University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Piotr Wygocki

On Nearest Neighbors

PhD dissertation

Supervisor
dr hab. Piotr Sankowski
Institute of Informatics
University of Warsaw

February 2019

Author's declaration:
I hereby declare that this dissertation is my own work.

February 31, 2019 ....

Piotr Wygocki

Supervisor's declaration:
The dissertation is ready to be reviewed

February 31, 2019 ....

dr hab. Piotr Sankowski

Abstract

This dissertation is devoted to various aspects of the nearest neighbors search. The nearest neighbor related techniques are applied in diverse areas including computational geometry, databases, robotic sensing, DNA sequencing, spell checking, statistical classification, cluster analysis, chemical similarity, computer vision, plagiarism detection, recommendation systems, viral marketing, social networks, data compression, coding theory and pattern recognition. The basic concept is the following. Knowledge on a given object may be considerably increased by examining objects which are similar or in some way connected to it. The nearest neighbors search provides means to identify such objects. Depending on the meaning of "similar" or "connected" in a particular case, an appropriate definition of a neighbor may be introduced. In this study, we show efficient methods of searching for neighbors in different cases. Among the numerous contexts in which the nearest neighbors search is applied, we are going to focus on similarity search in metric spaces and information dissemination in social networks.

Metric Spaces

In this scenario, we are interested in efficient algorithms for similarity search in metric spaces. We are given a set of input points in $\mathbb{R}^d$. This set can be preprocessed to build a data structure which will be used for queries. The query consists of a single point in $\mathbb{R}^d$ and results in finding the nearest neighbor, i.e., an input point which is closest to the query point in a given metric. In this dissertation, we consider an equivalent[1] problem of finding a near neighbor, i.e., for a given input set and a radius R, we produce a data structure which answers the queries for a near neighbor. A near neighbor is

[1]Each of the problems can be reduced to the other one with small complexity overhead [1].

any point from the input set which is closer than R to the query point.

There are numerous applications of this problem. For instance, one might consider document search on the Internet. In the query phase, we upload a document and we wish to find different versions of the same document on the web. Another scenario involving the Internet search is finding the most similar photo to the one provided by the user. Moreover, a broad area of applications involves using the neighbor search for classification purposes. Classification of an object can be determined by searching for neighbors among already classified objects. Thus obtained information can also be used for prediction, which is a very dynamic field of study. One can make predictions about an object based on the knowledge on similar objects detected as neighbors.

In this work, we consider high–dimensional spaces and ℓ_p metrics. A naïve query algorithm is to scan all input points and find a near (the nearest) neighbor. However, the query time linear in the size of the input set is not acceptable for practical purposes. In applications, we are interested in algorithms with fast query times, thus we only consider algorithms where the query time is sub–linear in the size of the input set: $\mathcal{O}(n^{\gamma})$ for $\gamma < 1$. Also, since our space is high–dimensional, neither the pre–processing time nor the query time can be exponential in the dimension of the space. Unfortunately, there are no data structures which satisfy the above conditions.[2] Hence, in this work, we consider the c-approximate near neighbor problem in which the task is to find any point which is closer than R to the query point but we are allowed to return a point which is closer than cR.

The problem of the c–approximate near neighbor obtained large attention in the research community and many efficient data structures were designed [1, 2, 3, 4, 5, 6, 7]. Most of these algorithms satisfy Monte Carlo probabilistic guarantees. In this work, we focus on algorithms which satisfy stronger Las Vegas assumptions.

We present various algorithms for ℓ_p for $p \in [1, \infty]$. However, the best results were obtained for ℓ_2 metric. In particular, for $c < 2$ and $p = 2$ we present an algorithm with almost optimal pre–processing complexity $\mathcal{O}(d^{\omega-1}n)$ and query time $\mathcal{O}(d^{\omega-1} + dn^{\frac{1}{1+\mathcal{O}(\epsilon^2 \log^{-1} \frac{1}{\epsilon})}})$ for $\epsilon = c - 1$. These are, up to the writer's knowledge, the best results for Euclidean metric with Las Vegas guarantees.

We present many versions of the algorithms, which allows smooth tuning for different trade-offs between the pre–processing and the query time. In

[2] Specific conditions for the existence of such algorithms are described in further sections.

particular, we introduce a query–efficient algorithm, with the query time $\mathcal{O}(d^{\omega-1})$ and the pre-processing time $n^{1+\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$. These results are close to the best results obtained by Monte Carlo algorithms [6] for both the query–efficient and the preprocessing–efficient version. Moreover, we present algorithms for any $p \in [1, \infty]$. Unfortunately, the guarantees on the algorithms become worse, as p moves away from 2.

Some of the results were the effect of collective work and were previously published. The LSH–based results for $c = \Omega(\sqrt{d})$ were an outcome of collective work and were published in [8]. Enhanced results for $c = \Omega(\sqrt{\log\log n})$ were published in [9]. All of the remaining algorithms and proofs for algorithms for any $c > 1$ were not published before. These results not only relax the constraints on c but also improve the complexities of the algorithms.

Social Networks

In this section, we look at the nearest neighbors from a different perspective. We consider social networks in the context of information dissemination (or alternatively rumor spreading). In this scenario, the neighbors are not entities which are similar but entities which are in some way related. In this particular case, a neighbor of a user A is the user who is willing to spread information produced by A. This relation is asymmetric and cannot be directly embedded into the metric space framework presented in the previous section.

A typical application for computing neighborhoods is optimizing viral marketing. The aim is to choose a (possibly small) number of influencers, who will maximize the number of people reached by the marketing content. In particular, it is interesting to investigate the dynamics of the information cascades.[3] We want to know, what is a chance that a given cascade will be large.

The key part of computing such neighborhoods is to understand the process of information dissemination. The main results of this part of the dissertation consist of modeling of this process. We investigate known models and their adequacy to the Twitter network [10]. In particular, we consider the

[3]A cascade consists of all users who have spread a given information.

Susceptible, Infected, Recovered (SIR) model.[4] We propose a detailed method for comparing the rumor–spreading models. Moreover, we propose new models which better fit the real-life cascade distributions: the *exp–SIR* model and the multi–source model. In the *exp–SIR* model, we take into account the fact that the close relation with the followee increases the chance of spreading information. In the multi–source model, we assume that information might be spread outside of our network: by the word of mouth, television, radio, newspapers, phone etc. We show that both models avoid the "blow up" problem which was present in the previous models and that the distribution of the cascades is much more accurate.

Additionally, we consider a theoretical graph model and show the basic properties of the SIR model on this graph. We consider directed acyclic Erdős–Rényi graphs [11], which are based on the following concept. Assume that the nodes of a graph are labeled with natural numbers from 1 to n. Directed acyclic Erdős–Rényi graph is a random graph for which each edge (i, j) such that $i < j$ is sampled independently with probability p. Using this sampling mechanism we can produce any directed acyclic graph. We show basic properties of this graph and show that the distribution of information cascades in this graph satisfies the power law, which is often observed in the real–life cascade distributions.

The results presented in this part of the thesis are an outcome of collective work. The results concerning the modeling of information dissemination on Twitter were presented in [12]. The results concerning cascades in random graphs were presented in [13].

[4] In this model the rumor starts with one Infected node. Each Infected node can change each of its Susceptible followers to Infected with some constant probability and then moves to Recovered state. The process repeats until all Infected nodes change to Recovered. There are different variants of this model. The details are presented in Chapter 6.

Acknowledgements

I would like to thank my supervisor Piotr Sankowski for all the support and commitment to my scientific efforts. Thank you for the trust and belief that was given to me through all the years of my work at the University. I would like to express gratitude to my coauthors and team colleagues: Andrzej Pacuk and Adam Witkowski. Thank you for our discussions, collective thinking, experiments, code reviews. It certainly had a great impact on this thesis. I would also like to thank Piotr Skowron, Marcin Pilipczuk and Rafał Latała for valuable discussions concerning technical aspects of my thesis.

Contents

Part I

Nearest Neighbors Without False Negatives in $\mathbf{l_p}$

Chapter 1

Introduction

The near neighbor search has numerous applications in image processing, search engines, recommendation engines, predictions and machine learning [14]. We define the near neighbor problem as follows: for a given input set of points, a query point and a distance R, return a point (optionally all points) from the input set, which is closer to the query point than R in the given metric (typically l_p for $p \in [1, \infty]$), or report that such a point does not exist. The input set and the distance R are known in advance. Hence, the input set may be preprocessed what may result in reducing the query time. The problem in which the distance R is not known during the preprocessing and our task is to find the nearest neighbor can be efficiently reduced to the problem defined as above [1]. Unfortunately, the near neighbors search appears to be intractable for high dimensional spaces such as l_p^d for large d. The existence of an algorithm with a sub–linear in the data size and not exponential in d query time and with not exponential in d pre-processing would contradict the strong exponential time hypothesis [15]. In order to overcome this obstacle, the c–near neighbors problem with $c > 1$, was introduced. In this problem, the query result is allowed to contain points which are within the distance cR from the query point. In other words, the points within the distance R from the query point are classified as neighbors, the points farther than cR are classified as non-neighbors, while the rest may be classified into any of these two categories. This assumption makes the problem easier, for many metric spaces such as l_p when $p \in [1, 2]$ or the Hamming space [1]. On one hand, sub–linear in the input size queries are possible. On the other hand, the queries and pre-processing times are polynomial in the dimension of the space.

Locality sensitive hashing (LSH) is one of the major techniques for solving the c–near neighbors search. Many LSH functions are mappings which roughly preserve distances. A random LSH function maps two 'close' points to two 'close' hashes with 'large' probability. Analogously, two 'distant' points are mapped to two 'distant' hashes with 'large' probability. Roughly speaking, the LSH is used to reduce the dimension of the input space, what allows to solve the problem in the lower dimensional space. Thus, the efficiency of the algorithm strongly depends on the quality of LSH functions used. The crucial properties of the LSH functions are the probability of false positives and the probability of false negatives. A false negative is a point which is 'close' to the query point, but its hash is 'far away' from the hash of the query point. Analogously, the false positive is a point whose distance to the query point is 'large', but it is mapped to a 'close' hash.

The previously known algorithms for the c–near neighbors (see e.g., [2, 16]) give Monte Carlo guarantees for returned points, i.e., an input point close to the query point is returned with some probability. In other words, there might be false negatives. For example, a common choice of the hash functions is $f(x) = \langle x, v\rangle$ or $f(x) = \lfloor\langle x, v\rangle\rfloor$, where v is a vector of numbers drawn independently from some probability distribution [2, 1, 8]. For a Gaussian distribution, $\langle x, v\rangle$ is also Gaussian with zero mean and standard deviation equal to $\|x\|_2$. Hashing functions defined as above are in deed locality sensitive for l_2, but as mentioned above, they only give probabilistic guarantees. In this part of the thesis, we aim to enhance this by focusing on the c–near neighbors search without false negatives for l_2. In other words, we consider algorithms, where a point 'close' to the query point is guaranteed to be returned. These type of guaranties are often called Las Vegas.

The Las Vegas guaranties are stronger than the Monte Carlo ones. Moreover, an algorithm with Las Vegas guarantees can be adjusted to one with Monte Carlo guarantees with preserving the complexities. Usually, Las Vegas algorithms have probabilistic complexities, i.e., we bound the average complexity. Having the algorithm with the Las Vegas guarantees, we might obtain algorithm with the Monte Carlo guaranties with pessimistic time complexity equal to the expected time complexity of the original Las Vegas algorithm. Markov's inequality implies, that if the expected value of the computation time is small, then with large probability the computation time is also small. We can break the computation of the original algorithm after the certain amount of time passed and return the empty result which gives the algorithm with Monte Carlo guarantees.

Throughout this thesis, we assume that $n \gg d$ and $\exp(d) \gg n$, where n is the number of points in the input space and d is the dimensionality of the space. This represents a situation where the exhaustive scan through all the input points, as well as the usage of data structures exponentially dependent on d, become intractable. The typical values to consider could be $n = 10^9$ and $d = 100$.

1.1 Preliminaries

In this section, we present common definitions, simple facts and known constructions used throughout this part of the thesis. We start with some crucial definitions:

Definition 1.1.1. *In the c–near neighbors without false negatives problem, we are given a set of points $X \in \mathbb{R}^d$, $|X| = n$, a radius R and metric $\|\cdot\|$. The goal is to build a data structure, which supports fast query operations. In each query, the data structure receives a point $q \in \mathbb{R}^d$. The data structure returns all points $Y \subset X$ such that $\{x : x \in X : \|q - x\| \leq R\} \subset Y$. Additionally, all the returned points cannot be "far away" from the query point: $\forall_{x \in Y} \|q - x\| \leq cR$.*

We often consider the simplified version in which we need to find only one neighbor:

Definition 1.1.2. *In the c-near neighbor without false negatives problem, we are given a set of points $X \in \mathbb{R}^d$, $|X| = n$, a radius R and metric $\|\cdot\|$. The goal is to build a data structure, which supports fast query operations. In each query, the data structure receives a point $q \in \mathbb{R}^d$. If there is a point $x \in X$ such that $\|q - x\| \leq R$, then the data structure returns any point $x' \in X$ such that $\|q - x'\| \leq cR$. If there is no such a point, the data structure could either return NO or return any point $x' \in X$ such that $\|q - x'\| \leq cR$.*

All algorithms presented in this part works in the more general c–near neighbors without false negatives problem. Sometimes, for simplicity, we present results for the c-near neighbor without false negatives problem, i.e., the problem in which the algorithm returns only one near neighbor. As mentioned in the introduction, if the neighbor exists it is found with probability 1. The input set is always assumed to contain n points. The c–near neighbors search without false negatives with parameter $c > 1$ and the dimension of the space equal to d, is denoted as $\mathsf{NN}_{\text{wfn}}(c, d)$. The expected query and pre-processing time complexities of $\mathsf{NN}_{\text{wfn}}(c, d)$ will be denoted as $\text{query}(c, d)$ and $\text{preproc}(c, d)$ respectively.

In this paper, we use the term "pre-processing" to refer to the sum of the actual pre-processing time and the storage required. W.l.o.g, throughout this work we assume, that R – a given radius equals 1 (otherwise, all vectors' lengths might be rescaled by $1/R$). It is often convenient to use $\epsilon = c - 1$ instead of c. Whenever ϵ appears, it is assumed to be defined as above. Finally, we use ω to denote the exponent of the fast matrix multiplication (currently $\omega \approx 2.37$).

We will often need a data structure which is capable of storing some set of values for a given key. In the pre-processing phase, we will assign some values to a given key. In the query phase, we fetch all values assigned to a given key. Indyk and Motwani [1] provided a hash-map storage. Let us consider the following standard, fully deterministic construction. For each of the stored points, we store its bit representation in a binary tree. This way the length of the branch representing a point equals to its bit-length. Hence, the query time is proportional to the bit size of the query. The size of the whole tree is bounded by the total size of the binary representation of all the stored points.

1.1.1 Notation

The considered problems are solved in the standard ℓ_p space with metric $\|\cdot\|_p$:

$$\|x\|_p = (\sum_i |x_i|^p)^{1/p}.$$

By ℓ_p^d we denote ℓ_p in $\mathbb{R}^d$. Euclidean space is l_p for $p = 2$, i.e.:

$$\|x\|_2 = \sqrt{\sum_i |x_i|^2}.$$

For $x, y \in \mathbb{R}^d$, $\langle x, y \rangle$ denotes the standard scalar product, i.e.:

$$\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i.$$

The random variable X is bounded when:

$$\mathbb{P}\left[|X| \leq C\right] = 1, \text{ for some constant } C \in \mathbb{R}^d.$$

The random variable X is symmetric when:

$$\mathbb{P}\left[X \leq C\right] = \mathbb{P}\left[X \geq -C\right], \text{ for all } C \in \mathbb{R}^d.$$

$[d]$ denotes the set of all natural numbers from 1 to d:

$$[d] = \{1, 2, ..., d\}.$$

$\mathbb{S}_p^{(d-1)}$ denotes a unit sphere in ℓ_p, i.e.:

$$\mathbb{S}_p^{(d-1)} = \{x : x \in \mathbb{R}^d, \|x\|_p = 1\}.$$

We will write $\mathbb{S}^{(d-1)}$ instead of $\mathbb{S}_2^{(d-1)}$. $\mathbb{B}_p^d$ denotes a unit ball in l_p, i.e.:

$$\mathbb{B}_p^d = \{x : x \in \mathbb{R}^d, \|x\|_p \leq 1\}.$$

$U(a, b)$ denotes the uniform distribution on the interval $[a, b]$. The *i.i.d* is the abbreviation for independent and identically distributed. Rademacher distribution is a two–point symmetric discrete distribution on $\{-1, 1\}$, i.e., for random variable X with Rademacher's distribution, we have $\mathbb{P}\left[X = \pm 1\right] = 1/2$.

