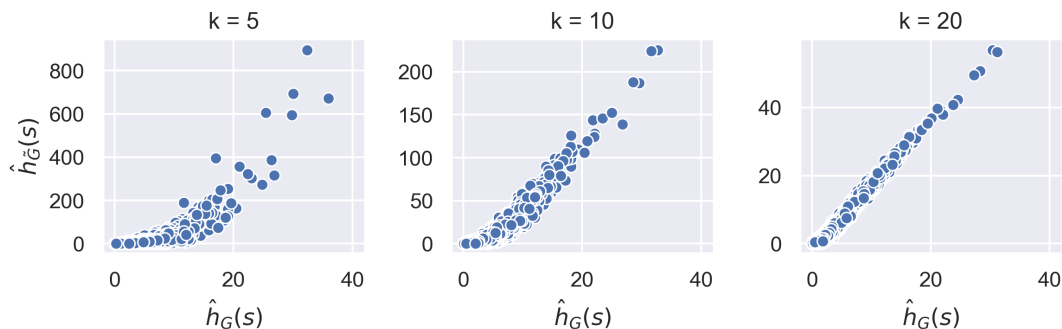


Figure 5.1: Correlation between approximate and real hubness values of dataset points, for different k values.

Each dot in the scatter plot represents a single data point.

CHAPTER 5. PROPOSED METHODS FOR IMPROVING
NN-DESCENT

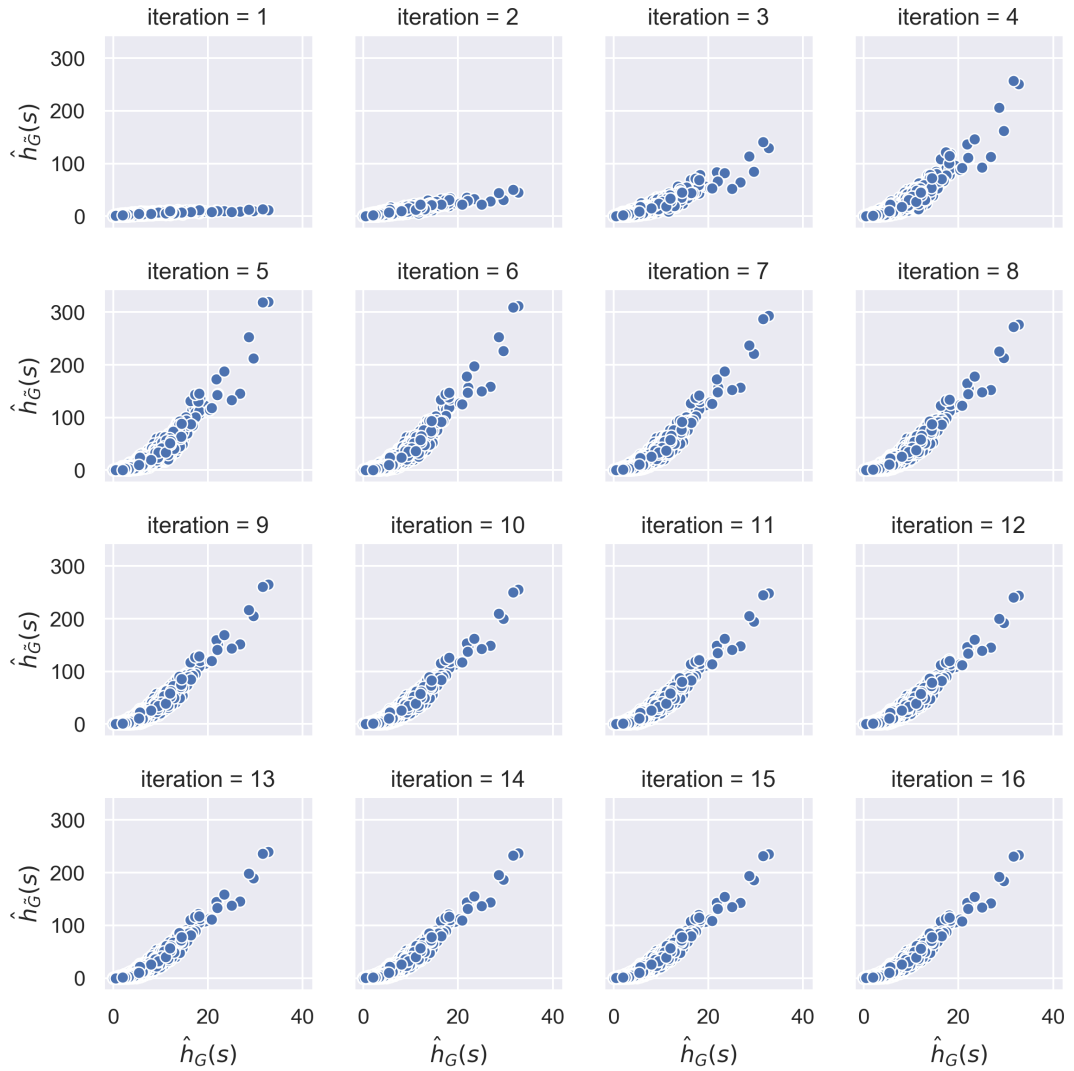


Figure 5.2: Correlation between approximate and real hubness values of dataset points, recorded after each iteration of *NN-Descent* algorithm.

Each dot in the scatter plot represents a single data point.

After the aforementioned small upgrade of *NN-Descent* algorithm, approximate hubness values are available during the whole algorithm execution. Figure 5.1 shows that the approximate hubness values extracted from an *NN-Descent* approximation are very correlated to the real hubness values. However, we are more interested in knowing if the approximate hubness values are having the same property during the algorithm execution (not after the algorithm has terminated). Therefore, correlations of approximate hubness values and real hubness values after each iteration of *NN-Descent* are presented in Figure 5.2. In the figure we presented the results for dataset S_{50} and neighborhood size $k = 10$, and we used the

5.2. HUBNESS-AWARE VARIANT

same *NN-Descent*'s parameters as in Figure 5.1. As can be seen, high correlation appeared after very few *NN-Descent*'s iterations. As the algorithm progressed, it started to overestimate hubness values of hubs. Nevertheless, the correlation is very high during the whole execution of *NN-Descent*.

There is one additional conclusion that can be inferred from the presented results. Since hubs have overestimated hubness values, it means that they are placed in more neighborhoods than needed. Consequently, hubs are actually “taking” other points’ reverse neighbors.

5.2 Hubness-aware variant

In this section, we will present an *NN-Descent* modification that uses hubness values to guide the choice of candidate points for inclusion in the NN list of a given data point. In further text we will refer to this *NN-Descent* modification as *hubness-aware NN-Descent variant (HA-NN-Descent)*. After implementing the simple algorithm for approximate hubness values described in Section 5.1, up-to-date hubness approximations become available at any given iteration of the algorithm execution. After each iteration, these values become more accurate; they are reasonably precise even after very first few iterations (see Figure 5.2).

Let us now describe in detail how this strategy works. In each iteration, we check whether a given point has hubs in its NN and *R*-NN lists. Let NN_{ij} and RNN_{ij} be lists used for local joins from a given point, initialized with elements from the point’s NN and *R*-NN lists, respectively, and let h_t be a threshold on the number of hub points to be considered from NN_{ij} and RNN_{ij} lists. The idea is to replace h_t points of high hubness value in NN_{ij} and RNN_{ij} lists with h_t new points chosen at random. The intuition behind this modification of *NN-Descent* is to diminish the impact of hubs on updates 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_t points are chosen. One way would be to fix h_t to some value and then to choose h_t points with the highest hubness values from each point’s NN_{ij} and RNN_{ij} lists. However, this approach is not flexible with regards to variability of neighborhood structures. Namely, it can happen that certain NN and *R*-NN lists (and hence NN_{ij} and RNN_{ij} lists initialized with them) do not have hubs at all, and therefore none of their points should be chosen by the process, while other NN and *R*-NN lists might have hubs that should be replaced. In order to alleviate this problem, for each point

CHAPTER 5. PROPOSED METHODS FOR IMPROVING NN-DESCENT

Table 5.1: List of parameters of hubness aware variant.

Parameter	Description
k	Neighborhood size (size of NN lists).
$dist$	Distance function.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (value between 0 and 1). Algorithm converges when there is less than $conv \cdot k \cdot n$ updates in the most recent iteration, where n is the number of k -NNG vertices. Note: present only for <i>convergence</i> termination condition.
ρ	Sampling (value between 0 and 1, not including 0). Only $\rho \cdot k$ points from NN and R -NN lists are considered for local joins.
h_{\min}	Minimum hubness value for replacement probability (see (5.1)).
h_{\max}	Maximum hubness value for replacement probability (see (5.1)).

in NN and R -NN lists we introduce a probability of choosing it for replacement, and we let its hubness value to determine that probability. To come up with a valid probability, we employed a transformation of raw hubness values into the interval $[0, 1]$. For the purpose of the transformation, we introduce values h_{\min} and h_{\max} . Hubness values within the range $[h_{\min}, h_{\max}]$ are linearly transformed to the probability range $[0, 1]$, while the values less than h_{\min} and values greater than h_{\max} , are transformed to the probabilities 0 and 1, respectively. The complete equation of the probability for a point to be selected for replacement is shown in (5.1). In the equation, s denotes a data point and $h_{\tilde{G}}(s)$ is its approximate hubness value in the current k -NNG approximation \tilde{G} . Finally, as can be seen, by introducing the probabilities on the point level, we managed to achieve adaptation to all neighborhood structures. Therefore, in this approach h_t is not fixed, but instead is determined by the number of chosen neighboring points (choice being led by points' probabilities).

$$\Pr[\text{replace}_s] = \begin{cases} 0, & \text{if } h_{\tilde{G}}(s) < h_{\min}, \\ 1, & \text{if } h_{\tilde{G}}(s) > h_{\max}, \\ \frac{h_{\tilde{G}}(s) - h_{\min}}{h_{\max} - h_{\min}}, & \text{otherwise.} \end{cases} \quad (5.1)$$

The outline of hubness aware *NN-Descent* variant is presented in Algorithm 4, while the list of its parameters is given in Table 5.1. The first four parameters are the same as in *NN-Descent*, while the last two parameters configure minimum and maximum hubness values that are used for the replacement probability given in

5.2. HUBNESS-AWARE VARIANT

Algorithm 4: Outline of hubness aware variant.

input : set of points S , distance function $dist$, neighborhood size k ,
sampling ρ
output: k -NNG approximation \tilde{G}
// Note: Functions $Sample$ and $RandKNNG$ are defined in Algorithms 1 and 2,
respectively.

- 1 $\tilde{G} \leftarrow RandKNNG(S, k, dist)$;
- 2 **repeat**
- 3 **foreach** data point $s \in S$ **do**
- 4 $NN_{ij} \leftarrow NN_{\tilde{G}}(s)$; $RNN_{ij} \leftarrow RNN_{\tilde{G}}(s)$;
- 5 $NN_{hubs} \leftarrow$ all points from NN_{ij} which are chosen according to their
replacement probability (see (5.1));
- 6 $RNN_{hubs} \leftarrow$ all points from RNN_{ij} which are chosen according to
their replacement probability (see (5.1));
- 7 $NN_{ij} \leftarrow (NN_{ij} \setminus NN_{hubs}) \cup Sample(S \setminus NN_{ij}, |NN_{hubs}|)$;
- 8 $RNN_{ij} \leftarrow (RNN_{ij} \setminus RNN_{hubs}) \cup Sample(S \setminus RNN_{ij}, |RNN_{hubs}|)$;
- 9 $R \leftarrow Sample(NN_{ij}, \rho \cdot k) \cup Sample(RNN_{ij}, \rho \cdot k)$;
- 10 **foreach** two points $v, w \in R$ **do**
- 11 $d \leftarrow dist(v, w)$;
- 12 Use $\langle v, d \rangle$ to update w 's NN list in \tilde{G} , and use $\langle w, d \rangle$ to update
 v 's NN list in \tilde{G} ;
- 13 **end**
- 14 **end**
- 15 **until** Termination condition is met
- 16 **return** \tilde{G} .

(5.1). These two parameters are advised to be configured with respect to k value, since hubness values are directly influenced by k .

A single point s can appear in multiple NN and in exactly k R -NN lists of a graph \tilde{G} . For a point with the replacement probability different from 0 and 1, it can happen that it gets replaced within one list, but within the other it does not. The expected number of a point's replacements is given in (5.2). As a consequence of the replacement probability (see (5.1)), the higher hubness value, the higher is the expected number of replacements, which is exactly what we wanted to achieve.

$$E[\text{replacements}_s] = k \cdot |RNN_{\tilde{G}}(s)| \cdot \Pr[\text{replace}_s] \quad (5.2)$$

5.3 Oversized NN list variant

As already explained in Section 3.2, accuracy of *NN-Descent* increases with k . In Figure 5.3 we demonstrate this *NN-Descent*'s behavior by running the algorithm for different k values. The datasets we used for this demonstration were S_{10}, S_{20}, S_{50} and S_{100} (see Section 2.3), distance was L_2 , while *NN-Descent* parameters were $conv = 0.01$ and $\rho = 1$. The figure clearly shows that the algorithm's accuracy increases as k increases.

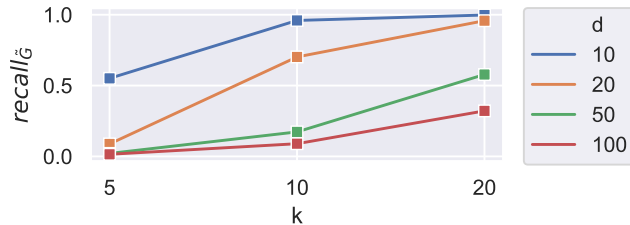


Figure 5.3: Recall values of *NN-Descent* k -NNG approximations for different k values.

Therefore, in order to achieve higher accuracy, *NN-Descent* could be run with some larger choice of neighborhood size $k' > k$, followed by a truncation of the NN lists to the target size k . During the truncation, only the k nearest points would be preserved. We will refer to the process of NN list truncation within a graph as *graph reduction*. However, there are two potential issues with the described approach. In further text we will present both of them.

The first issue is that with an increase of k value, besides the accuracy, execution time increases as well (see Section 3.2). This issue, as the authors of *NN-Descent* suggested, can be addressed with the sampling technique. If we expand the neighborhood from k to k' , for $k' > k$, the most convenient way to set the sampling is to use $\rho = \frac{k}{k'}$. In that way, larger neighborhoods are preserved during the algorithm execution (due to enlarged k value), while the number of local joins per iteration stays the same. It might seem that this configuration does not slow down *NN-Descent* at all, but that is not always the case. Namely, for the *fixed iterations* termination condition, the algorithm indeed does not slow down, however, for the convergence termination condition, this setting, due to enlarged neighborhoods, introduces slower *NN-Descent*'s convergence, and hence higher scan rate values. Nevertheless, this setting is, regardless of slower convergence, still faster than the one without sampling.

5.3. OVERSIZED NN LIST VARIANT

The second potential issue is that even though a graph built for the neighborhood size k' , for $k' > k$, has better recall than a graph built for neighborhood size k , it does not mean that a graph that is reduced from the one built for k' preserves the high recall. Differently said, the recall of the reduced graph might not be as high as the recall it had before the reduction; moreover it could be so much lower, that it does not beat the results of the original *NN-Descent*. In order to prove that this issue can appear, we will first give an insight into some relations between a reduced graph and the graph it was reduced from. Let G_k and $G_{k'}$ be real k -NNGs for neighborhood sizes k and k' , respectively, both of them created on the same dataset; \tilde{G}_k and $\tilde{G}_{k'}$ their arbitrary approximations; and let \tilde{G}_k^R be a graph reduced from $\tilde{G}_{k'}$ to the neighborhood size k . Let us also assume that all NN lists within the graphs are sorted by ascending order of distances. If, for some point s , the i^{th} element of $NN_{G_k}(s)$ is at j^{th} position in $NN_{\tilde{G}_{k'}}(s)$, then $j \leq i$. This trivially holds since there are $i - 1$ points closer to s than its i^{th} real neighbor—therefore, in s 's approximate NN list of any size, there can be at most $i - 1$ points in front of s 's i^{th} real neighbor. This leads us to a conclusion that s 's i^{th} real neighbor, for any $i \in [1, k]$, is in $NN_{\tilde{G}_k^R}(s)$ if and only if it is in $NN_{\tilde{G}_{k'}}(s)$.

As the previous conclusion implies, if s 's i^{th} real neighbor for $i \in [1, k]$ is in $NN_{\tilde{G}_{k'}}(s)$, it is in $NN_{\tilde{G}_k^R}(s)$, too. Therefore, in order for the recall of the reduced graph \tilde{G}_k^R to be at least as high as the recall of $\tilde{G}_{k'}$, it is necessary that the total number of points from $NN_{G_k}(s)$ that are also in $NN_{\tilde{G}_{k'}}(s)$ (where s runs through all the data points) is equal to or greater than $\text{recall}_{\tilde{G}_{k'}} \cdot k \cdot n$. Since this does not necessarily hold, we proved that $\text{recall}_{\tilde{G}_k^R}$ is not guaranteed to be as high as $\text{recall}_{\tilde{G}_{k'}}$. Moreover, whether $\text{recall}_{\tilde{G}_k^R}$ will be higher or lower than $\text{recall}_{\tilde{G}_{k'}}$ depends on the structure of NN lists in $\tilde{G}_{k'}$ —if they contain more i^{th} real neighbors for $i \leq k$, the recall of the reduced graph will be higher, and vice versa.

Figure 5.4 visualizes how $\text{recall}_{\tilde{G}_k^R}$ and $\text{recall}_{\tilde{G}_{k'}}$ can relate differently. The example is based on a graph reduction from $k' = 10$ to $k = 4$. Neighborhoods are visualized with arrays of squares, each square representing a single neighbor. It is assumed that neighbors are given in the increasing order of distances. Additionally, the thick vertical lines separate the target neighborhoods (of size $k = 4$) from the rest of the extended neighborhoods (of size $k' = 10$). First 4 nearest neighbors are presented with light green color, the next 6 with dark green color, while the misses in the approximated NN lists are presented with red color. The value $\text{recall}_{\tilde{G}_k^R}$ is then determined by the number of light green squares placed on the left of the thick vertical lines. Similarly, the value $\text{recall}_{\tilde{G}_{k'}}$ is determined by the total number of green squares (regardless of the color shade). The value $\text{recall}_{\tilde{G}_{k'}}$ is the

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

same in all the subfigures, amounting to 0.5. Contrary to that, the value $\text{recall}_{\tilde{G}_k^R}$ varies among the different subfigures, demonstrating how the two values can relate differently.

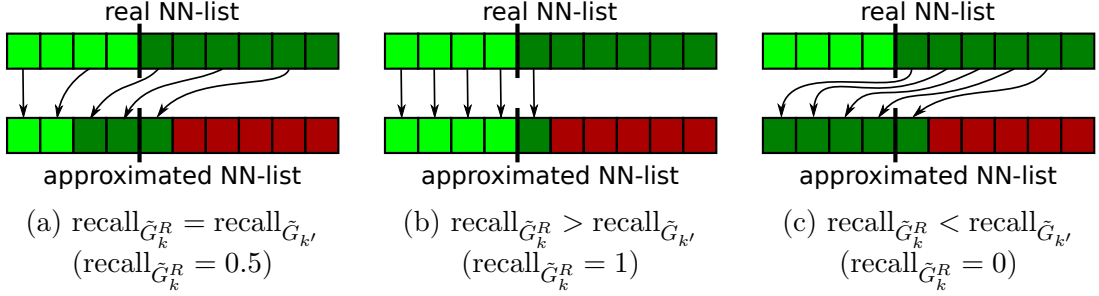


Figure 5.4: Different relations between $\text{recall}_{\tilde{G}_k^R}$ and $\text{recall}_{\tilde{G}_{k'}^R}$ for $k = 4$, $k' = 10$ and $\text{recall}_{\tilde{G}_{k'}^R} = 0.5$.

Now that we showed that the issue holds for a general case, we will conduct experiments in order to determine if the same holds for *NN-Descent*. Figure 5.5 presents recalls of graph reductions to all $k \leq k'$, k' being the neighborhood size of a k -NNG approximation created by *NN-Descent* algorithm. In the experiments we used S_{50} dataset (see Section 2.3), L_2 distance, and *NN-Descent* parameters $\text{conv} = 0.01$ and $\rho = 1$. Finally, let us point out that the case when $k = k'$ means that the graph was not reduced.

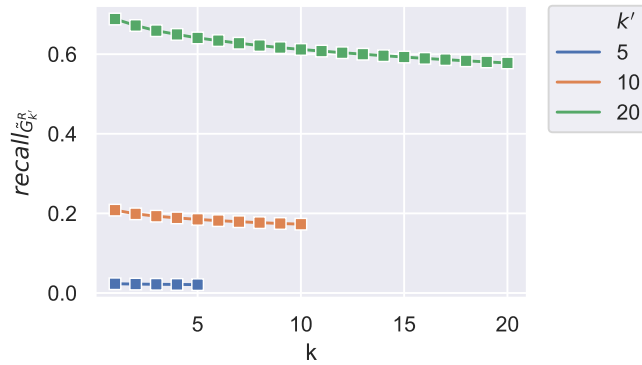


Figure 5.5: Recall values of graphs reduced from *NN-Descent* approximations $\tilde{G}_{k'}$ to neighborhood sizes $k \in [1, k']$.

The figure clearly shows that the recalls of the reduced graphs are not lower than the recalls of the graphs they were reduced from. Moreover, the recalls of the reduced graphs are even higher—as k decreases, the recall value increases. These empirical findings suggest that *NN-Descent* variant presented in this section can definitely increase accuracy of *NN-Descent*.

5.4. RANDOM WALK DESCENT VARIANT

In further text we will refer to this *NN-Descent* variant, which might as well be considered as a special way of choosing *NN-Descent*'s parameters, as *oversized NN list variant* (*O-NN-Descent*). The list of parameters for oversized NN list variant is presented in Table 5.2, while the algorithm itself is presented in Algorithm 5.

Table 5.2: List of parameters of oversized NN list variant.

Parameter	Description
k	Neighborhood size of the final k -NNG approximation, i.e., neighborhood size to which <i>NN-Descent</i> approximation will be reduced.
k'	Neighborhood size for which <i>NN-Descent</i> will be run.
$dist$	Distance function.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (value between 0 and 1). Algorithm converges when there is less than $conv \cdot k \cdot n$ updates in the most recent iteration, where n is the number of k -NNG vertices. Note: present only for <i>convergence</i> termination condition.
ρ	Sampling (value between 0 and 1, not including 0). Only $\rho \cdot k'$ points from NN and R -NN lists are considered for local joins.

Algorithm 5: Outline of oversized NN list variant.

input : set of points S , distance function $dist$, neighborhood sizes k and k' , sampling ρ
output: k -NNG approximation \tilde{G}

- 1 $\tilde{G} \leftarrow NN\text{-Descent}(S, dist, k', \rho)$;
- 2 **foreach** data point $s \in S$ **do**
- 3 Update $NN_{\tilde{G}}(s)$ by preserving k points that are nearest to s and discarding the rest;
- 4 **end**
- 5 **return** \tilde{G} .

5.4 Random walk descent variant

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

CHAPTER 5. PROPOSED METHODS FOR IMPROVING NN-DESCENT

Table 5.3: List of parameters of *RW-Descent*.

Parameter	Description
k	Neighborhood size (size of NN lists).
$dist$	Distance function.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (a value from range $(0, 1]$). A point converges when there is in average less than $conv \cdot b(s)$ updates in the hd most recent iterations. The algorithm converges when all points converge. Note: present only for <i>convergence</i> termination condition.
hd	History depth (a value from range $[0, \infty)$)—see parameter $conv$ for more information. Note: present only for <i>convergence</i> termination condition.
$b(s)$	Balancing function that returns the number of local joins for a point s .

The algorithm, *random walk descent* (*RW-Descent*), employs a random walk strategy for determining candidates for improvement. Like *NN-Descent*, *RW-Descent* constructs an initial k -NN graph by means of random selection. Thereafter, the algorithm iterates either a predetermined number of times (i.e., *fixed iterations* termination condition), or until a convergence criterion is satisfied (i.e., *convergence* termination condition). In each iteration, each data point is allocated a number of local joins 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 local joins that it will initiate. The weighting strategy is then incorporated in the balancing function $b(s) : S \rightarrow \mathbb{N}^+$, which returns the number of local joins for a data point s . After allocating a number of local joins $c = b(s)$ to a given data point s , the points to be compared with are the stopping points of c short random walks that start from s . The main objective while designing a balancing function, is to find a way to determine which points have incorrect neighborhoods; these points should be assigned with larger number of random walks. The random walks are applied on the current k -NNG's underlying simple graph; that is, each successive point in a 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 s 's neighbors, and thus *RW-Descent* would perform essentially as

5.5. NEAREST WALK DESCENT VARIANT

Algorithm 6: Outline of *RW-Descent* algorithm

input : set of points S , distance function $dist$, neighborhood size k ,
balancing function $b(s)$
output: k -NNG approximation \tilde{G}

// Note: Functions *Sample* and *RandKNNNG* are defined in Algorithms 1 and 2,
respectively.

- 1 $\tilde{G} \leftarrow \text{RandKNNNG}(S, k, dist)$;
- 2 **repeat**
- 3 **foreach** data point $s \in S$ **do**
- 4 $c \leftarrow b(s)$;
- 5 **for** $i = 1$ to c **do**
- 6 $w \leftarrow s$;
- 7 **for** $j = 1$ to 2 **do**
- 8 $w \leftarrow \text{Sample}(NN_{\tilde{G}}(w) \cup RNN_{\tilde{G}}(w) \setminus \{s\}, 1)$;
- 9 **end**
- 10 Use $\langle s, dist(s, w) \rangle$ to update w 's NN list, and use $\langle w, dist(s, w) \rangle$
to update s 's NN list;
- 11 **end**
- 12 **end**
- 13 **until** Termination condition is met
- 14 **return** \tilde{G} .

NN-Descent. In this thesis we will use only random walks of length 2, the reason being locality preservation—in the walks of length 2 the middle node is a shared neighbor of the starting and ending node.

If the termination condition is chosen to be convergence, the parameters $conv$ from range $[0, 1)$ and hd (*history depth*) from range $[0, \infty)$, determine when the algorithm terminates. Let us denote by $updates_i(s)$ the number of random walks that resulted with an update of point s 's NN list in the iteration i . A point converges in the iteration i_c if the average of $updates_i(s)$ values for $i \in [\max(0, i_c - hd), i_c]$ is less than $conv \cdot b(s)$. When all the points converge, the algorithm terminates. For a pseudocode description of *RW-Descent*, see Algorithm 6, while the list of the algorithm's parameters is given in Table 5.3.

5.5 Nearest walk descent variant

The idea of this *NN-Descent* variant, that we will call *nearest walk descent* (*NW-Descent*), is to improve previously introduced *RW-Descent* in such a way that the walks are not performed completely at random, but they are influenced

CHAPTER 5. PROPOSED METHODS FOR IMPROVING NN-DESCENT

by certain observations that we will introduce in this section. The observations hold only for Euclidean (L_2) distance, meaning that this method might not produce good results for other metrics.

This method has exactly the same basis as *RW-Descent*, the only difference being the way walks are performed. In *RW-Descent*, $b(s)$ random walks are used for local joins in which a point s participates. However, *NW-Descent* does not use random walks for that purpose, but it uses “best” walks instead. Namely, all possible walks of length 2 from a point s that do not end in s itself or in one of s ’s neighboring points, are assigned with a weight. After that, $b(s)$ walks with highest weights are used for local joins in the same manner as in *RW-Descent*. The weight of a walk is actually the probability that the walk’s ending point gets inserted in s ’s NN list.

Under the assumption that the distance function used for k -NNG construction is L_2 distance and that the dataset points are drawn from the uniform distribution, we will define how the mentioned probability is calculated. Let us say that $\langle s, s', s'' \rangle$ is a walk that starts in s , passes through s ’s neighbor s' , and ends in s' ’s neighbor s'' . Let $x = \text{dist}(s, s')$, $y = \text{dist}(s', s'')$, and $r = \text{dist}(s, s_k)$ where s_k is s ’s k^{th} nearest neighbor in the current k -NNG approximation (i.e. s ’s furthest neighbor). Moreover, we will define two hyperspheres H_s and $H_{s'}$. The center of hypersphere H_s is s , while its radius is r , meaning that all the points from s ’s current NN list are inside H_s . The center of hypersphere $H_{s'}$ is s' , while its radius is y , meaning that the point s'' lies somewhere on the hypersphere $H_{s'}$. The two hyperspheres can be positioned (relatively to each other) in four different ways: 1) $H_{s'}$ is contained in the interior of H_s ; 2) H_s is contained in the interior of $H_{s'}$; 3) H_s and $H_{s'}$ have disjoint interiors; 4) H_s and $H_{s'}$ have a nonempty intersection. More formally, the first case is defined by the inequality $x + y < r$, the second case by $x + r < y$, the third case by $y + r \leq x$, and, finally, if none of the inequalities hold, the fourth case is assumed. These cases are illustrated in the two-dimensional space in Figure 5.6. In the further text we will define how the probability is calculated for each of these four cases.

In the first case (Figure 5.6a), the probability of inserting s'' in s ’s NN list is 1. Having that the point s'' lies on $H_{s'}$, and that the whole $H_{s'}$ is in H_s ’s interior, s'' is in H_s ’s interior as well. This implies that s'' is nearer to s than s ’s furthest neighbor is, which means that s'' will certainly be inserted in s ’s NN list. Hence, the probability is 1, as already said.