Assume that we are given a classificator X which examines if the given object is in a certain class. If the object is classified as belonging to the class we have $X = 1$ and if it's classified as not belonging to the class, we have $X = 0$. Assume that Y is the true class of the object, i.e., $Y = 1$ if the object is in the class and $Y = 0$ otherwise. Our classificator might be wrong. It can predict $X = 1$, while $Y = 0$. Such a prediction is false positive. When $X = 0$ and $Y = 1$, we have false negative. The following table illustrate this notions:

		prediction	
		$X = 1$	$X = 0$
true label	$Y = 1$	True positive	False negative
	$Y = 0$	False positive	True negative

In our case, we classify object as neighbor. In Monte Carlo versions of the c–near neighbors problem, classificator could be wrong – there might be false negatives. In other words, the algorithm might not return point which is a near neighbor. In c–near neighbors without false negatives problem (a.k.a. c–near neighbors with Las Vegas guaranties), this cannot be the case, i.e., there are no false positives and false negatives. More precisely, in all our algorithms (and many other probabilistic algorithms for the c–near neighbors) the number of false positives is bounded, thus they are filter out from the final result. This is the reason why we often consider the probability of false positive (e.g., in context of some hashing function), bounding this probability allow bounding the number of false positives. The time of removing the false positives from the final results contributes to the total query complexity.

1.2 Related Work

There is extensive literature considering near neighbors problem. In this section, we focus on the most important and the most relevant publications.

1.2.1 Algorithms for constant dimension

There is a number of algorithms for the c–near neighbors problem that assume constant d [17, 18, 19, 1]. In each of them, either the pre-processing time or the query time depends exponentially on the dimension of the space. Nevertheless, these are the best fully deterministic algorithms that are known [17, 1]. A particularly interesting algorithm, presented in [1], has pre-processing time $n\mathcal{O}(1/\epsilon)^d$ and the query time equal to $\mathcal{O}(d)$, where $\epsilon = c - 1$. We use these results to obtain our algorithms.

1.2.2 Monte Carlo algorithms

There exists an efficient Monte Carlo c–near neighbors algorithm for ℓ_1 with the query and the pre-processing complexity equal to $\mathcal{O}(dn^{1/c})$ and $\mathcal{O}(n^{1+1/c} + dn)$, respectively [1]. By reduction, the algorithms for ℓ_1 works in the Hamming space. For ℓ_2 in turn, there exists a near to optimal algorithm with the query and the pre-processing complexity equal to $\mathcal{O}(dn^{1/c^2+o(1)})$ and $\mathcal{O}(n^{1+1/c^2+o(1)} + dn)$, respectively [2, 3].

Moreover, the algorithms presented in [1] work for l_p for any $p \in [1, 2]$. There are also data dependent algorithms that take into account the actual distribution of the input set. Such algorithms achieve query time $\mathcal{O}(dn^{\rho+o(1)})$ and space $\mathcal{O}(n^{1+\rho+o(1)} + dn)$, where $\rho = 1/(2c^2 - 1)$ [4].

An interesting algorithm was introduced in [5]. Alion et al. [5] considered slightly more general problem of Nearest Neighbors in l_2 and achieved $\mathcal{O}(d \log d + \log^3 n)$ query time and $n^{\mathcal{O}(1/\epsilon^2)}$ pre-processing time.

Recently, the optimal hashing–based time–space trade-offs for the c–near neighbors in l_2 were considered [6].[1] For any $p_u, p_q \geq 0$ such that:

$$c^2\sqrt{p_q} + (c^2-1)\sqrt{p_u} = \sqrt{2c^2-1},$$

there is a c–near neighbors algorithm with the storage $\mathcal{O}(n^{1+p_u+o(1)} + dn)$ and the query time $n^{p_q+o(1)} + dn^{o(1)}$.

1.2.3 Las Vegas algorithms

Pagh [20] considered the c–near neighbors search without false negatives for the Hamming space. He obtained results close to those presented in [1]. He showed that the exponents of his algorithm, for $cR = \log(n/k)$, differ by at most a factor of $\ln 4$ in comparison to the bounds in [1]. Recently, Ahle showed an algorithm for the c–near neighbors without false negatives for the Hamming space and Braun-Blanquet metric [21, 3]. The complexity of this algorithm matches the complexity of the Monte Carlo algorithm of Indyk and Motwani [1]. Indyk [7] provided a deterministic algorithm for l_∞ and $c = \Theta(\log_{1+\rho} \log d)$ with the storage $\mathcal{O}(n^{1+\rho} \log^{O(1)} n)$ and the query time $\log^{O(1)} n$ for some tunable parameter ρ. He proved that the c–near neighbors without false negatives for l_∞ and $c < 3$ is as hard as the subset query problem – a long-standing open combinatorial problem. This indicates that the c–near neighbors without false negatives for l_∞ might be hard to solve for any $c > 1$.

Indyk [22] considered deterministic mappings $l_2^n \to l_1^m$, for $m = n^{1+o(1)}$, which might be useful for constructing efficient algorithms for the c–near neighbors without false negatives. If we were able to efficiently embed l_1 into the Hamming space (which is just $\{0,1\}^d$ with l_1 distance function) with additional guarantees for false negatives, it would also give an algorithm for l_2 and l_1.

1.2.4 Dimension reduction

The dimension reduction with usage of random linear mappings was considered previously in a more general context. The concentration bounds used to prove

[1]The authors of this work assume $d = n^{o(1)}$.

this classic result will be very useful in our reductions:

Lemma 1.2.1 (Johnson-Lindenstrauss)**.** *Let $Y \in \mathbb{R}^d$ be chosen uniformly from the surface of the d-dimensional sphere. Let $Z = (Y_1, Y_2, \dots, Y_k)$ be the projection onto the first k coordinates, where $k < d$. Then for any $\alpha < 1$:*

$$\mathbb{P}\left[\frac{d}{k}\|Z\|_2^2 \leq \alpha\right] \leq \exp(\frac{k}{2}(1 - \alpha + \log\alpha)), \tag{1.1}$$

1.2.5 Anti-concentration measures

In this thesis, we show the anti-concentration measures for $\langle v, x\rangle$, for $x \in \mathbb{S}_p^{(d-1)}$, i.e., the upper bound for $\mathbb{P}\left[|\langle v, x\rangle| \leq \alpha\right]$. This is crucial subproblem in the LSH based techniques.

Let us start with a general bound for functions on a sphere. Particularly, in the *small ball probability theorem*, we bound $\mathbb{P}\left[|f(x)| \leq \alpha\right]$ for some function f on the unit sphere $\mathbb{S}^{(d-1)}$. The theorem conjectured in [23] and proved in [24] implies that for any Lipschitz function f, with Lipschitz constant L, whose average over the sphere is 1, we have $\mathbb{P}\left[|f(x)| \leq \alpha\right] \leq \alpha^{c/L^2}$, for some constant c and $x \in \mathbb{S}^{(d-1)}$.

Carbery and Wright [25] show the following bound for polynomial functions. There exists an absolute constant $c > 0$ such that, if $Q : \mathbb{R} \to \mathbb{R}$ is a polynomial of degree at most k and μ is a log-concave probability measure on $\mathbb{R}^m$, then for all $\alpha > 0$:

$$\Big(\int Q^2 d\mu\Big)^{\frac{1}{2k}} \mu\{x \in \mathbb{R}^m : |Q(x)| \leq \alpha\} \leq ck\alpha^{\frac{1}{k}}.$$

Since log-concave probability measures are strongly connected to the surface measure (see Lemma 2 in [23]), the above result gives an alternative way of proving the bounds for the probability of false positive presented in this thesis (see Section 3.3.6). The anti-concentration bound achievable using [25] gives worse constants than the alternative proof provided in this work. This is important since this constant is in the exponent of the complexities of the NN_{wfn} algorithm.

The anti-concentration measures are strongly connected with the Littlewood-Offord theory. Consider Lévy concentration function:

$$Q(X, \lambda) = \sup_x \mathbb{P}\left[X \leq x \leq X + \lambda\right].$$

We have $\mathbb{P}[|X| \le \alpha] \le Q(X, 2\alpha)$. So any bound on the Lévy concentration function is also a bound on our problem. Bobkov et al. [26] considered bounds on the Lévy concentration function for X being the sum of independent random variables with log-concave density function: (Theorem 1.1 in [26]):

Theorem 1.2.2. *If $X_1, \ldots, X_k$ are independent random variables with log-concave distribution and let $S = \sum_i X_k$. Then for all $\lambda \ge 0$*

$$Q(S, \lambda) \le \frac{\lambda}{\sqrt{Var(S) + \frac{\lambda^2}{12}}}.$$

The above theorem is used to prove anti-concentration bounds for log-concave bounded distributions. Let $M(X) = \operatorname{ess\,sup}_X p_X(x)$, where p_X is density function of X. A well-known result of Ball [27] allows us to prove the better result for uniform distribution (see also Proposition 3.1 in [26]):

Theorem 1.2.3. *If $X_1, \ldots, X_k$ are independent random variables with uniform distribution on $[-1, 1]$ and let $S = \sum_i X_k$, then $M(S) \le \frac{1}{\sqrt{2}}$.*

1.3 Results

In this sections, we summarize our new results for c–near neighbors without false negatives in ℓ_p for $p \in [1, \infty]$. The best results are achieved for Euclidean norm ($p = 2$), and they get gradually worse as p moves away from 2. In particular, for $p = 2$, we present efficient algorithm for any $c > 1$, while for $p = 1$ and $p = \infty$ our algorithms works for $c = \Omega(\sqrt{d})$. We also consider different trade-offs between pre-processing and query time. The most query–efficient algorithms achieve query time $\mathcal{O}(poly(d))$ and polynomial pre-processing time. The most pre-processing–efficient algorithms achieve pre-processing time $\mathcal{O}(n\ poly(d))$ and sub–linear query time.

1.3.1 Used Methods

We present two approaches for solving the $\mathsf{NN}_{\mathrm{wfn}}$:

- **The LSH approach.** In this approach, we design Las Vegas LSH functions, i.e., hash functions which have the property, that, with probability 1, two "close" points will remain "close" after hashing. Two "distant" points will remain "distant" with large probability. Using these hash functions we construct an algorithm which enables us to solve $\mathsf{NN}_{\mathrm{wfn}}$.

- **The counting approach.** In this approach, we quantize our space, so that all points have integer coefficients. After the quantization, there is a finite number of neighbors for the input points. Thus, we can store all of them, which gives simple algorithm for $\mathsf{NN}_{\mathrm{wfn}}$.

To achieve the best results, both of the techniques are combined with the dimension reduction technique which helps us decrease the dimension of the given space to $\mathcal{O}(\log n)$. This operation adds a factor of $d/\log n$ to complexities but it enables us to use the algorithms which are exponential in the dimension. The whole scheme of the used methods is presented in Figure 1.1 .

The crucial step in building efficient algorithms is dimension reduction. In Chapter 2, we show how to reduce the c–near neighbors in l_2^d ($\mathsf{NN}_{\mathrm{wfn}}(c, d)$) to $d/\log(n)$ instances of $\mathsf{NN}_{\mathrm{wfn}}(\mathcal{O}(c), \mathcal{O}(\log n))$. The reduction is based on the well-known Johnson-Lindenstrauss Lemma [28]. We introduce $d/\log(n)$ linear mappings, each reduces the dimension of the original problem. Each mapping roughly preserves the length of the vector and additionally at least one of them does not increase it. The property of not increasing the length of the vector is crucial. For two 'close' vectors $x, y \in \mathbb{R}^d$: $\|x - y\|_2 < 1$ and a linear mapping A, Ax and Ay are 'close' if and only if $\|Ax - Ay\|_2 = \|A(x - y)\|_2 < 1$, so A maps a 'small' vector $x - y$, to a 'small' vector $A(x - y)$.

We show further reductions, which enable us to relax the constraint to $c = \omega(\sqrt{\log\log n})$. We extend the reduction by using a number of mapping families. This leads to an interesting sub-problem of solving the c–near neighbors in $(\mathbb{R}^k)^L$, for norm l_∞–product$(x) := \max_{1 \le i \le L} \|x_i\|_2$ and the induced metric. This norm is present in literature and was denoted as max–product or l_∞–product. Apparently, the c–near neighbors search in l_∞–product might be solved using the LSH functions family introduced in Chapter 3.

Moreover, to show the properties of a hash function we study anti–concentration bounds which leads to interesting geometrical problems. In particular, we show tight, up to small constant factor, lower bound for the area of the spherical cup. We demonstrate, that $\mathbb{P}\left[|\langle x, y\rangle| < \alpha\right]$, where x, y are random points from $\mathbb{S}^{(d-1)}$ can be interpreted as the area of a spherical cup. We show, that $\mathbb{P}\left[|\langle x, y\rangle| < \alpha\right] \le \alpha\sqrt{d}$ (see Observation 3.3.16) and we argue that this inequality is tight up to constant.

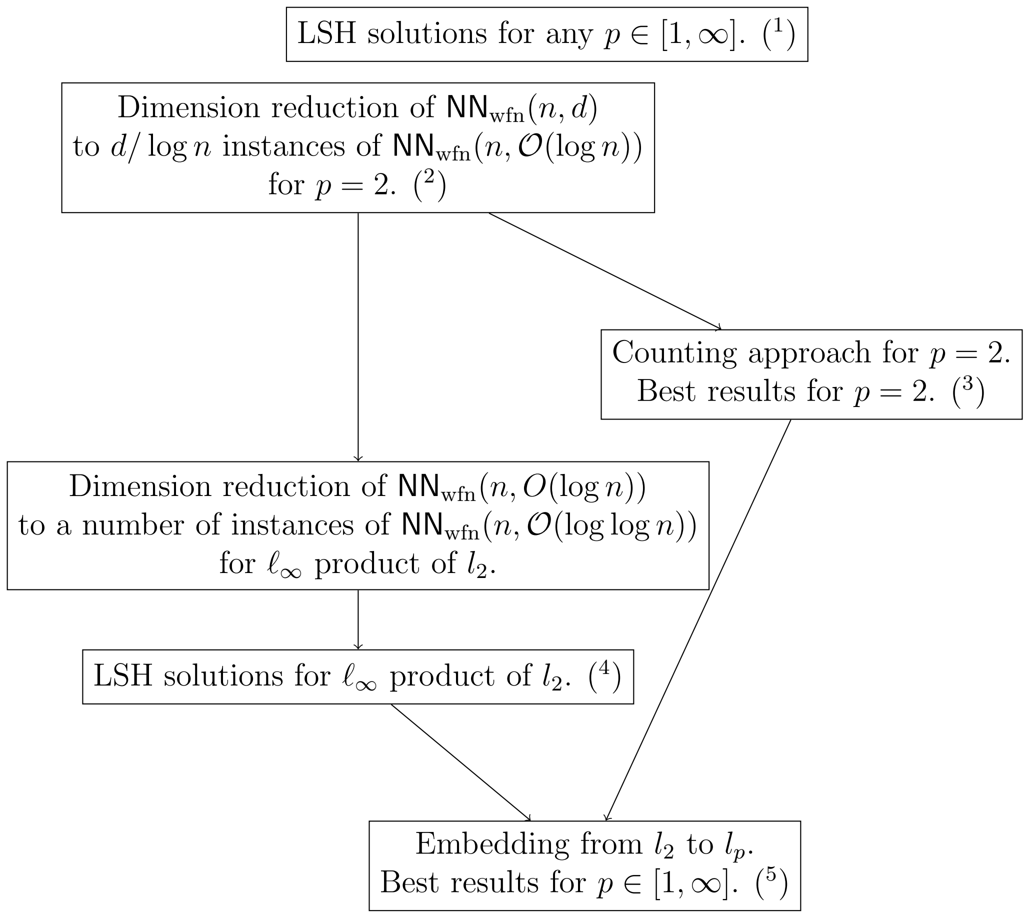
LSH solutions for any $p \in [1, \infty]$. (1)
Dimension reduction of $\mathsf{NN}_{\text{wfn}}(n, d)$ to $d/\log n$ instances of $\mathsf{NN}_{\text{wfn}}(n, \mathcal{O}(\log n))$ for $p = 2$. (2)
Counting approach for $p = 2$. Best results for $p = 2$. (3)
Dimension reduction of $\mathsf{NN}_{\text{wfn}}(n, O(\log n))$ to a number of instances of $\mathsf{NN}_{\text{wfn}}(n, \mathcal{O}(\log\log n))$ for ℓ_∞ product of l_2.
LSH solutions for ℓ_∞ product of l_2. (4)
Embedding from l_2 to l_p. Best results for $p \in [1, \infty]$. (5)

Figure 1.1: Scheme of the techniques used to achieve final results. (1) - results are presented in Table 1.1 ($p \neq 2$). (2) - results are presented in Corollary 1.3.1. (3) - results are presented in Table 1.2. (4) - results are presented in Table 1.1 for ($p = 2$). (5) - results are presented in Table 1.3. We only present results for embedding the counting approach.

In Chapter 4, we round all points and consider NN_{wfn} in integer domain. In integer domain, there is finite number of points that can be neighbors of the points from input set. The naïve algorithm would store all of these near neighbors in a data structure. In the query, we just fetch the right answer from our storage. Although storing of all possible solutions is exponential in the dimension of the space, we can obtain the polynomial algorithm by applying the dimension reduction. This enables us to construct an algorithm for any $c > 1$ for ℓ_2. Applying standard metric embedding, we obtain results for any $p \in [1, \infty]$.

1.3.2 Dimension reduction

Let us state the crucial results for the dimension reduction. We show that any instance of c–near neighbors can be reduced to a small number of instances of dimension logarithmic in the number of input points.

Corollary 1.3.1. *For any $1 \leq \alpha < c$ and $\gamma \log n < d$, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to $\mathcal{O}(d/(\gamma \log n))$ instances of the $\mathsf{NN}_{\text{wfn}}(\alpha, \gamma \log n)$, where $\gamma < \frac{2(1-\nu)}{(\frac{\alpha}{c})^2 - 1 - 2\log\frac{\alpha}{c}}$ for a tunable parameter $\nu \in [0, 1)$ and:*

$$\text{query}(c, d) = \mathcal{O}(d^2 + n^\nu + d/(\gamma \log n)\ \text{query}(\alpha, \gamma \log n)),$$

$$\text{preproc}(c, d) = \mathcal{O}(d^{\omega-1} n + d/(\gamma \log n)\ \text{preproc}(\alpha, \gamma \log n)).$$

If the queries are provided in the batches of the size d, we obtain the algorithm with the query time:

$$\text{query}(c, d) = \mathcal{O}(d^{\omega-1} + n^\nu + d/(\gamma \log n)\ \text{query}(\alpha, \gamma \log n)).$$

The tunable parameter ν enables us to achieve different trade–offs between, the query time overhead in dimension reduction, and the dimension of the result space. If $\nu = 0$, there is no overhead dependent on n in the query. If ν is close to 1, the overhead is nearly linear in n, while the dimension of the resulting space is constant. This corollary will be the building block used to achieve efficient algorithms for NN_{wfn}.

1.3.3 Results Outline

Table 1.1 summarizes the results for our **LSH approach**. The most important results are for $p = 2$, for which we obtained efficient algorithm for $c = \omega(\sqrt{\log \log n})$. Moreover, we distinguish cases for $p > 2$ and $p < 2$ and the fast query and the fast pre-processing versions. In the fast query version, the query time is $\mathcal{O}(\log n)$ and the pre-processing time is polynomial and depends on the value of p. In the fast pre-processing version, the pre-processing time equals $\mathcal{O}(n \log n)$ and the query time complexity is sub-linear in the number of points and depends on p.

Version	**Preprocessing time**	**Query time**
$p = 2$ $c > \mu = \omega(\sqrt{\log\log n})$ (*)	$\tilde{\mathcal{O}}(n^{1+\frac{\ln 3}{\ln(c/\mu)}+o(1)})$	$\tilde{\mathcal{O}}(n^{o(1)})$
$p \geq 2$ fast query $c > \tilde{\tau}_p = 2d^{1-1/p}$ (**)	$\mathcal{O}(n^{1+\frac{\ln 3}{\ln(c/\tilde{\tau}_p)}})$	$\mathcal{O}(\log n)$
$p \geq 2$, fast pre–processing $c > \tilde{\tau}_p = 2d^{1-1/p}$ (**)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{\frac{\ln 3}{\ln(3c/\tilde{\tau}_p)}})$
$p < 2$ fast query $c > \hat{\tau}_p = 2\sqrt{2d}$ (**)	$\mathcal{O}(n^{1+\frac{\ln 3}{\ln(c/\hat{\tau}_p)}})$	$\mathcal{O}(\log n)$
$p < 2$, fast pre–processing $c > \hat{\tau}_p = 2\sqrt{2d}$ (**)	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{\frac{\ln 3}{\ln(3c/\hat{\tau}_p)}})$

Table 1.1: Results outline for the LSH approach. Results marked with (*) follow from Theorem 3.4.3. Results marked with (**) follow from Theorem 3.4.1. For simplicity, we omit the dependence of d in complexities.

Table 1.2 summarizes the results for the **counting approach** for $p = 2$. In this table, we distinguish cases for $c > 2$ and $c < 2$ and the fast query and the fast pre-processing versions. In the fast pre-processing version, the pre-processing time equals $\mathcal{O}(nd^{\omega-1})$ and is very close to the optimal time $\mathcal{O}(nd)$. In the fast query version, we present complexities for a tunable parameter ν. When $\nu = 0$, the query complexity equals $\mathcal{O}(d^{\omega})$ which is close to the optimal $\mathcal{O}(d)$.

The results presented in Table 1.2 for $c \geq 2$ can be generalized to any $p \in [1, \infty]$ which is presented in Table 1.3. If $p = 2$, we get sames results as these presented in Table 1.2. These results gets gradually worse while p moves away from 2. However, for $p \neq 2$, for the fast pre-processing version, the pre-processing time is still close to optimal and for the fast query version, the query time is close to the optimal.

For simplicity, we omit the results that can be obtained by applying Lemma 4.0.3 to results of Theorem 3.4.3. This is, we do not show the results for $p \in [1, \infty]$ which can be derived for the best results using LSH approach.

Table 1.4 shows the comparison of the important cases of fast pre–processing and fast query for $c = 2$ with previous results. We demonstrate the state of the art algorithms for Monte Carlo, Las Vegas and deterministic algorithms. Moreover, we show different trade–offs between pre-precessing and query times. In particular, our pre-processing time efficient Las Vegas

Version	**Preprocessing time**	**Query time**
fast pre–processing $c < 2$ (*)	$\mathcal{O}(d^{\omega-1}n)$	$\mathcal{O}(d^{\omega-1} + dn^{\frac{1}{1+\mathcal{O}(\epsilon^2 \log^{-1}\frac{1}{\epsilon})}})$
fast query $c < 2$ (**)	$n^{1+\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon})}$	$\mathcal{O}(d^{\omega-1} + n^{\nu})$
fast pre–processing $c \geq 2$ (*)	$\mathcal{O}(d^{\omega-1}n)$	$\mathcal{O}(d^{\omega-1} + dn^{\mathcal{O}(\frac{1}{\log c})})$
fast query $c \geq 2$ (**)	$\mathcal{O}(d^{\omega-1}n + dn^{1+\mathcal{O}(\frac{1-\nu}{\log c})}/\log n)$	$\mathcal{O}(d^{\omega-1} + n^{\nu})$

Table 1.2: table
Results outline for the counting approach for $p = 2$. We set $\epsilon = c - 1$ and ν is a tunable parameter. Results marked with (*) follow from Theorem 4.0.1. Results marked with (**) follow from Theorem 4.0.2.

Version	**Preprocessing time**	**Query time**
fast pre–processing $c \geq 2d^{\lvert 1/2-1/p \rvert}$ (*)	$\mathcal{O}(d^{\omega-1}n)$	$\mathcal{O}(d^{\omega-1} + dn^{\mathcal{O}(\frac{1}{\alpha})})$
fast query $c \geq 2d^{\lvert 1/2-1/p \rvert}$ (**)	$\mathcal{O}(d^{\omega-1}n + dn^{1+\mathcal{O}(\frac{1-\nu}{\alpha})}/\log n)$	$\mathcal{O}(d^{\omega-1} + n^{\nu})$

Table 1.3: table
Results outline for the counting approach for any $p \in [1, \infty]$. Results follow from Lemma 4.0.3 applied to Theorems 4.0.1 and 4.0.2.
$\alpha = \log c - \lvert 1/2 - 1/p \rvert \log d$.

author	guarantees	query	pre-processing	space
[1]	Monte Carlo	$\mathcal{O}(dn^{1/c})$	$\mathcal{O}(n^{1+1/c} + dn)$	l_p for $c > 1$, $p \in [1, 2]$
[2]	Monte Carlo	$\mathcal{O}(dn^{1/c^2+o(1)})$	$\mathcal{O}(n^{1+1/c^2+o(1)} + dn)$	l_2 for $c > 1$
[4]	Monte Carlo	$\mathcal{O}(dn^{1/(2c^2-1)+o(1)})$	$\mathcal{O}(n^{1+1/(2c^2-1)+o(1)} + dn)$	l_2 for $c > 1$
[7]	Deterministic	$\log^{\mathcal{O}(1)} n$	$\mathcal{O}(n^{1+\rho} \log^{\mathcal{O}(1)} n)$	l_∞ for $c = \Omega(\log_{1+\rho} \log d)$
[20]	Las Vegas	$\mathcal{O}(dn^{\ln(4)/c})$	$\mathcal{O}(n^{1+\ln(4)/c} + dn)$	Hamming for $c > 1$
[21]	Las Vegas	$\mathcal{O}(dn^{1/c+o(1)})$	$\mathcal{O}(dn^{1+1/c+o(1)})$	Hamming for $c > 1$
[6]	Monte Carlo	$\mathcal{O}(d + n^{1-\mathcal{O}(\epsilon^2)+o(1)})$	$\mathcal{O}(dn + n^{1+o(1)})$	l_2 for $c > 1$
our results	Las Vegas	$\mathcal{O}(d^{\omega-1} + dn^{1-\mathcal{O}(\epsilon^2 \log^{-1} \frac{1}{\epsilon})})$	$\mathcal{O}(d^{\omega-1}n)$	l_2 for $c > 1$
[6]	Monte Carlo	$n^{o(1)}$	$n^{\mathcal{O}(1/\epsilon^2)+o(1)}$	l_2 for $c > 1$
[5]	Monte Carlo	$\mathcal{O}(d \log d + \log^3 n)$	$n^{\mathcal{O}(1/\epsilon^2)}$	l_2 for $c > 1$
our results	Las Vegas	$\mathcal{O}(d^{\omega-1})$	$n^{\mathcal{O}(1/\epsilon^2 \log 1/\epsilon)}$	l_2 for $c > 1$

Table 1.4: Comparison of the results for the c–near neighbors. We present only "fast query" and "fast pre-processing" parts of results for possibly small c. Also, results presented in [6] are under assumption that $d = n^{o(1)}$). Results in [7] are for a tunable parameter ρ. The parameter ϵ equals $c - 1$ and ω is the exponent of fast matrix multiplication.

algorithm in ℓ_2 works with pre-processing time equal to $\mathcal{O}(nd^{\omega-1})$ and the query time equal to $\mathcal{O}(d^{\omega-1} + dn^{1-\mathcal{O}(\epsilon^2 \log^{-1} \frac{1}{\epsilon})})$. While the best results with Monte Carlo guaranties are of [6] with $\mathcal{O}(d+n^{1-\mathcal{O}(\epsilon^2)+o(1)})$ and $\mathcal{O}(dn+n^{1+o(1)})$ query and pre-processing times respectively.

Our query time efficient Las Vegas algorithm works with $\mathcal{O}(d^{\omega-1})$ and $n^{\mathcal{O}(1/\epsilon^2 \log 1/\epsilon)}$ query and pre-processing time respectively. While the best results with Monte Carlo guaranties are of [6, 5] with $n^{o(1)}$ and $n^{\mathcal{O}(1/\epsilon^2)+o(1)}$ query and pre-processing times respectively for [6] and $\mathcal{O}(d \log d + \log^3 n)$ and $n^{\mathcal{O}(1/\epsilon^2)}$ query and pre-processing times respectively for [5].

Chapter 2

Dimension Reduction

The natural approach to handle the c–near neighbors problem in a high dimensional space is to reduce the problem to a number of problems with smaller dimension. This idea was widely used in algorithms with Monte Carlo guaranties (see e.g. [1]). In order to obtain Las Vegas guaranties, we show how to reduce a c–near neighbors problem to a number of c–near neighbors problems with reduced dimension, in such a away, that for given two points, the distance between these points is not increased in at least one instance. In other words, if the points were neighbors in the original space, they will remain neighbors in at least one of the resulting spaces. This way, we eliminate the false negatives. Our results are based on the well–known distribution Johnson-Lindenstrauss Lemma:

Lemma 2.0.1 (Johnson-Lindenstrauss)**.** *Let $Y \in \mathbb{R}^d$ be chosen uniformly from the surface of a d-dimensional sphere. Let $Z = (Y_1, Y_2, \dots, Y_k)$ be the projection onto the first k coordinates, where $k < d$. Then for any $\alpha < 1$:*

$$\mathbb{P}\left[\frac{d}{k}\|Z\|_2^2 \leq \alpha\right] \leq \exp(\frac{k}{2}(1 - \alpha + \log \alpha)). \tag{2.1}$$

In some sense we present the Las Vegas version of the above lemma.

In this chapter, we focus only on reductions. In Chapter 3, we show how to handle the problems in the reduced space using locality sensitive hashing. In Chapter 4, we show better results using quantization approach.

2.1 Reduction to dimension $\mathcal{O}(\log n)$

We will introduce d/k linear mappings $A^{(1)}, A^{(2)}, \ldots, A^{(d/k)} : R^d \to R^k$, where $k < d$ and show the following properties[1]:

1. for each point $x \in \mathbb{R}^d$, such that $\|x\|_2 \leq 1$, there exists $1 \leq i \leq d/k$, such that $\|A^{(i)}x\|_2 \leq 1$,
2. for each point $x \in \mathbb{R}^d$, such that $\|x\|_2 \geq c$, where $c > 1$, the probability that there exists $1 \leq i \leq d/k$, such that $\|A^{(i)}x\|_2 \leq 1$ is bounded away from 1.

The Property 1. states, that for a given 'short' vector (with a length smaller than 1), there exists always at least one mapping, which transforms this vector to a vector of length smaller than 1. Moreover, we will show, that there exists at least one mapping $A^{(i)}$, which does not increase the length of the vector, i.e., such that $\|A^{(i)}x\|_2 \leq \|x\|_2$. The property 2. states, that we can bound the probability of a 'long' vector ($\|x\|_2 > c$), being mapped to a 'short' one ($\|A^{(i)}x\|_2 \leq 1$). Using the standard concentration measure arguments, we will prove that this probability decays exponentially in k.