For the second and the third case (Figures 5.6b and 5.6c), the probability is 0, because $H_{s'}$ is disjoint from H_s ’s interior, and therefore s'' , lying on $H_{s'}$, can not

5.5. NEAREST WALK DESCENT VARIANT

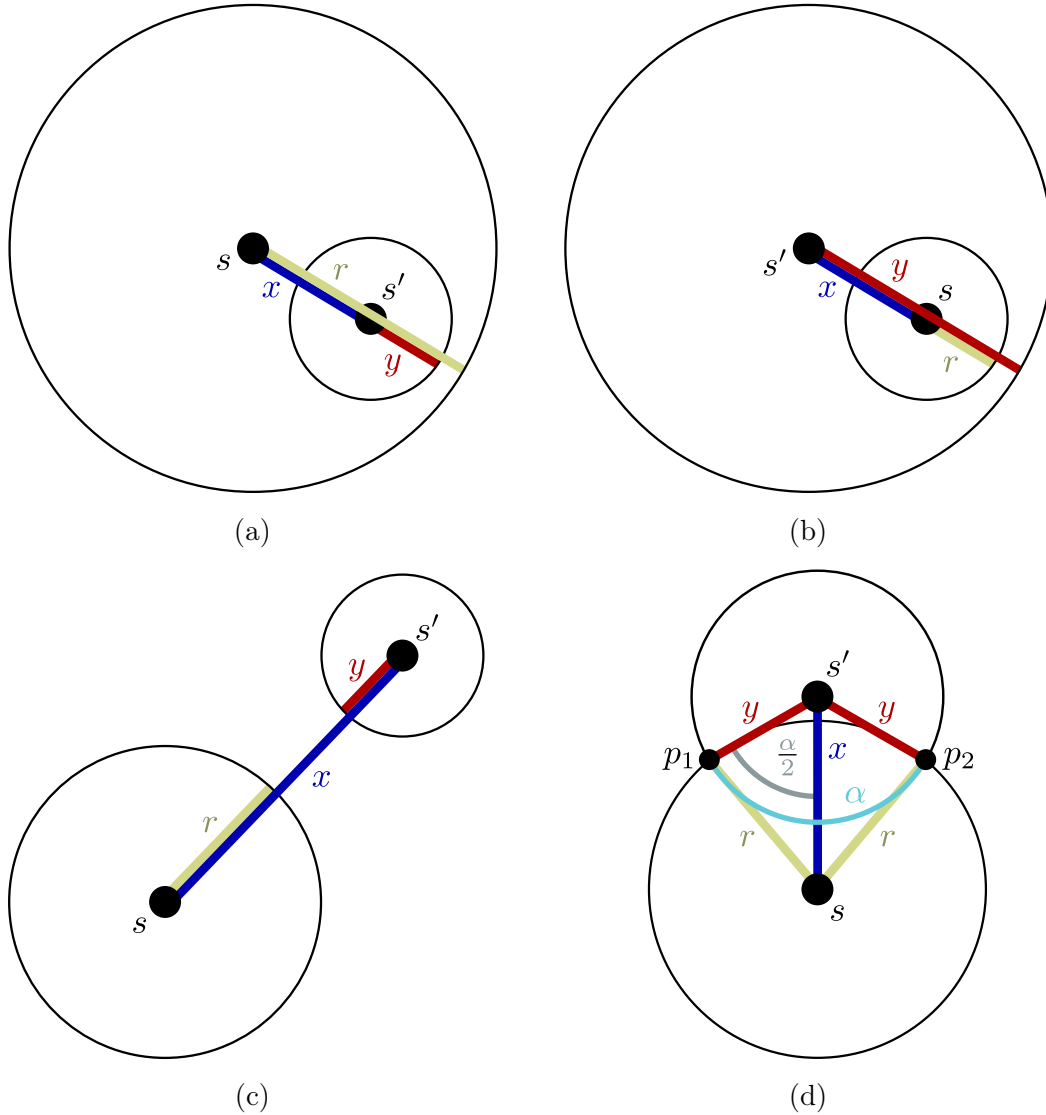


Figure 5.6: Possible relative positions of the points that participate in a walk. The relative position determines the probability that the walk leads to an update of its starting point's NN list.

be in H_s 's interior as well. This means that s'' is further from s than s 's furthest neighbor is. Hence, the considered probability is 0.

Finally, the probability in the fourth case (Figure 5.6d) is not as trivial as in the previous cases. The point s'' will be inserted in s 's NN list only if it lies on the part of $H_{s'}$ that is in H_s 's interior (this part is marked with light blue in Figure 5.6d). The probability is therefore equal to the ratio between the size of the mentioned $H_{s'}$'s portion and the size of the whole $H_{s'}$. Let P be any plane that contains both s and s' , and let p_1 and p_2 be the points of intersection of P ,

**CHAPTER 5. PROPOSED METHODS FOR IMPROVING
NN-DESCENT**

H_s and $H_{s'}$ (in a degenerate case, p_1 and p_2 may coincide). The mentioned ratio is then equal to $\frac{\alpha}{2\pi}$, where $\alpha = \angle p_1 s' p_2$. Additionally, α can be trivially calculated from the triangle $\triangle s s' p_1$, whose all sides are known (see (5.3)). We conclude that the probability in this case is $\frac{\alpha}{2\pi}$.

$$\alpha = 2 \arccos \frac{x^2 + y^2 - r^2}{2xy} \quad (5.3)$$

Finally, (5.4) summarizes calculation of probability that s'' gets inserted in s 's NN list.

$$\Pr[\text{dist}(s, s'') < r] = \begin{cases} 1, & \text{if } x + y < r; \\ 0, & \text{if } x + r < y; \\ 0, & \text{if } y + r \leq x; \\ \frac{\alpha}{2\pi}, & \text{otherwise.} \end{cases} \quad (5.4)$$

Now that the probabilities are known, for each point s , in each iteration, the algorithm simply chooses $b(s)$ walks whose ending points have the highest probabilities of being inserted in s 's NN list. However, this approach has a major problem of repeated distance calculations for certain pairs of points. The problem occurs when the ending point of a walk starting from a point s has high probability, but turns out not to trigger an update of s 's NN list. In that case, due to its high probability, this point will get chosen for a local join over and over again throughout different iterations, causing unnecessary distance calculations. Trivial solution to this problem would be simply to cache results of all distance calculations, but that would significantly increase the space complexity of the algorithm. Therefore, we introduce different solution to this problem, which preserves linear space complexity.

For each point s we store minimum of probabilities of all the $b(s)$ neighbors' neighbors that were chosen for local joins in the most recent iteration. We will denote it by pr_s . Moreover, for each point s we memorize all the points that, in the most recent iteration, got inserted in s 's NN or R -NN lists. We will refer to such points as *new neighbors*. Let $\langle s, s', s'' \rangle$ be a walk that starts from s , and let us assume that s' is not s 's new neighbor, and s'' is not s' 's new neighbor. Now, if s'' 's probability is greater than pr_s , that means that this walk has already been chosen before, and therefore should be immediately discarded. However, if either s' is a new neighbor of s , or s'' is a new neighbor of s' , the walk might not have been chosen before, regardless of its probability, and therefore, should not be discarded.

5.5. NEAREST WALK DESCENT VARIANT

This approach significantly reduces the number of repeated distance calculations without increasing asymptotic space complexity.

Algorithm 7: Outline of *NW-Descent* algorithm

```

input : set of points  $S$ , distance function  $dist$ , neighborhood size  $k$ ,
         balancing function  $b(s)$ 
output:  $k$ -NNG approximation  $\tilde{G}$ 

// Note: Function RandKNNNG is defined in Algorithm 2.
1  $\tilde{G} \leftarrow \text{RandKNNNG}(S, k, dist)$ ;
2 repeat
3   foreach data point  $s \in S$  do
4      $candidates \leftarrow$  empty map;
5      $r \leftarrow$  distance from  $s$  to  $s$ 's furthest neighbor;
6     foreach data point  $s' \in NN_{\tilde{G}}(s) \cup RNN_{\tilde{G}}(s)$  do
7        $x \leftarrow dist(s, s')$ ;
8       foreach data point  $s'' \in NN_{\tilde{G}}(s') \cup RNN_{\tilde{G}}(s')$  do
9          $y \leftarrow dist(s', s'')$ ;
10        if  $Pr[dist(s, s'') < r] > 0$  then
11          if  $s'$  or  $s''$  are new neighbors, or  $Pr[dist(s, s'') < r] < pr_s$ 
12            then
13               $candidates[s''] =$ 
14                 $max(candidates[s''], Pr[dist(s, s'') < r])$ ;
15            end
16          end
17        end
18      Reduce map  $candidates$  to contain at most  $b(s)$  map entries with the
19        highest map values;
20      for data point  $w \in keys$  of map  $candidates$  do
21        Use  $\langle s, dist(s, w) \rangle$  to update  $w$ 's NN list, and use  $\langle w, dist(s, w) \rangle$ 
22        to update  $s$ 's NN list;
23      end
24    end
25  until Termination condition is met
26 return  $\tilde{G}$ .

```

The overview of the complete *NW-Descent* is given in Algorithm 7, while its list of parameters is the same as in *RW-Descent* (see Table 5.3).

5.6 Randomized *NN-Descent* variant

The *randomized NN-Descent variant*¹ (*R-NN-Descent*) takes into account the phenomenon explained in Section 4.2 and illustrated in Figure 4.4. Knowing that the position in the initial random graph has an important role in construction of *NN-Descent* approximation, our idea was to give each point additional chances to find its “good” initial neighborhood.

Table 5.4: List of parameters of randomized *NN-Descent* variant.

Parameter	Description
k	Neighborhood size (size of NN lists).
$dist$	Distance function.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (value between 0 and 1). Algorithm converges when there is less than $conv \cdot k \cdot n$ updates in the most recent iteration, where n is the number of k -NNG vertices. Note: present only for <i>convergence</i> termination condition.
ρ	Sampling (value between 0 and 1, not including 0). Only $\rho \cdot k$ points from NN and <i>R-NN</i> lists are considered for local joins.
r	Number of random comparisons of a single point in a randomization phase.

In this *NN-Descent* variant, we introduce a randomization phase, which is performed before each *NN-Descent*’s iteration. At the beginning of the algorithm, before the first randomization phase, a new list of data points S' is created and initialized with all the points from the input set S . In a randomization phase itself, each point s from S' participates in r comparisons with r points that are randomly chosen from S , updating corresponding NN lists accordingly. By performing random comparisons we aim to position points in their “good” neighborhoods. However, some points might not get their NN lists updated by the random comparisons, in which case we assume that the neighborhoods of such points are with high probability already “good”. Namely, the points with “good” neighborhoods have lower chance of their neighborhoods to be updated by the random comparisons, and vice versa, the points that are not in “good” neighborhoods have higher chance

¹Original *NN-Descent* is already a randomized algorithm, hence the name of this variant might be misleading. However, while the original *NN-Descent* employs randomization only in the creation of the initial random graph (randomization takes place also in the sampling, but sampling is not a part of the original algorithm), this variant employs randomization during the core iterative part of the algorithm, as well.

5.6. RANDOMIZED NN-DESCENT VARIANT

Algorithm 8: Outline of randomized *NN-Descent* variant.

```

input : set of points  $S$ , distance function  $dist$ , neighborhood size  $k$ ,
         sampling  $\rho$ , random comparisons count  $r$ 
output:  $k$ -NNG approximation  $\tilde{G}$ 
// Note: Functions Sample and RandKNNNG are defined in Algorithms 1 and 2,
         respectively.
1  $\tilde{G} \leftarrow RandKNNNG(S, k, dist)$ ;
2  $S' \leftarrow S$ ;
3 repeat
4   foreach data point  $s \in S'$  do
5      $R \leftarrow Sample(S \setminus \{s\}, r)$ ;
6     foreach data point  $u \in R$  do
7        $d \leftarrow dist(s, u)$ ;
8       Use  $\langle s, d \rangle$  to update  $u$ 's NN list in  $\tilde{G}$ , and use  $\langle u, d \rangle$  to update  $s$ 's
          NN list in  $\tilde{G}$ ;
9     end
10    if  $s$ 's NN list was not updated with any point from  $R$  then
11       $S' \leftarrow S' \setminus \{s\}$ 
12    end
13  end
14  foreach data point  $s \in S$  do
15     $R \leftarrow Sample(NN_{\tilde{G}}(s), \rho \cdot k) \cup Sample(RNN_{\tilde{G}}(s), \rho \cdot k)$ ;
16    foreach two points  $v, w \in R$  do
17       $d \leftarrow dist(v, w)$ ;
18      Use  $\langle v, d \rangle$  to update  $w$ 's NN list in  $\tilde{G}$ , and use  $\langle w, d \rangle$  to update
           $v$ 's NN list in  $\tilde{G}$ ;
19    end
20  end
21 until Termination condition is met
22 return  $\tilde{G}$ .

```

that their neighborhoods get improved by random comparisons. For that reason, after a randomization phase, we remove from S' all the points that did not benefit from the random comparisons. Consequently, in all the following randomization phases, S' will consist of only points that have high probability of being in wrong neighborhood. Eventually, the list S' will get empty, after which the algorithm comes down to the original *NN-Descent*.

The outline of randomized *NN-Descent* variant is presented in Algorithm 8, while the list of its parameters is shown in Table 5.4. It is advised to configure the parameter r with respect to dataset size. Larger datasets increase the search space, so, by the laws of probability, more random comparisons should be performed in

order to discover points’ “good” neighborhoods. Therefore, in such cases r value should be higher.

5.7 Methods evaluation

In order to validate the proposed approaches against the original *NN-Descent* algorithm, we ran experiments on high-dimensional synthetic and real datasets. In this section we will give an overview of the conducted experiments—in Subsection 5.7.1 we will introduce the setup of the experiments, while in the Subsection 5.7.2 we will give an overview of the experimental results.

5.7.1 Experimental setup

The experiments were run on real and synthetic datasets presented in Table 5.5. The table contains information about datasets name, type (whether the dataset is synthetic or real), number of instances, dimensionality, and the highest hubness values in dataset’s k -NNGs $G_k, k \in \{5, 10, 20\}$. Datasets **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 $[-1, 1]$. The two synthetic datasets are very representative: high dimensionality, together with independently generated features, implies existence of hubness phenomenon, which is needed for the experiments, while the two different dataset sizes are introduced to support analysis of dataset size influence on algorithms’ performance. **BCI5** [40] is a brain-computer interface dataset of brain signal recordings taken while the subject contemplated some action. Dataset **Google-23** [29] 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** [22] from the UCI repository [21] is a dataset of spoken letters containing 26 classes of 150 instances each (3 instances are missing in the dataset), 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** [38] dataset 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 effectiveness of the algorithms: recall value $\text{recall}_{\tilde{c}}$ (see (2.2)), scan rate value $\text{scanrate}_{\tilde{c}}$ (see (2.3)) and harmonic mean

5.7. METHODS EVALUATION

Table 5.5: Datasets that were used for the experimental validation of the five *NN-Descent* variants.

Name	Type	Inst.	Dim.	Max h_{G_5}	Max $h_{G_{10}}$	Max $h_{G_{20}}$
i10000d100	Synthetic	10,000	100	160	274	488
i100000d100	Synthetic	100,000	100	263	508	912
BCI5	Real	31,216	96	24	65	225
Google-23	Real	6,686	1,937	60	118	258
ISOLET	Real	7,797	617	36	69	121
MNIST	Real	70,000	784	32	54	102

harmonic \bar{c} (see (2.5)), which were already introduced in Section 2.2. The experiments were run on Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz with 96GB of RAM, 2GB of which being available per core. Each experiment was ran 5 times, meaning that all the presented values are the averages of 5 different runs.

All the algorithms were verified for neighborhood sizes $k \in \{5, 10, 20\}$. Distance function that was used was Euclidean (L_2) distance. *NN-Descent* was ran for parameters $conv = 0.01$ and $\rho = 1$. The chosen value of $conv$ parameter achieves the best tradeoff between recall and scan rate. Hubness aware *NN-Descent* variant was ran for the same $conv$ and ρ values, and additionally for $h_{\min} = 2 \cdot k$ and $h_{\max} = 20 \cdot k$. Oversized NN list variant was ran for the same $conv$ value as the previous two algorithms, together with parameters $k' = 20$ and $\rho = 0.05 \cdot k$. As explained in Section 5.3, this combination of k' and ρ parameters is such that the upper bound value of the number of local joins in the oversized NN list variant is the same as it is in the original *NN-Descent* for neighborhood size k and $\rho = 1$. Namely, even though the neighborhood size is increased to 20, the upper bound of local joins for each dataset point, inside a single iteration, is $\rho \cdot k' = 0.05 \cdot k \cdot 20 = k$. *RW-Descent* and *NW-Descent* were run for $conv = 0.001$, and for the simplest possible balancing function. This function returns the same value $c = 8 \cdot k$ for all data points. Finally, for the randomized *NN-Descent* we used exactly the same parameters as in *NN-Descent*, with the addition of parameter r which was set to $\frac{n}{500}$.

5.7.2 Results and discussion

The results of the original *NN-Descent* algorithm are presented in Table 5.6. All we discussed in the previous chapters was validated by these experiments: *NN-Descent* depends strongly on the neighborhood size (k), on the size of the

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

dataset and on the intrinsic dimensionality of the dataset. 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 imply more local joins (by the nature of the algorithm), and if there are more local joins, the recall is more likely to be higher, while the scan rate is higher by the definition.

Table 5.6: Performance of *NN-Descent* algorithm expressed with recall and scan rate.

The results were generated with L_2 distance, $conv = 0.01$ and $\rho = 1$.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.09	0.36	0.73	0.02	0.1	0.36
scan rate	0.03	0.13	0.48	0	0.01	0.06
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.57	0.97	0.99	0.63	0.92	0.98
scan rate	0.01	0.04	0.13	0.05	0.14	0.43
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.83	0.98	1	0.74	0.96	0.99
scan rate	0.04	0.12	0.37	0.01	0.02	0.06

For a larger dataset (i.e., for a dataset 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 5.6 by comparing the results of the datasets i10000d100 and i100000d100. These two datasets have exactly the same characteristics except that the dataset i10000d100 has 10,000 instances, while the dataset i100000d100 has 100,000 instances. The explanation of this *NN-Descent*'s behavior is connected with the phenomenon described in Section 4.2 and shown in Figure 4.4. Namely, the position of the point in the initial random k -NNG is very important for the point's final recall value—if the point is in a “good” initial neighborhood, its recall value is more likely to be higher. However, the probability that a point is placed in a “good” initial neighborhood decreases as the dataset size increases. Therefore, the recalls of *NN-Descent*'s approximations decrease with the increase of the dataset size. The scan rate values also decrease due to the fact that

5.7. METHODS EVALUATION

the denominator of the scan rate equation depends quadratically on the dataset size.

Finally, our claim that the hubness highly influences *NN-Descent* was also validated by the presented results. Synthetic datasets (i10000d100 and i100000d100) have the highest intrinsic dimensionality among all the presented datasets, since all their values are independently generated. Therefore they also have the highest maximum hubness value. It can be seen in Table 5.6 that these datasets have also the worst recall values, which is aligned with our statements.

Table 5.7: Performance of hubness-aware *NN-Descent* variant expressed with recall and scan rate.

The results were generated with L_2 distance, $conv = 0.01$, $\rho = 1$,
 $h_{\min} = 2 \cdot k$ and $h_{\max} = 20 \cdot k$.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.15	0.46	0.79	0.02	0.12	0.42
scan rate	0.03	0.15	0.52	0	0.02	0.07
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.61	0.97	0.99	0.71	0.93	0.98
scan rate	0.01	0.04	0.13	0.05	0.16	0.47
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.85	0.98	1	0.75	0.96	0.99
scan rate	0.05	0.13	0.38	0.01	0.02	0.06

The experimental results of **hubness-aware *NN-Descent* variant** are presented in Table 5.7. For datasets with lower maximum hubness, such as BCI5, ISOLET and MNIST, this method performs the same as *NN-Descent*. Contrary to that, when the hubness phenomenon is more evident in the dataset, this method improves recall values. The improvements are usually followed by slightly higher scan rate values. For some cases, scan rate values even stay the same, while only recall values increase. For example, this happened for the dataset i10000d100 and $k = 5$, where the recall increased from 0.09 to 0.15 while the scan rate did not change. As a conclusion, for high dimensional datasets, this method achieved higher recall values than *NN-Descent*, at a small cost in terms of higher scan rate.

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

The experimental results obtained for **oversized NN list variant** are given in Table 5.8, from which we can observe a trade-off between scan rate and recall. This approach achieves really high increase of recall values, but at an evident cost in terms of increased scan rate values. However, the obtained scan rates are still much smaller than 1, which makes this approach much faster than the naive k -NNG construction.

Table 5.8: Performance of oversized NN list variant expressed with recall and scan rate.

The results were generated with L_2 distance, $conv = 0.01$, $\rho = 0.05 \cdot k$ and $k' = 20$.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.43	0.52	0.73	0.13	0.18	0.36
scan rate	0.18	0.27	0.48	0.02	0.03	0.06
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.89	0.99	0.99	0.8	0.94	0.98
scan rate	0.04	0.06	0.13	0.14	0.23	0.43
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.94	0.99	1	0.91	0.98	0.99
scan rate	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. Contrary to that, 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 per iteration stays the same as in *NN-Descent* algorithm. But, even though the upper bound of local joins per iteration is the same in the both cases, the total number of local joins in *NN-Descent* is smaller than it is in the oversized NN list variant, because, as already explained in Section 5.3, the oversized NN list variant converges slower, that is, it has more iterations. This is a consequence of the sampling. The sampling implies that in a single iteration no more than $\rho \cdot k$ neighbors are taken from a point's NN or R -NN list, and used in local joins. The neighbors that do get chosen are 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 from point's NN list is exactly k , which

5.7. METHODS EVALUATION

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 the previous iterations. For that reason, the overall number of local joins per iteration decreases at a slower rate, making the algorithm convergence slower, too. Consequently iterations count increases, leading to a 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 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*.

Table 5.9: Performance of *RW-Descent* expressed with recall and scan rate. The results were generated with L_2 distance, $conv = 0.001$ and balancing function that returns $8 \cdot k$ for all data points.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.1	0.38	0.73	0.01	0.1	0.35
scan rate	0.08	0.23	0.53	0.01	0.03	0.08
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.76	0.98	0.99	0.81	0.95	0.98
scan rate	0.03	0.04	0.08	0.11	0.19	0.35
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.9	0.98	1	0.8	0.97	0.99
scan rate	0.09	0.14	0.25	0.01	0.02	0.04

The results of *RW-Descent* are presented in Table 5.9. With this approach we also obtained higher recalls, except for dataset i100000d100 combined with k values 5 and 20, where this approach has slightly lower recall values than the original *NN-Descent*. Again, the increase of recall values is followed by a small increase of scan rates values. The advantage of *RW-Descent* is that the trade-off between recall and scan rates can easily be managed by adjusting the total number

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

Table 5.10: Performance of *NW-Descent* expressed with recall and scan rate. The results were generated with L_2 distance, $conv = 0.001$ and balancing function that returns $8 \cdot k$ for all data points.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.11	0.41	0.74	0.02	0.12	0.33
scan rate	0.06	0.2	0.51	0.01	0.03	0.07
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.65	0.95	0.99	0.75	0.93	0.98
scan rate	0.01	0.03	0.06	0.06	0.13	0.3
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.87	0.98	1	0.76	0.96	0.99
scan rate	0.04	0.09	0.18	0.01	0.02	0.04

of comparisons to be performed. Furthermore, *RW-Descent* has a 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 all 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—instead of calculating k -NNG approximation from the beginning, *RW-Descent* can be used only on changed nodes.

Table 5.10 summarizes the results of *NW-Descent*. The results show that *NW-Descent*’s more focused walks lead to scan rate values that are smaller than the ones produced by *RW-Descent*, while preserving approximately the same recall values in most cases. The recall values significantly dropped only for $k = 5$ in the datasets BCI5 and Google-23. In the synthetic datasets, *NW-Descent* even increases recall values, while decreasing scan rate. Generally speaking, it can be said that this approach outperforms *RW-Descent*, however, its major drawback is its limitation to L_2 distance.

5.7. METHODS EVALUATION

The results of the experiments for **randomized *NN-Descent* variant** are presented in Table 5.11. As suggested in Section 5.6, the parameter r is in the experiments directly proportional to the dataset size. This is because the size of the dataset determines the easiness of finding a “good” initial neighborhood—if the dataset is larger, the probability of finding a “good” initial 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, especially for the larger datasets, having that the parameter r is directly proportional to the dataset size.

The overall comparison of all the five approaches is shown in Figure 5.7. As can be seen, the *O-NN-Descent* usually achieves highest recalls, which are most evident for lower k values. At the same time this approach has in most of the cases high scan rate values, as well. Nevertheless, these scan rates are much lower than 1, which makes this approach a legitimate approximation algorithm—if the scan rate values were 1 or greater, the naive brute-force k -NNG construction would be a better choice. As a conclusion, when accuracy matters more than time, this approach should be considered. Additionally, *O-NN-Descent* might be the only reasonable

Table 5.11: Performance of randomized *NN-Descent* variant expressed with recall and scan rate.

The results were generated with L_2 distance, $conv = 0.01$, $\rho = 1$ and $r = \frac{n}{500}$.

	i10000d100			i100000d100		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.21	0.43	0.75	0.15	0.27	0.47
scan rate	0.14	0.22	0.54	0.13	0.16	0.2
	BCI5			Google-23		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.82	0.98	0.99	0.78	0.93	0.98
scan rate	0.09	0.11	0.18	0.11	0.2	0.48
	ISOLET			MNIST		
	$k=5$	$k=10$	$k=20$	$k=5$	$k=10$	$k=20$
recall	0.88	0.98	1	0.82	0.97	0.99
scan rate	0.11	0.18	0.41	0.09	0.09	0.12

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

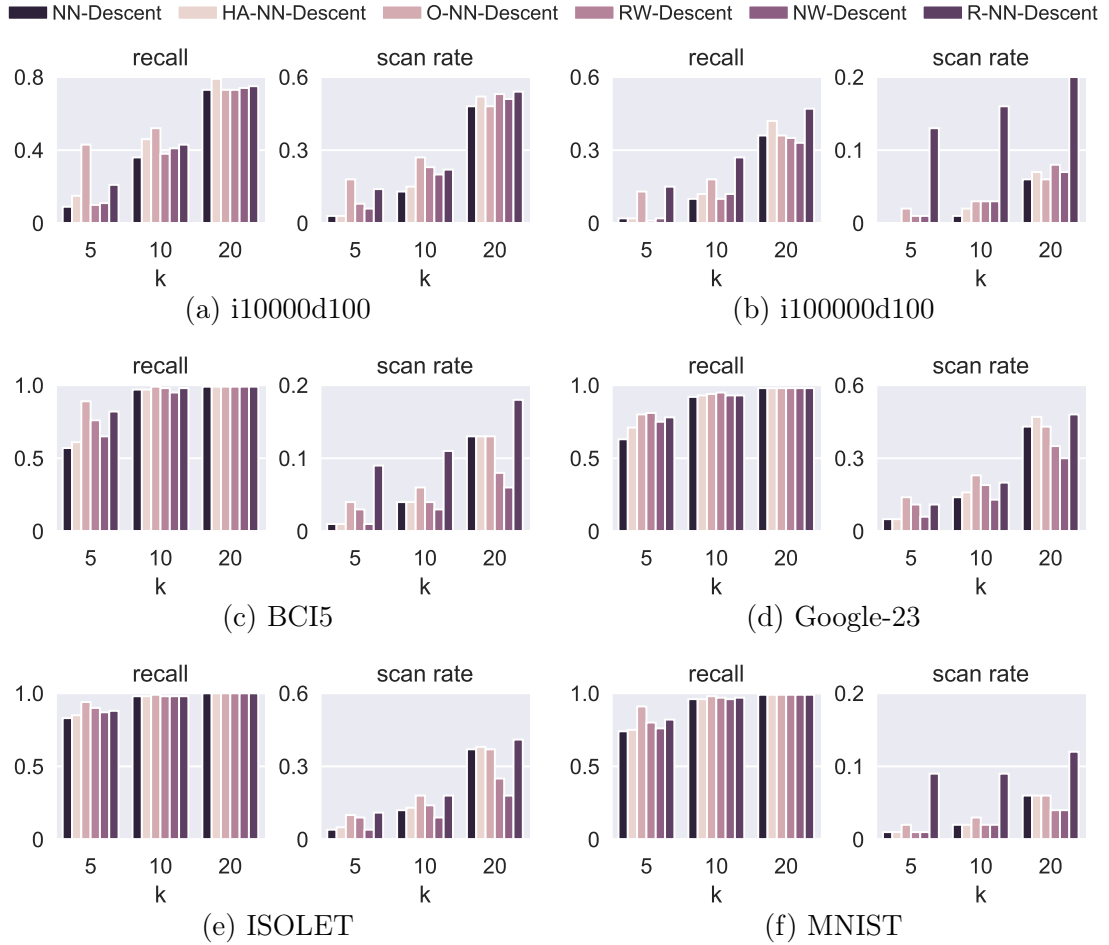


Figure 5.7: Performance of *NN-Descent* and all its variants, expressed with recall and scan rate.

option in time-sensitive applications as well, which happens when no other approach produces an approximation that is usable. Namely, as shown in Figure 5.7, for some datasets, the other approaches produce highly inaccurate approximations. For example, for dataset *i10000d100* and $k = 5$ (see Subfigure 5.7a), the recalls of other approaches are 0.21 and lower, while the recall of *O-NN-Descent* is 0.43. Therefore, even though the other algorithms are faster, they might be unusable since their approximations are highly inaccurate.

On the other side, when time is more important than accuracy, *HA-NN-Descent* and *NW-Descent* are the best choices. *HA-NN-Descent* increases recall values for the high dimensional datasets, while preserving scan rate values the same or almost the same. Compared to the other approaches, *HA-NN-Descent* does not improve recall value as much, but since it almost comes without any cost, it is certainly

5.7. METHODS EVALUATION

worth considering. *NW-Descent* comes with a slightly higher scan rates and usually with higher recall values, as well. These two approaches behave similarly in most cases, however, there are certain situations when one outperforms the other. For example, in i10000d100 *HA-NN-Descent* is better than *NW-Descent*, while in BCI5 the opposite holds. Additionally, while *HA-NN-Descent* is designed only for high dimensional datasets (otherwise it comes down to *NN-Descent*), *NW-Descent* has a potential to improve *NN-Descent* in low dimensional datasets, too.

As previously discussed, *RW-Descent* produces similar recall values as *NW-Descent*, but with a slightly higher scan rate values. However, for specific datasets, this approach outperformed all the others. This is the case for dataset Google-23, in which *RW-Descent* outperformed even *O-NN-Descent*, having higher recall, but at the same time lower scan rate.

Finally, *R-NN-Descent* is most of the times the second best with respect to recall values (the first one is, as already said, *O-NN-Descent*), but increase in scan rate values is usually very high. For example, in the case of datasets i100000d100 and MNIST, which are the largest datasets, *R-NN-Descent* increased scan rate values drastically, while achieving recall values that are similar to the ones achieved by *O-NN-Descent*. As a conclusion, this approach is rarely the best choice.