2.1.1 Linear mappings

In this section, we will introduce linear mappings satisfying properties 1. and 2. Our technique will depend on the concentration bound used to prove the classic Johnson-Lindenstrauss Lemma. In Lemma 2.0.1, we take a random vector and project it onto the first k vectors of the standard basis of $\mathbb{R}^d$. In our settings, we will project the given vector to a random orthonormal basis which gives the same guaranties. The mapping $A^{(i)}$ consists of k consecutive vectors from the random basis of the $\mathbb{R}^d$ space scaled by $\sqrt{\frac{d}{k}}$. The following reduction describes the basic properties of our construction:

Lemma 2.1.1. *For any parameter $\alpha \geq 1$ and $k < d$, there exist d/k linear mappings $A^{(1)}, A^{(2)}, \ldots, A^{(d/k)}$, from $\mathbb{R}^d$ to $\mathbb{R}^k$, such that:*

1. *for each point $x \in \mathbb{R}^d$ such that $\|x\|_2 \leq 1$, there exists $1 \leq i \leq d/k$, which satisfies $\|A^{(i)}x\|_2 \leq 1$,*

[1]For simplicity, let us assume that k divides d, this can be achieved by padding extra dimensions with 0's.

2. *for each point $x \in \mathbb{R}^d$ such that $\|x\|_2 \geq c$, where $c > 1$, for each i: $1 \leq i \leq d/k$, we have*

$$\mathbb{P}\left[\|A^{(i)}x\|_2 \leq \alpha\right] < e^{\frac{k}{2}\left(1-(\frac{\alpha}{c})^2+\log((\frac{\alpha}{c})^2)\right)}$$

Proof. Let $a_1, a_2, \ldots, a_d$ be a random basis of R^d. Each of the $A^{(i)}$ mappings is represented by a $k \times d$ dimensional matrix. We will use $A^{(i)}$ for denoting both the mapping and the corresponding matrix. The jth row of the matrix $A^{(i)}$ equals $A^{(i)}_j = \sqrt{\frac{d}{k}} a_{(i-1)k+j}$. In other words, the rows of $A^{(i)}$ consist of k consecutive vectors from the random basis of the $\mathbb{R}^d$ space scaled by $\sqrt{\frac{d}{k}}$.

To prove the first property, observe that $A = \sum_{i=1}^{d} \langle a_i, x \rangle^2 \leq 1$, since the distance is independent of the basis. Assume on the contrary, that for each i, $\|A^{(i)}_2 x\| > 1$. It follows that $d \geq dA = k \sum_{i=1}^{d} \|A^{(i)}x\|_2^2 > d$. This contradiction ends the proof of the first property.

For any $x \in \mathbb{R}^d$, such that $\|x\|_2 > c$, the probability:

$$\mathbb{P}\left[\|A^{(i)}x\|_2 \leq \alpha\right] = \mathbb{P}\left[\frac{\|A^{(i)}x\|_2^2}{c^2} \leq (\frac{\alpha}{c})^2\right] \leq \mathbb{P}\left[\frac{\|A^{(i)}x\|_2^2}{\|x\|_2^2} \leq (\frac{\alpha}{c})^2\right].$$

Applying Lemma 2.0.1 ends the proof. □

Since $\log x < x - 1 - (x-1)^2/2$ for $x < 1$, the above bound is not trivial for $\alpha < c$. The algorithm works as follows: for each i, we project $\mathbb{R}^d$ to $\mathbb{R}^k$ using A_i and solve the corresponding problem in the smaller space. For each query point, we need to merge the solutions obtained for each sub-problem. This results in reducing the $\mathsf{NN}_{\text{wfn}}(c, d)$ to d/k instances of $\mathsf{NN}_{\text{wfn}}(\alpha, k)$.

Lemma 2.1.2. *For $1 < \alpha < c$ and $k < d$, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to d/k instances of the $\mathsf{NN}_{\text{wfn}}(\alpha, k)$. The expected pre-processing time equals $\mathcal{O}(d^{\omega-1}n + d/k\ \text{preproc}(\alpha, k))$ and the expected query time equals $\mathcal{O}(d^{\omega-1} + d/k\ e^{\frac{k}{2}\left(1-(\frac{\alpha}{c})^2+\log((\frac{\alpha}{c})^2)\right)} n + d/k\ \text{query}(k, \alpha))$.*

Proof. We use the assumption that $k < d$ and $d^{4-\omega} < n$ to simplify the complexities. The pre-processing time consists of:

- $\mathcal{O}(d^3)$: the time of computing a random orthonormal basis of $\mathbb{R}^d$.
- $\mathcal{O}(d^{\omega-1}n)$: the time of changing the basis to $a_1, a_2, \ldots, a_d$.

- $\mathcal{O}(dnk)$: the time of computing $A^{(i)}x$, for all $1 \le i \le d$, and for all n points.
- $\mathcal{O}(d/k \text{ preproc}(\alpha, k))$: the expected pre-processing time of all sub-problems.

The query time consists of:

- $\mathcal{O}(d^{\omega-1})$: the amortized time of changing the basis to $a_1, a_2, \ldots, a_d$.[2]
- $\mathcal{O}(d/k\ e^{\frac{k}{2}\left(1-(\frac{\alpha}{c})^2+\log((\frac{\alpha}{c})^2)\right)} n)$: the expected number of false positives (by Lemma 2.1.1).
- $\mathcal{O}(d/k \text{ query}(k, \alpha))$: the expected query time for all sub-problems.

$\square$

The following corollary simplifies the formulas used in Lemma 2.1.2 and shows that the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to a number of problems of dimension $\log n$ in an efficient way. Namely, setting $k = \left(\frac{2(1-\nu)}{1-(\frac{\alpha}{c})^2+\log((\frac{\alpha}{c})^2)}\right)\log n$ we get:

Corollary 1.3.1. *For any $1 \le \alpha < c$ and $\gamma \log n < d$, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to $\mathcal{O}(d/(\gamma \log n))$ instances of the $\mathsf{NN}_{\text{wfn}}(\alpha, \gamma \log n)$, where $\gamma < \frac{2(1-\nu)}{(\frac{\alpha}{c})^2-1-2\log\frac{\alpha}{c}}$ for a tunable parameter $\nu \in [0, 1)$ and:*

$$\text{query}(c, d) = \mathcal{O}(d^2 + n^\nu + d/(\gamma \log n) \text{ query}(\alpha, \gamma \log n)),$$
$$\text{preproc}(c, d) = \mathcal{O}(d^{\omega-1}n + d/(\gamma \log n) \text{ preproc}(\alpha, \gamma \log n)).$$

If the queries are provided in the batches of the size d, we obtain the algorithm with the query time:

$$\text{query}(c, d) = \mathcal{O}(d^{\omega-1} + n^\nu + d/(\gamma \log n) \text{ query}(\alpha, \gamma \log n)).$$

Using the bound $\log x < x - 1 - (x-1)^2/2$, we obtain the following inequality:

$$\gamma < \left(\frac{2c}{c-\alpha}\right)^2 (1-\nu),$$

which will be often more convenient to use.

[2] We assume that query comes in batches, this is explicitly explained in Corollary 1.3.1.

2.2 Reduction to ℓ_∞–product

In this section, we give another algorithm which reduces the c–near neighbors problem to a number of instances in ℓ_∞–product of l_2 spaces with dimension $\mathcal{O}(\log\log n)$. Lemma 2.0.1 implies that the $\mathsf{NN}_{\mathrm{wfn}}(c, d)$ problem can be reduced to $d/\log(n)$ problems of dimension logarithmic in n. In order to reduce the dimension even more, we will employ L independent families of linear mappings introduced in Section 2.1. In each of these families, there will be at least one mapping, which does not increase the length of the input vector. As a result, there exists a combination of L mappings (each mapping taken from a distinct family) which does not increase the input vector length. Also, for any combination of L mappings, the probability that all the mappings transform a 'long' vectors to a 'short; one is small. The structure of the mappings is presented in Figure 2.1.

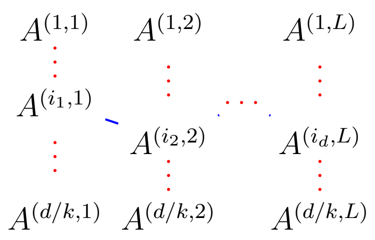

Figure 2.1: Each column describes one family of linear mappings, constructed based on one random, orthonormal basis. The blue path describes one combination of mappings.

To formalize the above line of thinking, we introduce the following lemma:

Lemma 2.2.1. *For any natural number $L > 0$, there exist Ld/k linear mappings $A^{(i,j)} : R^d \to R^k$, where $k < d$, $1 \leq i \leq d/k$ and $1 \leq j \leq L$, such that*

1. *for each point $x \in R^d$ which satisfies $\|x\|_2 \leq 1$, there exist $1 \leq i_1, i_2, \ldots, i_L \leq d$ such that $\|A^{(i_j,j)}x\|_2 \leq 1$, for each $1 \leq j \leq L$.*
2. *for each point $x \in R^d$ which satisfies $\|x\|_2 \geq c$, where $c > 1$, for each $i_1, i_2, \ldots, i_L$: $1 \leq i_1, i_2, \ldots, i_L \leq d/k$, we have*

$$\mathbb{P}\left[\forall_j : \|A^{(i_j,j)}x\|_2 \leq \alpha\right] < \exp\big(-\frac{kL}{4}(\frac{c-\alpha}{c})^2\big).$$

Proof. For each j: $1 \leq j \leq L$ we independently sample the orthonormal basis of $\mathbb{R}^d$: $a_1, a_2, \ldots, a_d$. The $A^{(i,j)}$ will be created in the same way as in Lemma 2.1.1, namely, the t–th row of $A^{(i,j)}$ equals $A_t^{(i,j)} = \sqrt{\frac{d}{k}} a_{(i-1)k+t}$. The properties (1) and (2) follow directly from Lemma 2.1.1.

$\square$

In order to employ Lemma 2.2.1 for a given query q, we need to be able to find all points in X such that a given combination of mappings transforms these points and the query point to 'close' vectors. In other words, we need to find all c-approximate near neighbors for the transformed input set $\tilde{X} \subset (\mathbb{R}^k)^L$ in the space equipped with metric: l_∞–product$(x, y) = \max_{1 \leq i \leq L}(\|x_i - y_i\|)$, which is formally defined as follows:

Definition 2.2.1 (the c–near neighbors search in l_∞–product)**.** *The* $\bigoplus_{l_\infty} l_p\text{\_}NN(c, L, k)$ *is defined as follows: given a query point* $q \in (\mathbb{R}^k)^L$ *and a set* $\tilde{X} \subset (\mathbb{R}^k)^L$ *of* n *input points, find all input points, such that for each* $1 \leq i \leq L$: $\|q_i - \tilde{x}_i\|_2 \leq 1$. *Moreover, each* $\tilde{x}$ *satisfying* $\forall_i \|q_i - \tilde{x}_i\|_2 \leq c$, *might be returned as well. Finally, each* x *such that* $\exists_i \|q_i - \tilde{x}_i\| > c$, *must not be returned.*

Using the construction from Lemma 2.2.1, the $\mathsf{NN}_{\text{wfn}}(c, d)$ problem can be reduced to d^L instances of the $\bigoplus_{l_\infty} l_p\text{\_}NN(\alpha, L, k)$. Each of the instances is represented by indices: $\{i_1, i_2, \ldots, i_L\}$ and the corresponding mappings $A^{(i_j,j)}$ for $1 \leq j \leq L$. Each input point $\tilde{x} \in \tilde{X}$ comes from the point $x \in X$ by applying the mappings: $\tilde{x} = (A^{(i_1,1)}x, \ldots, A^{(i_L,L)}x)$. Similarly, the query point $\tilde{q}$, in l_∞–product, is created from the query point q in $\mathsf{NN}_{\text{wfn}}(c, d)$, as $\tilde{q} = (A^{(i_1,1)}q, \ldots, A^{(i_L,L)}q)$.

Lemma 2.2.2. *The* $\mathsf{NN}_{\text{wfn}}(c, d)$ *can be reduced to* $(d/k)^L$ *instances of* $\bigoplus_{l_\infty} l_p\text{\_}NN(\alpha, L, k)$. *The expected pre-processing time equals:*

$$\mathcal{O}(Ld^2 n + (d/k)^L \ \text{preproc}_{l_\infty\text{-}product}(\alpha, L, k))$$

and the expected query time equals:

$$\mathcal{O}(Ld^2 + (d/k)^L e^{-kL(\frac{c-\alpha}{2c})^2} n + (d/k)^L \ \text{query}_{l_\infty\text{-}product}(\alpha, L, k)).$$

The proof of the Lemma is analogical to the proof of Lemma 2.1.2. The following corollary presents the simplified version of Lemma 2.2.2. Setting $k = \lceil L^{-1} \left(\frac{2c}{c-\alpha}\right)^2 \log n \rceil$ we get:

Corollary 2.2.2.1. *For any $1 \le \alpha < c$, the $\mathsf{NN}_{\mathrm{wfn}}(c,d)$ can be reduced to d^L instances of $\bigoplus_{l_\infty} l_p\text{-}NN(\alpha, L, L^{-1}\gamma\log n)$, where $\gamma = \left(\frac{2c}{c-\alpha}\right)^2$ and:*

$$\mathrm{query}(c,d) = \mathcal{O}(Ld^2 + (d/\log(n))^L\ \mathrm{query}_{l_\infty\text{–}product}(\alpha, L, \lceil L^{-1}\gamma\log n\rceil)),$$

$$\mathrm{preproc}(c,d) = \mathcal{O}(Ld^2n + (d/\log(n))^L\ \mathrm{preproc}_{l_\infty\text{–}product}(\alpha, L, \lceil L^{-1}\gamma\log n\rceil)).$$

In this section, we presented mechanisms, which enable us to decrease the dimension of the initial problem. In Chapters 3 and 4, we show efficient algorithms for the c–near neighbors problem in this reduced space.

Chapter 3

Locality Sensitive Hashing

In this section, we describe how Locality Sensitive Hashing (LSH) with Las Vegas guaranties can be used to solve the c–near neighbors without false negatives problem. In the following Sections 3.3.2–3.3.6, we show the specific families of such functions.

Definition 3.0.1. *The LSH function family without false negatives (or with Las Vegas guaranties) $\mathbb{H}$ is a family of hashing functions $h : \mathbb{R}^d \to \mathbb{Z}$ such that, for given metric $\|\cdot\|$, radius R and constant $c > 1$:*

- *(close items have close hashes) For $x, y \in \mathbb{R}^d$, if $\|x - y\| < R$, then $|h(x) - h(y)| \leq 1$ for each $h \in \mathbb{H}$.*
- *(distant items have distant hashes) For $x, y \in \mathbb{R}^d$, if $\|x - y\| > cR$, then $|h(x) - h(y)| > 1$ with probability p_{fp} independent of x and y.*

At first, we assume that we have the LSH family with certain properties. In Section 3.1 and 3.2, we show algorithms for c–near neighbors without false negatives for l_p and l_∞–product of l_p, respectively. In Section 3.3, we describe various LSH families and their properties. In Section 3.4, we combine the results of this chapter into a concise form.

3.1 Algorithm for l_2

As mentioned before, w.l.o.g we assume that radius $R = 1$. The idea for solving the c-nearest neighbors without false negatives is to hash all input points and for each hash value store all points with this hash value. For a query point, we compute its hash and fetch the corresponding bucket from the storage. We examine all input points which are hashed to the same value as the query point. This bucket will certainly contain all points close to the query point and some number of false positives (distant points with close hashes). In order to decrease the probability of a false positive, we introduce a new hash function g which is the concatenation of k random h functions. Each of the input points is hashed by g and the reference to this point is kept in a single storage data structure.