Additionally, we calculated harmonic means of recall and scan gain values (see (2.5)) and presented them in Figure 5.8. As we discussed, *O-NN-Descent* improves recall values with the cost in terms of increased scan rate values. However, according to the harmonic means presented in Figure 5.8, this trade-off is cost-effective. This especially holds for smaller k values, where harmonic mean of this approach is usually the best among harmonic means of all the other approaches. Moreover, *O-NN-Descent* reported outstandingly higher harmonic means the synthetic datasets (i10000d100 and i100000d100), which differ from other datasets by having very high intrinsic dimensionalities.

In the dataset i100000d100, which is a large dataset of high intrinsic dimensionality, the best approach with respect to harmonic mean was *R-NN-Descent*. Despite the fact that the relative differences between *R-NN-Descent*'s scan rate values and scan rate values of other approaches are considerably high, the absolute differences are usually not much higher than 0.1. Having that the harmonic mean is dominated by the minimum of its arguments, recall value dominates scan gain value in the harmonic mean only if the recall is lower than scan gain. As mentioned *R-NN-Descent*'s absolute increase of scan rate is not high enough in the dataset i100000d100 to make scan gain lower than recall, meaning that, in this case, recall value dominates scan gain. Since the recall value of this approach

CHAPTER 5. PROPOSED METHODS FOR IMPROVING *NN-DESCENT*

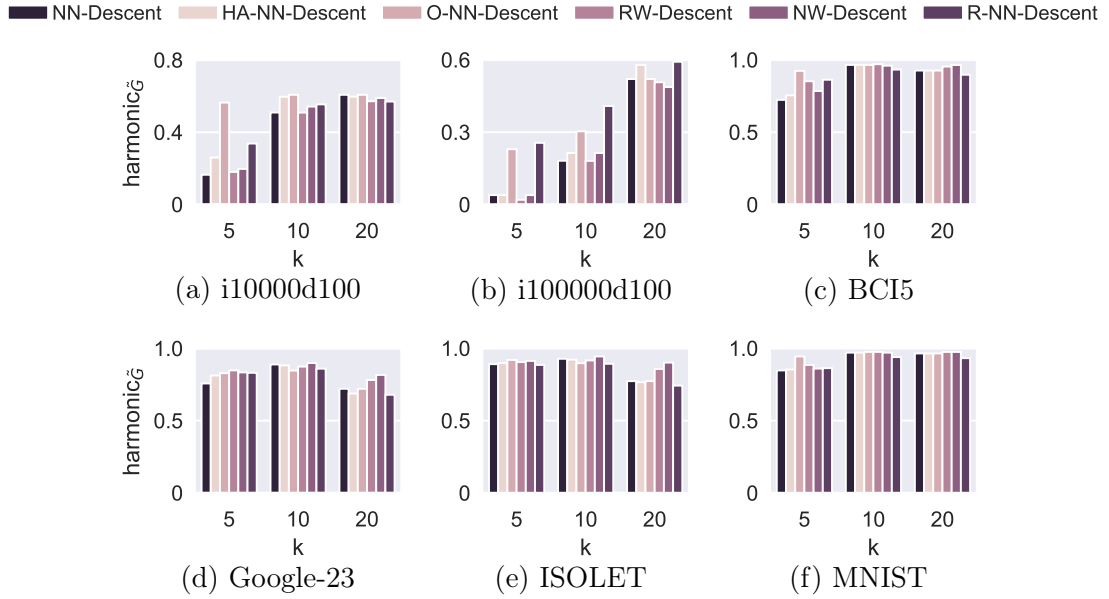


Figure 5.8: Performance of *NN-Descent* and all its variants, expressed with harmonic mean of recall and scan gain.

is in this case the highest among recalls of all the other approaches, the harmonic mean ended up being the highest as well. Therefore, from this perspective, *R-NN-Descent* indeed outperformed other approaches in a large high dimensional dataset.

For larger k values harmonic means of *RW-Descent* and *NW-Descent* tend to be the best. Moreover, harmonic means of these approaches are quite good for lower k values as well. For example, in the datasets *Google-23* and *ISOLET* they are the best performing approaches for all k values.

Finally, as expected, with respect to the harmonic mean, *HA-NN-Descent* behaves better in the datasets with higher presence of hubness phenomenon, which are *i10000d100*, *i100000d100* and *Google-23*. In the other datasets, *HA-NN-Descent* reports similar harmonic means as *NN-Descent*.

As a conclusion, all the presented *NN-Descent* variants introduce certain advancements over *NN-Descent* on high dimensional datasets. Moreover, there is no ultimately the best variant—the experiments demonstrated that each of them has its plus and minus sides, making a variant suitable only under certain assumptions. However, even though none of the variants gives an ultimate solution, all of them together provide a powerful tool for variety of possible problem setups. It is also important to point out that, generally speaking, the improvements over original

5.7. METHODS EVALUATION

NN-Descent are most evident for smaller k values, while for $k = 20$ *NN-Descent* already performed well enough.

Chapter 6

NN-Descent-based approximation algorithms for k -NN graph updates

In real world scenarios, datasets often change over time: new instances are being added and existing instances are being changed or removed. If a k -NNG had been built on a dataset that changed afterwards, the graph must be updated accordingly, because changes in the data result in new nodes and edges appearing, old nodes and edges disappearing, and information associated with nodes and edges being updated. The simplest way to update the graph is to construct a new k -NNG on the updated dataset from scratch, which could be done by using any algorithm for k -NNG construction. However, this approach is slow and does not make use of the information that was integrated in the previous k -NNG. Therefore, instead of completely discarding the previous k -NNG, one could use it as a starting point for a construction of a new k -NNG.

In this chapter we propose two *NN-Descent*-based approximation algorithms for k -NNG updates. The proposed algorithms update the existing k -NNG after a subset of its nodes has changed. Even though this algorithms do not support node additions nor deletions, they could be trivially adapted in order to support these operations as well.

Section 6.1 gives an overview of the naive, brute-force, k -NNG update algorithm. In Section 6.2 we will introduce our *NN-Descent*-based approximation algorithms for k -NNG updates. Finally, in Section 6.3 we will evaluate the performance of all algorithms by conducting an experimental analysis.

6.1 Naive k -NN graph update algorithm

When a dataset changes, the k -NNG built on it should be updated accordingly. As already said, the updates of the k -NNG should be performed in an incremental

6.1. NAIVE k -NN GRAPH UPDATE ALGORITHM

manner, rather than constructing the graph from scratch. The reason behind this lies in the fact that datasets usually do not change completely, but only partially. Hence, the graph should also be updated partially.

Algorithm 9: Function that returns a set of points whose NN lists should be updated.

```
// Function takes a  $k$ -NN graph ( $G$ ) together with a list of its changed nodes
// ( $changed\_nodes$ ), and returns a set of all the points that are affected by the
// changed nodes, i.e., the set of points whose NN lists should be updated.
1 function GetAffectedNodes ( $G$ ,  $changed\_nodes$ )
2   |  $affected \leftarrow changed\_nodes$ ;
3   | foreach data point  $s \in changed\_nodes$  do
4     |   |  $affected \xleftarrow{insert} RNN_G(s)$ ;
5     |   | end
6     |   | return  $affected$ .
7 end
```

Algorithm 10: Outline of the naive k -NNG update algorithm.

```
input : data set  $S$ , distance function  $dist$ , neighborhood size  $k$ ,  $k$ -NNG  $G$ ,
        list of  $G$ 's changed nodes  $changed\_nodes$ 
output: updated  $k$ -NNG  $G$ 
// Note: Function  $GetAffectedNodes$  is defined in Algorithm 9.
1 for  $\_update \leftarrow GetAffectedNodes(G, changed\_nodes)$ ;
2 foreach data point  $s \in \_update$  do
3   |   | foreach data point  $v \in S$  do
4     |     | Use  $\langle v, dist(s, v) \rangle$  to update  $s$ 's NN list in  $G$ , and use  $\langle s, dist(s, v) \rangle$  to
5     |     |   | update  $v$ 's NN list in  $G$ ;
6     |     |   | end
7   |     | end
8 end
9 return  $G$ .
```

Naive brute-force k -NNG construction algorithm can easily be adapted to support partial k -NNG updates. Let us say that node s of k -NNG G has changed. In that case, distances between s and all the other points must be calculated again, and s 's NN list, together with NN lists of all the other dataset points, must be updated accordingly. There are two possible changes of k -NNG which could be a result of this procedure: 1) s 's NN list might be different than before, 2) s might end up in other points' NN lists in which it had not been before. The very same procedure must be applied for all the points from $RNN_G(s)$ as well—distances from these points to all the other points must be calculated, and their NN lists

must be updated accordingly. Namely, by performing this procedure, each point from $RNN_G(s)$ determines if s should be removed from its NN list. No other distances should be calculated during the algorithm.

For datasets without hubness phenomenon, in which sizes of R -NN lists are approximately k , the described incremental update of a k -NNG would result with worst case complexity of $O(pnk)$, p being the number of updated dataset points, n being the size of the dataset, and k being the neighborhood size. In the following text we will refer to this algorithm as *naive k -NNG update*. An outline of the approach is given in Algorithm 10.

6.2 Online variants of random walk descent and nearest walk descent

The algorithm presented in the previous section, performs a partial update of an exact k -NNG which results with a new, updated exact k -NNG. If this is still not fast enough, an approximation algorithm for k -NNG updates could be an option. In this case the result of an update would not be the exact k -NNG, but a k -NNG approximation. In this section we will propose two such algorithms that perform incremental updates of k -NNG, resulting with a k -NNG approximation.

The two algorithms are based on *RW-Descent* and *NW-Descent*, which are presented in Sections 5.4 and 5.5, respectively. The idea behind this algorithms is rather simple. In the same manner as in the naive k -NNG update algorithm, we pre-compute the set of the points whose NN lists should be updated: that set contains the changed points themselves and the points that have at least one changed point in their NN lists (these points are actually reverse neighbors of the changed points). Unlike in the naive k -NNG update algorithm, in these algorithms the points from the pre-computed set are not being compared against all the dataset points. Instead, the graph walk approaches from *RW-Descent* and *NW-Descent* are applied. The algorithms iteratively apply a certain number of short walks that start from each point of the pre-computed set. The starting and the ending point of each walk participate in a local join. The algorithms iterate until the termination condition is met (see Section 5.4).

The only difference between the two algorithms is the way they determine which walks to perform. In the first algorithm, that we will call *online random walk descent (online RW-Descent)*, the walks are performed completely at random. In the second algorithm, that we will call *online nearest walk descent (online*

6.2. ONLINE VARIANTS OF RANDOM WALK DESCENT AND NEAREST WALK DESCENT

NW-Descent), all possible 2-length walks from a given point are evaluated, and the “best” ones are chosen. *online RW-Descent* and *online NW-Descent* perform walks in completely analogous way as *RW-Descent* and *NW-Descent*, respectively. The outline of the two algorithms is presented in Algorithm 11.

Algorithm 11: Outline of *online RW-Descent* and *online NW-Descent* algorithms.

input : data set S , distance function $dist$, neighborhood size k , balancing function $b(s)$, previous k -NNG G , list of G 's changed nodes $changed_nodes$

output: k -NNG approximation \tilde{G}

// Note: Function *GetAffectedNodes* is defined in Algorithm 9.

- 1 $\tilde{G} \leftarrow$ copy of G ;
- 2 $for_update \leftarrow GetAffectedNodes(G, changed_nodes)$;
- 3 **repeat**
- 4 **foreach** data point $s \in for_update$ that did not converge **do**
- 5 $walks \leftarrow$ choose $b(s)$ walks in \tilde{G} that start in s ;
- 6 **foreach** $walk \in walks$ **do**
- 7 $w \leftarrow$ ending point of $walk$;
- 8 Use $\langle s, dist(s, w) \rangle$ to update $NN_{\tilde{G}}(w)$, and use $\langle w, dist(s, w) \rangle$ to update $NN_{\tilde{G}}(s)$;
- 9 **end**
- 10 **end**
- 11 **until** *Termination condition is met*
- 12 **return** \tilde{G} .

Online RW-Descent and *online NW-Descent* have one crucial issue. They work under the assumption that the points did not change considerably, i.e., that they stayed in their extended neighborhoods. The algorithms make use of this assumption when they look for points' new nearest neighbors by performing short walks—by doing this, the algorithms try to update points' NN lists by looking at their extended neighborhoods. When a point did not change significantly, that is, when it stayed in the same extended neighborhood, the algorithms perform very well and very fast. However, the problem arises when the point did change significantly. In that case the point's new nearest neighbors can not be found in its extended neighborhood, and, therefore, there is a high probability that the point will get confined in wrong local minimum, without finding its real neighbors.

After the previous statement, an additional question arises. In the original *RW-Descent* and *NW-Descent*, points also start from wrong initial neighborhoods, but the aforementioned problem does not hold there. Therefore, the question is:

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION
ALGORITHMS FOR *K*-NN GRAPH UPDATES

Algorithm 12: Outline of the complete *online RW-Descent* and *online NW-Descent* algorithms.

input : data set S , distance function $dist$, neighborhood size k , balancing function $b(s)$, convergence $conv$, random comparisons count r , previous k -NNG G , list of G 's changed nodes $changed_nodes$

output: k -NNG approximation \tilde{G}

// Note: Function *GetAffectedNodes* is defined in Algorithm 9.

```

1  $\tilde{G} \leftarrow$  copy of  $G$ ;
2  $for\_update \leftarrow GetAffectedNodes(G, changed\_nodes)$ ;
3  $S' \leftarrow for\_update$ ;
4 repeat
5   foreach data point  $s \in S'$  do
6      $R \leftarrow Sample(S \setminus \{s\}, r)$ ;
7     foreach data point  $u \in R$  do
8        $d \leftarrow dist(s, u)$ ;
9       Use  $\langle s, d \rangle$  to update  $u$ 's NN list in  $\tilde{G}$ , and use  $\langle u, d \rangle$  to update  $s$ 's
       NN list in  $\tilde{G}$ ;
10    end
11    if  $s$ 's NN list was not updated more than  $r \cdot conv$  times then
12       $S' \leftarrow S' \setminus \{s\}$ 
13    end
14  end
15  foreach data point  $s \in for\_update$  that did not converge do
16     $walks \leftarrow$  choose  $b(s)$  walks in  $\tilde{G}$  that start in  $s$ ;
17    foreach  $walk \in walks$  do
18       $w \leftarrow$  ending point of  $walk$ ;
19      Use  $\langle s, dist(s, w) \rangle$  to update  $NN_{\tilde{G}}(w)$ , and use  $\langle w, dist(s, w) \rangle$  to
      update  $NN_{\tilde{G}}(s)$ ;
20    end
21  end
22 until Termination condition is met
23 return  $\tilde{G}$ .

```

why the problem exists in *online RW-Descent* and *online NW-Descent*, while it does not exist in *RW-Descent* and *NW-Descent*? The reason for this is that *RW-Descent* and *NW-Descent* start with a random initial graph. In a very first iteration, a point s is compared with $b(s)$ completely random points, among which the nearest ones are inserted in s 's NN list. Contrary to that, online variants of the algorithms do not start with a random initial graph, but with the previous version of k -NNG. Consequently, the first iteration of *online RW-Descent* and *online NW-Descent* does not imply comparisons with $b(s)$ uniformly random

6.2. ONLINE VARIANTS OF RANDOM WALK DESCENT AND NEAREST WALK DESCENT

points, but with $b(s)$ points from the s 's old extended neighborhood. Unlike in *RW-Descent* and *NW-Descent*, where a point initially gets to choose among different neighborhoods (multiple uniformly random points come, with high probability, from multiple neighborhoods), in online variants a point starts with a single neighborhood, which is a very nice starting point if that neighborhood is “good”, but otherwise it is a very bad place to start from.

Table 6.1: List of parameters of *online RW-Descent* and *online NW-Descent*.

Parameter	Description
G	Existing k -NNG.
k	Neighborhood size (size of NN lists).
$dist$	Distance function.
$changed_nodes$	G 's nodes that have changed.
it	Number of iterations. Note: present only for <i>fixed iterations</i> termination condition.
$conv$	Convergence criterion (a value from range $(0, 1]$). This parameter configures two different things. 1) Convergence criterion (a value from range $(0, 1]$). A point converges when there is in average less than $conv \cdot b(s)$ updates in the hd most recent iterations. The algorithm converges when all points converge. Note: present only for <i>convergence</i> termination condition. 2) A point participates in randomization phases as long as the number of its NN list updates in most recent randomization phase is greater than $conv \cdot r$.
hd	History depth (a value from range $[0, \infty)$)—see parameter $conv$ for more information. Note: present only for <i>convergence</i> termination condition.
$b(s)$	Balancing function that returns number of local joins for a point s .
r	Number of random comparisons of a single point in the randomization phase.

For this reason, previously presented algorithms are extended with the idea on which *R-NN-Descent* is based (see Section 5.6)—more precisely, *online RW-Descent* and *online NW-Descent* are extended with the randomization phase. Before each iteration, the randomization phase is performed, in which points from pre-computed set are compared with r other dataset points that are chosen uniformly at random. This allows points to consider more than one neighborhood in the initial phase of the algorithm, which is exactly what is happening in *RW-Descent* and *NW-Descent*, but was missing from their online variants. Additionally, in *R-NN-Descent*, a point is excluded from future randomization phases as soon as

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR K -NN GRAPH UPDATES

the random comparisons from the current phase fail to update the point’s NN list. As we already discussed in Section 5.6, if a point’s NN list does not get updated, the probability that the point is in a “good” neighborhood is reasonably high, and for that reason the point does not have to participate in the future randomization phases. The same idea is implemented in *online RW-Descent* and *online NW-Descent*, but with a minor technical change—a point is excluded from future randomization phases when the number of NN list updates is not greater than a predefined threshold. If that threshold is zero, then the approach comes down to the one included in *R-NN-Descent*. The threshold is defined by *conv* parameter—if, for a point s , less than $conv \cdot r$ comparisons with random points resulted with s ’s NN list update, s is excluded from the future randomization phases. The same parameter *conv* is used for convergence termination condition, as well. The updated version of *online RW-Descent* and *online NW-Descent* is presented in Algorithm 12, while the overview of their parameters can be found in Table 6.1.

6.3 Methods evaluation

As previously explained, the main goal of *online RW-Descent* and *online NW-Descent* algorithms is to fasten the k -NNG update. In order to verify if the goal was met, intensive experiments were conducted. The algorithms were compared with the naive partial k -NNG update, presented in Section 6.1, but also with construction of k -NNG from scratch by using an existing fast approximation algorithm *NN-Descent*, presented in Section 3.2.

Online RW-Descent and *online NW-Descent* can be useful in any scenario in which multiple k -NNGs must be created on similar datasets—first dataset’s k -NNG graph is in that case created with any k -NNG construction algorithm, while k -NNGs of all the other datasets are created by *online RW-Descent* or *online NW-Descent*. Moreover, the most natural application of *online RW-Descent* and *online NW-Descent* is probably the construction of temporal k -NNGs (as explained in Section 3.3, a temporal k -NNG is a k -NNG that is built on temporal data). Each time data change over time, *online RW-Descent* or *online NW-Descent* can be used to create a new k -NNG approximation.

Therefore, for the experiments, we simulated a real-world system in which data change over time, and we verified *online RW-Descent* and *online NW-Descent* in that context. When it comes to temporal data, time series are usually the first choice, hence these simulations are based on time series data. In the Sub-

section 6.3.1 we introduce datasets that we used in the experiments, in Subsection 6.3.2 we thoroughly describe the design of the simulations, Subsection 6.3.3 gives an overview of the experimental setup, and last but not least, Subsection 6.3.4 presents the final results.

6.3.1 Datasets

As already said, we used time series data as the basis of our experiments (see Section 2.1 for more insights about time series). If k -NNG was built on such data, it has to be updated as the time series change, which is an ideal use case for verification of *online RW-Descent* and *online NW-Descent*.

The most famous and most complete time series data repository is the UCR Time Series Classification Archive [16]. For that reason we verified *online RW-Descent* on all the datasets from UCR repository. The repository contains 128 datasets, each of them having different properties.

In order to adapt the datasets for our experiments, we had to perform a small preprocessing step. The preprocessing included only removal of entries with missing values. The reason behind this was that missing values, as such, could not be interpreted by the distance functions we used—these distance functions can accept only numerical values. There are two possibilities for dealing with this problem: 1) to replace each missing value with some appropriate numerical value (such as average of non-missing time series' values) and 2) to ignore/remove entries with missing values. At the end we went for the second option, which is to remove entries with missing values.

The properties of datasets after the preprocessing phase are given in Table 6.2. The table contains information about dataset name (dataset), minimum and maximum time series length (min/max ts), and the number of time series within a dataset (size). Before the preprocessing phase, all the time series within a single dataset were of equal length. However, after removal of entries with missing values, time series lengths within a dataset may vary.

6.3.2 Simulation design

In this subsection we will present the design of simulations used to verify *online RW-Descent* and *online NW-Descent*. Let us start by presenting the real-world scenario that we simulated in our experiments. First of all, there is a dataset whose instances are time series. Data that is preserved in the time series are measurements of a certain phenomenon at different moments in time—each time series

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

Table 6.2: Properties of UCR datasets after the preprocessing step.

dataset	min/max ts	size	dataset	min/max ts	size
ACSF1	1460/1460	200	ItalyPowerDemand	24/24	1096
Adiac	176/176	781	LargeKitchenAppliances	720/720	750
AllGestureWiimoteX	2/385	1000	Lightning2	637/637	121
AllGestureWiimoteY	2/385	1000	Lightning7	319/319	143
AllGestureWiimoteZ	2/385	1000	Mallat	1024/1024	2400
ArrowHead	251/251	211	Meat	448/448	120
Beef	470/470	60	MedicalImages	99/99	1141
BeetleFly	512/512	40	MelbournePedestrian	24/24	3650
BirdChicken	512/512	40	MiddlePhalanxOutlineAgeGroup	80/80	554
BME	128/128	180	MiddlePhalanxOutlineCorrect	80/80	891
Car	577/577	120	MiddlePhalanxTW	80/80	553
CBF	128/128	930	MixedShapesRegularTrain	1024/1024	2925
Chinatown	24/24	365	MixedShapesSmallTrain	1024/1024	2525
ChlorineConcentration	166/166	4307	MoteStrain	84/84	1272
CinCECGTorso	1639/1639	1420	NonInvasiveFetalECGThorax1	750/750	3765
Coffee	286/286	56	NonInvasiveFetalECGThorax2	750/750	3765
Computers	720/720	500	OliveOil	570/570	60
CricketX	300/300	780	OSULeaf	427/427	442
CricketY	300/300	780	PhalangesOutlinesCorrect	80/80	2658
CricketZ	300/300	780	Phoneme	1024/1024	2110
Crop	46/46	24000	PickupGestureWiimoteZ	29/361	100
DiatomSizeReduction	345/345	322	PigAirwayPressure	2000/2000	312
DistalPhalanxOutlineAgeGroup	80/80	539	PigArtPressure	2000/2000	312
DistalPhalanxOutlineCorrect	80/80	876	PigCVP	2000/2000	312
DistalPhalanxTW	80/80	539	PLAID	101/1344	1074
DodgerLoopDay	215/288	158	Plane	144/144	210
DodgerLoopGame	215/288	158	PowerCons	144/144	360
DodgerLoopWeekend	215/288	158	ProximalPhalanxOutlineAgeGroup	80/80	605
Earthquakes	512/512	461	ProximalPhalanxOutlineCorrect	80/80	891
ECG200	96/96	200	ProximalPhalanxTW	80/80	605
ECG5000	140/140	5000	RefrigerationDevices	720/720	750
ECGFiveDays	136/136	884	Rock	2844/2844	70
ElectricDevices	96/96	16637	ScreenType	720/720	750
EOGHorizontalSignal	1250/1250	724	SemgHandGenderCh2	1500/1500	900
EOGVerticalSignal	1250/1250	724	SemgHandMovementCh2	1500/1500	900
EthanolLevel	1751/1751	1004	SemgHandSubjectCh2	1500/1500	900
FaceAll	131/131	2250	ShakeGestureWiimoteZ	40/385	100
FaceFour	350/350	112	ShapeletSim	500/500	200
FacesUCR	131/131	2250	ShapesAll	512/512	1200
FiftyWords	270/270	905	SmallKitchenAppliances	720/720	750
Fish	463/463	350	SmoothSubspace	15/15	300
FordA	500/500	4921	SonyAIBORobotSurface1	70/70	621
FordB	500/500	4446	SonyAIBORobotSurface2	65/65	980
FreezerRegularTrain	301/301	3000	StarLightCurves	1024/1024	9236
FreezerSmallTrain	301/301	2878	Strawberry	235/235	983
Fungi	201/201	204	SwedishLeaf	128/128	1125
GestureMidAirD1	80/360	338	Symbols	398/398	1020
GestureMidAirD2	80/360	338	SyntheticControl	60/60	600
GestureMidAirD3	80/360	338	ToeSegmentation1	277/277	268
GesturePebbleZ1	100/455	304	ToeSegmentation2	343/343	166
GesturePebbleZ2	100/455	304	Trace	275/275	200
GunPoint	150/150	200	TwoLeadECG	82/82	1162
GunPointAgeSpan	150/150	451	TwoPatterns	128/128	5000
GunPointMaleVersusFemale	150/150	451	UMD	150/150	180
GunPointOldVersusYoung	150/150	451	UWaveGestureLibraryAll	945/945	4478
Ham	431/431	214	UWaveGestureLibraryX	315/315	4478
HandOutlines	2709/2709	1370	UWaveGestureLibraryY	315/315	4478
Haptics	1092/1092	463	UWaveGestureLibraryZ	315/315	4478
Herring	512/512	128	Wafer	152/152	7164
HouseTwenty	2000/2000	159	Wine	234/234	111
InlineSkate	1882/1882	650	WordSynonyms	270/270	905
InsectEPGRegularTrain	601/601	311	Worms	900/900	258
InsectEPGSmallTrain	601/601	266	WormsTwoClass	900/900	258
InsectWingbeatSound	256/256	2200	Yoga	426/426	3300

6.3. METHODS EVALUATION

holds measurements for an object it represents. Moreover, time series preserve at most d last measurements of the phenomenon; meaning that, if a time series already holds d values and a new value comes, the oldest value has to be discarded. Therefore, the upper bound of time series length is d . This dataset is then used as a node set of a k -NNG. The k -NNG must always be up to date, meaning that when any of the time series change, the graph must be updated as well.

As can be seen, such scenario implies data flow. To simulate such data flow by using static datasets presented in the previous section, we will employ *sliding window technique*. Generally speaking, sliding window technique is applied when an array of values is never of interest as a whole. The current area of interest within the array is then defined by a sliding window. A sliding window has its size denoted by d , which suggests how many array elements it represents, and its starting position within the array denoted by i . Finally, a sliding window of size d and starting position i represents the sub-array that starts at index i and ends at index $i + d - 1$. Moreover, the sliding windows have property that their starting positions change during the execution of an algorithm that uses them. The algorithm itself dictates how sliding window's starting position changes, however, very often sliding window position is initially 0, and then it monotonically increases until reaching $l - d$, where l is the time series length. Figure 6.1 visualizes an example of sliding window technique application. The sliding window is depicted by the colored array cells, while each of the numbered six rows represents a different state of the same array. An algorithm in its first step initializes the sliding window to the leftmost starting position within the array. Then, in each step the algorithm moves the sliding window by three steps towards right, until the sliding window reaches the rightmost position (notice that in the last step, the algorithm could not move the sliding window by three steps, since there was no room for it). The purpose of the whole process is that the algorithm in each step could perform some operation on the sub-array represented by the current sliding window position.

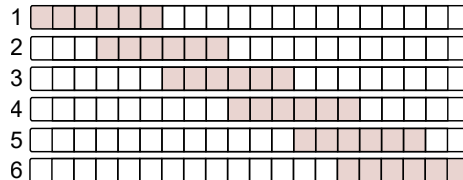


Figure 6.1: Visualization of the sliding window technique.

In the simulations we will use the sliding window technique inside individual time series. For that purpose we introduce parameter **sliding window size**, in further text denoted by *sw*. Each time series within a dataset will have its

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

own sliding window of size sw . During the simulation, k -NNGs will never be constructed on the whole time series, but only on their segments defined by their sliding windows. At the beginning of a simulation, sliding windows inside each time series have starting position 0. As the simulation proceeds, sliding windows are moved towards right, in a manner we will describe later. Each time some sliding window(s) move(s), a k -NNG must be updated.

A simulation is executed iteratively. In each iteration a certain number of time series is selected, and their sliding windows are moved towards right for a certain number of steps, after which, as said, a k -NNG update follows. The number of steps by which a sliding window is moved, is defined by parameters **minimum and maximum batch size**, in further text denoted by b_{min} and b_{max} , respectively. For a sliding window which needs to be moved, a random value r from the range $[b_{min}, b_{max}]$ is taken, and its starting position i increases by r . Additionally, as already said, i is upper-bounded by $l - sw$ (l being time series length), therefore, if $i + r > l - sw$, then i is set to $l - sw$. If b_{min} and b_{max} are equal, then we will denote them simply by b . As a conclusion, batch size value determines how much time series change in a single simulation iteration—the higher batch size value, the greater is the time series change. The reason for having the minimum and maximum batch size values instead of a single batch size is to support behavior of systems in which time series values do not come in regular manner.

The number of time series whose sliding windows move inside a single iteration, is defined by parameters **minimum and maximum points count**, in further text p_{min} and p_{max} , respectively. Similarly as with batch size, if p_{min} and p_{max} are equal, we refer to them simply as p . The actual number of time series that are selected in a single iteration is then randomly chosen from the range $[p_{min}, p_{max}]$ —let us denote that number by r . These r time series are selected uniformly at random from all the datasets's time series whose sliding windows did not reached rightmost position. If there are less than r such time series, all of them are selected.

There are two additional simulation parameters: an algorithm for initial k -NNG construction, in further text **initial algorithm**, and an algorithm for k -NNG update, in further text **update algorithm**. Moreover, the input of a simulation is a single time series dataset.

Let us now summarize the flow of a single simulation. Each time series within the input dataset has its sliding window whose length is sw , and initial starting position is 0. Before proceeding any further, an initial k -NNG is constructed by the initial algorithm. The k -NNG is not constructed on the whole dataset's time series, but only on their segments defined by their sliding windows. A simulation then

6.3. METHODS EVALUATION

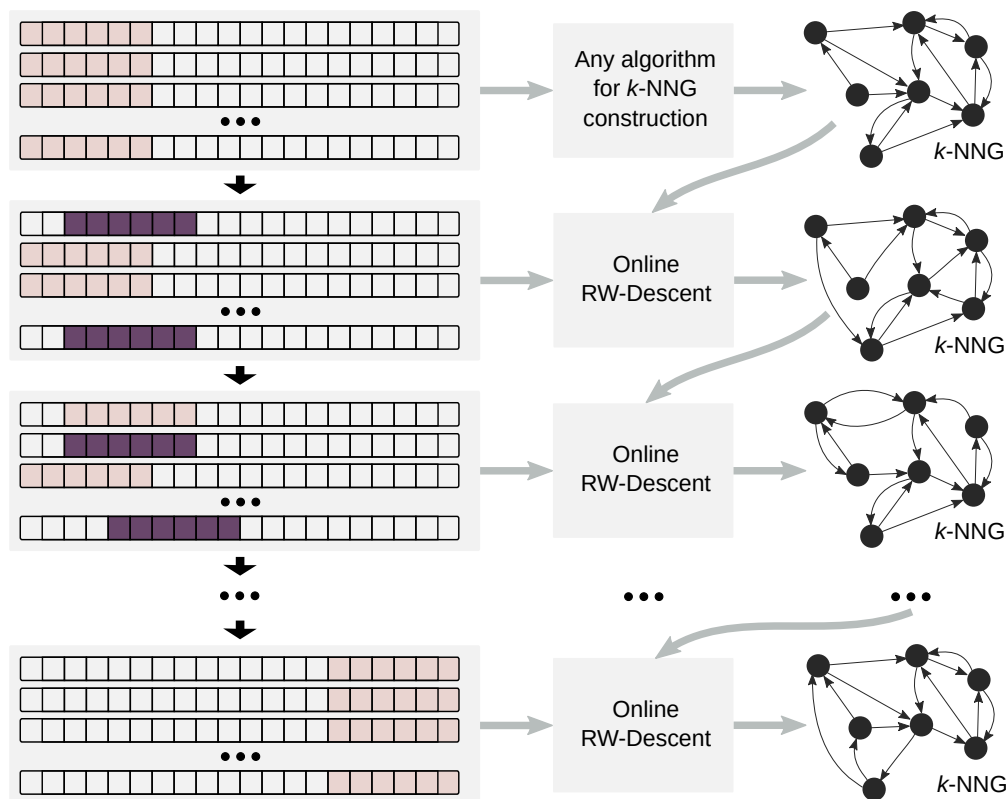


Figure 6.2: Outline of the simulation flow.

proceeds with an iterative execution. In each iteration a number r is randomly chosen from range $[p_{min}, p_{max}]$, and then r random time series are selected for update. Each time series' update implies movement of its sliding window towards right, by the number of steps which is randomly chosen from range $[b_{min}, b_{max}]$. After the updates of all r randomly chosen time series, the current version of k -NNG is updated by the update algorithm. This finalizes an iteration. Simulation continues iterating until all the time series' sliding windows are in the rightmost position.

Figure 6.2 visualizes the simulation flow. The simulation from the figure is configured as follows: $sw = 6$, $b = 2$, $p = 2$, the initial algorithm is an arbitrary chosen k -NNG construction algorithm, while the update algorithm is *online RW-Descent*. Left hand side of the figure depicts sliding window updates. Snapshots of sliding window positions are presented inside gray rectangles—each rectangle presents sliding window positions of a single iteration. Similarly as in Figure 6.1, sliding windows are depicted by the colored cells. Sliding windows colored with darker shade are the ones that are selected for update, and therefore moved to the right. In the middle of the figure there are rectangles which represent algorithms. Arrows

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

that go inside these rectangles are the algorithms inputs, while the arrows that go outside of the rectangles are the algorithms outputs. *k*-NNGs on the right hand side are the ones that were constructed on the colored part of dataset from their left hand side. Finally, as can be seen, *online RW-Descent* always takes the current dataset and the previous *k*-NNG, and creates a new *k*-NNG approximation.

Finally, the list of all simulation parameters is given in Table 6.3. Moreover, there are certain constraints and restrictions among the simulation parameters. They are given in (6.1), (6.2) and (6.3), in which n stands for dataset size, while ts_{min} stands for the length of the shortest time series within the dataset.

Table 6.3: List of simulation parameters.

Parameter	Description
initial algorithm	The algorithm that is used for construction of initial <i>k</i> -NNG, together with all its parameters.
update algorithm	The algorithm that is used for <i>k</i> -NNG updates, together with all its parameters.
sw	Sliding window size.
b_{min}	Minimum batch size.
b_{max}	Maximum batch size.
p_{min}	Minimum points count.
p_{max}	Maximum points count.

$$0 < sw \leq ts_{min} \tag{6.1}$$

$$0 < b_{min} \leq b_{max} \leq sw \tag{6.2}$$

$$0 < p_{min} \leq p_{max} \leq n \tag{6.3}$$

6.3.3 Experimental setup

The main purpose of the experiments was to compare four different approaches for solving *k*-NNG update problem, which are: naive partial *k*-NNG update, *NN-Descent*, *online RW-Descent* and *NW-Descent*. The datasets used in the experiments are presented in Subsection 6.3.1. The experiments are completely based on simulations introduced in Subsection 6.3.2, therefore, in this subsection we will give an overview of the simulation parameters settings.

Update algorithms of the simulation were set to be exactly the four algorithms mentioned above. Moreover, each of these algorithms comes with its parameters, which are configured as follows. For all the algorithms we used values 5 and 10 for

6.3. METHODS EVALUATION

parameter k , and Euclidean distance and DTW¹ as distance measures. Both distance measures are chosen for their popularity—Euclidean distance is commonly used in general, while DTW is commonly used with time series data. Furthermore, parameters for *NN-Descent* were: $\rho = 1$, $conv = 0.01$. For both, *online RW-Descent* and *online NW-Descent*, we used two different parameter setups: 1) $conv = 0.001$, $hd = 3$, $r = \frac{n}{4k^2}$, and balancing function that always returns 5, 2) $conv = 0.001$, $hd = 3$, $r = \frac{n}{4k^2}$, and balancing function that always returns 10. The parameter r depicts the number of disjoint, distinct extended neighborhoods in the dataset². Namely, if a dataset does not contain hubness phenomenon, each point has around $2k$ direct and reverse neighbors. Since the extended neighborhood contains neighbors of neighbors, its size is $4k^2$. Ruffly said, a dataset then has $\frac{n}{4k^2}$ distinct extended neighborhoods. Therefore, with such parameter r , a point gets a chance to find its best neighborhood among nearly all dataset’s extended neighborhoods.

In the further text we will denote naive partial k -NNG update algorithm by *naive*, *NN-Descent* by *nndes*, the first setting of *online RW-Descent* by *orwdes-5*, the second setting by *orwdes-10*, the first setting of *NW-Descent* by *onwdes-5*, and the second one by *onwdes-10*.

Talking about initial algorithm setting, we paired each of the update algorithms with a different initial algorithm. The initial algorithm that we paired with the naive partial k -NNG update was the regular naive brute-force k -NNG construction algorithm. *NN-Descent* was paired with *NN-Descent* itself—we used it both for initial and for update algorithm, by configuring it in the same way in both cases. Finally, with *online RW-Descent* we used *RW-Descent*, and with *online NW-Descent* we used *NW-Descent* as the initial algorithm. Moreover, *RW-Descent* and *NW-Descent* in a role of the initial algorithm were configured analogously to the algorithms they were paired with.

For the sliding window size we used values 10 and 50. Due to (6.1), for both sw values, we could not use datasets AllGestureWiimoteX, AllGestureWiimoteY and AllGestureWiimoteZ, while for $sw = 50$ we additionally could not use datasets Chinatown, Crop, ItalyPowerDemand, MelbournePedestrian, PickupGestureWiimoteZ, ShakeGestureWiimoteZ and SmoothSubspace, because of

¹Note that DTW is not actually a real distance function, since it does not satisfy triangle inequality condition.

²The notion of disjoint extended neighborhood is very vague in this context, since we do not refer here to any formal graph substructure (e.g. strongly connected component). Here we are assuming artificial linear division of graph nodes into disjoint subsets of size $4k^2$, that we refer to as extended neighborhoods.

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

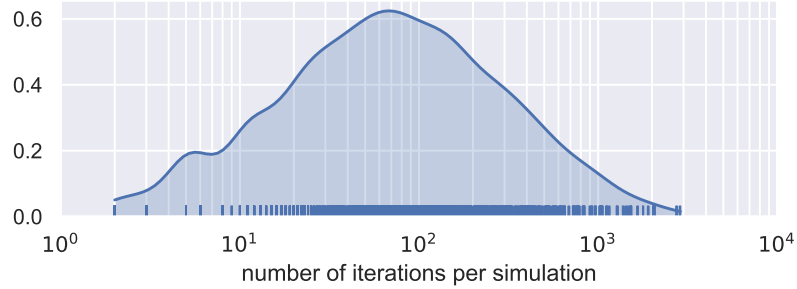


Figure 6.3: Distribution of simulation iterations counts.

their ts_{min} value. For the batch size b we used values $0.5 \cdot sw$ and sw , while for the points count p we used $0.2 \cdot n$ and $0.5 \cdot n$, where n is dataset size.

The overview of all values used for all the parameters is given in Table 6.4. The total number of simulations per dataset were 192, except for the 7 aforementioned datasets (which were run only for $sw = 10$) where the number of simulations per dataset was 96. This leads to a total of 23,328 simulations that were run inside this experiment. In each simulation the four examined algorithms were run inside each iteration. The exact number of iterations depends on simulation parameters and time series length. The histogram of number of iterations among all simulations that were run, is given in Figure 6.3 (note that the x axis of the figure is logarithmic). As can be seen, the simulation iterations counts are spread between 1 and 3,000, most of them being around 70.

Table 6.4: Simulation parameters values that were used in experiments.

parameter	values
algorithms	(total: 6) <i>naive</i> , <i>nndes</i> , <i>orwdes-5</i> , <i>orwdes-10</i> , <i>onwdes-5</i> , <i>onwdes-10</i> .
dataset	(total: 125 for $sw = 10$ and 118 for $sw = 50$) Datasets from Table 6.2.
k	(total: 2) 5, 10
$dist$	(total: 2) L_2 , DTW
sw	(total: 2) 10, 50
b	(total: 2) $0.5 \cdot sw$, sw
p	(total: 2) $0.2 \cdot n$, $0.5 \cdot n$

The algorithms are evaluated by using three different measures: recall (see (2.2)), scan rate (see (2.3)) and harmonic mean (see (2.5)). A good algorithm should maximize recall and harmonic mean, while minimizing scan rate.

6.3.4 Results and discussion

In this subsection we will present results produced by all the simulations. The main focus of this analysis is to compare the four approaches. The first part will be devoted to the overall results. In the second part we will investigate the influence of different parameters on the algorithms’ performance. At the end, we will analyse the simulation flow, with the main focus on determining if the results change over time. Namely, there is a possibility that the recall of *online RW-Descent* and *online NW-Descent* decreases over time, as a consequence of the accumulated error.

The overall averages of the recall, scan rate and harmonic mean for all the four approaches are presented in Table 6.5 and Figure 6.4. The presented averages were obtained in the following way. Firstly, the averages for each simulation are calculated—the average values for a simulation is calculated upon the results of all simulation’s iterations. Then, the values for each simulation were used for the final average values presented in the table and the figure.

Table 6.5: Average recall, scan rate and harmonic mean for all the presented k -NNG update approaches.

	naive	nndes	orwdes-5	orwdes-10	onwdes-5	onwdes-10
recall	1	0.98	0.84	0.91	0.78	0.86
scan rate	0.91	1.06	0.13	0.2	0.16	0.23
harmonic	0.16	0.43	0.84	0.84	0.79	0.8

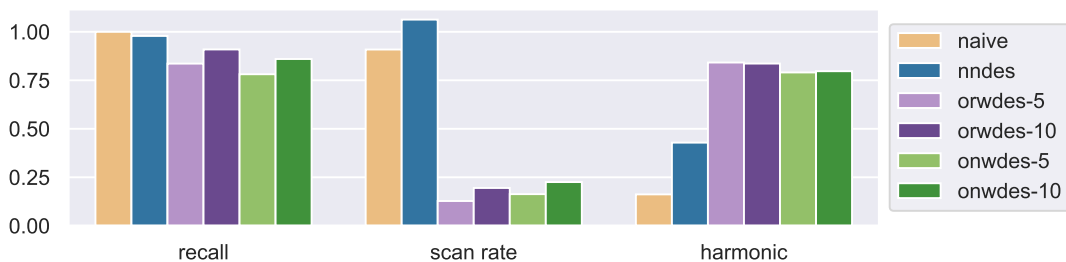


Figure 6.4: Average recall, scan rate and harmonic mean for all the presented k -NNG update approaches.

The average recall value for naive k -NNG update algorithm is the highest possible, since this algorithm is not an approximation algorithm. For the case of *NN-Descent* the average recall is 0.98, which is very high, as expected. As we already discussed, *NN-Descent* produces highly accurate approximations in datasets

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

with low ID. Having that the values in a time series are usually highly correlated, intrinsic dimensionality of time series datasets is small, which justifies high average recall value of *NN-Descent*. For the case of *online RW-Descent*, average recalls are 0.84 for parameter configuration *orwdes-5*, and 0.91 for *orwdes-10*. These two average recall values are very high as well, especially for the case of *orwdes-10*. However, they are smaller than the average recall obtained by *NN-Descent*. *Online NW-Descent* performs slightly worse than *online RW-Descent* in terms of average recall. It reported average recall of 0.78 for parameter configuration *onwdes-5*, and 0.86 for *onwdes-10*. As a conclusion, the accuracy is, naturally, the highest for the naive *k*-NNG update algorithm, then for *NN-Descent*, which produces extremely high recall values, then for *online RW-Descent*, and it is lowest for *online NW-Descent*. Moreover, the worst obtained average recall value (obtained by *online NW-Descent*) is still reasonably high, which means that all the approaches are reasonably accurate.

Talking about the average scan rate values, it is evident that the *online RW-Descent* and *online NW-Descent* significantly reduce the number of distance calculations. The average scan rate of naive *k*-NNG update algorithm is very high, amounting to 0.91. *NN-Descent* reports scan rate that is even higher than 1, which is caused by the small-sized datasets. Namely, *NN-Descent* reports very high scan rates for very small datasets, and these values highly influence the average. However, as we will discuss later, *NN-Descent*'s scan rates drastically decrease with the increase of dataset size. Finally, *online RW-Descent* and *online NW-Descent* report very low average scan rates. Moreover, *online RW-Descent* reports smaller average scan rates than *online NW-Descent*, which we will additionally discuss later. For the parameter settings *orwdes-10* and *onwdes-10* average scan rate values are naturally higher than for the *orwdes-5* and *onwdes-5*. This increase in scan rate values is followed by the increase in recall values.

Finally, let us analyze average harmonic mean values, that take into account both recall and scan rate values. Average harmonic mean is evidently highest for the *online RW-Descent* and *online NW-Descent* algorithms. The worst harmonic mean is, of course, associated with naive *k*-NNG update algorithm. *NN-Descent* is considerably better than naive approach, but its average harmonic mean is still much lower than the harmonic means of the online algorithms. What might seem unexpected is that, while having higher recall and lower scan rate than *NN-Descent*, naive *k*-NNG update algorithm has lower harmonic mean. The reason for this is that the harmonic mean treats equally all the scan rate values equal or higher than 1. Consequently, extremely high scan rate values of a few small

6.3. METHODS EVALUATION

datasets do not influence harmonic mean at great extent. Regarding the online algorithms, as said, both report high average harmonic mean, *online RW-Descent* being better than *online NW-Descent*. Interestingly, the harmonic means of different parameter configurations of *online RW-Descent*, and also of *online NW-Descent*, are the same, or almost the same—both *orwdes-5* and *orwdes-10* have average harmonic mean of 0.84, while *onwdes-5* and *onwdes-10* have 0.79 and 0.8, respectively. This implies that the increased number of walks leads to nearly proportional increase of recall. Consequently, higher recall could easily be achieved at the cost of increased scan rate, simply by increasing the number of walks. However, there is an upper bound of the number of walks per point—there is no purpose for walks number to be higher than the number of a point’s extended neighborhood (which is around $4k^2$ for a dataset which does not have hubness phenomenon).

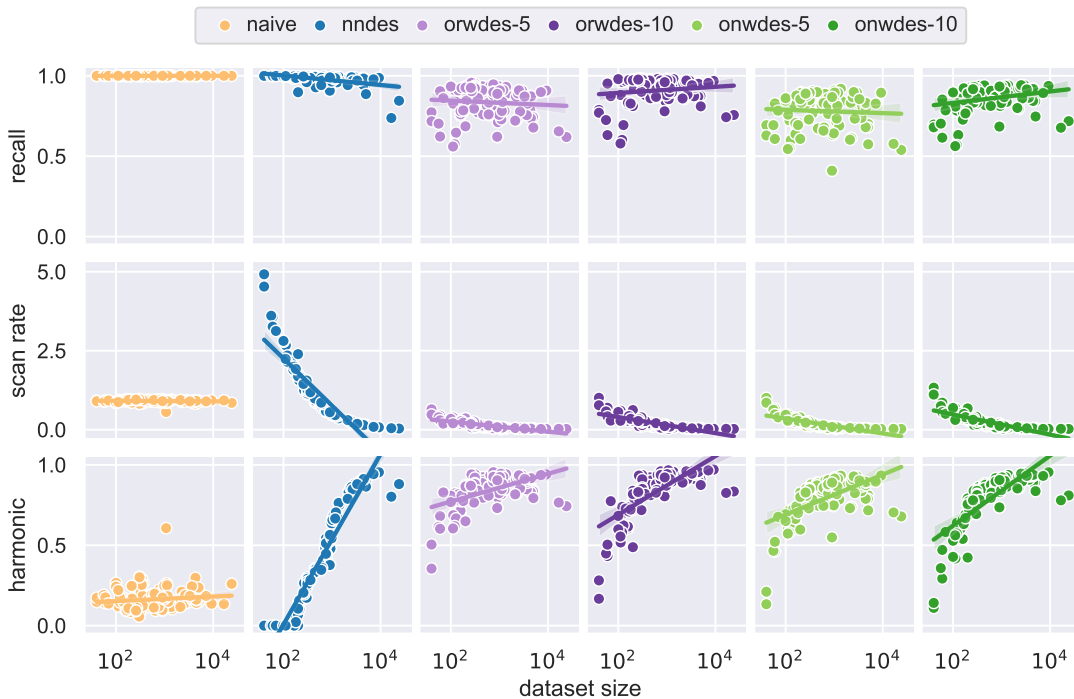


Figure 6.5: Influence of dataset size on performance of k -NNG update algorithms.

We already said that dataset size influences *NN-Descent*’s speedup. Let us now see if the same holds for *online RW-Descent* and *online NW-Descent*. In Figure 6.5 we show how recall, scan rate and harmonic mean are influenced by dataset size for all the four algorithms (note that the x axis of the figure is logarithmic). Each dot in the scatter plot represents a single dataset, while the plotted lines are regression lines. Result for a single dataset represents an average of all dataset’s simulations’ values. As can be seen, dataset size does not influence naive

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR K -NN GRAPH UPDATES

k -NNG update algorithm. Contrary to that, dataset size influences all the other algorithms, in certain extent. Dataset size influence is most evident for scan rate values of *NN-Descent*. For smaller datasets, *NN-Descent* produces extremely bad scan rate values, that go even up to 5, making the algorithm unusable. However, as dataset size gets higher, the scan rate exponentially decays. Talking about *NN-Descent*'s recall values, they also depend on dataset size, but at much lower extent. *NN-Descent*'s recall values decrease at slow rate as dataset size increases. For the online algorithms, scan rate values are also influenced by dataset size, but much less than for *NN-Descent*. Unlike with *NN-Descent*, scan rates that are higher than 1 are really rare with the online algorithms, especially with orwdes-5 and onwdes-5. Consequently, even for very small datasets, the online algorithms outperform naive k -NNG update algorithm in this matter. Recall values of orwdes-5 and onwdes-10 are not influenced by dataset size, while for the orwdes-10 and onwdes-10 a slight influence emerged—as dataset size gets higher, recall value gets higher as well. Finally, harmonic means summarize everything. Harmonic mean of *NN-Descent* exponentially grows, which is dictated by the exponential decay of scan rate value. For all the online algorithms, harmonic means also grow with dataset size; growth being more evident for orwdes-10 and onwdes-10. As a conclusion, the online algorithms are undoubtedly the best choice in small to medium sized datasets, while in the large ones, *NN-Descent* approaches the performance of the online algorithms.

As can be seen in Figure 6.5, dataset size does not significantly influence the online algorithms' recall values. However, some datasets still have higher, while the others have lower average recall. For that reason, we further investigated how dataset properties influence the algorithms. Before going any further, let us point out one important fact—larger the nodes changes are, harder it is for the online algorithms to find the nodes' new neighborhoods. The dataset property that dictates the extent of nodes changes, is time series variability. If the successive values in time series are very different, the node changes during the simulation are large, and an online algorithm is faced with a harder task. Figure 6.6 shows that time series variability indeed influences the online algorithms (note that the x axis of the figure is logarithmic). The dots in scatter plot are again representing individual datasets. The values on x axis are obtained in the following way. Each time series in dataset is differenced, meaning that its values become the differences between the old consecutive values. Afterwards, standard deviation is calculated on the differenced time series. Finally, average value of all time series' standard deviations is calculated, which results with a value shown on the figure's x axis.

6.3. METHODS EVALUATION

Moreover, to reduce the impact of dataset size, in Figure 6.6 we show only 20 datasets of similar sizes; more precisely, only datasets which size is in the range [800, 1200] are shown.

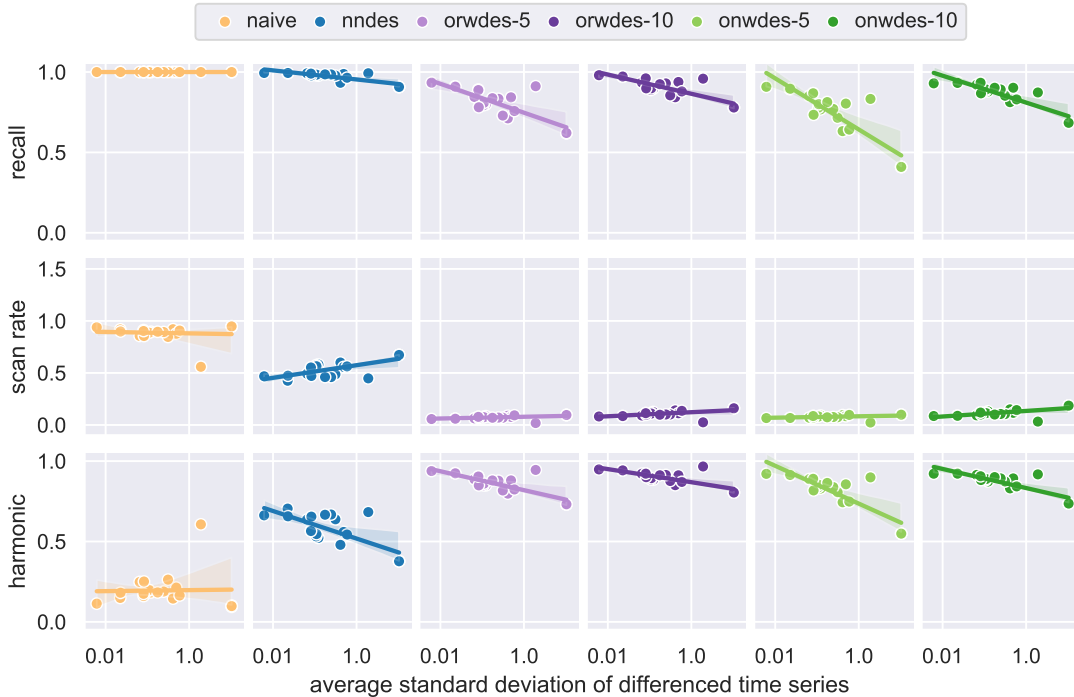


Figure 6.6: Influence of time series variability on performance of k -NNG update algorithms.

The influence of time series variability on the online algorithms’ recall values is quite evident—as the variability increases, recall decreases. The problem is even more evident for the parameter settings with lower number of walks, which are *orwdes-5* and *onwdes-5*. Moreover, it can also be seen that *online NW-Descent* is more sensitive to time series variability, than *online RW-Descent* is. Regarding the scan rate values, the influence is present, but not very strong. It is expressed as a slight increase of scan rate values as variability increases. One interesting phenomenon appeared in recall values of *NN-Descent*—increase of time series variability is followed by slight decrease of *NN-Descent*’s recall values. The reason for this is the higher intrinsic dimensionality of time series with high variability, which then negatively influences *NN-Descent*.

Let us now analyze the influence of the different parameters on the final results. We will start with the parameter k . The Figure 6.7 shows how k affects the algorithms. As expected, the influence of k on *NN-Descent*’s scan rate is very high. However, even though for the smaller k values *NN-Descent*’s scan rate is

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR K -NN GRAPH UPDATES

much smaller, it is still significantly higher than the scan rates obtained by the online algorithms. Additionally, increased k value negatively influences *online NW-Descent*—for larger k , *online NW-Descent*'s recall is smaller, while the scan rate is higher. The problem of the decreased recall can be solved by increasing the walks count. Actually, it is natural that the walks count depend on k —larger neighborhoods should be explored with more walks. If we look at the *onwdes-5* with $k = 5$, and *onwdes-10* with $k = 10$, we see that the increase of walks count for larger k does solve the problem, having that the recall is greater for the second case. On the other side, the scan rate increased due to a slower algorithm convergence. Talking about *online RW-Descent*, interestingly, the recall of this approach does not change with k , meaning that this approach does not need additional walks for larger neighborhoods. Moreover, scan rate of the *online RW-Descent* increases only slightly. As a conclusion, for smaller k values, *online NW-Descent* behaves very good—for *onwdes-10* parameter configuration, it even outperforms *RW-Descent* by achieving similar recall for the lower scan rate. However, for the larger k values, *online RW-Descent* is the best option.

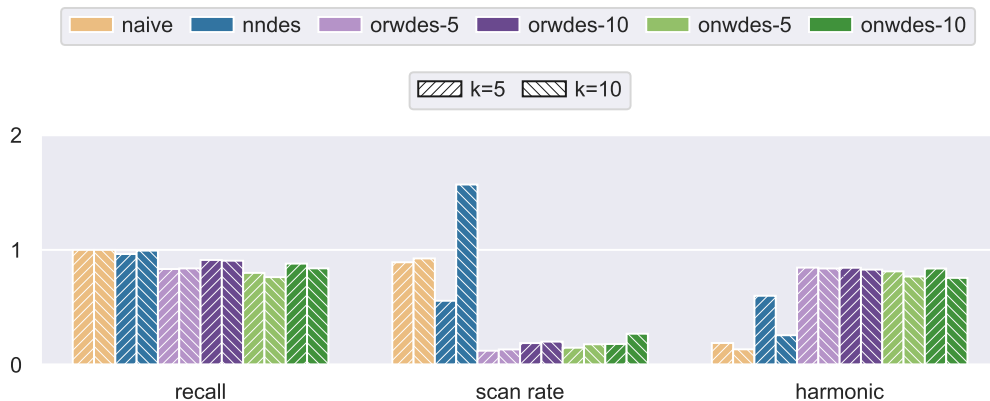


Figure 6.7: Influence of parameter k on performance of k -NNG update algorithms.

Figure 6.8 show how results depend on distance function. All the approximation algorithms have slightly lower recall for the case of DTW. Moreover, *NN-Descent* report slightly higher scan rate value for DTW, while the other approaches did not show dependence of scan rate on distance function. What is especially important to point out is that the *online NW-Descent* did not show significant decrease of recall values for DTW, even though it was originally designed only for L_2 distance function.

Dependence on simulation parameter sw is given in Figure 6.9. This parameter actually influences time series' dimensionalities. For that reason, all the approx-

6.3. METHODS EVALUATION



Figure 6.8: Influence of distance function on performance of k -NNG update algorithms.

imation algorithms reported slightly lower recall values for higher sw . For the scan rate values, the dependence is also consistent among all the approximation algorithms—all of them reported slight increase of scan rate values for higher sw .



Figure 6.9: Influence of simulation parameter sw on performance of k -NNG update algorithms.

Simulation parameters p and b both determine the extent to which data change inside a single simulation iteration. For that reason we presented these two parameters together in Figure 6.10. As can be seen, these parameters do not influence NN -Descent, since NN -Descent constructs k -NNG from the scratch, and therefore is not aware of data changes. Regarding the update algorithms, their recall values were expected to be negatively influenced by the increase of these two parameters, for the same reason they are negatively influenced by times series variability. Surprisingly, the experiments did not confirm this reasoning—online algorithms did not report decrease of recall values for higher values of parameters p and b . The only influence regarded these two parameters is that the increase of the parameter p negatively influences scan rate values, which must be the case since higher p value implies the update of NN lists of more dataset points.

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR K -NN GRAPH UPDATES

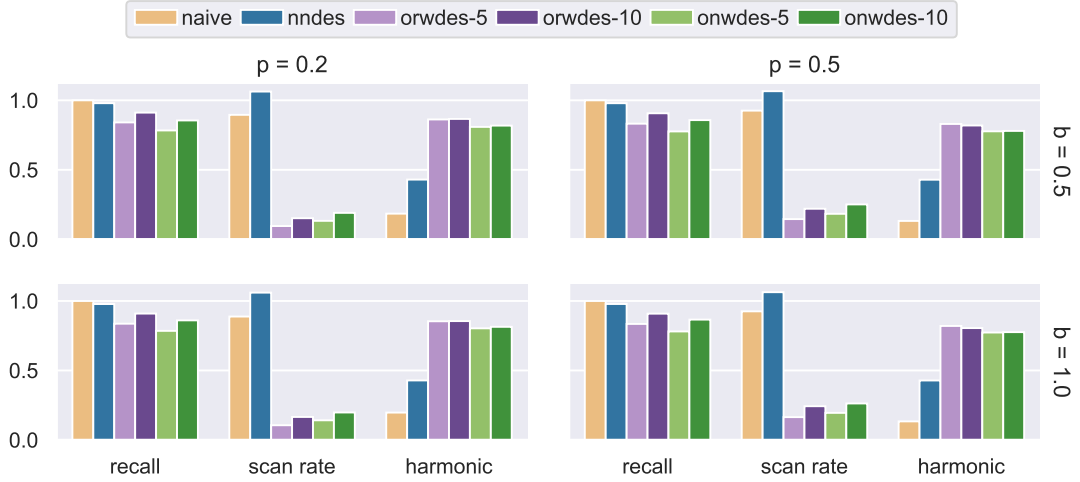


Figure 6.10: Influence of simulation parameters p and b on performance of k -NNG update algorithms.