Namely, each $x \in \mathbb{R}^d$ will be hashed by $g(x) \coloneqq (h_1(x), \ldots, h_k(x))$, where g is a hash function defined as a concatenation of k random LSH functions h. If two points are 'close' in the considered metric, then g transforms these points to hashes $p^{(1)}, p^{(2)} \in \mathbb{Z}^k$, such that $|p_i^{(1)} - p_i^{(2)}| \leq 1$ for all $i \in k$. The pre-processing algorithm is summarized in the following pseudocode:

Algorithm 1: The pre-processing algorithm

```
Data: $X \subset \mathbb{R}^d$ – the set of $n$ input points
Result: $M : \mathbb{Z}^k \to 2^X$ – the map storing for each hash $\alpha \in \mathbb{Z}^k$ the
        subset of input points with hashes close to $\alpha$
$M = \emptyset$;
for $x \in X$ do
    $\alpha = g(x)$;
    for $\alpha'$ such that $\|\alpha - \alpha'\|_\infty \leq 1$ do
        $M(\alpha').push(x)$;
    end
end
```

The query algorithm consists of examining the bucket for $g(query\_point)$:

```
Algorithm 2: The query algorithm
Data: $q \in \mathbb{R}^d$ – the query point
      $M : \mathbb{Z}^k \to 2^X$ – the map storing for each hash $\alpha \in \mathbb{Z}^k$ the
subset of input points with hashes close to $\alpha$
Result: $P \subset X$ – the set of neighbors of $q$
$P = \emptyset$;
for $x \in M(g(q))$ do
    if $x$ is a neighbor of $q$ then
        $P.push(x)$;
    end
end
```

The following theorem gives the properties of the above algorithm:

Theorem 3.1.1. *For any $c > \tau$ and the number of iterations $k > 0$, there exists a c–near neighbors algorithm without false negatives for ℓ_p, where $p \in [1, \infty]$, with:*

- *Preprocessing time: $\mathcal{O}(n(kd + 3^k))$,*
- *Memory usage: $\mathcal{O}(n3^k)$,*
- *Expected query time: $\mathcal{O}(d(|P| + k + np_{fp}^k))$.*

Where $|P|$ is the size of the result and p_{fp} is the upper bound of probability of false positives, where p_{fp} and $\tau = \tau(d)$ depend on a choice of hash functions family and the dimension of the space.

Also, there exists c–near neighbors algorithm without false negatives for ℓ_p(fast pre-processing version):

- *Preprocessing time: $\mathcal{O}(nkd)$,*
- *Memory usage: $\mathcal{O}(n)$,*
- *Expected query time: $\mathcal{O}(d(|P| + np_{fp}^k) + 3^k)$.*

Proof. Since we consider two hashes to be 'close', when they differ by at most one, for each hash $\alpha \in \mathbb{Z}^k$ we need to store the reference to every point, that satisfies $\|\alpha - g(x)\|_\infty \leq 1$. Thus, the hash map size is $\mathcal{O}(n3^k)$. Computing a single h function in $\mathbb{R}^d$ takes $\mathcal{O}(d)$. The pre-processing consists of computing the 3^k hashes for each point in the input set. The query consists of computing the hash of the query point, looking up all the points with colliding hashes, filtering out the false positives and returning the neighbors.

In the fast pre-processing version, we store only the initial n points and during the query phase we iterate through all 3^k close hashes. □

To bound the query time, we need to limit the number of false positives np_{fp}^k. Setting $k = -\log_{p_{fp}} n$ in the "fast query" version of this theorem gives the following corollary:

Theorem 3.1.2. *For any $c > \tau$ and for large enough n, there exists a c–near neighbors without false negatives algorithm for ℓ_p, where $p \in [1, \infty]$ (fast query version), with:*

- *Preprocessing time: $\mathcal{O}(n(d \log n + n^{\gamma(1-\eta)}))$,*
- *Memory usage: $\mathcal{O}(n^{1+\gamma(1-\eta)})$,*
- *Expected query time: $\mathcal{O}(d(|P| + \log(n) + n^{\eta}))$.*

Where $|P|$ is the size of the result, $\gamma = \frac{\ln 3}{-\ln p_{fp}}$ and p_{fp} and τ are chosen as in Theorem 3.1.1 and $0 \leq \eta < 1$ is a tunable parameter.

Proof. The number of iterations k can be chosen arbitrarily, so we will choose the optimal value. Denote $a = -\ln p_{fp}$ and $b = \ln 3$, then set k to be:

$$k = \left\lceil -(1-\eta) \log_{p_{fp}} n \right\rceil .$$

Let us assume that n is large enough so that $k \geq 1$. Then, we have:

$$3^k \leq 3 \cdot (3^{-(1-\eta)\frac{\log n}{\log p_{fp}}}) = 3 \cdot n^{(1-\eta)b/a} = 3 \cdot n^{(1-\eta)b/a} = 3 \cdot n^{(1-\eta)\gamma},$$

$$np_{fp}^k \leq np_{fp}^{-(1-\eta)a} = n^{\eta}.$$

Substituting 3^k and np_{fp}^k values in Theorem 3.1.1 gives the complexity guaranties. □

In the "fast pre-processing" version, we need to set k in such a way that number of false positives dnp_{fp}^k and number of hashes that need to be searched 3^k is balanced.

Theorem 3.1.3. *For any $c > \tau$ and for large enough n, there exists a c–near neighbors algorithm without false negatives for ℓ_p, where $p \in [1, \infty]$ (fast pre-processing version), with:*

- *Preprocessing time:* $\mathcal{O}(dn \log(dn))$,
- *Memory usage:* $\mathcal{O}(n \log(dn))$,
- *Expected query time:* $\mathcal{O}(d|P| + (dn)^{\frac{a}{a+b}})$.

Where $|P|$ is the size of the result, $a = -\ln p_{fp}$ and $b = \ln 3$ and p_{fp} and τ are chosen as in Theorem 3.1.1 and $0 \leq \eta < 1$ is a tunable parameter.

Proof. The number of iterations k can be chosen arbitrarily, so we will choose the optimal value and set k to be:

$$k = \left\lceil \frac{\log(nd)}{a+b} \right\rceil .$$

Let us assume that n is large enough so that $k \geq 1$. We have:

$$3^k \leq 3 \cdot (3^{\frac{\log(dn)}{a+b}}) = 3 \cdot e^{\frac{\log(dn)a}{a+b}} = 3 \cdot (dn)^{\frac{a}{a+b}},$$

$$dnp_{fp}^k \leq dnp_{fp}^{\frac{\log(nd)}{a+b}} = dne^{-\frac{b\log(nd)}{a+b}} = dn(dn)^{-\frac{b}{a+b}} = (dn)^{\frac{a}{a+b}}.$$

Substituting 3^k and dnp_{fp}^k values in Theorem 3.1.1 gives the complexity guaranties. □

3.2 Solving the c-approximate near neighbors in ℓ_∞-products of ℓ_p

In Chapter 2, we showed how to reduce c–near neighbors in $\ell_2$to a number of instances of c–near neighbors in l_∞-products of ℓ_2. The $\bigoplus_{l_\infty} l_p\text{-}NN(\alpha, L, k)$ can be trivially solved by dealing with each of the $\mathsf{NN}_{\mathrm{wfn}}(\alpha, k)$ problems separately. Unfortunately, this gives unacceptable complexities. In order to improve it, we need to be able to solve the $\bigoplus_{l_\infty} l_p\text{-}NN$ more efficiently. We show that c–near neighbors in l_∞-products of ℓ_p can be solved using the locality sensitive hash functions for any $p \in [1, \infty]$.

Again, we assume that we are given an LSH family without false negatives. We consider two hashes to be 'close' if $|h(x) - h(x')| \leq 1$ for a hash function h. Based on h, we introduce a new hash function g. Each of the input points is hashed by g and the reference to this point is kept in a single hash map. For a given query point, we examine all input points which are hashed to the same value as the query point.

Namely, each $\tilde{x} \in (\mathbb{R}^k)^L$ will be hashed by $g(\tilde{x}) \coloneqq (g_1(\tilde{x}_1), \ldots, g_L(\tilde{x}_L))$, where $g_i(x) \coloneqq (h_1(x), h_2(x), \ldots, h_w(x))$ is a hash function defined as a concatenation of w random LSH functions h. The function g can be also seen as a concatenation of wL random hash functions h. If two points are 'close' in the considered l_∞–product metric, then g transforms these points to hashes $p^{(1)}, p^{(2)} \in \mathbb{Z}^{wL}$, such that $|p_i^{(1)} - p_i^{(2)}| \leq 1$ for all $i \in wL$. The pre-processing algorithm is summarized in the following pseudocode:

```
Algorithm 3: The pre-processing algorithm
  Data: $X \subset (\mathbb{R}^k)^L$ – the set of $n$ input points
  Result: $H : \mathbb{Z}^{wL} \to 2^X$ – the map storing for each hash $\alpha \in \mathbb{Z}^{wL}$ the
          subset of input points with hashes close to $\alpha$
  $H = \emptyset$;
  for $x \in X$ do
      $\alpha = g(x)$;
      for $\alpha'$ such that $\|\alpha - \alpha'\|_\infty \leq 1$ do
          $H(\alpha').push(x)$;
      end
  end
```

The query algorithm consists of examining the bucket for $g(query\_point)$:

Algorithm 4: The query algorithm

```
Data: $q \in (\mathbb{R}^k)^L$ – the query point
Result: $P \subset X$ – the set of neighbors of $q$
$P = \emptyset$;
for $x \in H(g(q))$ do
    if $x$ is a neighbor of $q$ then
        $P.push(x)$;
    end
end
```

The following theorem describes the properties the above algorithm:

Theorem 3.2.1. *For $L = o(\log n)$ and $c > \tau(k)$, the $\bigoplus_{l_\infty} l_p\_NN$(c,L,k) can be solved in the $\mathcal{O}(kL|P| + k \ln n)$ query time and $\mathcal{O}(n^{1+\frac{\log 3}{-\log(p_{fp})}+o(1)})$ pre-processing time complexity.*

Proof. Since we consider two hashes to be 'close', when they differ at most by one, for each hash $\alpha \in \mathbb{Z}^{wL}$ we need to store the reference to every point, that satisfies $\|\alpha - g(x)\|_\infty \leq 1$. Thus, the hash map size is $\mathcal{O}(n3^{wL})$. Computing a single h function in $\mathbb{R}^k$ takes $\mathcal{O}(k)$, so evaluating the $g(x)$ for $x \in (\mathbb{R}^k)^L$ takes $O(wkL)$. The pre-processing consists of computing the 3^{wL} hashes for each point in the input set. The query consists of computing the hash of the query point, looking up all the points with colliding hashes, filtering out the false positives and returning the neighbors.

For any $c > \tau(k)$ and the number of iterations $w \geq 1$, there exists a $\bigoplus_{l_\infty} l_p\_NN$(c,L,k) algorithm with the following properties:

- the pre-processing time: $\mathcal{O}(n(wkL + 3^{wL}))$, where wkL is the time needed to compute the $g(input\_point)$ and $\mathcal{O}(3^{wL})$ is the number of the updated hashes for one input point,
- the expected query time: $\mathcal{O}(kL(|P| + w + np_{fp}^{wL}))$, where wkL is the time needed to compute the $g(query\_point)$, np_{fp}^{wL} is the number of false positives which need to be ignored, $|P|$ denotes the size of the result set. For each of the candidates, we need to perform a check of complexity $\mathcal{O}(kL)$ to classify the point as a true positive or a false positive.

Above p_{fp} and τ are as in Theorem 3.1.1. The number of iterations w can be chosen arbitrarily, so we will choose the optimal value. Denote $a = -\ln p_{fp}$ and $b = \ln 3$, then set w to be:

$$w = \left\lceil \frac{\log n}{La} \right\rceil .$$

Let us assume that n is large enough so that $w \geq 1$. We have:

$$3^{wL} \leq 3^L \cdot 3^{\frac{\log n}{a}} = 3 \cdot n^{b/a+o(1)},$$

$$np_{fp}^{wL} = ne^{-awL} \leq ne^{-\log n} = \mathcal{O}(1).$$

Hence, for constant c the expected query time is $\mathcal{O}(kL|P| + k \ln n)$. Subsequently, the pre-processing time is: $\mathcal{O}(n3^{wL}) = \mathcal{O}(n^{1+b/a+o(1)})$. Substituting a, b and p_{fp} values gives the complexity guaranties. □

3.3 Hash Functions

We will consider hash functions of form

$$h_p(x) = \lfloor \langle x, v \rangle \rfloor ,$$

where $v\mathbb{R}^d$ is drawn from some distribution. In order to argue that properties of the hash functions, we need will use following two observations:

Observation 3.3.1.

$$\mathbb{P}\left[|h_p(x) - h_p(y)| \leq 1\right] \leq \mathbb{P}\left[| \langle x - y, v \rangle | < 2\right].$$

Observation 3.3.2.

$$\mathbb{P}\left[|h_p(x) - h_p(y)| \leq 1\right] \geq \mathbb{P}\left[| \langle x - y, v \rangle | < 1\right].$$

In other words, $|h_p(x) - h_p(y)$ is "small" iff $| \langle x - y, v \rangle |$ is also "small". In order to prove that Las Vegas property holds, i.e., to ensure that "close" points are hashed to "close" values, the distribution of v must be bounded. In this section, we consider different families of bounded functions. We start with general bounds for independent variables in Section 3.3.1. We show slightly better bounds for Rademacher's variables in Section 3.3.2. Then, we consider bounded variables with log–concave distributions in Section 3.3.3. We show that log–concaveness is crucial to improve the performance of hashing functions. In Section 3.3.4, we consider particular log–concave distribution

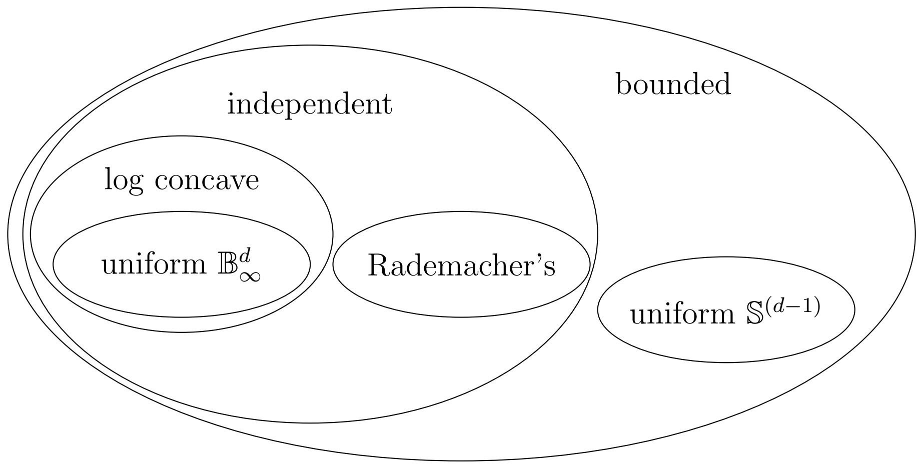

Figure 3.1: The structure of hash functions families.

$U(-1,1)$, and show bounds for it. Since vector v with $v_i \sim U(-1,1)$, for $i \in [d]$, can be interpreted as a random point from the unit ball in l_∞, we call this LSH family a uniform $\mathbb{B}^d_\infty$ family. This family gives the best results for $p \in [1,2)$. In Section 3.3.6, we consider hash functions in which v is drawn uniformly from $\mathbb{S}^{(d-1)}_p$. Thus, the vector v is dependent. This family gives the best results for $p \in [2,\infty]$. All of the families are presented in Figure 3.1.

The bounds for the probability of false positives are summarized in Table 3.1. We introduce constants $\kappa = \kappa_{c,p} = \frac{2d^{\max\{1-1/p,1/2\}}}{c}$ and $\tilde{\kappa} = \tilde{\kappa}_{c,p} = \frac{2d^{1/2+|1/2-p|}}{c}$. The bounds for the presented LSH families will work for κ (or $\tilde{\kappa}$ of dependent hashing family) bounded by some universal constant. The bounds become better when κ (or $\tilde{\kappa}$) is closer to 0. In the uniform $\mathbb{S}^{(d-1)}$ case, for $p \geq 2$, we have $\tilde{\kappa} = \kappa$. Let us remind that since p_{fp} is present in the exponent of the nearest neighbor algorithm, it is important to optimize the constant.

3.3.1 Bounded independent hash functions

In this section, we show the LSH family defined as:

$$h_p(x) = \left\lfloor \frac{\langle x, v\rangle}{\rho_p} \right\rfloor,$$

where $\rho_p = d^{1-\frac{1}{p}}$ and v is a vector of $i.i.d$ from some symmetric probability distribution such that $|v_i| \leq 1$. These assumptions are enough to prove the

family	p_{fp}
bounded independent (h_p)	$\frac{2}{3} + \frac{2\kappa^2}{3H^2}$
Rademacher's (h_p)	$\frac{1}{2} + \sqrt{2}\kappa$
log–concave ($\hat{h}_p$)	$\frac{2\kappa}{sd(W)}$
uniform $\mathbb{B}^d_\infty$ ($\hat{h}_p$)	$\sqrt{2}\kappa$
uniform $\mathbb{S}^{(d-1)}$ ($\tilde{h}_p$)	$\tilde{\kappa}$

Table 3.1: Comparison of the results for probability of false positives for different bounded LSH families. The bound for independent variables is for $\mathbb{E}(v_i) = 0$, $\mathbb{E}(v_i^2) = H^2$, $\mathbb{E}(v_i^4) \leq 3H^4$ (for any $H > 0$). The bound for the log–concave distribution is for log–concave distribution W and $sd(W)$ is the standard deviation of W.

reasonable bounds for false positives.

Observation 3.3.3 (Close points have close hashes)**.** *For any random bounded vector $v \in \mathbb{R}^d$ of independent random variables such that: $|v_i| \leq 1$ for $i = 1, \ldots, d$ and for $x, y \in \mathbb{R}^d$, if $\|x-y\|_p < 1$ then $\forall_{h_p} |h_p(x) - h_p(y)| \leq 1$.*

Proof. We know that $\|z\|_1 \leq \rho_p \|z\|_p$, consequently:

$$\rho_p \|x-y\|_p \geq \|x-y\|_1 = \sum_i |x_i - y_i| \geq \sum_i |v_i(x_i - y_i)| = |\langle x-y, v\rangle|.$$

Now, when points are close in ℓ_p:

$$\|x-y\|_p < 1 \iff \rho_p\|x-y\|_p < \rho_p \implies |\langle x-y, v\rangle| < \rho_p.$$

Next, by Observation 3.3.2:

$$1 = \mathbb{P}\left[|\langle x-y, v\rangle| < \rho_p\right] \leq \mathbb{P}\left[|h_p(x) - h_p(y)| \leq 1\right].$$

This ends the proof.

□

This observation holds for any bounded distribution. In order to analyze the properties of the hash functions, we need the following technical observations:

Observation 3.3.4. *For any $z \in \mathbb{R}^d$ where, $\delta_q = d^{\min\{1/2-1/q,0\}}$ and $1/p + 1/q = 1$:*

$$\|z\|_p \delta_p \leq \|z\|_2 \leq \|z\|_p \delta_q^{-1}.$$

This observation is a direct consequence of the inequality between means. The proofs for false positive bounds for independent variables are based on so-called anti-concentration bounds, which are formalized in the following lemma:

Lemma 3.3.5. *Assume, that we are given the hashing function of the form:*

$$h_p(x) = \left\lfloor \frac{\langle v, x \rangle}{d^{1-1/p}} \right\rfloor.$$

If v satisfies the following anti-concentration bound:

$$\mathbb{P}\left[\langle v, x \rangle \leq \alpha\right] \leq \beta_\alpha \text{ for } \alpha \text{ satisfying condition } \Phi(\alpha) \text{ for } \|x\|_2 = 1,$$

and $\|x - y\|_p > c$, $x, y \in \mathbb{R}^d$ then:

$$p_{fp} = \mathbb{P}\left[|h_p(x) - h_p(y)| \leq 1\right] < \beta_{\kappa_{c,p}} \text{ for } \Phi(\kappa_{c,p}),$$

i.e., the probability of false positive in the c–near neighbors equals $\beta_{\kappa_{c,p}}$ for $\Phi(\kappa_{c,p})$.

Proof. By Observation 3.3.1, we have.

$$\mathbb{P}\left[|h_p(x) - h_p(y)| > 1\right] \geq \mathbb{P}\left[|\langle v, z \rangle| > 2\rho_p\right] = \mathbb{P}\left[|\frac{\langle v, z \rangle}{\|z\|_2}| > \frac{2\rho_p}{\|z\|_2}\right],$$

where $z = x - y$. By Observation 3.3.4,

$$\mathbb{P}\left[|h_p(x) - h_p(y)| > 1\right] \geq \mathbb{P}\left[|\frac{\langle v, z \rangle}{\|z\|_2}| > \frac{2d^{\max\{\frac{1}{2}, 1-1/p\}}}{\|z\|_p}\right]$$

$$\geq \mathbb{P}\left[|\frac{\langle v, z \rangle}{\|z\|_2}| > \frac{2d^{\max\{\frac{1}{2}, 1-1/p\}}}{c}\right] = \mathbb{P}\left[|\frac{\langle v, z \rangle}{\|z\|_2}| > \kappa_{c,p}\right].$$

Applying anti–concentration bound finishes the proof. □

Now, we proceed to prove the anti–concatenation bound for false positives for general symmetric and bounded distribution:

Lemma 3.3.6 (Anti–concentration bound for general distribution)**.** *For any random bounded vector $v \in \mathbb{R}^d$ of independent, symmetric random variables such that $\mathbb{E}(v_i) = 0$, $\mathbb{E}(v_i^2) = H^2$, $\mathbb{E}(v_i^4) \le 3H^4$ (for any $H > 0$). If $\alpha < H$ and $\|x\|_2 = 1$, then*

$$\mathbb{P}\left[|\langle v, x\rangle| \le \alpha\right] \le 1 - \frac{(1 - \frac{\alpha^2}{H^2})^2}{3},$$

for every metric ℓ_p, where $p \in [1, \infty]$.

Proof. Let $X = \|v, x\|$, we have:

$$\mathbb{P}\left[|X| \le \alpha\right] \le 1 - \mathbb{P}\left[X^2 > \frac{\alpha^2}{H^2} H^2\right],$$

Using the assumption $\alpha < H$, by Paley-Zygmunt inequality (analogously to [29]) for variable X^2 and $\theta = \frac{\alpha^2}{H^2} < 1$:

$$\mathbb{P}\left[|X| \le \alpha\right] \le 1 - (1 - \frac{\alpha^2}{H^2})^2 \frac{H^4}{\mathbb{E}X^4},$$

Using the assumption on the forth moment ends the proof. □

Now, we proceed to prove the bound for the false positive for general symmetric bounded distribution. By Lemma 3.3.5 and Lemma 3.3.6, we have:

Lemma 3.3.7 (The probability of false positives for general distribution)**.** *For any random bounded vector $v \in \mathbb{R}^d$ such as in Lemma 3.3.6. When $\|x - y\|_p > c$, $x, y \in \mathbb{R}^d$ and $\kappa_{c,p} < H$ then:*

$$p_{fp_1} = \mathbb{P}\left[|h_p(x) - h_p(y)| \le 1\right] < 1 - \frac{\left(1 - (\frac{\kappa_{c,p}}{H})^2\right)^2}{3},$$

for every metric ℓ_p, where $p \in [1, \infty]$ (p_{fp_1} is the probability of false positive).

3.3.2 Rademacher hash functions

In this section, we consider specific hash functions for Rademacher's distributions, i.e., $\mathbb{P}\left[v_i = \pm 1\right] = \frac{1}{2}$. This is the optimal choice of distribution in Lemma 3.3.7 for bounded distribution $|v_i| \le 1$. The optimality follows from the fact that Rademacher's variable has the largest variance for bounded distributions. In this section, we show better false positives bound for these functions. We start with the anti–concentration bound:

Lemma 3.3.8. *For any random vector $v \in \mathbb{R}^d$ of independent random variables such that $\mathbb{P}[v_i = \pm 1] = \frac{1}{2}$, for every $p \in [1, \infty]$ and $\|x\|_2 = 1$ and $\kappa_{c,p} < \frac{1}{\sqrt{2}}$, it holds:*

$$\mathbb{P}[| \langle v, x\rangle | \le \alpha] \le 1 - \frac{(1-\sqrt{2}\alpha)^2}{2}.$$

Proof. Let $X = \langle v, x\rangle$. The *Khintchine* inequality [30] states $\mathbb{E}|X| \ge \frac{\|x\|_2}{\sqrt{2}} = \frac{1}{\sqrt{2}}$, so:

$$\mathbb{P}[|X| \le \alpha] \le 1 - \mathbb{P}\left[|X| > \frac{\alpha}{\mathbb{E}|X|}\mathbb{E}|X|\right],$$

Since $\alpha < \frac{1}{\sqrt{2}}$, by Paley-Zygmunt inequality for variable $|X|$ and $\theta = \frac{\alpha}{\mathbb{E}|X|} < 1$:

$$\mathbb{P}[|X| \le \alpha] \le 1 - (1 - \frac{\alpha}{\mathbb{E}|X|})^2 \frac{(\mathbb{E}|X|)^2}{\mathbb{E}(X^2)},$$

By the Khintchine inequality and the fact that $\mathbb{E}X^2 = 1$, the the statement of the theorem follows. □

By Lemma 3.3.5 and Lemma 3.3.8, we have:

Lemma 3.3.9. *For any random vector $v \in \mathbb{R}^d$ of independent random variables such that $\mathbb{P}[v_i = \pm 1] = \frac{1}{2}$, for every $p \in [1, \infty]$, $x, y \in \mathbb{R}^d$ and $\kappa_{c,p} < \frac{1}{\sqrt{2}}$ such that $\|x - y\|_p > c$, it holds:*

$$p_{fp_2} = \mathbb{P}[|h_p(x) - h_p(y)| \le 1] < 1 - \frac{(1-\sqrt{2}\kappa_{c,p})^2}{2}.$$

3.3.3 Bounded log–concave hash functions

In Section 3.3.2, we have proved bounds for hash functions defined as: $h_p(x) = \lfloor d^{1/p-1}\langle x, v\rangle \rfloor$, where $v \in \{-1, 1\}^d$ is a random vector satisfying: $\mathbb{P}[v_i = 1] = 1/2$. In this section, we will introduce new hash functions $\hat{h}_p$, which improve over h_p for $p \in [1, \infty]$. Particularly, the probability of false positives is decreased, which leads to better complexities of the c–near neighbors algorithm for $c = \Theta(d^{\max\{1/2, 1-1/p\}})$.

Given a vector $x \in \mathbb{R}^d$ such that $\|x\|_p > c$, the probability of a false positives for h_p can be bounded as follows:

$$p_{fp} = \mathbb{P}[|h_p(x) - h_p(y)| \le 1] < 1 - \frac{(1 - \frac{\sqrt{8d}}{c})^2}{2}.$$

Even for very large c, p_{fp} is always greater than 1/2. This must be the case, since for an arbitrarily large vector $x = (C, C, 0, 0, \ldots, 0)$, the probability that this vector will be mapped to 0 equals 1/2. To overcome this obstacle, we introduce new hash function:

$$\hat{h}_p(x) = \left\lfloor d^{1/p-1} \langle w, x\rangle \right\rfloor,$$

where w is a vector of independent random variables with log–concave distribution.

In particular, the optimal choice o w_i is $w_i \sim U(-1, 1)$. In this section, we prove general bounds for log–concave distribution. In the next section, we prove better bounds for $w_i \sim U(-1, 1)$.

To bound the probability of false positives, we need to be able to bound the probability of $\mathbb{P}[|\langle w, x\rangle| < \alpha]$. The anti-concentration bounds can be proved using general results of [26], because they are strongly connected with the Littlewood-Offord theory. Consider Lévy concentration function:

$$Q(X, \lambda) = \sup_x \mathbb{P}[X \le x \le X + \lambda].$$

We have $\mathbb{P}[|X| \le \alpha] \le Q(X, 2\alpha)$. Actually, if X is log–concave and symmetric, then this inequality turns into equality. So any bound on the Lévy concentration function is also a bound for our problem. Bobkov et al. [26] considered bounds on the Lévy concentration function for X being the sum of independent random variables with the log–concave density function. Particularly (Theorem 1.1 in [26]):

Theorem 3.3.10. *If $X_1, \ldots, X_k$ are independent random variables with log–concave distribution, set $S = \sum_i X_k$. Then for all $\lambda \ge 0$*

$$Q(S, \lambda) \le \frac{\lambda}{\sqrt{Var(S) + \frac{\lambda^2}{12}}}.$$

The above theorem is used to prove the anti-concentration bounds for the log–concave bounded distributions:

Lemma 3.3.11 (Anti-concentration bound for any log–concave distributions)**.** *Let $x \in \mathbb{S}^{(d-1)}$ be a given fixed unit vector and $w \in \mathbb{R}^d$ be a vector of independent random variables with log–concave distribution W, then*

$$\mathbb{P}[|\langle w, x\rangle| < \alpha] \le \frac{2\alpha}{sd(W)},$$

where $sd(W)$ is a standard deviation of W.

Proof. To proof this observation, we apply the bounds for the Lévy concentration function for log–concave distributions presented in [26]. Let $X_i = w_i x_i$ and $S = \sum_i X_k$. We have

$$\mathbb{P}\left[|\langle w, x\rangle| < \alpha\right] = \mathbb{P}\left[|S| < \alpha\right] \leq Q(S, 2\alpha).$$

By applying Theorem 3.3.10 we get:

$$\mathbb{P}\left[|\langle w, x\rangle| \leq \alpha\right] \leq \frac{2\alpha}{\sqrt{Var(S) + \frac{\alpha^2}{3}}} \leq \frac{2\alpha}{\sqrt{Var(S)}}.$$

Since $Var(X_i) = x_i^2 Var(W)$ and $Var(S) = \|x\|_2^2 Var(W) = Var(W)$, we have:

$$\mathbb{P}\left[|\langle w, x\rangle| \leq \alpha\right] \leq \frac{2\alpha}{sd(W)}.$$

$\square$

If we assume that W is bounded: $|W| \leq 1$, the best bound is achieved for uniform distribution. In this case ($W = U(-1, 1)$) we have $sd(W) = \sqrt{1/3}$ and $\mathbb{P}\left[|\langle w, x\rangle| \leq \alpha\right] \leq 2\sqrt{3}\alpha$. The above bound for the uniform distribution is not tight. For this special case, the better bound is introduced in Section 3.3.4. By Lemma 3.3.5 we get:

Lemma 3.3.12. *Let $x \in \mathbb{S}^{(d-1)}$ be a given fixed unit vector and $w \in \mathbb{R}^d$ be as in Lemma 3.3.11, then for every $p \in [1, \infty]$, $x, y \in \mathbb{R}^d$ such that $\|x - y\|_p > c$, it holds:*

$$p_{fp} = \mathbb{P}\left[|\hat{h}_p(x) - \hat{h}_p(y)| \leq 1\right] < \frac{2\kappa_{c,p}}{sd(W)}.$$

3.3.4 Uniform l_∞ ball hash functions

In this section, we prove results for special class of log–concave functions, i.e., the $W \sim U(-1, 1)$. Let $M(X) = \sup_X p_X(x)$[1], where p_X is the density function of X. A well-known result of Ball [27] will allow us to prove better result for uniform distribution (see also Proposition 3.1 in [26]):

[1] $M(X)$ is usually defined using more general essential supremum $M(X) = \operatorname{ess\,sup}_X p_X(x)$ but this is not necessary in our case.

Theorem 3.3.13. *If $X_1, \dots, X_k$ are independent random variables with uniform distribution on $[-1, 1]$ and let $\lambda \in \mathbb{S}^{(d-1)}$ be some fixed unit vector in ℓ_2^d and set $S = \sum_i \lambda_i X_i$, then $M(S) \leq \frac{1}{\sqrt{2}}$.*

Now we can prove the anti-concentration bound for uniform distributions.

Observation 3.3.14 (Anti-concentration bound for uniform distributions)**.** *Let $x \in \mathbb{S}^{(d-1)}$ be a given fixed unit vector and $w \in \mathbb{R}^d$ be a vector of independent random variables with $U(-1, 1)$ distribution, then*

$$\mathbb{P}\left[| \langle w, x \rangle | < \alpha\right] \leq \sqrt{2}\alpha,$$

Proof. In order to proof this observation, we apply the bounds for cube slicing of Ball [27]. Let $X_i = w_i x_i$ and $S = \sum_i X_k$. We have

$$\mathbb{P}\left[| \langle w, x \rangle | < \alpha\right] = \int_{-\alpha}^{\alpha} p_S(\zeta) d\zeta,$$

where p_S is density function of S. Consequently, we have:

$$\mathbb{P}\left[| \langle w, x \rangle | < \alpha\right] \leq 2\alpha M(S) \leq \sqrt{2}\alpha.$$

The last inequality follows from Theorem 3.3.13.