Figure 6.11 shows how the performance of the algorithms changes among different iterations of the simulations. The data shown in the figure is created in the following way. Let i_{cnt} be the number of iterations in a simulation. Each simulation then produces i_{cnt} recall, scan rate and harmonic mean values—one value per each iteration. Let a *resulting array* of a simulation be an array that contains values of some measure (either recall, scan rate or harmonic mean) obtained for individual simulation’s iterations. Resulting arrays are then always of size i_{cnt} . Let $i_{\text{cnt}}^{\text{max}}$ be maximum i_{cnt} among all the simulations. We now increase sizes of resulting arrays of all the other simulations to $i_{\text{cnt}}^{\text{max}}$, by evenly inserting linearly interpolated values between each two successive array elements. Now that all simulations have evenly sized resulting arrays, we are able to create averaged resulting array for each measure. The averaged resulting array is such that the value on the index i represents an average of elements positioned as well on the index i in all the simulations’ resulting arrays. Finally the figure presents averaged resulting arrays for different measures and different algorithms.

First of all, let us notice that *NN-Descent*’s performance did not significantly change during the simulations execution. As already discussed, this is due to the fact that *NN-Descent* constructs k -NNGs independently in each iteration, and hence its results do not depend on the previously created k -NNG, nor on the node changes.

The second thing to notice is the decrease of scan rate values near the end of simulations execution, for all the algorithms except for *NN-Descent*. This decrease is caused only by the nature of the simulations. When a simulation approaches

6.3. METHODS EVALUATION

its end, sliding windows of many points are already very near, or exactly at their rightmost positions. That implies lower extent of point changes near the end of a simulation, which further implies lower scan rate values. Moreover, Figure 6.11 gives us a new insight into naive k -NNG update algorithm—its scan rate is actually very near to the 1 throughout simulation, while only at the end it significantly drops. Therefore, previously presented average scan rate values for this approach might be a bit misleading, having that the average scan rate is highly influenced by the small values appearing near the simulations' end. The same does not hold for *online RW-Descent* and *online NW-Descent*, because the drop of scan rate is not high for these algorithms, meaning that it does not influence their average values as much. Additionally, for these two algorithms, higher scan rates followed by a sudden drop can be detected at the beginning of the simulations. The reason for this is that these algorithms must create complete k -NNG approximation, while later they only update NN lists for real subsets of graph nodes.

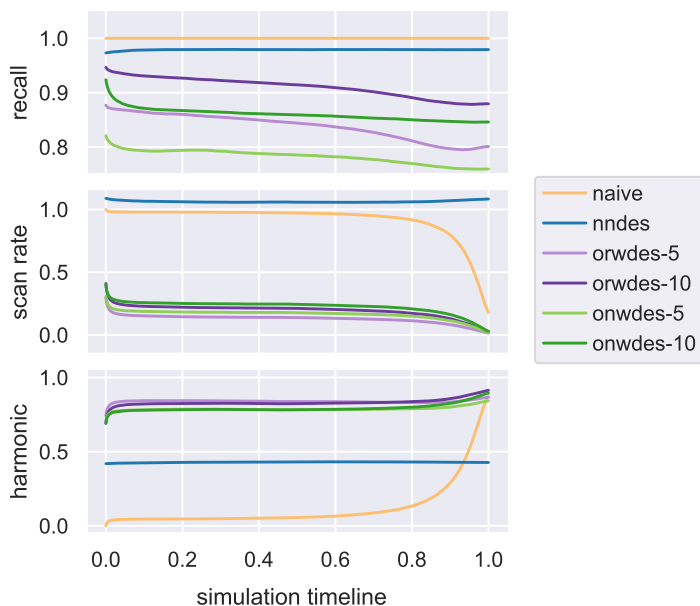


Figure 6.11: Performance of the k -NNG update algorithms during the simulations execution.

The main reason for the analysis of the simulation flow, presented in Figure 6.11, is to determine if *online RW-Descent* and *online NW-Descent* have a problem of accumulating error. Namely, these algorithms are very dependent on the previous k -NNG, so if the algorithms keep adding new erroneous neighbors, while preserving the old ones, the quality of the approximation will drop over time. As can be seen in the figure, the two algorithms do have this problem at

CHAPTER 6. *NN-DESCENT*-BASED APPROXIMATION ALGORITHMS FOR *K*-NN GRAPH UPDATES

some extent—the recall values decrease over time. Luckily, the decrease is not high. The problem is even less evident for *online NW-Descent* than for *online RW-Descent*—*online NW-Descent* has the initial high drop in recall values, which is not related to the accumulating error, while afterwards the recall values only slightly decrease over time. The reason for the initial drop is that the first *k*-NNG is created by iteratively improving a random graph, so the points are introduced with many different random neighborhoods. Afterwards, the algorithm, due to its nature which implies reduced randomness, more often fails to introduce points with their “good” neighborhoods. Even though the randomization phase helps, it still does not achieve the results obtained when starting from a random graph. Finally, even though the problem of accumulating error exists at some extent for both online algorithms, there is a way to deal with it. After a certain number of iterations, *k*-NNG can be generated from scratch, just like at the beginning of the simulations. After that, the accumulated error is eliminated, and the update process can be continued.

As a conclusion, the results of the *online RW-Descent* and *online NW-Descent* proved to be very competitive. While reducing the recall values slightly, these approaches introduce immense decrease of scan rate values. Moreover, *online RW-Descent* performed generally better than *online NW-Descent*, while *online NW-Descent* still has a benefit of lower accumulated error. The results also suggest that these algorithms are naturally more suitable when data change at lower extent. Moreover, the algorithms are performing much better than *NN-Descent* in small to medium sized datasets, while for large datasets *NN-Descent* performs equally well.

6.3. METHODS EVALUATION

Chapter 7

Conclusions

Having that k -NNG is used as a building block in many algorithms and problems, fast approximation algorithms for its construction are a very important topic. This thesis focused on such algorithms. We analyzed and explained the problems of one existing approximation algorithm called *NN-Descent*, and we proposed its modifications that overcome these problems to a certain extent. Additionally, we proposed two *NN-Descent*-based approximation algorithms for k -NNG updates.

In the first part of the thesis we analyzed the poor performance of *NN-Descent*, which produces highly inaccurate approximations on high-dimensional data. We showed that the poor performance of *NN-Descent* is related to the phenomenon called hubness. Hubness refutes basic *NN-Descent*'s assumption that two points sharing a neighbor are also likely to be neighbors. As a consequence, the algorithm performs poorly on data with hubness phenomenon. Additionally, we confirmed these statements with an experimental analysis.

In order to address this *NN-Descent*'s shortcoming, we proposed five different variants of the original algorithm: 1) hubness aware variant (*HA-NN-Descent*), 2) oversized NN list variant (*O-NN-Descent*), 3) random walk descent variant (*RW-Descent*), 4) nearest walk descent variant (*NW-Descent*) and 5) randomized *NN-Descent* variant (*R-NN-Descent*). To validate the results of the proposed algorithms, we conducted the experiments on six high-dimensional datasets, two of which are synthetic. The results of the experimental analysis show that all new *NN-Descent* variants achieve better recalls at the certain expense of increased scan rate. Moreover, the suitability of the individual algorithms depends on the nature of a problem setup—some algorithms might be suitable for one problem setup, but not for the other.

Since *HA-NN-Descent* increases scan rate values only slightly, it is most suitable when the execution time is highly important. Contrary to that, *O-NN-Descent* should be used when accuracy is more important than the execution time—this

7.1. DIRECTIONS FOR FUTURE WORK

approach achieves high increase of recall values, but with more evident increase of scan rates. *RW-Descent* and *NW-Descent* are very convenient when one needs to configure the amount of “work” on individual nodes’ neighborhood approximations. *RW-Descent* achieves slightly higher recall than *NW-Descent*, but it also has higher scan rate. Moreover, *NW-Descent* should be used in a similar manner as *HA-NN-Descent*, i.e. when execution time is highly important. Finally, *R-NN-Descent* is usually second best with respect to recall values, the only better algorithm being *O-NN-Descent*. However, this algorithm produces scan rate values that are usually higher than the ones obtained by *O-NN-Descent*, which makes it rarely the best choice.

The second research direction of this thesis were approximation algorithms for k -NNG updates. To the best of our knowledge, there are not many algorithms for k -NNG updates—the focus of the literature are the algorithms that build k -NNG from scratch. In this thesis we proposed two online approximation algorithms for k -NNG updates: *online RW-Descent* and *online NW-Descent*. We conducted extensive simulation-based experimental analysis on time series data. The experiments show that these two algorithms outperform brute-force update algorithm and *NN-Descent*. In general, *online RW-Descent* and *online NW-Descent* report slightly lower recall values than *NN-Descent*, but the decrease of scan rate values is significant, especially for the smaller datasets.

7.1 Directions for future work

The algorithms presented in this thesis can be further explored and improved in number of ways. In this section we will present some of them.

NN-Descent’s sampling technique can be improved by replacing random selection of $\rho \cdot k$ points by some more appropriate selection that makes certain assessments and chooses $\rho \cdot k$ best candidates. Since *O-NN-Descent* is advised to be used together with the sampling technique, the improved sampling would improve *O-NN-Descent* results as well.

In *HA-NN-Descent* algorithm, the replacement probability relies on linear mapping of hubness values to the interval $[0, 1]$. However, some non-linear mappings, such as logarithmic transformations, could also be used instead. Therefore, various mappings can be explored, in order to get the best performing one.

In our experiments we used the simplest possible balancing function for *RW-Descent* and *NW-Descent*. This balancing function assigns the same number of walks to all dataset points. However, these two algorithms might be significantly

CHAPTER 7. CONCLUSIONS

improved with a better balancing function that should probably rely on hubness values approximations of dataset points. Therefore, another direction for future work would be to explore different balancing functions.

In this thesis, in Chapter 5, we analyzed all the proposed algorithms separately, even though some of the approaches are complementary and could be combined together. By combining multiple approaches together, one might achieve even better results. For example, *HA-NN-Descent*, *O-NN-Descent* and *R-NN-Descent* could be all used together, each having its own role—*HA-NN-Descent* would make sure that hubs are not being extensively compared with other points, *O-NN-Descent* would increase NN lists, aiming for a higher recall, while *R-NN-Descent* would try to place points in their "good" initial neighborhoods. Moreover, *RW-Descent*, as well as *NW-Descent*, could be combined with *O-NN-Descent* and *R-NN-Descent*—it would be interesting to analyze how *RW-Descent* and *NW-Descent* behave in extended neighborhoods introduced in *O-NN-Descent*, and also to see whether the randomization phase introduced in *R-NN-Descent* would be beneficial.

One additional research direction is theoretical analyses of the proposed algorithms. Such analyses would give more insights into the performance of the algorithms.

Finally, regarding the online algorithms for k -NNG updates, there is a possibility to improve these algorithms by using a balancing function that assigns more walks and random comparisons to the points that changed more.

7.1. DIRECTIONS FOR FUTURE WORK

Appendix A

Console application for algorithms related to k -NN graphs

In order to conduct the experiments presented in this thesis, we developed a C++ library for algorithms related to k -NNG. Moreover, we developed a console application that provides an easy way to use the library. In this appendix, we present the commands supported by the console application. Each command, together with its parameters, is passed to the console application through its arguments. The commands are described with the following properties: 1) command's name (*command*), 2) syntax of the command (*syntax*), 3) list of descriptions of each command's parameter (*params*), and 4) general command's description (*desc*).

Complete C++ source code of the library can be found on the GitHub repository <https://github.com/brankicabratric/knng>.

Command: `knng`

Syntax: `knng ds_path ds_type ds_inst_type out_path dist k`

Params: **ds_path** - Path to the dataset file.
ds_type - Dataset type. Possible values:
 `csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)
ds_inst_type - Type of dataset instances. Possible values:
 `point, timeseries`.
out_path - Path to the file where k -NN graph will be stored.
dist - Distance function. Possible values: `l2, dtw`.
k - Number of neighbors in k -NN graph.

Desc: Creates k -NN graph.

Command: `nndescent`

Syntax: `nndescent ds_path ds_type ds_inst_type out_path dist k
nndes_type (it_count | conv_ratio) [sampling]`

Params: `ds_path` - Path to the dataset file.

`ds_type` - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

`ds_inst_type` - Type of dataset instances. Possible values:

`point, timeseries.`

`out_path` - Path to the file where k-NN graph approximation will be stored.

`dist` - Distance function. Possible values: `l2, dtw.`

`k` - Number of neighbors in k-NN graph.

`nndes_type` - NN-Descent termination condition. Possible values: `"it"` (NN-Descent terminates when given number of iterations is reached), `"conv"` (NN-Descent terminates when number of updates is less than threshold).

`it_count` - This value is present when `nndes_type` is `"it"`.

It represents the number of algorithm's iterations.

`conv_ratio` - This value is present when `nndes_type` is `"conv"`. If there are less than `conv_ratio*n*k` updates of NN lists in most recent iteration, the algorithm terminates.

`sampling` - The portion of neighbors to be used in local joins. If not given, value 1 is assumed.

Desc: Creates k-NN graph approximation by using NN-Descent algorithm.

Command: `hanndescent`

Syntax: `hanndescent ds_path ds_type ds_inst_type out_path dist k
ha_nndes_type (it_count | conv_ratio) h_min h_max [sampling]`

Params: `ds_path` - Path to the dataset file.

`ds_type` - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

APPENDIX A. CONSOLE APPLICATION FOR ALGORITHMS RELATED TO K -NN GRAPHS

ds_inst_type - Type of dataset instances. Possible values: point, timeseries.

out_path - Path to the file where k-NN graph approximation will be stored.

dist - Distance function. Possible values: l2, dtw.

k - Number of neighbors in k-NN graph.

ha_nndes_type - Hubness aware NN-Descent termination condition. Possible values: "it" (HA-NN-Descent terminates when given number of iterations is reached), "conv" (HA-NN-Descent terminates when number of updates is less than threshold).

it_count - This value is present when ha_nndes_type is "it". It represents the number of algorithm's iterations.

conv_ratio - This value is present when ha_nndes_type is "conv". If there are less than conv_ratio*n*k updates of NN lists in most recent iteration, the algorithm terminates.

h_min - Minimum hubness value for replacement probability.

h_max - Maximum hubness value for replacement probability.

sampling - The portion of neighbors to be used in local joins. If not given, value 1 is assumed.

Desc: Creates k-NN graph approximation by using hubness aware NN-Descent algorithm.

Command: `osnndescent`

Syntax: `osnndescent ds_path ds_type ds_inst_type out_path dist k os_nndes_type (it_count | conv_ratio) k2 [sampling]`

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values: csv[delimiter]{label_index} (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: point, timeseries.

out_path - Path to the file where k-NN graph approximation will be stored.

dist - Distance function. Possible values: l2, dtw.

k - Number of neighbors in k-NN graph.

os_nndes_type - O-NN-Descent termination condition. Possible values: "it" (O-NN-Descent terminates when given number of iterations is reached), "conv"

(O-NN-Descent terminates when number of updates is less than threshold).

it_count - This value is present when `os_nndes_type` is "it". It represents number of algorithm's iterations.

conv_ratio - This value is present when `os_nndes_type` is "conv". If there are less than `conv_ratio*n*k` updates of NN lists in most recent iteration, the algorithm terminates.

k2 - Enlarged neighborhood size.

sampling - The portion of neighbors to be used in local joins. If not given, value 1 is assumed.

Desc: Creates k-NN graph approximation by using O-NN-Descent algorithm.

Command: `rwdescent`

Syntax: `rwdescent ds_path ds_type ds_inst_type out_path dist k rws_count rand_count rwdes_type (it_count | max_it_count) [conv_ratio] [rw_pr]`

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: `point`, `timeseries`.

out_path - Path to the file where k-NN graph approximation will be stored.

dist - Distance function. Possible values: `l2`, `dtw`.

k - Number of neighbors in k-NN graph.

rws_count - Random walks count. It is the number of random walks that start from each node (it is recommended to be some factor of k value).

rand_count - Number of random comparisons per point in the randomization phase.

rwdes_type - RW-Descent termination condition. Possible values: "it" (RW-Descent terminates when given number of iterations is reached), "conv" (RW-Descent terminates when number of updates of each node's NN list is less than threshold).

it_count - This value is present when `rwdes_type` is "it". It represents the number of algorithm's iterations.

APPENDIX A. CONSOLE APPLICATION FOR ALGORITHMS RELATED TO K -NN GRAPHS

max_it_count - This value is present when `rwdes_type` is "conv". It represents the upper bound on algorithm's iterations count. Namely, the algorithm terminates after `max_it_count` iterations, regardless of convergence.

conv_ratio - This value is present when `rwdes_type` is "conv". A point converges when there is in average less than `conv_ratio*rw_count` updates in a few recent iterations. The algorithm converges when all points converge.

rw_pr - Algorithm for edge traversal probabilities.
Possible values: uniform, edge_maturity.

Desc: Creates k -NN graph approximation by using RW-Descent algorithm.

Command: `nwdescent`

Syntax: `nwdescent ds_path ds_type ds_inst_type out_path dist k ws_count rand_count nwdes_type (it_count | max_it_count) [conv_ratio]`

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: point, timeseries.

out_path - Path to the file where k -NN graph approximation will be stored.

dist - Distance function. Possible values: l2, dtw.

k - Number of neighbors in k -NN graph.

ws_count - Walks count. It is the number of walks that start from each node (it is recommended to be some factor of k value).

rand_count - Number of random comparisons per point in the randomization phase.

nwdes_type - NW-Descent termination condition. Possible values: "it" (NW-Descent terminates when given number of iterations is reached), "conv" (NW-Descent terminates when number of updates of each node's NN list is less than threshold).

it_count - This value is present when `nwdes_type` is "it". It represents the number of algorithm's iterations.

max_it_count - This value is present when `nwdes_type` is "conv". It represents the upper bound on algorithm's iterations count. Namely, the algorithm terminates after `max_it_count` iterations, regardless of convergence.

conv_ratio - This value is present when `nwdes_type` is "conv". A point converges when there is in average less than `conv_ratio*nws_count` updates in a few recent iterations. The algorithm converges when all points converge.

Desc: Creates k-NN graph approximation by using NW-Descent algorithm.

Command: `rnnndescent`

Syntax: `rnnndescent ds_path ds_type ds_inst_type out_path dist k r_nndes_type (it_count | conv_ratio) r [sampling]`

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: `point`, `timeseries`.

out_path - Path to the file where k-NN graph approximation will be stored.

dist - Distance function. Possible values: `l2`, `dtw`.

k - Number of neighbors in k-NN graph.

r_nndes_type - Randomized NN-Descent termination condition. Possible values: "it" (Randomized NN-Descent terminates when given number of iterations is reached), "conv" (Randomized NN-Descent terminates when number of updates is less than threshold).

it_count - This value is present when `r_nndes_type` is "it". It represents number of algorithm iterations.

conv_ratio - This value is present when `r_nndes_type` is "conv". If there are less than `conv_ratio*n*k` updates of NN lists in most recent iteration, the algorithm terminates.

r - Number of random comparisons of a single point in the randomization phase.

sampling - The portion of neighbors to be used in local

APPENDIX A. CONSOLE APPLICATION FOR ALGORITHMS RELATED TO K -NN GRAPHS

joins. If not given, value 1 is assumed.
Desc: Creates k -NN graph approximation by using randomized NN-Descent algorithm.

Command: `partial_knng`
Syntax: `partial_knng ds_path ds_type ds_inst_type out_path dist k min_id max_id`
Params: **ds_path** - Path to the dataset file.
ds_type - Dataset type. Possible values:
 `csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)
ds_inst_type - Type of dataset instances. Possible values:
 `point, timeseries.`
out_path - Path to the file where partial k -NN graph will be stored.
dist - Distance function. Possible values: `l2, dtw.`
k - Number of neighbors in k -NN graph.
min_id - NN lists of points with id less than `min_id` will not be calculated.
max_id - NN lists of points with id greater than `max_id` will not be calculated.
Desc: Creates NN lists for all nodes whose id is a value in range `[min_id,max_id]`.

Command: `reduce_knng`
Syntax: `reduce_knng ds_path ds_type ds_inst_type knng_path out_path k`
Params: **ds_path** - Path to the dataset file.
ds_type - Dataset type. Possible values:
 `csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)
ds_inst_type - Type of dataset instances. Possible values:
 `point, timeseries.`
knng_path - Path to the file where original k -NN graph is stored.
out_path - Path to the file where reduced k -NN graph will be stored.

k - Number of neighbors in reduced k-NN graph.
Desc: Reduces k-NN graph to given k.

Command: **recall**

Syntax: recall real_knng_path approx_knng_path [out]

Params: **real_knng_path** - Path to the file where real k-NN graph is stored.

approx_knng_path - Path to the file where k-NN graph approximation is stored.

out - If present, the recall will be output to the file called the same as k-NNG approximation file, but with suffix "_recall". Possible values: "out".

Desc: Outputs the recall of the k-NN graph approximation.

Command: **simulation**

Syntax: simulation ds_path ds_type ds_inst_type sliding_window runs out k dist (batch_size | (batch_size_min batch_size_max [points_min points_max]))

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values:

csv[delimiter]{label_index} (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: timeseries.

sliding_window - Sliding window size.

runs - Number of simulation runs.

out - If value "cout" is given, the results will be written in the standard output. Otherwise, the file path should be given, in which case the results will be written in that file. The results of a single iteration are written in a single line, formatted in the following way: "time,dist_calcs{;time,dist_calcs,recall}", where curly braces are not a part of the output, but denote repeating part (each repeated string represents results of one approximation algorithm). The part before curly braces represents results of exact k-NN graph construction.

k - k value of the k-NN Graph.

dist - Distance function. Possible values: l2, dtw.

batch_size - The size of the batch that will be loaded into time series during the updates.

APPENDIX A. CONSOLE APPLICATION FOR ALGORITHMS RELATED TO K -NN GRAPHS

batch_size_min - When this value is given, the size of the batch that will be loaded into time series during the updates is randomly chosen. This value is the lower bound of randomly chosen batch value.

batch_size_max - When this value is given, the size of the batch that will be loaded into time series during the updates is randomly chosen. This value is the upper bound of randomly chosen batch value.

points_min - When this value is given, the number of points that will be updated during one update is randomly chosen. This value is then lower bound of number of updated points.

points_max - When this value is given, the number of points that will be updated during one update is randomly chosen. This value is then upper bound of number of updated points.

Desc: Executes simulation whose purpose is to evaluate performance of approximation algorithms for k -NN graph updates.

Command: `prepare_simulation`

Syntax: `prepare_simulation ds_path ds_type ds_inst_type sliding_window runs k dist (batch_size | (batch_size_min batch_size_max [points_min points_max]))`

Params: **ds_path** - Path to the dataset file.

ds_type - Dataset type. Possible values:

`csv[delimiter]{label_index}` (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: `timeseries`.

sliding_window - Sliding window size.

runs - Number of simulation runs.

k - k value of the k -NN Graph.

dist - Distance function. Possible values: `l2`, `dtw`.

batch_size - The size of the batch that will be loaded into time series during the updates.

batch_size_min - When this value is given, the size of the batch that will be loaded into time series during the updates is randomly chosen. This value is the lower bound of randomly chosen batch value.

batch_size_max - When this value is given, the size of the batch that will be loaded into time series during the updates is randomly chosen. This value is the upper bound of randomly chosen batch value.

points_min - When this value is given, the number of points that will be updated during one update is randomly chosen. This value is then lower bound of number of updated points.

points_max - When this value is given, the number of points that will be updated during one update is randomly chosen. This value is then upper bound of number of updated points.

Desc: Does a preparation for a simulation by creating all k-NN graphs that are needed for calculation of approximations' recall values. If this command is not ran before command "simulation", command "simmulation" will construct these graphs itself. These graphs are stored in cache_sim folder, which is created for this purpose on the same location where executable file is stored.

Command: **clear_sim_cache**

Syntax: clear_sim_cache ds_name

Params: **ds_name** - Name of the dataset, which is considered to be the file name of the dataset, without extension.

Desc: Removes all simulation cache for the given dataset.

Command: **create_ds**

Syntax: create_ds ds_path ds_type ds_inst_type inst_cnt dim [min_val max_val]

Params: **ds_path** - Output path.

ds_type - Dataset type. Possible values:

csv[delimiter]{label_index} (Represents a CSV format. Delimiter should be given in square brackets. Square brackets can be omitted, in which case delimiter is comma. Label index is given in curly brackets. Curly brackets can also be omitted in which case it is assumed that the dataset is not labeled.)

ds_inst_type - Type of dataset instances. Possible values: point, timeseries.

inst_cnt - Number of instances.

dim - Dataset dimensionality.

min_val (default -1) - Instances' values are generated randomly, minimum value being min_val.

max_val (default 1) - Instances' values are generated randomly, maximum value being max_val.

APPENDIX A. CONSOLE APPLICATION FOR ALGORITHMS RELATED TO K -NN GRAPHS

Desc: Creates new dataset by generating values randomly (by using uniform distribution).

Command: **script**

Syntax: script script_path

Params: **script_path** - Path of the script file.

Desc: Executes commands from the external script file.

Command: ;

Syntax: command {; command }

Params: **command** - Arbitrary command with all its belonging parameters.

Desc: All given commands will be executed one after the other.

Command: **&&**

Syntax: command {&& command }

Params: **command** - Arbitrary command with all its belonging parameters.

Desc: Executes commands one by one. The current command is executed only if the preceding command has executed successfully.

Command: **||**

Syntax: command {|| command }

Params: **command** - Arbitrary command with all its belonging parameters.

Desc: Executes commands one by one. The current command is executed only if the preceding command has executed unsuccessfully.

Command: **help**

Syntax: help [command]

Params: **command** - Arbitrary command.

Desc: Outputs help. If "command" parameter is not present, all available commands will be listed together with their descriptions. Otherwise, if command parameter is present, only description of that command will be shown.



Appendix B

Detailed results of the experiments related to k -NN graph update algorithms

In Chapter 6 we proposed two algorithms for k -NNG update. The algorithms were evaluated by the extensive experimental analysis. In this appendix we show more detailed insight into the results of the experiments. Table B.1 present average recall and scan rate values for each dataset and each examined algorithm.

Resulting files of all conducted simulations can be found on the GitHub repository: <https://github.com/brankicabratric/knng-update-experiments>.

Table B.1: Average simulations results per dataset.

dataset	naive		nndes		orwdes-5		orwdes-10		onwdes-5		onwdes-10	
	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr
ACSF1	1.0	0.95	1.0	1.62	0.91	0.23	0.97	0.35	0.88	0.27	0.92	0.35
Adiac	1.0	0.87	0.99	0.61	0.84	0.08	0.93	0.12	0.83	0.09	0.91	0.12
ArrowHead	1.0	0.92	1.0	1.63	0.92	0.2	0.97	0.31	0.9	0.25	0.93	0.34
Beef	1.0	0.92	1.0	3.57	0.9	0.44	0.95	0.69	0.83	0.66	0.85	0.85
BeetleFly	1.0	0.92	1.0	4.92	0.77	0.64	0.79	1.01	0.69	1.01	0.7	1.33
BirdChicken	1.0	0.9	1.0	4.53	0.72	0.48	0.77	0.77	0.63	0.85	0.68	1.11
BME	1.0	0.94	0.99	1.82	0.78	0.21	0.83	0.32	0.71	0.27	0.77	0.36
Car	1.0	0.92	1.0	2.32	0.84	0.25	0.88	0.39	0.72	0.34	0.78	0.44
CBF	1.0	0.92	0.93	0.6	0.71	0.09	0.84	0.14	0.63	0.09	0.81	0.15
Chinatown	1.0	0.84	0.99	1.18	0.77	0.16	0.88	0.24	0.79	0.18	0.89	0.25
ChlorineConcentration	1.0	0.82	0.99	0.15	0.79	0.02	0.89	0.03	0.8	0.02	0.89	0.03
CinCECGTorso	1.0	0.95	0.99	0.37	0.89	0.05	0.96	0.07	0.84	0.06	0.91	0.08
Coffee	1.0	0.89	1.0	3.61	0.7	0.39	0.72	0.58	0.69	0.63	0.7	0.75
Computers	1.0	0.94	0.97	0.83	0.89	0.06	0.94	0.1	0.8	0.08	0.87	0.12
CricketX	1.0	0.93	0.98	0.64	0.85	0.09	0.93	0.13	0.78	0.1	0.89	0.15
CricketY	1.0	0.93	0.98	0.64	0.84	0.09	0.93	0.13	0.77	0.1	0.89	0.15
CricketZ	1.0	0.93	0.98	0.64	0.84	0.09	0.93	0.14	0.78	0.1	0.89	0.15
Crop	1.0	0.85	0.84	0.04	0.62	0.02	0.76	0.02	0.54	0.02	0.72	0.02
DiatomSizeReduction	1.0	0.88	0.99	1.15	0.87	0.13	0.94	0.2	0.87	0.16	0.91	0.2
DistalPhalanxOutlineAgeGroup	1.0	0.89	0.98	0.85	0.8	0.11	0.89	0.17	0.79	0.12	0.88	0.18
DistalPhalanxOutlineCorrect	1.0	0.9	0.98	0.58	0.8	0.08	0.9	0.12	0.77	0.09	0.88	0.13
DistalPhalanxTW	1.0	0.89	0.98	0.85	0.8	0.11	0.89	0.17	0.79	0.12	0.88	0.18
DodgerLoopDay	1.0	0.91	0.98	2.28	0.76	0.3	0.83	0.49	0.7	0.37	0.81	0.57
DodgerLoopGame	1.0	0.91	0.98	2.28	0.75	0.3	0.84	0.49	0.7	0.37	0.81	0.58

Continued on the next page.