□

If we assume that variables in w are *i.i.d.* and bounded, then $\langle w, x \rangle$ satisfies assumptions of the Hoefding inequality [31]. This implies that $\langle w, x \rangle$ is highly concentrated in the interval $(-|x|_2, |x|_2)S$, where S is the standard deviation of w_i. Consequently, if w is a vector of variables with *i.i.d.* then $\hat{h}_p$ is optimal, up to a constant factor. As the inequality in Ball's cube slicing is tight, the inequality in Observation 3.3.14 is also tight, for small α. The tightness of this bound is further discussed in Section 3.3.5. To improve the constant in inequality, we need to change used hash functions. For $p \geq 2$, such hash functions are introduced in Section 3.3.6.

Using Lemma 3.3.5 and Lemma 3.3.14, we show the bound for the probability of false positives for $\hat{h}_p$:

Lemma 3.3.15 (Probability of false positives for $\hat{h}_p$)**.** *For every $p \in [1, \infty]$, $x, y \in \mathbb{R}^d$ and $\kappa_{c,p} \leq \frac{1}{\sqrt{2}}$ such that $\|x - y\|_p > c$, the following holds:*

$$\hat{\mathsf{p}}_{\mathsf{fp}} = \mathbb{P}\left[|\hat{h}_p(x) - \hat{h}_p(y)| \leq 1\right] < \sqrt{2}\kappa_{c,p}.$$

3.3.5 Tightness of bounds

In this section, we study the tightness of inequalities for bounded independent hash functions. We showed that for two distant points $x, y : \|x - y\|_p > cr$, the probability of a collision is "small" when $c = \Theta(d^{\max\{1-1/p,1/2\}})$. The natural question arises: Can we bound the probability of a collision for points $\|x - y\|_p > c'r$ for some c' strictly smaller than c? More formally, is there $c' = O(cd^{-\lambda})$ for some constant λ?

We will show that such c' does not exist, i.e., there always exists $\tilde{x}$ such that $\|\tilde{x}\|_p$ is arbitrarily close to cr, and both $\tilde{x}$ and $\vec{0}$ will end up in the same or adjacent hash with high probability. More formally, let $p \in [1, \infty]$ and $h_p(x) = \left\lfloor \frac{\langle x,v \rangle}{\rho_p} \right\rfloor$, where coordinates of d-dimensional vector v are random variables v_i, such that $|v_i| \leq 1$ with $\mathbb{E}(v_i) = 0$. We will show that there always exists $\tilde{x}$ such that $\|\tilde{x}\|_p \approx d^{\max\{1-1/p,1/2\}}$ and $|h_p(\tilde{x}) - h_p(\vec{0})| \leq 1$ with high probability.

For $p \geq 2$ denote $x_0 = (\rho_p - \epsilon, 0, 0, \ldots, 0)$. We have $\|x_0 - \vec{0}\|_p = \rho_p - \epsilon$ and:

$$|h_p(x_0) - h_p(\vec{0})| = \left| \left\lfloor \frac{\rho_p - \epsilon}{\rho_p} \cdot v_1 \right\rfloor - 0 \right| \leq 1.$$

For $p \in [1, 2)$, denote $x_1 = d^{-\frac{1}{p}+\frac{1}{2}-\epsilon}\vec{1}$. We have $\|x_1\|_p = d^{\frac{1}{2}-\epsilon}$ and by applying Observation 3.3.2 for complementary probabilities:

$$\begin{aligned} \mathbb{P}\left[|h_p(x_1) - h_p(\vec{0})| > 1\right] &\leq \mathbb{P}\left[|\langle x_1, v\rangle| \geq \rho_p\right] = \mathbb{P}\left[\left|\left\langle \vec{1}, v \right\rangle\right| \geq d^{\frac{1}{2}+\epsilon}\right] \\ &= \mathbb{P}\left[\left|\frac{\sum_{i=1}^d v_i}{d}\right| \geq d^{-\frac{1}{2}+\epsilon}\right] \leq 2 \cdot \exp\left(\frac{-d^{2\epsilon}}{2}\right). \end{aligned}$$

The last inequality follows from the well–known Hoeffding [31] inequality which we state below. Let $X_1, \ldots, X_d$ be bounded independent random variables: $a_i \leq X_i \leq b_i$ and $\overline{X}$ be the mean of these variables $\overline{X} = \sum_{i=1}^d X_i/d$. Theorem 2 of Hoeffding [31] states:

$$\mathbb{P}\left[|\overline{X} - \mathbb{E}\left[\overline{X}\right]| \geq t\right] \leq 2 \cdot \exp\left(-\frac{2d^2t^2}{\sum_{i=1}^d (b_i - a_i)^2}\right).$$

In our case, $v_1, \ldots, v_d$ are bounded by $a_i = -1 \leq v_i \leq 1 = b_i$ with $\mathbb{E}v_i = 0$. Hoeffding inequality implies:

$$\mathbb{P}\left[\left|\frac{\sum_{i=1}^{d} v_i}{d}\right| \geq t\right] \leq 2 \cdot \exp\left(-\frac{2d^2t^2}{\sum_{i=1}^{d}(b_i - a_i)^2}\right) = 2 \cdot \exp\left(-\frac{dt^2}{2}\right).$$

Setting $t = d^{-1/2+\epsilon}$ we obtain the claim:

$$\mathbb{P}\left[\left|\frac{\sum_{i=1}^{d} v_i}{d}\right| \geq d^{-1/2+\epsilon}\right] \leq 2 \cdot \exp\left(-\frac{d^{2\epsilon}}{2}\right).$$

The aforementioned probability for $p \in [1, 2)$ is bounded by an expression exponential in $d^{2\epsilon}$. Even if we would concatenate k random hash functions (see proof of Theorem 3.1.1 for more details), the chance of a collision would be at least $(1 - 2e^{\frac{-d^{2\epsilon}}{2}})^k$. To bound this probability by a constant, the number k needs to be at least $\Theta(e^{\frac{d^{2\epsilon}}{2}})$, which is exponential for any constant ϵ. The probability bounds do not work for ϵ arbitrary close to 0: we proved that introduced hash functions for $c = d^{1/2-\epsilon}$ do not work (may give false positives). One may try to obtain tighter bound, e.g., $c = d^{1/2}/\log(d)$ or show that for every $\epsilon > 0$, the approximation factor $c = d^{1/2-\epsilon}$ does not work.

Hence, to obtain a significantly better approximation factor c, one must introduce a completely new family of hash functions.

3.3.6 Uniform $\mathbb{S}^{(d-1)}$ hash functions

In this section, we introduce a new LSH function family: $\tilde{h}_p$ which is tuned up for $p \geq 2$. Although the improvement is in the constant present in anti-concentration bound, this is important since this constant is in the exponent of complexities for the *c-approximate nearest neighbor without false negatives*. We define $\tilde{h}_p$ as follows:

$$\tilde{h}_p(x) = \lfloor \delta_q \langle w, x \rangle \rfloor, \text{ where } w \text{ is a random vector from the unit sphere } \mathbb{S}^{(d-1)}.$$

Let us remind that $\delta_q = d^{\min\{1/p-1/2,0\}}$. In order to bound the probability of false positive, we need to be able to bound the probability of $\mathbb{P}[|\langle w, x \rangle| < \alpha]$. We cannot use the techniques introduced in previous sections, because random variables in w are not independent. Instead, the probability can be elegantly expressed in geometrical terms. The term $\langle w, x\rangle$ can be seen as the first coefficient of a random point from $\mathbb{S}^{(d-1)}$. The probability of the complementary event is proportional to the area of two spherical caps of distance α from the origin of $\mathbb{S}^{(d-1)}$. The fraction between the area of these spherical caps and the area of the unit ball can be expressed as $I_{\alpha^2}(1/2, (d-1)/2)$ for $|x|_2 = 1$, where $I_x(a, b)$ is a regularized incomplete beta function [32]. Bounding the incomplete beta function gives the following observation:

Observation 3.3.16 (The anti-concentration bound for $\mathbb{S}^{(d-1)}$)**.** *Let $x \in \mathbb{S}^{(d-1)}$ be a given unit vector and $w \in \mathbb{S}^{(d-1)}$ be a random unit vector, then*

$$\mathbb{P}\left[|\langle w, x\rangle| < \alpha\right] \le \alpha\sqrt{d}.$$

Proof. As stated before, the complement of the above probability equals the area of two spherical caps of the normalized $(d-1)$-dimensional sphere (i.e. the area of the sphere equals 1). For a spherical cap, let $0 \le \phi \le \pi/2$ denote a colatitude angle, i.e. the largest angle between e_1 and a vector from the spherical cap. As stated in [32], the area of the spherical cap is given by $1/2 I_{\sin^2\phi}((d-1)/2, 1/2)$. Substituting $\alpha = \cos\phi$, we have:

$$\begin{aligned} f(\alpha) = \mathbb{P}\left[|\langle w, x\rangle| < \alpha\right] &= I_{\sin^2\phi}((d-1)/2, 1/2) \\ &= I_{1-\alpha^2}((d-1)/2, 1/2) = I_{\alpha^2}(1/2, (d-1)/2), \end{aligned}$$

where the last equality follows from the fact that $I_x(a, b) = I_{1-x}(b, a)$. By the definition of $I_x(a, b)$, we have:

$$f'(\alpha) = \frac{2\alpha\alpha^{-1}(1-\alpha^2)^{\frac{d-1}{2}}}{B(1/2, (d-1)/2)} = \frac{2(1-\alpha^2)^{\frac{d-1}{2}}}{B(1/2, (d-1)/2)}$$

and

$$f''(\alpha) = \frac{-2\alpha(d-3)(1-\alpha^2)^{\frac{d-5}{2}}}{B(1/2, (d-1)/2)},$$

where $B(a, b)$ is a beta function. For $d = 2$ the function f is convex, so

$$f(\alpha) \le (1-\alpha)f(0) + \alpha f(1) = \alpha.$$

For $d > 2$, the function is concave and:

$$f(\alpha) \le f(0) + \alpha f'(0) = \frac{2\alpha}{B(1/2, (d-1)/2)}.$$

The last step is proving, that $B(1/2,(d-1)/2) \geq \frac{2}{\sqrt{d}}$. Grenié et al. [33] proved that:

$$B(x,y) \geq \frac{x^{x-1}y^{y-1}}{(x+y)^{x+y-1}}.$$

Applying this inequality gives the following bound:

$$B(1/2,(d-1)/2) \geq \frac{(1/2)^{-1/2}(\frac{d-1}{2})^{\frac{d-3}{2}}}{(\frac{d}{2})^{\frac{d-2}{2}}} = \frac{(1/2)^{-1/2}(\frac{d-1}{d})^{\frac{d-3}{2}}}{(\frac{d}{2})^{\frac{1/2}{2}}} = \frac{2(\frac{d-1}{d})^{\frac{d-3}{2}}}{\sqrt{d}},$$

which ends the proof, since $g(d) = (\frac{d-1}{d})^{\frac{d-3}{2}}$ is decreasing for $d \geq 3$ and $g(3) = 1$.

□

For large d, $g(d) \approx e^{-1/2}$, what gives a slightly better bound. Given the above anti-concentration bound we prove the crucial properties of $\tilde{h}_p$:

Observation 3.3.17 (Close points have close hashes for $\tilde{h}_p$)**.** *For $x, y \in \mathbb{R}^d$, if $\|x-y\|_p < 1$ then $\forall_{\tilde{h}_p} |\tilde{h}_p(x) - \tilde{h}_p(y)| \leq 1$.*

Proof. We have:

$$\mathbb{P}\left[|\tilde{h}_p(x) - \tilde{h}_p(y)| \leq 1\right] \geq \mathbb{P}\left[|\langle x-y, w\rangle|\delta_q \leq 1\right].$$

Applying, in turn, the Schwarz inequality and Observation 3.3.4 we get:

$$\delta_q|\langle x-y,w\rangle| \leq \delta_q\|x-y\|_2 \leq \|x-y\|_p \leq 1.$$

Hence, the points will inevitably hash into the same or adjacent buckets. □

Lemma 3.3.18 (Probability of false positives for $\tilde{h}_p$)**.** *For every $p \in [1,\infty]$, $x, y \in \mathbb{R}^d$ and $\tilde{\kappa}_{c,p} < 1$ such that $\|x-y\|_p > c$, it holds:*

$$\tilde{\mathsf{p}}_{\mathsf{fp}} = \mathbb{P}\left[|\tilde{h}_p(x) - \tilde{h}_p(y)| \leq 1\right] < \tilde{\kappa}_{c,p}.$$

Proof. Let $z = x - y$ and $X = \|z\|_2^{-1}\langle w, z\rangle$, be a random variable. By Observation 3.3.1:

$$\tilde{\mathsf{p}}_{\mathsf{fp}} \leq \mathbb{P}\left[|X|\|z\|_2 \leq 2\delta_q^{-1}\right] \leq \mathbb{P}\left[|X| \leq 2(\|z\|_p\delta_q\delta_p)^{-1}\right].$$

The second inequality follows from the Observation 3.3.4. Since $\delta_q\delta_p = d^{-|1/2-1/p|}$, we have:

$$\tilde{\mathsf{p}}_{\mathsf{fp}} \leq \mathbb{P}\left[|X| \leq 2\|z\|_p^{-1}d^{|1/2-1/p|}\right] \leq \mathbb{P}\left[|X| \leq 2c^{-1}d^{|1/2-1/p|}\right].$$

Applying the anti-concentration bound ends the proof. □

For $p \in [2, \infty]$ we have asymptotically the same constraints on c ($c = \mathcal{O}(d^{1-1/p})$). In addition, for any $p \in [2, \infty]$ we have $\tilde{\mathsf{p}}_{\mathsf{fp}} < \hat{\mathsf{p}}_{\mathsf{fp}}$. Although the improvement in the bound for p_{fp} is only in constant, this might be important for practical cases, because this constant is present in the exponent of the complexities of the c–near neighbors algorithm. For $p \in [1, 2)$ there are discrepancies between the constraints on c, depending on the hash functions used. Particularly, the hash functions h_p and $\hat{h}_p$ work for any $c = \Omega(\sqrt{d})$ for $p \in [1, 2)$, while the $\tilde{h}_p$ works for $c = \Omega(d^{1/p})$.

A natural approach for optimizing both the probability of false positives and the constraint on c would be to consider hash functions of the form $\check{h}_p = \lfloor \langle w, x \rangle \rfloor$, where w is a random point from $\mathbb{S}_q^{(d-1)}$ for $1/q + 1/p = 1$.[2] The Hölder inequality implies that 'close' points are hashed to adjacent buckets. In order to prove the bounds for false positives, we need to bound $\mathbb{P}[|\langle w, x \rangle| < \epsilon]$. We conjecture that this probability can be bounded by $\mathcal{O}(\epsilon\sqrt{d})$ for any $p \in [1, 2]$. This is true for $p = 2$, since $\check{h}_2 = \tilde{h}_2$. Also for large d, $\check{h}_1 \approx \hat{h}_1$, because these two functions differ only by the factor of $\max_i |u_i|$, where $u_i \sim U(-1, 1)$. This factor will be close to 1 for large d. Still, techniques used to prove bounds for $\tilde{h}_p$ and $\hat{h}_p$ seem to be insufficient to prove more general bounds for $\check{h}_p$.

3.4 Main results

In this section, we present the main results of this chapter. We combine the schema for solving the c–near neighbors using the LSH and the bounds obtained for particular LSH functions. For ℓ_2, we apply the dimension reduction techniques introduced in Chapter 2.