Table B.1: Average simulations results per dataset (continued).

dataset	naive		nndes		orwdes-5		orwdes-10		onwdes-5		onwdes-10	
	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr
DodgerLoopWeekend	1.0	0.91	0.98	2.28	0.76	0.3	0.82	0.48	0.71	0.37	0.78	0.56
Earthquakes	1.0	0.93	0.93	0.89	0.76	0.14	0.86	0.22	0.67	0.14	0.82	0.23
ECG200	1.0	0.93	0.99	1.77	0.88	0.27	0.96	0.41	0.86	0.32	0.92	0.45
ECG5000	1.0	0.86	0.96	0.14	0.74	0.03	0.87	0.04	0.7	0.03	0.86	0.04
ECGFiveDays	1.0	0.88	0.99	0.56	0.84	0.08	0.94	0.11	0.8	0.08	0.9	0.12
ElectricDevices	1.0	0.93	0.74	0.05	0.65	0.03	0.74	0.03	0.58	0.03	0.68	0.03
EOGHorizontalSignal	1.0	0.95	0.99	0.61	0.92	0.06	0.97	0.09	0.83	0.08	0.86	0.1
EOGVerticalSignal	1.0	0.95	0.99	0.61	0.92	0.06	0.97	0.09	0.84	0.08	0.87	0.11
EthanolLevel	1.0	0.94	0.99	0.47	0.93	0.06	0.98	0.08	0.91	0.07	0.93	0.09
FaceAll	1.0	0.89	0.95	0.29	0.72	0.05	0.86	0.07	0.61	0.05	0.79	0.07
FaceFour	1.0	0.93	0.99	2.67	0.83	0.38	0.88	0.6	0.76	0.47	0.81	0.68
FacesUCR	1.0	0.88	0.95	0.28	0.73	0.05	0.86	0.07	0.61	0.05	0.8	0.07
FiftyWords	1.0	0.91	0.99	0.55	0.89	0.08	0.96	0.11	0.87	0.09	0.93	0.12
Fish	1.0	0.9	1.0	1.12	0.9	0.14	0.97	0.2	0.9	0.16	0.94	0.21
FordA	1.0	0.92	0.95	0.15	0.74	0.03	0.87	0.04	0.68	0.04	0.84	0.05
FordB	1.0	0.92	0.94	0.16	0.74	0.04	0.87	0.05	0.68	0.04	0.84	0.05
FreezerRegularTrain	1.0	0.91	0.98	0.19	0.88	0.03	0.95	0.04	0.83	0.03	0.9	0.04
FreezerSmallTrain	1.0	0.91	0.98	0.2	0.88	0.03	0.95	0.04	0.83	0.03	0.9	0.04
Fungi	1.0	0.93	1.0	1.66	0.92	0.21	0.97	0.31	0.88	0.26	0.92	0.35
GestureMidAirD1	1.0	0.83	1.0	1.14	0.93	0.1	0.98	0.16	0.89	0.13	0.91	0.18
GestureMidAirD2	1.0	0.85	1.0	1.14	0.93	0.1	0.98	0.16	0.88	0.13	0.9	0.19
GestureMidAirD3	1.0	0.85	1.0	1.11	0.94	0.09	0.98	0.14	0.88	0.12	0.89	0.16
GesturePebbleZ1	1.0	0.82	0.99	1.32	0.9	0.14	0.96	0.22	0.84	0.17	0.91	0.25
GesturePebbleZ2	1.0	0.82	0.99	1.32	0.9	0.14	0.96	0.22	0.84	0.17	0.91	0.25
GunPoint	1.0	0.94	1.0	1.63	0.93	0.18	0.98	0.26	0.89	0.23	0.92	0.3
GunPointAgeSpan	1.0	0.93	1.0	0.87	0.93	0.09	0.98	0.13	0.89	0.11	0.91	0.14
GunPointMaleVersusFemale	1.0	0.94	1.0	0.87	0.93	0.09	0.98	0.13	0.89	0.11	0.91	0.14
GunPointOldVersusYoung	1.0	0.94	1.0	0.87	0.93	0.09	0.98	0.13	0.89	0.11	0.91	0.14
Ham	1.0	0.91	0.99	1.7	0.88	0.24	0.96	0.37	0.87	0.28	0.93	0.4
HandOutlines	1.0	0.94	0.99	0.36	0.93	0.05	0.98	0.07	0.92	0.05	0.94	0.07
Haptics	1.0	0.91	1.0	0.9	0.91	0.11	0.97	0.16	0.9	0.12	0.93	0.17
Herring	1.0	0.9	1.0	2.23	0.65	0.2	0.69	0.32	0.6	0.29	0.63	0.36
HouseTwenty	1.0	0.95	0.99	1.9	0.78	0.18	0.83	0.29	0.64	0.25	0.73	0.36
InlineSkate	1.0	0.95	1.0	0.67	0.94	0.07	0.98	0.1	0.89	0.09	0.9	0.12
InsectEPGRegularTrain	1.0	0.94	1.0	1.12	0.95	0.07	0.98	0.1	0.82	0.11	0.83	0.14
InsectEPGSmallTrain	1.0	0.94	1.0	1.25	0.96	0.07	0.98	0.11	0.83	0.12	0.83	0.15
InsectWingbeatSound	1.0	0.92	0.99	0.27	0.87	0.05	0.95	0.06	0.83	0.05	0.92	0.07
ItalyPowerDemand	1.0	0.85	0.98	0.49	0.73	0.07	0.85	0.1	0.72	0.07	0.85	0.1
LargeKitchenAppliances	1.0	0.94	0.98	0.57	0.95	0.04	0.97	0.05	0.86	0.05	0.9	0.06
Lightning2	1.0	0.92	1.0	2.35	0.91	0.24	0.96	0.39	0.84	0.36	0.88	0.53
Lightning7	1.0	0.92	1.0	2.15	0.91	0.25	0.97	0.41	0.83	0.35	0.9	0.52
Mallat	1.0	0.9	0.99	0.24	0.89	0.04	0.96	0.05	0.87	0.04	0.93	0.05
Meat	1.0	0.9	1.0	2.15	0.57	0.19	0.6	0.29	0.55	0.28	0.57	0.33
MedicalImages	1.0	0.89	0.98	0.46	0.83	0.07	0.93	0.1	0.77	0.08	0.89	0.11
MelbournePedestrian	1.0	0.86	0.98	0.17	0.76	0.03	0.87	0.04	0.74	0.03	0.86	0.04
MiddlePhalanxOutlineAgeGroup	1.0	0.9	0.99	0.83	0.83	0.11	0.91	0.17	0.82	0.12	0.9	0.17
MiddlePhalanxOutlineCorrect	1.0	0.89	0.98	0.57	0.79	0.08	0.89	0.12	0.78	0.08	0.88	0.12
MiddlePhalanxTW	1.0	0.89	0.99	0.83	0.81	0.11	0.9	0.16	0.82	0.12	0.89	0.17
MixedShapesRegularTrain	1.0	0.92	0.99	0.2	0.91	0.04	0.97	0.05	0.9	0.04	0.94	0.05
MixedShapesSmallTrain	1.0	0.93	0.99	0.23	0.91	0.04	0.97	0.05	0.9	0.04	0.94	0.05
MoteStrain	1.0	0.91	0.96	0.43	0.78	0.07	0.9	0.11	0.69	0.08	0.84	0.11

Continued on the next page.

APPENDIX B. DETAILED RESULTS OF THE EXPERIMENTS RELATED TO K -NN GRAPH UPDATE ALGORITHMS

Table B.1: Average simulations results per dataset (continued).

dataset	naive		nndes		orwdes-5		orwdes-10		onwdes-5		onwdes-10	
	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr	rc	sr
NonInvasiveFetalECGThorax1	1.0	0.91	0.97	0.17	0.83	0.03	0.92	0.04	0.79	0.04	0.9	0.05
NonInvasiveFetalECGThorax2	1.0	0.91	0.98	0.17	0.85	0.03	0.94	0.04	0.81	0.03	0.91	0.04
OliveOil	1.0	0.9	1.0	3.26	0.62	0.3	0.63	0.42	0.61	0.48	0.62	0.56
OSULeaf	1.0	0.93	0.99	0.99	0.91	0.13	0.97	0.2	0.9	0.15	0.94	0.21
PhalangesOutlinesCorrect	1.0	0.89	0.97	0.24	0.78	0.04	0.88	0.06	0.73	0.04	0.86	0.06
Phoneme	1.0	0.94	0.94	0.3	0.72	0.05	0.86	0.08	0.6	0.05	0.8	0.08
PickupGestureWiimoteZ	1.0	0.85	1.0	2.79	0.94	0.29	0.98	0.47	0.85	0.41	0.89	0.6
PigAirwayPressure	1.0	0.95	1.0	1.15	0.94	0.1	0.98	0.15	0.85	0.14	0.86	0.18
PigArtPressure	1.0	0.96	1.0	1.22	0.93	0.15	0.98	0.23	0.89	0.19	0.92	0.26
PigCVP	1.0	0.96	0.99	1.27	0.92	0.16	0.98	0.24	0.89	0.19	0.92	0.27
PLAID	1.0	0.56	0.99	0.45	0.91	0.02	0.96	0.03	0.83	0.02	0.87	0.03
Plane	1.0	0.86	1.0	1.56	0.85	0.2	0.92	0.3	0.83	0.23	0.88	0.3
PowerCons	1.0	0.93	0.99	1.18	0.86	0.18	0.94	0.27	0.77	0.2	0.9	0.3
ProximalPhalanxOutlineAgeGroup	1.0	0.9	0.99	0.77	0.83	0.1	0.91	0.15	0.83	0.11	0.91	0.15
ProximalPhalanxOutlineCorrect	1.0	0.89	0.98	0.56	0.81	0.07	0.9	0.11	0.8	0.08	0.89	0.11
ProximalPhalanxTW	1.0	0.9	0.99	0.77	0.83	0.1	0.91	0.15	0.83	0.11	0.91	0.15
RefrigerationDevices	1.0	0.94	0.97	0.63	0.88	0.08	0.95	0.12	0.76	0.09	0.87	0.13
Rock	1.0	0.92	1.0	3.13	0.86	0.19	0.87	0.29	0.78	0.39	0.78	0.5
ScreenType	1.0	0.94	0.97	0.59	0.9	0.05	0.95	0.07	0.81	0.06	0.87	0.08
SemgHandGenderCh2	1.0	0.95	0.91	0.67	0.62	0.1	0.78	0.16	0.41	0.1	0.68	0.19
SemgHandMovementCh2	1.0	0.95	0.91	0.67	0.62	0.1	0.78	0.16	0.41	0.1	0.68	0.19
SemgHandSubjectCh2	1.0	0.95	0.91	0.67	0.62	0.1	0.78	0.16	0.41	0.1	0.68	0.19
ShakeGestureWiimoteZ	1.0	0.86	1.0	2.81	0.93	0.34	0.98	0.56	0.86	0.47	0.9	0.69
ShapeletSim	1.0	0.95	0.9	2.39	0.69	0.36	0.81	0.62	0.61	0.39	0.78	0.67
ShapesAll	1.0	0.92	0.99	0.43	0.91	0.06	0.97	0.09	0.89	0.07	0.94	0.09
SmallKitchenAppliances	1.0	0.94	0.96	0.59	0.9	0.04	0.94	0.06	0.82	0.05	0.87	0.07
SmoothSubspace	1.0	0.97	0.96	1.56	0.8	0.25	0.9	0.39	0.73	0.26	0.86	0.41
SonyAIBORobotSurface1	1.0	0.89	0.97	0.82	0.76	0.11	0.87	0.17	0.68	0.12	0.84	0.19
SonyAIBORobotSurface2	1.0	0.91	0.96	0.56	0.76	0.09	0.88	0.14	0.64	0.1	0.83	0.14
StarLightCurves	1.0	0.93	0.99	0.08	0.91	0.02	0.97	0.03	0.89	0.02	0.94	0.03
Strawberry	1.0	0.86	0.99	0.49	0.84	0.06	0.93	0.09	0.85	0.07	0.92	0.09
SwedishLeaf	1.0	0.85	0.98	0.47	0.78	0.07	0.9	0.1	0.73	0.07	0.87	0.1
Symbols	1.0	0.9	0.99	0.47	0.91	0.06	0.97	0.09	0.9	0.07	0.93	0.09
SyntheticControl	1.0	0.94	0.94	0.86	0.76	0.13	0.87	0.21	0.66	0.14	0.81	0.22
ToeSegmentation1	1.0	0.94	0.99	1.47	0.9	0.21	0.97	0.33	0.88	0.25	0.93	0.36
ToeSegmentation2	1.0	0.95	1.0	1.99	0.92	0.27	0.97	0.43	0.88	0.35	0.92	0.5
Trace	1.0	0.94	0.99	1.69	0.93	0.19	0.97	0.29	0.86	0.25	0.89	0.34
TwoLeadECG	1.0	0.9	0.99	0.46	0.84	0.07	0.92	0.1	0.81	0.07	0.9	0.1
TwoPatterns	1.0	0.92	0.89	0.14	0.68	0.04	0.81	0.05	0.57	0.04	0.75	0.05
UMD	1.0	0.94	0.99	1.92	0.79	0.22	0.83	0.34	0.69	0.28	0.77	0.4
UWaveGestureLibraryAll	1.0	0.92	0.98	0.15	0.86	0.03	0.94	0.04	0.8	0.03	0.91	0.04
UWaveGestureLibraryX	1.0	0.91	0.98	0.15	0.87	0.03	0.95	0.04	0.83	0.03	0.92	0.04
UWaveGestureLibraryY	1.0	0.91	0.98	0.15	0.87	0.03	0.95	0.04	0.83	0.03	0.92	0.04
UWaveGestureLibraryZ	1.0	0.91	0.98	0.15	0.87	0.03	0.95	0.04	0.83	0.03	0.92	0.04
Wafer	1.0	0.9	0.98	0.09	0.9	0.02	0.96	0.02	0.83	0.02	0.89	0.03
Wine	1.0	0.87	1.0	2.24	0.56	0.2	0.58	0.28	0.54	0.28	0.56	0.31
WordSynonyms	1.0	0.9	0.99	0.55	0.89	0.08	0.96	0.11	0.87	0.09	0.93	0.12
Worms	1.0	0.95	1.0	1.45	0.93	0.18	0.98	0.27	0.89	0.22	0.92	0.32
WormsTwoClass	1.0	0.95	1.0	1.45	0.93	0.18	0.98	0.27	0.89	0.22	0.92	0.31
Yoga	1.0	0.9	0.97	0.18	0.87	0.03	0.94	0.04	0.85	0.03	0.92	0.04

Prošireni izvod

Međusobno slični (bliski) objekti su u manjoj ili većoj meri sastavni deo mnogih problema. Pri rešavanju pomenutih problema vrlo često se koristi graf k najbližih suseda (k -NN graf) u ulozi strukture podataka koja modelira veze sličnosti između objekata [2, 3, 5, 10, 14, 24, 25, 32, 35, 44, 45, 60]. k -NN graf je usmeren graf čiji su čvorovi sami objekti. Svaki čvor je povezan usmerenim granama sa njegovih k najbližih suseda. Najjednostavniji način generisanja k -NN grafa jeste računanje distanci između svaka dva objekta, nakon čega se svakom objektu dodeli njegovih k najbližih suseda. Ovakav pristup podrazumeva $\binom{n}{2}$ računanja distanci, što uzrokuje kvadratnu vremensku složenost.

Postoje brojni algoritmi za generisanje k -NN grafova čiji je cilj smanjenje vremenske složenosti. Ovakvi algoritmi se mogu podeliti u tri klase. Prva klasa podrazumeva algoritme koji u problem uvode restrikcije povoljnih karakteristika, koje onda omogućavaju dalju optimizaciju [1, 2, 19, 39, 41, 50, 54, 55]. Drugi način optimizacije generisanja k -NN grafova je paralelizacija, te druga klasa predstavlja paralelne algoritme za generisanje k -NN grafova [11, 12, 15, 34, 45]. Treća klasa podrazumeva aproksimativne algoritme, koji za cilj imaju minimizovanje vremenske složenosti algoritma i maksimizovanje tačnosti finalne aproksimacije k -NN grafa [13, 20, 30, 42, 43, 51, 57, 59].

Upravo aproksimativni algoritmi za generisanje k -NN grafova su fokus ove teze. *NN-Descent* je jedan takav algoritam koji se pokazao vrlo efikasnim [20]. Zasnovan je na pretpostavci: „sused mog suseda je s velikom verovatnoćom i moj sused”. Ovaj algoritam počinje od slučajnog k -NN grafa, kog potom iterativno unapređuje. Unutar jedne iteracije, algoritam računa distance između tački koje imaju zajedničkog suseda, a potom te distance koristi za ažuriranje aproksimacije k -NN grafa. Iako vrlo efikasan u većini slučajeva, ovaj algoritam se nije dobro pokazao nad visokodimenzionalnim podacima.

Prvi pravac istraživanja ove teze podrazumeva detekciju i objašnjavanje razloga lošeg ponašanja *NN-Descent* algoritma nad podacima visoke dimenzionalnosti. Pokazali smo da je ovakvo ponašanje algoritma mahom uslovljeno fenomenom

zvanim *habnes* [8, 9, 48]. *Habnes* je jedna od posledica kletve dimenzionalnosti, koja podrazumeva postojanje čvorova izuzetno visokih ulaznih stepena. Čvorovi visokih ulaznih stepena umanjuju verovatnoću da dve tačke koje imaju zajedničkog suseda takođe budu susedi. Budući da je upravo ova verovatnoća srž pretpostavke *NN-Descent* algoritma, jasno je da *habnes* ima negativan uticaj. Sve pomenuto je u okviru teze potvrđeno eksperimentalnim analizama.

U okviru teze predstavili smo pet novih modifikacija *NN-Descent* algoritma, koje za cilj imaju ublažavanje gorepomenutog problema. Prva modifikacija je zasnovana na činjenici da čvorovi visokih ulaznih stepena imaju dobre aproksimacije najbližih suseda, dok su glavni problem zapravo čvorovi niskih ulaznih stepena, čije su aproksimacije susedstva vrlo loše. U skladu sa tim, algoritam za cilj ima da usmeri računarske resurse upravo na određivanje suseda čvorova niskih ulaznih stepena. Druga modifikacija uzima u obzir činjenicu da je tačnost *NN-Descent* algoritma veća za veće k vrednosti. Ideja je onda da se generiše k -NN graf za veću k vrednost, a da se potom redukuje na željenu k vrednost. Treća i četvrta modifikacija omogućavaju da se količina utrošenih računarskih resursa podešava na nivou pojedinačnih čvorova, što onda može dalje da se iskoristi tako da se više resursa dodeli čvorovima koji imaju loša susedstva. Peta, a ujedno i poslednja, modifikacija, zasnovana je na činjenici da pozicija čvora u inicijalnom slučajnom grafu igra veliku ulogu u tačnosti finalne aproksimacije susedstva. Stoga ova modifikacija sprovodi dodatna nasumična poređenja tački koje su u pogrešnom susedstvu u okviru inicijalnog grafa. Svih pet modifikacija *NN-Descent* algoritma su evaluirane eksperimentalnom analizom nad dva sintetička i četiri realna visokodimenzionalna skupa podataka.

Drugi pravac istraživanja je usmeren na problem ažuriranja k -NN grafa. Naime, podaci često imaju tendenciju da se menjaju vremenom. Zbog toga postoji potreba za algoritmima koji bi efikasno ažurirali k -NN graf nakon što se podskup njegovih čvorova promeni. Najjednostavniji način da se k -NN graf ažurira jeste da se iznova kreira nad novim podacima. Ovakav pristup svakako nije efikasan, budući da se podaci često samo parcijalno menjaju, te bi mnogo efikasnije bilo ažurirati samo relevantni deo grafa. Jedan način da se to realizuje jeste primenom metoda grube sile koji ponovo računa samo distance koje su afektovane izmenom podataka, a potom na osnovu novoizračunatih distanci ažurira graf. Problem ovog algoritma grube sile je što može biti nedovoljno brz, naročito u slučajevima kada izmena podataka obuhvata veliki broj čvorova.

U okviru ove teze, predstavili smo i dva aproksimativna algoritma za ažuriranje k -NN grafa, koji su zasnovani na *NN-Descent* algoritmu. Unutar oba algoritma

sprovode se kratke šetnje koje počinju od čvorova koji su izmenjeni. Potom se računaju distance između početnih i krajnjih čvorova ovih šetnji, a na osnovu tih distanci se onda i k -NN graf ažurira. Predloženi algoritmi su evaluirani obimnim eksperimentima nad vremenskim serijama. Eksperimenti zapravo predstavljaju simulacije realnog scenarija, u kom vremenske serije dobijaju nove vrednosti kako vreme prolazi.

Osnovni pojmovi i definicije

Graf k najbližih suseda (k -NN graf) G je usmeren graf čiji čvorovi predstavljaju objekte ulaznog skupa S , nad kojima je definisana neka funkcija razdaljine. Čvor $s \in S$ povezan je usmerenom granom sa čvorom $s' \in S$ samo ukoliko je čvor s' jedan od k najbližih čvorova čvoru s . U tom slučaju je s' sused čvora s , a s reverzni sused čvora s' . Listu svih suseda čvora s unutar grafa G nazvaćemo *NN (nearest neighbor) listom*, a označiti sa $NN_G(s)$; listu svih reverznih suseda nazvaćemo *R-NN (reverse nearest neighbor) listom*, a označiti sa $RNN_G(s)$. Za evaluaciju aproksimativnih algoritama za generisanje k -NN grafova, korišćićemo mere $\text{recall}_{\tilde{G}}$, $\text{scanrate}_{\tilde{G}}$ i $\text{harmonic}_{\tilde{G}}$. Prva navedena mera, tj. $\text{recall}_{\tilde{G}}$, ocenjuje tačnost aproksimacije \tilde{G} grafa G —veća vrednost sugerise veću tačnost aproksimacije; vrednost $\text{scanrate}_{\tilde{G}}$ direktno je proporcionalna broju izračunatih razdaljina, te su u ovom slučaju manje vrednosti poželjnije; $\text{harmonic}_{\tilde{G}}$ evaluira algoritam uzimajući u obzir i tačnost ($\text{recall}_{\tilde{G}}$) i broj izračunatih razdaljina ($\text{scanrate}_{\tilde{G}}$).

Habnes je aspekt kletve dimenzionalnosti koji se javlja u k -NN grafovima. Neka je $h_G(s) = |RNN_G(s)|$ *habnes vrednost* čvora s iz grafa G , $S_d \subset \mathbb{R}^d$ skup od n nasumice izabranih tački, i neka je G_d k -NN graf generisan nad S_d za neko fiksno k . Kako se dimenzionalnost d povećava, tako se distribucija vrednosti funkcije h_{G_d} značajno menja—neki čvorovi, koje ćemo nazvati *habovima*, nalaze se u velikom broju NN lista drugih čvorova. Habnes fenomen je onda u datom skupu podataka prisutan ukoliko taj skup podataka sadrži habove.

NN-Descent

NN-Descent [20] je brz, aproksimativni algoritam za generisanje k -NN grafova. Glavna pretpostavka ovog algoritma glasi: „sused mog suseda je s velikom verovatnoćom i moj sused”. Ovaj algoritam se može koristiti u kombinaciji sa bilo kojom funkcijom razdaljine; štaviše, može se koristiti i u kombinaciji sa funkcijom koja

ne zadovoljava neki od metričkih uslova, međutim, u tom slučaju tačnost aproksimacije zavisi od stepena usklađenosti funkcije sa gorepomenutom pretpostavkom algoritma.

Algoritam počinje od slučajnog k -NN grafa, koji se potom iterativno unapređuje. Unutar jedne iteracije se računaju razdaljine između svake dve tačke koje imaju zajedničkog suseda. To se realizuje tako što svaka tačka unutar jedne iteracije odigra ulogu *pivot tačke*. Za svaku pivot tačku se onda računaju razdaljine između svaka dva njena suseda. Na osnovu ovih razdaljina se aproksimacija k -NN grafa ažurira. *NN-Descent* iterira dok god se ne ispuni kriterijum za završetak algoritma, koji može biti definisan na jedan od sledeća dva načina: 1) izvršava se unapred zadati broj iteracija, nakon čega se algoritam završava, ili 2) algoritam se završava onda kada se u poslednjoj iteraciji aproksimacija k -NN grafa nije previše izmenila, u kom slučaju kažemo da je *NN-Descent* konvergirao.

NN-Descent u velikom broju slučajeva veoma efikasno proizvodi vrlo tačne aproksimacije k -NN grafa. Međutim, ipak postoje situacije u kojima *NN-Descent* ne daje dobre rezultate. Prvo, vremenska složenost algoritma ima kvadratnu zavisnost od parametra k . Stoga, za dovoljno veliko k , algoritam postaje sporiji od metoda grube sile. Drugi problem *NN-Descent* algoritma je taj što tačnost njegovih aproksimacija opada kako se dimenzionalnost ulaznih podataka povećava; te nad visokodimenzionalnim podacima algoritam daje veoma loše aproksimacije. Unutar ove teze ćemo se između ostalog baviti i drugim problemom *NN-Descent* algoritma—opisaćemo razloge koji stoje iza lošeg ponašanja algoritma nad visokodimenzionalnim podacima, a potom ćemo uvesti nekoliko novih njegovih verzija, čiji je cilj prvenstveno prevazilaženje pomenutog problema.

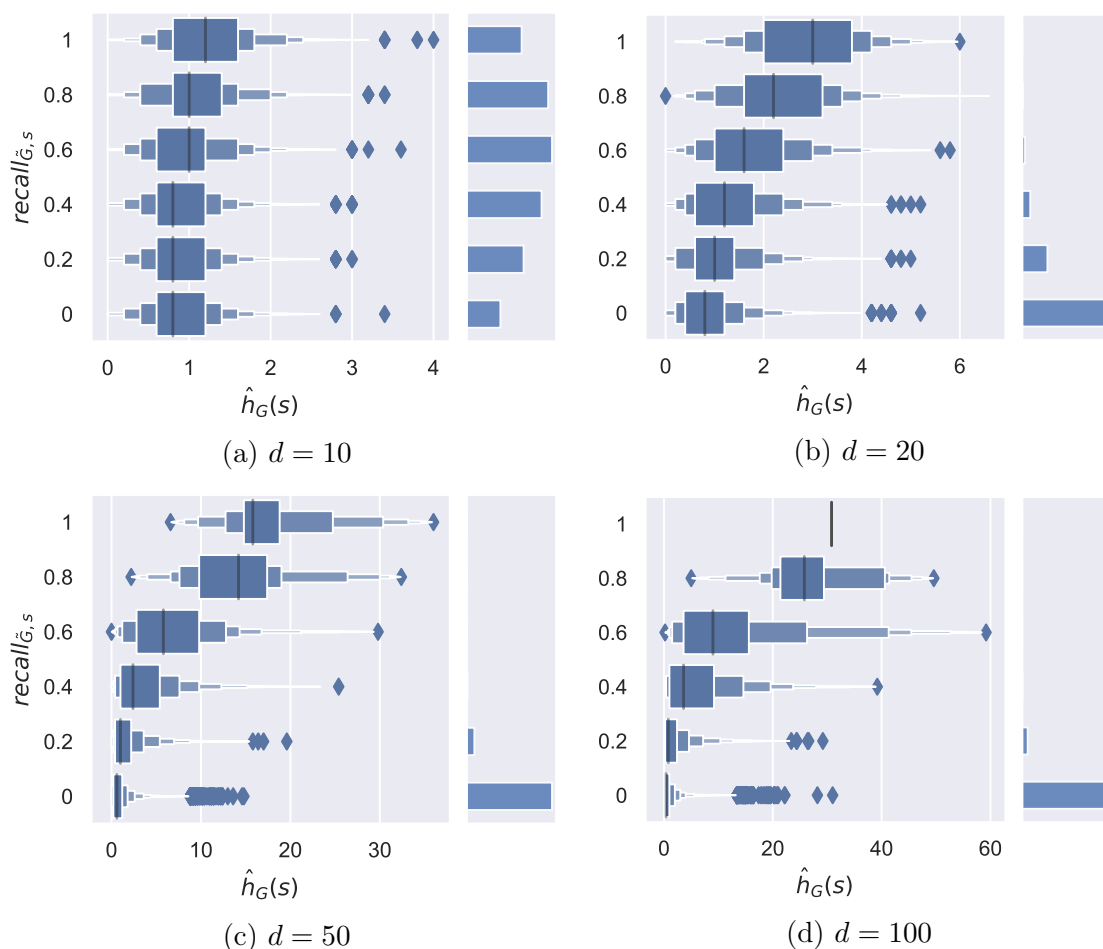
NN-Descent nad visokodimenzionalnim podacima

Kao što je već rečeno, *NN-Descent* ne generiše dobre aproksimacije k -NN grafa nad podacima velike dimenzionalnosti. Čest uzročnik lošeg rada algoritama mašinskog učenja nad visokodimenzionalnim podacima je gorepomenuti fenomen zvani habnes. Zbog toga je bilo važno ispitati kako ovaj fenomen utiče na *NN-Descent* algoritam.

Na slici 4.2 su prikazane distribucije habnes vrednosti pojedinačnih tački za različite *recall* vrednosti. Slika sadrži četiri grafikona za četiri različite dimenzionalnosti podataka. Sa histograma, koji se nalaze sa desne strane unutar svakog grafikona, može se videti da performanse *NN-Descent* algoritma nisu zadovoljavajuće s obzirom na to da veliki broj tački ima malu *recall* vrednost. Takođe

Prošireni izvod

se može videti da se performanse dodatno pogoršavaju kako se dimenzionalnost povećava. Pored toga, sa boks plotova, koji se nalaze sa leve strane unutar svakog grafikona, može se zapaziti jedan vrlo interesantan fenomen—kako se $recall$ vrednosti povećavaju, tako se povećavaju i habnes vrednosti. Na osnovu ovoga se može izvesti zaključak da, u visokodimenzionalnim podacima, habovi imaju veću verovatnoću da ostvare visoku $recall$ vrednost.

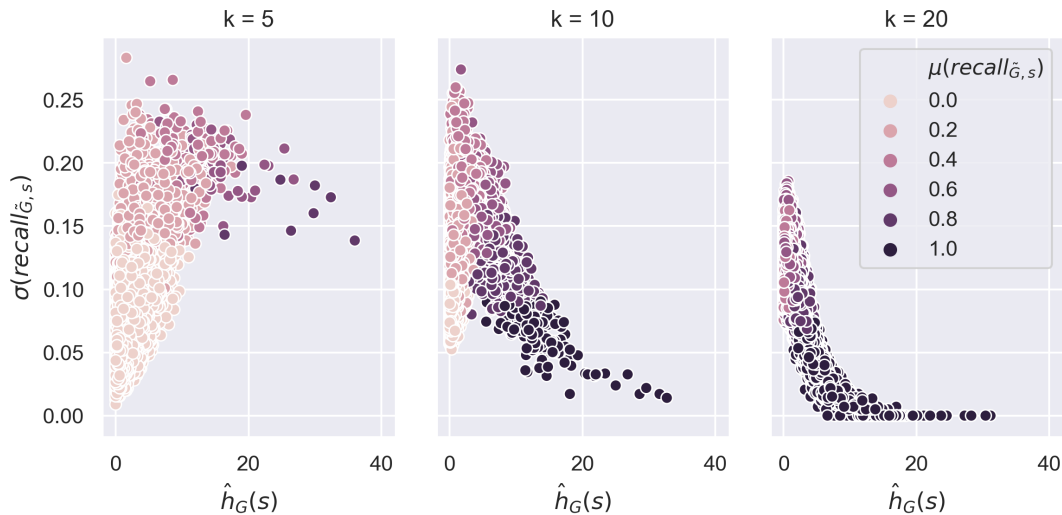


Slika 4.2 (ponovljeno): Distribucije habnes vrednosti pojedinačnih tački za različite $recall$ vrednosti.

Pored uticaja habnes vrednosti, na $recall$ vrednosti pojedinačnih tački utiče i njihova pozicija u inicijalnom slučajnom grafu. Da bismo ovu tvrdnju verifikovali, izvršili smo algoritam 100 puta, pri čemu se slučajne promenljive nezavisno generišu pri svakom izvršavanju, a potom smo posmatrali kako se $recall$ vrednosti pojedinačnih tački menjaju kroz različita izvršavanja. Na slici 4.4 predstavljeni su rezultati ovog eksperimenta. Intenzitet boje na ovoj slici predstavlja aritmetičku sredinu $recall$ vrednosti dobijenih kroz 100 nezavisnih puštanja algoritma, pri čemu

veći intenzitet boje predstavlja veću vrednost. Na slici se može videti da za $k = 10$ i $k = 20$, tačke malih habnes vrednosti imaju relativno visoku standardnu devijaciju $recall$ vrednosti, dok standardna devijacija za tačke velikih habnes vrednosti teži nuli. Budući da se $recall$ vrednost pojedinačnih tački malih habnes vrednosti drastično menja kroz različita izvršavanja algoritma, može se zaključiti da pozicija ovih tački unutar inicijalnog slučajnog grafa, igra bitnu ulogu za njihove $recall$ vrednosti.

Međutim, važno je istaći da pomenuto ne važi za $k = 5$. Naime, kao što se može videti na slici 4.4, za $k = 5$ uočavamo da standardne devijacije $recall$ vrednosti nisu nužno velike za tačke malih habnes vrednosti. Razlog za ovakvo ponašanje je taj što za male k vrednosti $NN-Descent$ algoritam daje izuzetno loše rezultate za veliku većinu tački malih habnes vrednosti. Samim tim, šansa da se tačka male habnes vrednosti nađe među onima koje imaju visoku $recall$ vrednost je drastično manja. Posledica toga je da je algoritam potrebno izvršiti mnogo veći broj puta da bi data tačka najzad došla do tačnijeg susedstva.



Slika 4.4 (ponovljeno): Zavisnost habnes vrednosti pojedinačnih tački i standardnih devijacija njihovih $recall$ vrednosti dobijenih za 100 nezavisnih izvršavanja $NN-Descent$ algoritma.

Zaključak ovih analiza je da tačke malih habnes vrednosti imaju manju verovatnoću da njihovo susedstvo u aproksimaciji k -NN grafa bude ispravno, i obrnuto, tačke velikih habnes vrednosti imaju veću verovatnoću da njihovo susedstvo bude tačno. Pored toga, $recall$ vrednost pojedinačnih tački zavisi od njihove pozicije u inicijalnom slučajnom grafu.

Modifikacije *NN-Descent* algoritma

Da bismo u određenoj meri prevazišli problem *NN-Descent* algoritma koji se javlja u visokodimenzionalnim podacima, uvešćemo pet njegovih modifikacija.

NN-Descent varijanta svesna habnesa

U okviru ove modifikacije *NN-Descent* algoritma, koju ćemo skraćeno oslovljavati sa *HA-NN-Descent*, habnes vrednosti se koriste pri određivanju parova tački između kojih će se računati razdaljina. Naime, pre ove modifikacije, unutar jedne iteracije *NN-Descent* algoritma, poredile su se sve tačke koje imaju zajedničkog suseda. S obzirom na to da habovi imaju dobra susedstva u aproksimaciji k -NN grafa, a tačke malih habnes vrednosti nemaju, ideja ove modifikacije jeste da se habovi manje porede sa drugim tačkama, a da se umesto toga više međusobno porede tačke malih habnes vrednosti.

Opisano ponašanje se realizuje ubacivanjem jednog dodatnog koraka koji nastupa neposredno pre nego što se susedi date pivot tačke međusobno uporede. Neka je L lista inicijalizovana direktnim i reverznim susedima date pivot tačke. Novi korak algoritma podrazumeva zamenu habova iz liste L nasumice izabranim tačkama ulaznog skupa. Nakon ovog koraka se svake dve tačke iz liste L međusobno porede, u skladu sa originalnim *NN-Descent* algoritmom.

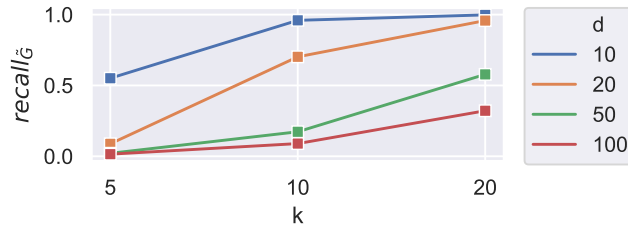
Jedino što u opisanom postupku ostaje nerazjašnjeno jeste kako se odlučuje da li je neka tačka hab, tj. kako se odlučuje da li će data tačka biti zamenjena nasumice izabranom tačkom. Za tu svrhu uvodimo verovatnoću da tačka bude zamenjena (5.1). Za svaku tačku iz liste L se onda računa ova verovatnoća, i po njoj se nasumice odlučuje da li će tačka biti zamenjena.

$$\Pr[\text{replace}_s] = \begin{cases} 0, & \text{if } h_{\tilde{G}}(s) < h_{\min}, \\ 1, & \text{if } h_{\tilde{G}}(s) > h_{\max}, \\ \frac{h_{\tilde{G}}(s) - h_{\min}}{h_{\max} - h_{\min}}, & \text{otherwise.} \end{cases} \quad (5.1)$$

NN-Descent varijanta proširenih NN lista

Već je rečeno da se *NN-Descent* algoritam sporije izvršava za veće k vrednosti. Međutim, sporije izvršavanje dolazi u paru sa većom tačnošću finalne aproksimacije k -NN grafa. Ovakvo ponašanje algoritma se može videti na slici 5.3, gde su prikazane *recall* vrednosti (y osa) za različite k vrednosti (x osa), i različite dimen-

zionalnosti ulaznih podataka (d vrednost; različite dimenzionalnosti su obojene različitim bojama).



Slika 5.3 (ponovljeno): *Recall* vrednosti aproksimacija k -NN grafa dobijenih *NN-Descent* algoritmom, za različite k vrednosti.

Ova varijanta *NN-Descent* algoritma, koju ćemo skraćeno oslovljavati sa *O-NN-Descent*, funkcioniše vrlo jednostavno. *NN-Descent* se pušta za veću k vrednost, a potom se tako dobijena aproksimacija k -NN grafa redukuje na željeno k . Redukcija aproksimacije k -NN grafa je vrlo jednostavan proces u kom se za svaku tačku odbacuje višak suseda, tj. zadržava se samo k najbližih.

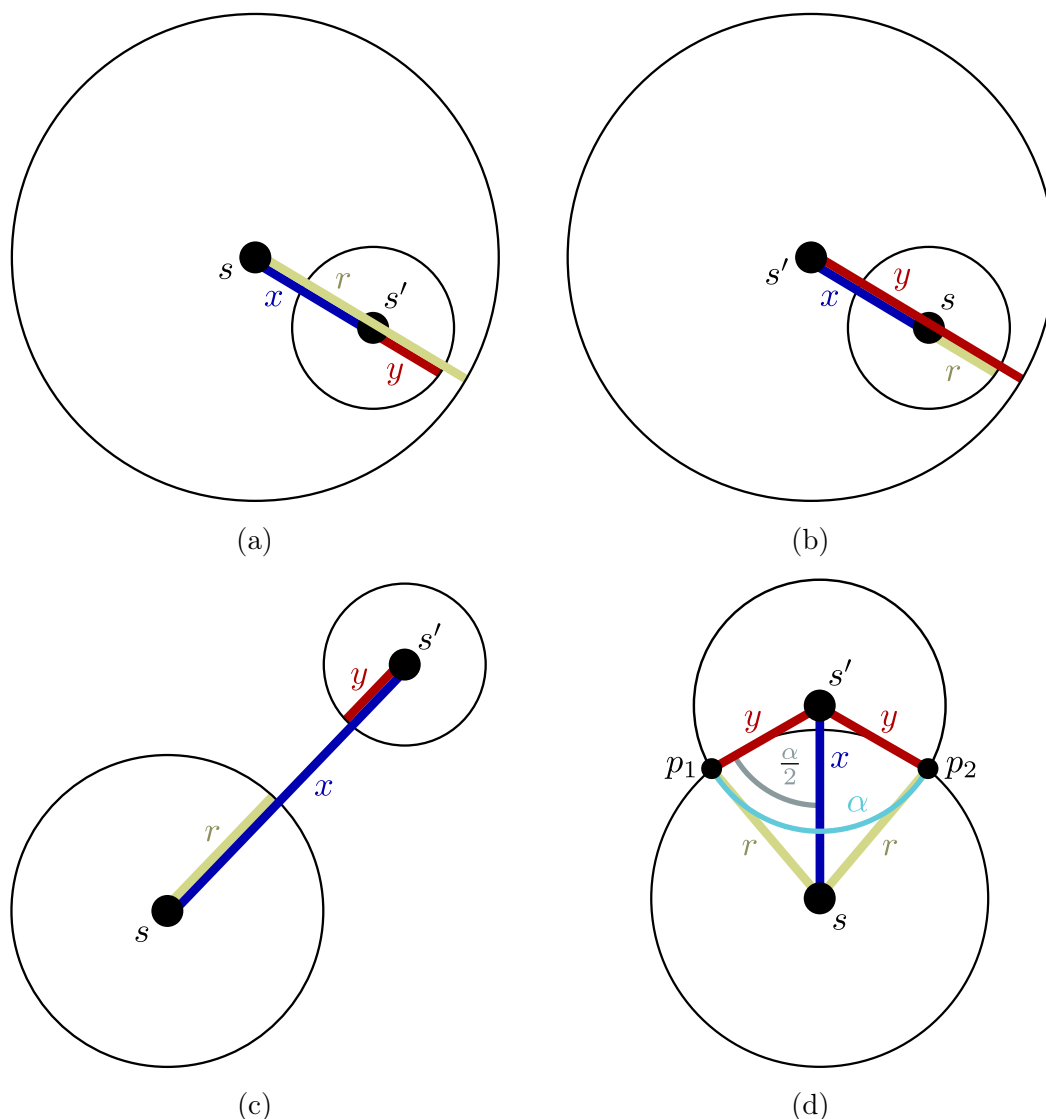
NN-Descent varijanta bazirana na slučajnim šetnjama

Glavna svrha ove *NN-Descent* varijante jeste da omogući da se broj poređenja konfigurise na nivou pojedinačnih tački. Na taj način bi se mogle isprobati različite strategije balansiranja pri raspodeli raspoloživih poređenja tački: na primer, mogao bi se dodeliti jednak broj poređenja svim tačkama, ili bi se recimo mogao dodeliti veći broj poređenja tačkama malih habnes vrednosti, itd.

Da bi se njena svrha realizovala, ova *NN-Descent* varijanta se zasniva na slučajnim šetnjama, zbog čega njena skraćunica glasi *RW-Descent* (*random walk descent*). Naime, umesto da se u jednoj iteraciji porede svake dve tačke koje imaju zajedničkog suseda, u *RW-Descent* algoritmu se iz svake tačke pušta određeni broj kratkih slučajnih šetnji. Nakon svake slučajne šetnje porede se njena početna i krajnja tačka. Pominjane strategije balansiranja onda određuju broj slučajnih šetnji za svaku tačku ulaznog skupa unutar svake iteracije algoritma.

NN-Descent varijanta bazirana na bliskim šetnjama

Ova *NN-Descent* varijanta je vrlo slična *RW-Descent* varijanti. Jedina razlika u odnosu na *RW-Descent* jeste da šetnje nisu slučajne, već su odabrane na osnovu



Slika 5.6 (ponovljeno): Međusobni odnosi tački koje učestvuju u šetnji. Upravo međusobni odnos ovih tački određuje verovatnoću da data šetnja vodi ka poboljšanju NN liste njene početne tačke.

određenih obzervacija. Skraćenica ove varijante je *NW-Descent* (*nearest walk descent*). U njoj se razmatraju sve moguće šetnje koje počinju u datoj tački, a koje su dužine dva, a onda se među njima biraju one koje su najbolje po kriterijumu kog ćemo u narednom tekstu opisati. Ono što je važno napomenuti jeste da ova *NN-Descent* varijanta funkcioniše samo sa L_2 merom razdaljine.

Neka je $\langle s, s', s'' \rangle$ šetnja koja počinje u s , prolazi kroz s' i završava se u s'' . Neka je $x = \text{dist}(s, s')$, $y = \text{dist}(s', s'')$ i $r = \text{dist}(s, s_k)$, gde s_k predstavlja k -tog suseda od s (tj. najdaljeg suseda iz NN liste tačke s). Dodatno, uvešćemo dve hipersfere

S i S' . Centar hipersfere S je u s , a njen poluprečnik je r , što znači da su sve tačke iz trenutnog susedstva tačke s , unutar S . Centar hipersfere S' je u s' , dok je njen poluprečnik y , što znači da se tačka s'' nalazi negde na površini hipersfere S' . Ove dve hipersfere mogu međusobno biti pozicionirane na četiri različita načina: 1) S' se nalazi unutar S ; 2) S se nalazi unutar S' ; 3) S i S' su međusobno disjunktne; 4) S i S' se seku. Dodatno, prvi slučaj je definisan nejednakošću $x + y < r$, drugi slučaj nejednakošću $x + r < y$, treći $y + r \leq x$ i najzad, ukoliko nijedna od navedenih nejednakosti nije zadovoljena, podrazumeva se četvrti slučaj. Ova četiri slučaja su ilustrovana u dvodimenzionalnom prostoru na slici 5.6.

U prvom slučaju (slika 5.6a) je verovatnoća da s'' upadne u NN listu od s jednaka 1, u drugom i trećem slučaju (slike 5.6b i 5.6c) ta verovatnoća je 0, dok u četvrtom slučaju (slika 5.6d) verovatnoća nije trivijalna. Naime, u četvrtom slučaju, tačka s'' će biti ubačena u NN listu tačke s samo ako se nalazi na delu S' koji je unutar S (ovaj deo je na slici označen svetlo plavom bojom). Stoga je verovatnoća jednaka odnosu između veličine pomenutog dela hipersfere S' i veličine cele hipersfere S' . Neka je P neka ravan koja sadrži s i s' , i neka su p_1 i p_2 tačke preseka P , S i S' . Pomenuti odnos je u tom slučaju jednak $\frac{\alpha}{2\pi}$, gde $\alpha = \angle p_1 s' p_2$. Dodatno, ugao α se može trivijalno izračunati na osnovu trougla $\triangle ss'p_1$, čije su sve stranice poznate (5.3). Kao zaključak, verovatnoća u četvrtom slučaju je $\frac{\alpha}{2\pi}$.

$$\alpha = 2 \arccos \frac{x^2 + y^2 - r^2}{2xy} \quad (5.3)$$

Kao što je već rečeno, ova varijanta *NN-Descent* algoritma za datu tačku razmatra sve šetnje dužine dva, i bira one koje su najbolje. Najbolje šetnje definisane su goreopisanom verovatnoćom—veća verovatnoća sugeriše da je šetnja bolja.

***NN-Descent* varijanta sa dodatnim slučajnim poređenjima**

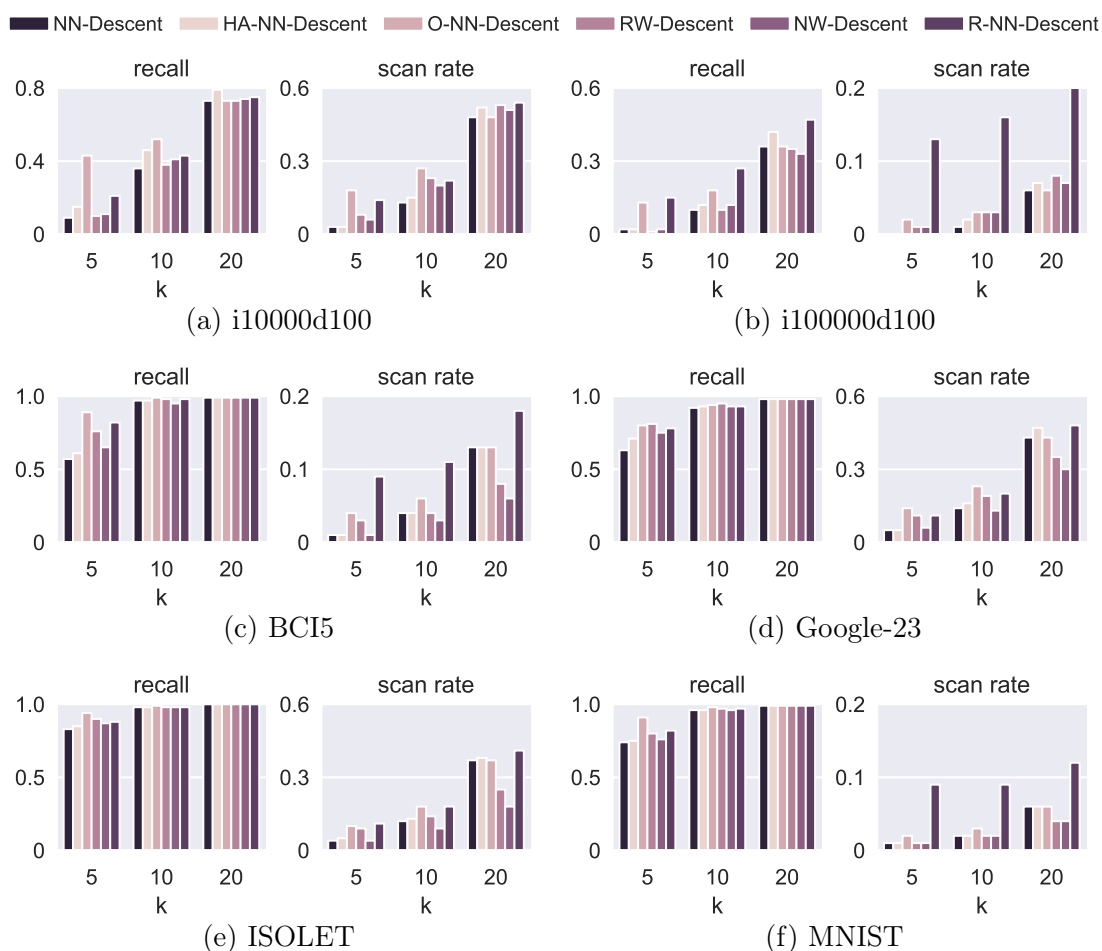
Ova *NN-Descent* varijanta, koju ćemo zvati i *R-NN-Descent*, uzima u obzir već opisani fenomen po kojem tačnost susedstva neke tačke zavisi od njene pozicije u inicijalnom slučajnom k -NN grafu (pogledati sliku 4.4). Unutar ove varijante, na početku svake iteracije, izvršava se novi korak algoritma, kog ćemo nazvati *fazom slučajnih poređenja*. Cilj ove faze jeste da tačke stavi u pravo susedstvo, kako bi se maksimizovala *recall* vrednost svake tačke.

Na samom početku izvršavanja algoritma inicijalizovaćemo novi skup S' svim tačkama ulaznog skupa S . Za svaku tačku s iz skupa S' će se na početku svake iteracije, unutar novog koraka algoritma, izvršiti poređenja sa unapred definisanim brojem nasumice izabranih tački iz S . Ukoliko dovoljan broj ovih poređenja ne

rezultuje ažuriranjem NN liste tačke s , s se izbacuje iz S' , te u sledećoj iteraciji neće biti poređena sa nasumice izabranim tačkama. Izbacivanje tačke iz skupa S' zapravo znači da je ona vrlo verovatno već u dobrom susedstvu k -NN grafa, te nije potrebno tražiti joj novo susedstvo.

Rezultati *NN-Descent* varijanti

Da bismo evaluirali nove *NN-Descent* varijante, sproveli smo eksperimentalnu analizu koja podrazumeva puštanje svih algoritama nad realnim i sintetičkim visokodimenzionalnim skupovima podataka. Svi algoritmi su puštane za $k \in \{5, 10, 20\}$ i L_2 funkciju razdaljine. Rezultati svih algoritama se mogu videti na slici 5.7.



Slika 5.7 (ponovljeno): Performanse *NN-Descent* algoritma i svih njegovih varijanti, izražene *recall* i *scan rate* vrednostima.

Kao što se može videti sa slike iznad, *O-NN-Descent* u većini slučajeva postiže najbolje *recall* vrednosti, ali u isto vreme ovaj algoritam ima i najveće *scan rate*

vrednosti. Stoga, kada je tačnost aproksimacije k -NN grafa važnija od brzine izvršavanja, *O-NN-Descent* je dobar izbor.

S druge strane, kada je vreme izvršavanja važnije od tačnosti, najbolje su se pokazali *HA-NN-Descent* i *NW-Descent*. Naime, *HA-NN-Descent* i *NW-Descent* povećavaju *recall* vrednost u slučaju visokodimenzionalnih skupova podataka, održavajući *scan rate* istim ili skoro istim.

RW-Descent ima slične *recall* vrednosti kao i *NW-Descent*, s tim što mu je *scan rate* vrednost nešto veća. Međutim, za neke pojedinačne skupove podataka, ovaj algoritam je dao bolje rezultate od svih ostalih.

Za kraj, *R-NN-Descent* je u većini slučaja drugi po *recall* vrednosti, ali je njegova mana to što su mu *scan rate* vrednosti uglavnom veoma visoke. Stoga ovaj algoritam obično nije prvi izbor.

Sve predložene varijante *NN-Descent* algoritma nad visokodimenzionalnim podacima imaju bolje rezultate od originalnog *NN-Descent* algoritma. Dodatno, nijedna od ovih varijanti nije ultimativno najbolja—eksperimenti su pokazali da svaka od njih ima svoje prednosti i mane, te se za različite primene, različite varijante pokazuju najboljim. Iako ne postoji ultimativno najbolje rešenje, predložene *NN-Descent* varijante zajedno pokrivaju razne probleme i scenarije, predstavljajući moćan alat za rešavanje problema generisanja aproksimacije k -NN grafa nad visokodimenzionalnim podacima.

Algoritmi za ažuriranje aproksimacije k -NN grafa

U realnom svetu, podaci se često menjaju vremenom. Ukoliko je k -NN graf generisan nad podacima koji su se potom izmenili, potrebno je generisani graf ažurirati. Najjednostavniji način ažuriranja k -NN grafa jeste da se prosto izgeneriše ispočetka bilo kojim algoritmom za generisanje k -NN grafova. Međutim, ovaj pristup je neoptimalan, s obzirom na to da se podaci uglavnom ne menjaju u celosti, već parcijalno, te je moguće iskoristiti prethodnu verziju grafa i u njoj izmeniti samo ono što je neophodno.

Metod grube sile za ažuriranje k -NN grafa

Prvi način da se odradi parcijalno ažuriranje k -NN grafa je metod grube sile. Pretpostavimo da se izmenio čvor s k -NN grafa. U tom slučaju se moraju iznova izračunati razdaljine između s i svih ostalih čvorova grafa, što će rezultovati izmenama NN listi nekih čvorova. Pored toga, potrebno je izračunati razdaljine i

između čvorova iz R -NN liste čvora s i svih ostalih čvorova. Razlog za to je što čvorovi koji imaju s u svojoj NN listi možda umesto s sada treba da imaju neki drugi čvor.

Onlajn varijante algoritama *RW-Descent* i *NW-Descent*

Opisani metod grube sile vrši parcijalno ažuriranje k -NN grafa, što rezultuje novim, egzaktnim k -NN grafom. Ukoliko to nije dovoljno brzo, jedna opcija je razvijanje aproksimativnog algoritma za ažuriranje k -NN grafa. U ovoj sekciji predstavimo onlajn verzije algoritama *RW-Descent* i *NW-Descent*, koji su po svojoj srži upravo aproksimativni algoritmi za ažuriranje k -NN grafa.

Onlajn verzije pomenutih algoritama funkcionišu na sledeći način. Na samom početku algoritma, pravi se skup čvorova čije NN liste treba ažurirati. Slično kao u metodu grube sile, taj skup sadrži sve izmenjene čvorove, ali i njihove reverzne susede. Potom se nad prethodnom verzijom k -NN grafa *RW-Descent* i *NW-Descent* primenjuju na uobičajen način, uz malu izmenu koja podrazumeva da šetnje ne započinju od svih čvorova, već samo od onih koji se nalaze u prethodno pomenutom skupu tački čije NN liste treba izmeniti.

Opisani onlajn *RW-Descent* i onlajn *NW-Descent* algoritmi imaju jedan ozbiljan problem. Naime, ovi algoritmi rade pod pretpostavkom da se čvorovi nisu značajno izmenili, tj. da su nakon izmene ostali u svom širem susedstvu. Ova pretpostavka se koristi onda kada se novi susedi izmenjenih tački traže šetnjama dužine dva—na ovaj način algoritam traži nove susede date tačke gledajući samo u njeno šire susedstvo. Međutim, ukoliko se neki čvor značajno izmenio, postoji velika šansa da će ovim postupkom ostati zaglavljnjen u nekom lokalnom minimumu, bez uspeha u pronalaženju svojih novih suseda.

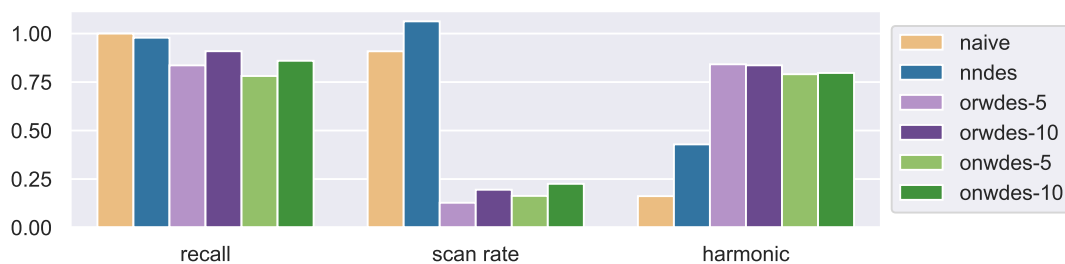
Ono što je zanimljivo jeste da se opisani problem ne javlja u originalnim verzijama *RW-Descent* i *NW-Descent* algoritama, tj. u originalnim verzijama se tačke sa manjom verovatnoćom zaglavljuju u lokalnim minimumima. Razlog za to je što originalni *RW-Descent* i *NW-Descent* počinju od slučajnog grafa, te se u prvoj iteraciji algoritama svaka tačka poredi sa određenim brojem slučajno odabranih tački (jer se šetnje vrše unutar slučajnog grafa). Ovo dalje implicira da se svaka tačka u prvoj iteraciji algoritama poredi sa većim brojem tački koje potiču iz različitih susedstva. Za razliku od toga, onlajn varijante ne počinju od slučajnog grafa, već od prethodne verzije k -NN grafa. Kao posledica toga, tačke se porede sa drugim tačkama koje potiču iz istog susedstva, čime se ne daje prilika izmenjenim tačkama da razmatraju više susedstva i odaberu najbolje.

Da bi se taj problem rešio, onlajn *RW-Descent* i *NW-Descent* su prošireni idejom na kojoj je baziran algoritam *R-NN-Descent*. Tačnije, pomenute algoritme smo proširili fazom slučajnih poređenja, koja se izvršava na početku svake iteracije, a koja podrazumeva poređenje izmenjenih tački sa unapred zadatim brojem slučajno odabranih tački. Na taj način se daje prilika izmenjenim tačkama da pronađu svoje šire susedstvo.

Rezultati onlajn *RW-Descent* i onlajn *NW-Descent* algoritama

Kao što smo već pomenuli, glavni cilj onlajn *RW-Descent* i *NW-Descent* algoritama jeste da ubrzaju proces ažuriranja k -NN grafova. Da bismo utvrdili da li je ovaj cilj uspešno realizovan, sproveli smo opširne eksperimente. Unutar eksperimenata poredili smo performanse pomenuta dva algoritma sa performansama metoda grube sile, ali i sa performansama brzog aproksimativnog algoritma *NN-Descent*.

Eksperimenti su zapravo bazirani na simulacijama realnog scenarija koji podrazumeva ažuriranje k -NN grafova koji su generisani nad vremenskim serijama. Naime, vremenske serije se po svojoj prirodi menjaju vremenom, te ukoliko je nad takvim podacima generisan k -NN graf, potrebno je ažurirati ga svaki put kada vremenske serije dobiju nove vrednosti. Stoga se unutar eksperimenata vrši simulacija izmena vremenskih serija, što je praćeno ažuriranjem k -NN grafa. Za eksperimente smo koristili 128 skupa podataka sačinjenih od vremenskih serija, koji su preuzeti iz čuvenog repozitorijuma *UCR Time Series Classification Archive* [16].



Slika 6.4 (ponovljeno): Prosečne *recall*, *scan rate* i *harmonic* vrednosti svih predstavljenih metoda za ažuriranje k -NN grafa.

Rezultati različitih algoritama su prikazani na slici 6.4. Na slici naziv *naive* predstavlja metod grube sile, *nndes* predstavlja *NN-Descent*, *orwdes-5* i *orwdes-10* predstavljaju onlajn *RW-Descent* algoritam za dve različite konfiguracije njegovih

parametara, dok *onwdes-5* i *onwdes-10* predstavljaju onlajn *NW-Descent*, takođe za dve različite konfiguracije parametara.

Na slici se može videti da je *recall* vrednost metoda grube sile maksimalna moguća, jer ovaj algoritam nije aproksimativni. *NN-Descent* algoritam takođe daje vrlo visoku *recall* vrednost od 0.98, što je svakako i bilo očekivano s obzirom na to da podaci nad kojima smo algoritme puštali nisu visokodimenzionalni. Onlajn *RW-Descent* algoritam proizvodi *recall* vrednosti od 0.84 i 0.91 za njegove dve konfiguracije, što pokazuje da je ovaj algoritam takođe vrlo kompetetivan. Za kraj, *recall* vrednosti onlajn *NW-Descent* algoritma su 0.78 i 0.86, što je lošije od onlajn *RW-Descent* algoritma, ali je ipak vrlo zadovoljavajuće.

Što se tiče *scan rate* vrednosti, vrlo jasno se vidi da onlajn *RW-Descent* i onlajn *NW-Descent* značajno smanjuju broj izračunatih razdaljina. Kao što je to i očekivano, prosečna *scan rate* vrednost metoda grube sile je vrlo visoka, i iznosi 0.91. *NN-Descent* ima prosečnu *scan rate* vrednost koja je čak i veća od 1, što je posledica skupova podataka koji su vrlo mali. Naime, *NN-Descent* ima karakteristiku da nad dovoljno malim skupovima podataka radi lošije od metoda grube sile. Stoga su mali skupovi podataka uticali na to da *NN-Descent* ima lošu prosečnu *scan rate* vrednost, ali treba imati u vidu da se njegove performanse drastično poboljšavaju nad većim skupovima podataka.

Za kraj ćemo analizirati vrednosti harmonijske sredine, koja uzima u obzir i *recall* i *scan rate* vrednosti, omogućavajući evaluaciju algoritama i iz perspektive tačnosti aproksimacije i iz perspektive brzine izvršavanja. Kao što se može videti, harmonijska sredina je vidno najbolja za onlajn *RW-Descent* i onlajn *NW-Descent*. Prirodno, najgoru harmonijsku sredinu ima metod grube sile. *NN-Descent* ima harmonijsku sredinu koja je bolja od metoda grube sile, ali ipak značajno lošija od predstavljenih onlajn algoritama.

Uopšteno govoreći, ovi eksperimenti sugerišu da su onlajn *RW-Descent* i onlajn *NW-Descent* vrlo kompetetivni algoritmi. Iako su njihove *recall* vrednosti neznatno manje od *NN-Descent* algoritma i metoda grube sile, ovi algoritmi vrlo značajno smanjuju *scan rate* vrednosti. Takođe, može se videti da onlajn *RW-Descent* ostvaruje nešto bolje rezultate od onlajn *NW-Descent* algoritma. Međutim, dodatne analize su pokazale da onlajn *NW-Descent* algoritam ima manju akumulacionu grešku (tj. uzastopnim primenjivanjem algoritma nad grafom koji se menja, greška aproksimacije se manje akumulira). Takođe, eksperimenti su pokazali da se ovi algoritmi bolje pokazuju u slučaju kada su izmene podataka manje. I za kraj je bitno napomenuti da onlajn algoritmi daju mnogo bolje rezul-

tate od *NN-Descent* algoritma nad manjim do srednjim skupovima podataka, dok je nad velikim skupovima podataka i *NN-Descent* dovoljno dobar.

Zaključak

S obzirom na to da su k -NN grafovi sastavni deo mnogih algoritama i problema, brzi aproksimativni algoritmi za njihovo generisanje su veoma važni. Ova teza je fokusirana upravo na aproksimativne algoritme za generisanje k -NN grafova. Analizirali smo i objasnili probleme jednog takvog algoritma, koji se zove *NN-Descent*, a pored toga smo uveli i njegove modifikacije koje za cilj imaju prevazilaženje pomenutih problema. Dodatno, uveli smo i dva aproksimativna algoritma za ažuriranje k -NN grafova.

Prvi deo teze je dakle posvećen analiziranju loših performansi *NN-Descent* algoritma u slučaju kada je pušten nad visokodimenzionalnim podacima. Pokazali smo da su loše aproksimacije posledica fenomena koji se zove *habnes*. *Habnes* pobija osnovnu pretpostavku *NN-Descent* algoritma da je sused suseda vrlo verovatno takođe sused. Kao posledica toga, *NN-Descent* se loše ponaša nad skupovima podataka koji unutar sebe imaju *habnes* fenomen. Sve to smo dokazali eksperimentalnim analizama.

Kako bismo prevazišli pomenutu manu *NN-Descent* algoritma, uveli smo njegovih pet modifikacija. Performanse ovih modifikacija evaluirane su eksperimentalnom analizom nad šest visokodimenzionalnih skupova podataka, od kojih su dva sintetička. Rezultati eksperimentalne analize su pokazali da sve modifikacije uvode porast *recall* vrednosti, ali na štetu *scan rate* vrednosti, koja je uglavnom nešto lošija od *scan rate* vrednosti originalnog algoritma.

Drugi pravac istraživanja odnosio se na aproksimativne algoritme za ažuriranje k -NN grafova. U literaturi nema mnogo radova na ovu temu—fokus je većinski na algoritmima za generisanje novih k -NN grafova, a ne na ažuriranju postojećih. U ovoj tezi smo predložili dva aproksimativna algoritma za ažuriranje k -NN grafova. Sproveli smo opširne eksperimente nad 128 skupa podataka koji sadrže vremenske serije. Rezultati eksperimenata sugerišu da novi algoritmi imaju nešto lošije *recall* vrednosti, ali da zato značajno smanjuju *scan rate*.

Bibliography

- [1] Alok Aggarwal, Leonidas J Guibas, James Saxe, and Peter W Shor. “A linear-time algorithm for computing the Voronoi diagram of a convex polygon”. In: *Discrete & Computational Geometry* 4.6 (1989), pp. 591–604.
- [2] David C Anastasiu and George Karypis. “L2knnng: Fast exact k-nearest neighbor graph construction with l2-norm pruning”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM. 2015, pp. 791–800.
- [3] Mehmet Serkan Apaydin, Douglas L Brutlag, Carlos Guestrin, David Hsu, Jean-Claude Latombe, and Chris Varma. “Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion”. In: *Journal of Computational Biology* 10.3-4 (2003), pp. 257–281.
- [4] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 1961.
- [5] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. “Label propagation and quadratic criterion”. In: *Semi-Supervised Learning*. Ed. by Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. MIT Press, 2006.
- [6] Marshall Bern, David Eppstein, and John Gilbert. “Provably good mesh generation”. In: *Journal of computer and system sciences* 48.3 (1994), pp. 384–409.
- [7] Daniel Boley. “Principal direction divisive partitioning”. In: *Data mining and knowledge discovery* 2.4 (1998), pp. 325–344.
- [8] Brankica Bratić, Michael E Houle, Vladimir Kurbalija, Vincent Oria, and Miloš Radovanović. “NN-descent on high-dimensional data”. In: *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics*. 2018, pp. 1–8.

-
- [9] Brankica Bratić, Michael E Houle, Vladimir Kurbalija, Vincent Oria, and Miloš Radovanović. “The Influence of Hubness on NN-Descent”. In: *International Journal on Artificial Intelligence Tools* 28.06 (2019).
- [10] MR Brito, EL Chavez, AJ Quiroz, and JE Yukich. “Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection”. In: *Statistics & Probability Letters* 35.1 (1997), pp. 33–42.
- [11] Paul B Callahan. “Optimal parallel all-nearest-neighbors using the well-separated pair decomposition”. In: *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*. IEEE. 1993, pp. 332–340.
- [12] Paul B Callahan and S Rao Kosaraju. “A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields”. In: *Journal of the ACM (JACM)* 42.1 (1995), pp. 67–90.
- [13] Jie Chen, Haw-ren Fang, and Yousef Saad. “Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection”. In: *Journal of Machine Learning Research* 10.Sep (2009), pp. 1989–2012.
- [14] Ulrich Clarenz, Martin Rumpf, and Alexandru Telea. “Finite elements on point based surfaces”. In: *SPBG*. 2004, pp. 201–211.
- [15] Michael Connor and Piyush Kumar. “Fast construction of k-nearest neighbor graphs for point clouds”. In: *IEEE transactions on visualization and computer graphics* 16.4 (2010), pp. 599–608.
- [16] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. 2018.
- [17] Thibault Debatty, Pietro Michiardi, and Wim Mees. “Fast online K-NN graph building”. In: *arXiv preprint arXiv:1602.06819* (2016).
- [18] Thibault Debatty, Pietro Michiardi, Olivier Thonnard, and Wim Mees. “Building k-NN graphs from large text data”. In: *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE. 2014, pp. 573–578.
- [19] Matthew T Dickerson and David Eppstein. “Algorithms for proximity problems in higher dimensions”. In: *Computational Geometry* 5.5 (1996), pp. 277–291.

BIBLIOGRAPHY

- [20] Wei Dong, Moses Charikar, and Kai Li. “Efficient k-nearest neighbor graph construction for generic similarity measures”. In: *Proceedings of the 20th international conference on World wide web*. ACM. 2011, pp. 577–586.
- [21] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [22] Mark Fanty and Ronald Cole. “Spoken letter recognition”. In: *Advances in Neural Information Processing Systems*. 1991, pp. 220–226.
- [23] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. “Similarity search in high dimensions via hashing”. In: *Vldb*. Vol. 99. 6. 1999, pp. 518–529.
- [24] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. “Fast approximate nearest-neighbor search with k-nearest neighbor graph”. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 1. 2011, pp. 1312–1317.
- [25] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. “Outlier detection using k-nearest neighbour graph”. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 3. IEEE. 2004, pp. 430–433.
- [26] Douglas M Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980.
- [27] Heike Hofmann, Karen Kafadar, and Hadley Wickham. *Letter-value plots: Boxplots for large data*. Tech. rep. had.co.nz, 2011.
- [28] Michael E Houle, Xiguo Ma, Vincent Oria, and Jichao Sun. “Improving the quality of K-NN graphs for image databases through vector sparsification”. In: *Proceedings of International Conference on Multimedia Retrieval*. ACM. 2014, pp. 89–96.
- [29] Michael E Houle, Vincent Oria, Shin’Ichi Satoh, and Jichao Sun. “Annotation propagation in image databases using similarity graphs”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 10.1 (2013), p. 7.
- [30] Peter Wilcox Jones, Andrei Osipov, and Vladimir Rokhlin. “Randomized approximate nearest neighbors algorithm”. In: *Proceedings of the National Academy of Sciences* 108.38 (2011), pp. 15679–15686.
- [31] F Juhász and K Mályusz. “Problems of cluster analysis from the viewpoint of numerical analysis”. In: *Numerical Methods, Colloquia Mathematica Societatis Janos Bolyai*. Vol. 22. 1977, pp. 405–415.

-
- [32] David R Karger and Matthias Ruhl. “Finding nearest neighbors in growth-restricted metrics”. In: *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. ACM. 2002, pp. 741–750.
- [33] Donald Ervin Knuth. “Sorting and searching”. In: *The art of computer programming* 3 (1973), pp. 506–549.
- [34] Ivan Komarov, Ali Dashti, and Roshan M D’Souza. “Fast k-NNG construction with GPU-based quick multi-select”. In: *PloS one* 9.5 (2014), e92409.
- [35] Wei-Shinn Ku, Roger Zimmermann, Haojun Wang, and Chi-Ngai Wan. “Adaptive nearest neighbor queries in travel time networks”. In: *Proceedings of the 13th annual ACM international workshop on Geographic information systems*. 2005, pp. 210–219.
- [36] Cornelius 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.
- [37] Neal Lathia, Stephen Hailes, and Licia Capra. “kNN CF: a temporal social network”. In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM. 2008, pp. 227–234.
- [38] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [39] Der-Tsai Lee. “On k-nearest neighbor Voronoi diagrams in the plane”. In: *IEEE transactions on computers* 100.6 (1982), pp. 478–487.
- [40] Jdel 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)*. Vol. 4. IEEE. 2004, pp. 2877–2882.
- [41] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. “Practical construction of k-nearest neighbor graphs in metric spaces”. In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2006, pp. 85–97.
- [42] Youngki Park, Heasoo Hwang, and Sang-goo Lee. “A novel algorithm for scalable k-nearest neighbour graph construction”. In: *Journal of Information Science* 42.2 (2016), pp. 274–288.

BIBLIOGRAPHY

- [43] Youngki Park, Sungchan Park, Sang-goo Lee, and Woosung Jung. “Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction”. In: *International Conference on Database Systems for Advanced Applications*. Springer. 2014, pp. 327–341.
- [44] Mark Pauly, Markus Gross, and Leif P Kobbelt. “Efficient simplification of point-sampled surfaces”. In: *IEEE Visualization, 2002. VIS 2002*. IEEE. 2002, pp. 163–170.
- [45] Erion Plaku and Lydia E Kavradi. “Distributed computation of the knn graph for large high-dimensional point sets”. In: *Journal of parallel and distributed computing* 67.3 (2007), pp. 346–359.
- [46] Milos Radovanović, Alexandros Nanopoulos, and Mirjana 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*. ACM. 2010, pp. 186–193.
- [47] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. “Time-series classification in many intrinsic dimensions”. In: *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM. 2010, pp. 677–688.
- [48] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. “Hubs in space: Popular nearest neighbors in high-dimensional data”. In: *Journal of Machine Learning Research* 11.Sep (2010), pp. 2487–2531.
- [49] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. “Reverse nearest neighbors in unsupervised distance-based outlier detection”. In: *IEEE transactions on knowledge and data engineering* 27.5 (2015), pp. 1369–1382.
- [50] Michael Ian Shamos and Dan Hoey. “Closest-point problems”. In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE. 1975, pp. 151–162.
- [51] Sami Sieranoja and Pasi Fränti. “Constructing a High-Dimensional k NN-Graph Using a Z-Order Curve”. In: *Journal of Experimental Algorithmics (JEA)* 23.1 (2018), pp. 1–9.
- [52] Nenad Tomašev, Miloš Radovanović, Dunja Mladenić, and Mirjana Ivanović. “The role of hubness in clustering high-dimensional data”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2011, pp. 183–195.

- [53] David Tritchler, Shafagh Fallah, and Joseph Beyene. “A spectral clustering method for microarray data”. In: *Computational statistics & data analysis* 49.1 (2005), pp. 63–76.
- [54] Pravin M Vaidya. “An $O(n \log n)$ algorithm for the all-nearest-neighbors problem”. In: *Discrete & Computational Geometry* 4.2 (1989), pp. 101–115.
- [55] Olli Virmajoki and Pasi Franti. “Divide-and-conquer algorithm for creating neighborhood graph for clustering”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*. Vol. 1. IEEE. 2004, pp. 264–267.
- [56] Fei Wang and Changshui Zhang. “Label propagation through linear neighborhoods”. In: *IEEE Transactions on Knowledge and Data Engineering* 20.1 (2007), pp. 55–67.
- [57] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. “Scalable k-nn graph construction for visual descriptors”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1106–1113.
- [58] Ian H Witten and Eibe Frank. “Data mining: practical machine learning tools and techniques with Java implementations”. In: *Acm Sigmod Record* 31.1 (2002), pp. 76–77.
- [59] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. “Fast kNN graph construction with locality sensitive hashing”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 660–674.
- [60] Wan-Lei Zhao. “k-NN Graph Construction: a Generic Online Approach”. In: *arXiv preprint arXiv:1804.03032* (2018).
- [61] Xiaojin Zhu and Zoubin Ghahramani. “Learning from labeled and unlabeled data with label propagation”. In: (2002).

List of Figures

2.1	An example of k -NNG ($k = 2$) on a set of 2-dimensional points in Euclidean plane.	8
2.2	Dependence of hubness values (extracted from k -NNGs with $k = 5$ and L_2 distance) on data dimensionality (d).	10
4.1	Recall values of <i>NN-Descent</i> k -NNG approximations ($k = 5$) created on datasets of dimensionalities 10, 20, 50 and 100.	29
4.2	Distributions of points' hubness values for different recall values. . .	30
4.3	Average recall values for different hubness values.	32
4.4	Dependence between points' hubness values and standard deviations of their recall values over 100 runs of <i>NN-Descent</i>	34
4.5	Dependence between points' distance to dataset mean and standard deviations of their recall values over 100 runs of <i>NN-Descent</i>	35
5.1	Correlation between approximate and real hubness values of dataset points, for different k values.	38
5.2	Correlation between approximate and real hubness values of dataset points, recorded after each iteration of <i>NN-Descent</i> algorithm. . . .	39
5.3	Recall values of <i>NN-Descent</i> k -NNG approximations for different k values.	43
5.4	Different relations between recall $_{\tilde{G}_k^R}$ and recall $_{\tilde{G}_{k'}}$ for $k = 4$, $k' = 10$ and recall $_{\tilde{G}_{k'}} = 0.5$	45
5.5	Recall values of graphs reduced from <i>NN-Descent</i> approximations $\tilde{G}_{k'}$ to neighborhood sizes $k \in [1, k']$	45
5.6	Possible relative positions of the points that participate in a walk. The relative position determines the probability that the walk leads to an update of its starting point's NN list.	50
5.7	Performance of <i>NN-Descent</i> and all its variants, expressed with recall and scan rate.	63

LIST OF FIGURES

5.8	Performance of <i>NN-Descent</i> and all its variants, expressed with harmonic mean of recall and scan gain.	65
6.1	Visualization of the sliding window technique.	76
6.2	Outline of the simulation flow.	78
6.3	Distribution of simulation iterations counts.	81
6.4	Average recall, scan rate and harmonic mean for all the presented <i>k</i> -NNG update approaches.	82
6.5	Influence of dataset size on performance of <i>k</i> -NNG update algorithms.	84
6.6	Influence of time series variability on performance of <i>k</i> -NNG update algorithms.	86
6.7	Influence of parameter <i>k</i> on performance of <i>k</i> -NNG update algorithms.	87
6.8	Influence of distance function on performance of <i>k</i> -NNG update algorithms.	88
6.9	Influence of simulation parameter <i>sw</i> on performance of <i>k</i> -NNG update algorithms.	88
6.10	Influence of simulation parameters <i>p</i> and <i>b</i> on performance of <i>k</i> -NNG update algorithms.	89
6.11	Performance of the <i>k</i> -NNG update algorithms during the simulations execution.	90

List of Tables

3.1	List of parameters of <i>NN-Descent</i>	24
5.1	List of parameters of hubness aware variant.	41
5.2	List of parameters of oversized NN list variant.	46
5.3	List of parameters of <i>RW-Descent</i>	47
5.4	List of parameters of randomized <i>NN-Descent</i> variant.	53
5.5	Datasets that were used for the experimental validation of the five <i>NN-Descent</i> variants.	56
5.6	Performance of <i>NN-Descent</i> algorithm expressed with recall and scan rate.	57
5.7	Performance of hubness-aware <i>NN-Descent</i> variant expressed with recall and scan rate.	58
5.8	Performance of oversized NN list variant expressed with recall and scan rate.	59
5.9	Performance of <i>RW-Descent</i> expressed with recall and scan rate.	60
5.10	Performance of <i>NW-Descent</i> expressed with recall and scan rate.	61
5.11	Performance of randomized <i>NN-Descent</i> variant expressed with recall and scan rate.	62
6.1	List of parameters of <i>online RW-Descent</i> and <i>online NW-Descent</i>	72
6.2	Properties of UCR datasets after the preprocessing step.	75
6.3	List of simulation parameters.	79
6.4	Simulation parameters values that were used in experiments.	81
6.5	Average recall, scan rate and harmonic mean for all the presented <i>k</i> -NNG update approaches.	82
B.1	Average simulations results per dataset.	109

LIST OF TABLES

Abbreviations

<i>dist</i>	A distance function
$NN_G(s)$	Nearest neighbor list of the point s in the graph G
$RNN_G(s)$	Reverse nearest neighbor list of the point s in the graph G
$h_G(s)$	Hubness value of the point s in the graph G
k	Neighborhood size in k -nearest neighbor graph
L_p	Minkowski distance of order p
k -NNG	k -nearest neighbor graph
k -NN	k -nearest neighbors
NN	Nearest neighbor(s)
R -NN	Reverse nearest neighbor(s)
DTW	Dynamic time warping
ID	Intrinsic dimensionality

ABBREVIATIONS

Kratka biografija

Brankica Bratić je rođena 12. 12. 1989. u Kikindi. Osnovnu školu „Ivo Lola Ribar” je završila u Novim Kozarcima 2004. godine, nakon čega je upisala gimnaziju „Dušan Vasiljev” u Kikindi, koju je završila 2008. godine. Po završetku gimnazije, upisala se na osnovne studije na Prirodno-matematičkom fakultetu u Novom Sadu, smer informatika. Osnovne studije je završila u junu 2011. godine sa prosečnom ocenom 9,85. Odmah potom je upisala master studije na istom fakultetu, koje je završila u junu 2013. godine sa prosečnom ocenom 10,00. Doktorske studije informatike je upisala 2014. godine, a sve ispite sa ovog stepena studija položila je sa prosečnom ocenom 10,00.



Od 2014. godine je, kao asistent na Departmanu za matematiku i informatiku Prirodno-matematičkog fakulteta u Novom Sadu, držala teorijske i praktične vežbe iz više informatičkih predmeta. Bila je član organizacionih odbora više međunarodnih konferencija. Ima četiri objavljena rada u međunarodnim časopisima sa ISI liste, a pored toga još tri rada sa međunarodnih konferencija i časopisa. Bila je član sedam naučnih projekata, koja su većinski iz oblasti mašinskog učenja. U toku studija bila je na pet studentskih razmena i škola: u Litvaniji, Austriji, Japanu i dva puta u Sloveniji.

Novi Sad, septembar 2020

Brankica Bratić

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:

RBR

Identifikacioni broj:

IBR

Tip dokumentacije:

Monografska dokumentacija

TD

Tip zapisa:

Tekstualni štampani materijal

TZ

Vrsta rada:

Doktorska disertacija

VR

Autor:

Brankica Bratić

AU

Mentor:

dr Vladimir Kurbalija

MN

Naslov rada:

Aproksimativni algoritmi za generisanje k -NN grafa

NR

Jezik publikacije:

Engleski

JP

Jezik izvoda:

Srpski/Engleski

JI

Zemlja publikovanja:

Republika Srbija

ZP

Uže geografsko područje:

Vojvodina

UGP

Godina:

2020

GO

Izdavač:

Autorski reprint

IZ

Mesto i adresa:

Novi Sad, Trg Dositeja Obradovića 4

MA

Fizički opis rada: 7/165 (xiv + 151)/61/18/26/0/2
(broj poglavlja/strana/lit. citata/ta-
bela/slika/grafika/priloga)

FO

Naučna oblast: Računarske nauke

NO

Naučna disciplina: Mašinsko učenje

ND

Predmetna odrednica / Ključne reči: k -NN graf, *NN-Descent*, aproksima-
tivni algoritmi, habnes

PO

UDK

Čuva se: Biblioteka Departmana za matematiku
i informatiku, Novi Sad

ČU

Važna napomena:

VN

Izvod: Graf najbližih suseda modeluje veze između objekata koji su međusobno bliski. Ovi grafovi se koriste u mnogim disciplinama, pre svega u mašin-
skom učenju, a potom i u pretraživanju informacija, biologiji, računarskoj
grafici, geografskim informacionim sistemima, itd. Fokus ove teze je graf
 k najbližih suseda (k -NN graf), koji predstavlja posebnu klasu grafova
najbližih suseda. Svaki čvor k -NN grafa je povezan usmerenim granama
sa njegovih k najbližih suseda.

Metod grube sile za generisanje k -NN grafova podrazumeva $O(n^2)$ raču-
nanja razdaljina između dve tačke. Ova teza se bavi problemom efikasni-
jeg generisanja k -NN grafova, korišćenjem aproksimativnih algoritama.
Glavni cilj aproksimativnih algoritama za generisanje k -NN grafova jeste
smanjivanje ukupnog broja računanja razdaljina između dve tačke, uz
održavanje visoke tačnosti krajnje aproksimacije.

NN-Descent je jedan takav aproksimativni algoritam za generisanje k -NN
grafova. Iako se pokazao kao veoma dobar u većini slučajeva, ovaj algo-
ritam ne daje dobre rezultate nad visokodimenzionalnim podacima. Un-
utar prvog dela teze, detaljno je opisana suština problema i objašnjeni
su razlozi za njegovo nastajanje.

U drugom delu predstavljeno je pet različitih modifikacija *NN-Descent* algoritma, koje za cilj imaju njegovo poboljšavanje pri radu nad visokodimenzionalnim podacima. Evaluacija ovih algoritama je data kroz eksperimentalnu analizu.

Treći deo teze se bavi algoritmima za ažuriranje k -NN grafova. Naime, podaci se vrlo često menjaju vremenom. Ukoliko se izmene podaci nad kojima je prethodno generisan k -NN graf, potrebno je graf ažurirati u skladu sa izmenama. U okviru ovog dela teze predložena su dva aproksimativna algoritma za ažuriranje k -NN grafova. Ovi algoritmi su evaluirani opširnim eksperimentima nad vremenskim serijama.

IZ

Datum prihvatanja teme od strane NN veća: 24. 2. 2020.

DP

Datum odbrane:

DO

Članovi komisije:

Predsednik: Dr Mirjana Ivanović, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Mentor: Dr Vladimir Kurbalija, vanredni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Član: Dr Miloš Radovanović, vanredni profesor, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Član: Dr Majkl Houl (Michael Houle), gostujući profesor, Nacionalni institut informatike, Tokio, Japan

Član: Dr Zoltan Geler, docent, Filozofski fakultet, Univerzitet u Novom Sadu

KO

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
KEY WORDS DOCUMENTATION

Accession number:

ANO

Identification number:

INO

Document type:

Monograph type

DT

Type of record:

Printed text

TR

Contents code:

Ph.D. thesis

CC

Author:

Brankica Bratić

AU

Mentor:

Vladimir Kurbalija, Ph.D.

MN

Title:

Approximation algorithms for k -NN
graph construction

TI

Language of text:

English

LT

Language of abstract:

Serbian/English

LA

Country of publication:

Republic of Serbia

CP

Locality of publication:

Vojvodina

LP

Publication year:

2020

PY

Publisher:

Author's reprint

PU

Publication place:

Novi Sad, Trg Dositeja Obradovića 4

PP

Physical description: 7/165 (xiv + 151)/61/18/26/0/2
(chapters/pages/literature/tables/
pictures/graphics/appendices)

PD
Scientific field: Computer science

SF
Scientific discipline: Machine learning

SD
Subject / Key words: *k*-NN graph, *NN-Descent*, approxima-
tion algorithms, hubness

SKW

UC
Holding data: Library of Department of Mathematics
and Informatics, Novi Sad

HD

Note:

N

Abstract: Nearest neighbor graphs are modeling proximity relationships between objects. They are widely used in many areas, primarily in machine learning, but also in information retrieval, biology, computer graphics, geographic information systems, etc. The focus of this thesis are *k*-nearest neighbor graphs (*k*-NNG), a special class of nearest neighbor graphs. Each node of *k*-NNG is connected with directed edges to its *k* nearest neighbors.

A brute-force method for constructing *k*-NNG entails $O(n^2)$ distance calculations. This thesis addresses the problem of more efficient *k*-NNG construction, achieved by approximation algorithms. The main challenge of an approximation algorithm for *k*-NNG construction is to decrease the number of distance calculations, while maximizing the approximation's accuracy.

NN-Descent is one such approximation algorithm for *k*-NNG construction, which reports excellent results in many cases. However, it does not perform well on high-dimensional data. The first part of this thesis summarizes the problem, and gives explanations for such a behavior. The second part introduces five new *NN-Descent* variants that aim to improve *NN-Descent* on high-dimensional data. The performance of the proposed algorithms is evaluated with an experimental analysis.

Finally, the third part of this thesis is dedicated to k -NNG update algorithms. Namely, in the real world scenarios data often change over time. If data change after k -NNG construction, the graph needs to be updated accordingly. Therefore, in this part of the thesis, two approximation algorithms for k -NNG updates are proposed. They are validated with extensive experiments on time series data.

AB

Accepted by Scientific Board on: 24. 2. 2020.

ASB

Defended:

DE

Thesis defend board:

President: Mirjana Ivanović, Ph.D., full professor, Faculty of Sciences, University of Novi Sad

Advisor: Vladimir Kurbalija, Ph.D., associate professor, Faculty of Sciences, University of Novi Sad

Member: Miloš Radovanović, Ph.D., associate professor, Faculty of Sciences, University of Novi Sad

Member: Michael Houle, Ph.D., visiting professor, National Institute of Informatics, Tokyo, Japan

Member: Zoltan Geler, Ph.D., assistant professor, Faculty of Philosophy, University of Novi Sad

DB

Овај Образац чини саставни део докторске дисертације, односно докторског уметничког пројекта који се брани на Универзитету у Новом Саду. Попуњен Образац укоричити иза текста докторске дисертације, односно докторског уметничког пројекта.

План третмана података

Назив пројекта/истраживања
Approximation algorithms for k-NN graph construction (Апроксимативни алгоритми за генерисање k-NN графа)
Назив институције/институција у оквиру којих се спроводи истраживање
а) Природно-математички факултет, Универзитет у Новом Саду
Назив програма у оквиру ког се реализује истраживање
1. Опис података
<i>1.1 Врста студије</i> <i>У овој студији нису прикупљани подаци.</i>
2. Прикупљање података
3. Третман података и пратећа документација
4. Безбедност података и заштита поверљивих информација
5. Доступност података
6. Улоге и одговорност