3.4.1 Algorithms for any p

Theorem 3.1.2 and Theorem 3.1.3 with hash functions $\hat{h}_p$ (Lemma 3.3.15) and $\tilde{h}_p$ (Lemma 3.3.18) gives:

Theorem 3.4.1. *For any $p \in [1, \infty]$ and for any $c > \tau_p$, we show data structures for the c–near neighbors with*

[2] There are many possibilities of choosing a random point from a sphere in ℓ_p. We conjecture that the bounds should hold for both geometric surface measure and cone measure.

- *$\mathcal{O}(n^{1+\frac{\ln 3}{\ln(c/\tau_p)}})$ preprocessing time and $\mathcal{O}(\log n)$ query time for the 'fast query' algorithm,*
- *$\mathcal{O}(n \log n)$ preprocessing time and $\mathcal{O}(n^{\frac{\ln 3}{\ln(3c/\tau_p)}})$ query time for the 'fast preprocessing' algorithm.*[3]

We distinguish two cases of the theorem for hash functions $\hat{h}_p$ and $\tilde{h}_p$ respectively:

1. $\tau_p = \hat{\tau}_p = 2\sqrt{2}d^{\max\{1-1/p,1/2\}}$,
2. $\tau_p = \tilde{\tau}_p = 2d^{1/2+|1/2-1/p|}$.

3.4.2 Algorithms for $p = 2$

Combining Corollary 1.3.1 with Theorem 3.1.2, we can obtain the algorithm with the polynomial pre-processing time and the sub-linear query time. Theorem 3.4.1 states, that for any $c > 2\sqrt{d}$, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be solved in the $\mathcal{O}(n^{1+\frac{\log 3}{\log(c/\tilde{\tau})}})$ pre-processing time and the query time equal to $\mathcal{O}(d|P| + d \log n + d^2)$, where P is the size of the result set and $\tilde{\tau} = 2\sqrt{d}$. Altogether, setting $\alpha = c/2$ in Corollary 1.3.1, we get:

Theorem 3.4.2. *The $\mathsf{NN}_{\text{wfn}}(c, d)$ can be solved for any $c > \tilde{\kappa} = 16\sqrt{\log n}$ with:*

$$\text{query}(c, d) = \mathcal{O}(d|P| + d \log n + d^2)$$

$$\text{preproc}(c, d) = \mathcal{O}(dn^{1+\frac{\ln 3}{\log(c/\tilde{\kappa})}} / \log(n)).$$

The time complexity of the algorithm is the same as for previously introduced algorithm which required $c = \Omega(\sqrt{d})$. For large d, the exponent in the pre-processing is much smaller comparing with the previously introduced algorithms. We focus on the query efficient version of the problem. It is easy to obtain results for pre-processing efficient version with pre-processing time $\mathcal{O}(n \log n)$.

[3] For simplicity, we omitted the factors dependent on d.

Algorithms for $p = 2$ and $c = \omega(\sqrt{\log\log n})$

In order to achieve an efficient algorithm for $c = \omega(\log(\log(n)))$, we will make a series of reductions. First, using Corollary 1.3.1, we reduce our problem to a number of $\mathsf{NN}_{\text{wfn}}(\mathcal{O}(c), \mathcal{O}(\log n))$ problems. Next, these problems are reduced to a number of $\bigoplus_{l_\infty} l_p\text{-}NN$ problems with dimension k of $O(\log\log n)$. In the end, we use Theorem 3.2.1 to solve the $\bigoplus_{l_\infty} l_p\text{-}NN$. Again one can produce multiple versions of algorithms giving different trade-offs between the pre-processing time and the query time. Particularly, the algorithm with the $\mathcal{O}(n \log n)$ processing time and the sub-linear query time can be obtained. The same can be done for Theorem 3.4.3. We omit this to avoid the unnecessary complications.

Theorem 3.4.3. *The $\mathsf{NN}_{\text{wfn}}(c, d)$ can be solved with:*

- *pre-processing time $\tilde{\mathcal{O}}(d^2 n + dn^{1+\frac{\ln 3}{\ln(c/\mu)}+1/f(n)})$,*
- *query time $\tilde{\mathcal{O}}(d^2 + dn^{1/f(n)}|P|)$,*

for any $c > \mu = D\sqrt{f(n)\log\log n}$,
where $f(n)$ is any function, which satisfies $1/f(n) = o(1)$ and D is some constant.

Proof. There are two consecutive reductions:

1. By Corollary 1.3.1, the $\mathsf{NN}_{\text{wfn}}(c, d)$ can be reduced to d instances of the $\mathsf{NN}_{\text{wfn}}(\alpha_1, k_1)$.
2. By Corollary 2.2.2.1, the $\mathsf{NN}_{\text{wfn}}(\alpha_1, k_1)$ can be reduced to k_1^L instances of the
$\bigoplus_{l_\infty} l_p\text{-}NN(\alpha_2, k_2, L)$.

Accordingly, we set:

1. $\alpha_1 = c/2$ and $k_1 = \lceil D_1 \log n \rceil$ in the first reduction,
2. $\alpha_2 = c/4$, $k_2 = \lceil D_2 L^{-1} \log n \rceil \leq \lceil D_2 f(n) \log\log n \rceil$ and $L = \lceil \frac{\log n}{f(n)\log\log n} \rceil$ in the second reduction.

The constants D_1 and D_2 are chosen to satisfy Corollaries 1.3.1 and 2.2.2.1. k_1^L can be bounded in the following way:

$$k_1^L = \lceil D_1 \log n \rceil^L = \tilde{\mathcal{O}}(n^{1/f(n)}) = \tilde{\mathcal{O}}(n^{o(1)}).$$

The final query complexity, up to factors logarithmic in n, equals:

$$\begin{gathered}
\text{query}(c,d) = \tilde{\mathcal{O}}(d^2 + d\ \text{query}(\alpha_1, k_1)) = \\
\tilde{\mathcal{O}}(d^2 + d\big(Lk_1^2 + k_1^L\ \text{query}_{l_\infty\text{-product}}(\alpha_2, L, k_2)\big)) = \\
\tilde{\mathcal{O}}(d^2 + dn^{1/f(n)}\ \text{query}_{l_\infty\text{-product}}(\alpha_2, L, k_2)) = \\
\tilde{\mathcal{O}}(d^2 + dn^{1/f(n)}(k_2L|P| + k_2\log n + k_2L) = \\
\tilde{\mathcal{O}}(d^2 + dn^{1/f(n)}|P|) = \\
\tilde{\mathcal{O}}(d^2 + dn^{o(1)}|P|).
\end{gathered}$$

The final pre-processing time equals:

$$\begin{gathered}
\text{preproc}(c,d) = \tilde{\mathcal{O}}(d^2n + d\ \text{preproc}(\alpha_1, k_1)) = \\
\tilde{\mathcal{O}}(d^2n + d\ \big(Lk_1^2n + k_1^L\ \text{preproc}_{l_\infty\text{-product}}(\alpha_2, L, k_2)\big) = \\
\tilde{\mathcal{O}}(d^2n + dn^{1/f(n)}\ \text{preproc}_{l_\infty\text{-product}}(\alpha_2, L, k_2) = \\
\tilde{\mathcal{O}}(d^2n + dn^{1+\frac{\ln(3)}{\ln(c/\kappa)}+1/f(n)}),
\end{gathered}$$

where $\kappa = 2\sqrt{k_2} = D\sqrt{f(n)\log\log n} = \mu$.

□

The function $f(n)$ may be chosen arbitrarily. Slowly increasing $f(n)$ will be chosen for small c close to $\Theta(\log\log n)$. For larger c, one should choose the maximal possible $f(n)$, to optimize the query time complexity.

Chapter 4

Quantization approach

In this chapter, we show another approach to c–near neighbors without false negatives. We start with the dimension reduction, presented in Chapter 2, and afterwards we apply the algorithm presented in [1] to the resulting low–dimensional instances. This leads to relaxation of the constraints on c and to improving the complexity of the algorithms. Particularly, for ℓ_2:

Theorem 4.0.1. *The $\mathsf{NN}_{\text{wfn}}(c, d)$ in ℓ_2 can be solved with the amortized query time $\mathcal{O}(d^{\omega-1} + n^{\nu})$ and the pre-processing time:*

- $n^{1+\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon})}$ *for any $c < 2$,*
- $\mathcal{O}(d^{\omega-1}n + dn^{1+\mathcal{O}(\frac{1-\nu}{\log c})}/\log n)$ *for any $c \geq 2$,*

for some tunable parameter $\nu \in [0, 1)$ and $\epsilon = c - 1$.

We assume, that the queries are provided in batches of size d. This assumption can be omitted, which leads to an algorithm with the query time of $\mathcal{O}(d^2 + n^{\nu})$. The above is also valid for the other presented algorithms (in particular, for Theorem 4.0.2). We focus on the batch version to avoid unnecessary complexity.

In particular, for $\nu = 0$ and $c \leq 2$, we obtain an algorithm with the query time $\mathcal{O}(d^{\omega-1})$ and the pre-processing time $n^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$. For small c, our results are incomparable with the previously discussed algorithms. In particular, our results are similar to the algorithm presented recently in [6], which works with the query time $n^{o(1)}$ and the pre-processing time $n^{\mathcal{O}(1/\epsilon^2)+o(1)}$. However, results presented in [6] give weaker Monte Carlo guaranties. Increasing the parameter ν allows us to reduce the preprocessing complexity. For $\nu = 1/c$ we achieve an algorithm with the query time $\mathcal{O}(n^{1/c})$ and the pre-processing time $n^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. Setting $\nu = 1 - \epsilon^2/\log\frac{1}{\epsilon}$ gives the algorithm with the polynomial pre-processing time independent of c.

In addition, we show the pre-processing efficient versions of the algorithms, which have an optimal in therms of n, linear complexity.

Theorem 4.0.2. *The $\mathsf{NN}_{\mathrm{wfn}}(c,d)$ in ℓ_2 can be solved with the pre-processing time $\mathcal{O}(d^{\omega-1}n)$ and the amortized query time:*

- $\mathcal{O}(d^{\omega-1} + dn^{\frac{1}{1+\mathcal{O}(\epsilon^2\log^{-1}\frac{1}{\epsilon})}})$ *for any $c < 2$,*
- $\mathcal{O}(d^{\omega-1} + dn^{\mathcal{O}(\frac{1}{\log c})})$ *for any $c \geq 2$,*

where $\epsilon = c - 1$.

This gives new results for probably the most interesting case from the practical point of view. In particular, for $c \leq 2$, we achieve the query time:

$$\mathcal{O}(d^{\omega-1} + dn^{\frac{1}{1+\mathcal{O}(\epsilon^2\log^{-1}\frac{1}{\epsilon})}}) = \mathcal{O}(d^{\omega-1} + dn^{1-\mathcal{O}(\epsilon^2\log^{-1}\frac{1}{\epsilon})}) = \mathcal{O}(d^{\omega-1} + dn^{1-\mathcal{O}(\epsilon^3)})$$

Again, our algorithm gives results similar to the ones presented in [6] which gives the query time equal to $\mathcal{O}(d + n^{1-\mathcal{O}(\epsilon^2)+o(1)})$.

The above results can be extended to ℓ_p for any $p \in [1, \infty]$.

Lemma 4.0.3. *Let us assume, that there is an algorithm for $\mathsf{NN}_{\mathrm{wfn}}(c,d)$ for ℓ_2, then there exists algorithm for $\mathsf{NN}_{\mathrm{wfn}}(cd^{|1/2-1/p|}, d)$ for ℓ_p any $p \in [1, \infty]$.*

Proof. Assume, that we are given a query point $q \in \mathbb{R}^d$. Consider two cases:

- Let us assume, that $p < 2$.
  If we have $\|q - x\|_p \leq 1$, for $x \in \mathbb{R}^d$, then the algorithm for $\mathsf{NN}_{\mathrm{wfn}}(c,d)$ in ℓ_2 will return x, because $\|x - y\|_2 \leq \|x - y\|_p \leq 1$.

  Assume that the algorithm returned some x. We have $\|x - y\|_2 < c$ and consequently:

  $$\|x - y\|_p \leq \|x - y\|_2 d^{|1/2-1/p|} < cd^{|1/2-1/p|}.$$

  Thus, $\|x - y\|_p < cd^{|1/2-1/p|}$.

- Now, let us assume, that $p \geq 2$.
  In this case, we will solve $\mathsf{NN}_{\text{wfn}}(c, d)$ in ℓ_2 in space rescaled by $d^{-|1/2-1/p|}$. If we have $\|q - x\|_p \leq 1$, for $x \in \mathbb{R}^d$, then the algorithm for $\mathsf{NN}_{\text{wfn}}(c, d)$ in ℓ_2 will return x, because $\|x - y\|_2 d^{-|1/2-1/p|} \leq \|x - y\|_p \leq 1$.

  Assume that the algorithm returned some x. We have $\|x - y\|_2 < c$ and consequently: $\|q - x\|_p \leq \|q - x\|_2 < cd^{|1/2-1/p|}$. Thus, $\|x - y\|_p < cd^{|1/2-1/p|}$.

□

The above lemma combined with Theorems 4.0.1 and 4.0.2 gives the best algorithms for ℓ_p for $p \in [1, \infty]$, that are presented in this work.

All of the presented algorithms give Las Vegas guaranties, which are stronger than the Monte Carlo guaranties considered previously by other authors. The provided algorithms are practical in terms of implementation.

4.1 Nearest neighbors without false negatives for any c

In this section, we show an efficient algorithm for solving the $\mathsf{NN}_{\text{wfn}}(c, d)$ in ℓ_2. Indyk and Motwani [1] showed an algorithm with the pre-processing time $n\mathcal{O}(1/\epsilon)^d$ and the query time equal to $\mathcal{O}(d)$, where $\epsilon = c - 1$. The idea of the algorithm is the following. We start with a quantization of the given space, which reduces the problem to finding the near neighbor in a space with integer coefficients. After the quantization, there is a finite number of points which have a neighbor in the input set. It is enough to provide the data structure which will store all such points with accompanying near neighbors from the input set. It is proved that the number of neighbors of each input point is $\mathcal{O}(1/\epsilon)^d$. So in total, we need to store $n\mathcal{O}(1/\epsilon)^d$ such points. We can fetch a point from the data structure in the time proportional to the size of this point, thus the query time is $\mathcal{O}(d)$. This storage data structure is presented in Section 1.1.

The above construction gives an algorithm for the c–near neighbors in ℓ_2 with the efficient query time. Unfortunately, unless $d = \mathcal{O}(\log n)$, the pre-processing time is exponential. If the dimension is larger than $\gamma \log n$, with γ defined as in Corollary 1.3.1, we may reduce the complexity of the pre-processing by reducing the dimension of the input space.

4.1.1 Fast query

In this section we prove Theorem 4.0.1:

Theorem 4.0.1. *The $\mathsf{NN}_{\text{wfn}}(c,d)$ in ℓ_2 can be solved with the amortized query time $\mathcal{O}(d^{\omega-1}+n^{\nu})$ and the pre-processing time:*

- $n^{1+\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon})}$ *for any $c<2$,*
- $\mathcal{O}(d^{\omega-1}n + dn^{1+\mathcal{O}(\frac{1-\nu}{\log c})}/\log n)$ *for any $c\geq 2$,*

for some tunable parameter $\nu\in[0,1)$ and $\epsilon=c-1$.

Proof. Consider two cases:

- $c<2$:

  For $c<2$, we set $\alpha=\frac{c+1}{2}$ in Corollary 1.3.1. It follows that $\mathsf{NN}_{\text{wfn}}(c,d)$ can be reduced to $\mathcal{O}(d/(\gamma\log n))$ instances of $\mathsf{NN}_{\text{wfn}}(c/2+1/2,\gamma\log n)$ and:

  – the query time equals $\mathcal{O}\big(d^{\omega-1}+n^{\nu}+d/(\gamma\log n)\ \mathrm{query}(\frac{c+1}{2},\gamma\log n)\big)$,
  – the pre-processing time equals $\mathcal{O}\big(d^{\omega-1}n+d/(\gamma\log n)\ \mathrm{preproc}(\frac{c+1}{2},\gamma\log n)\big)$.

  Since $\log x < x-1-(x-1)^2/2$ for $x<1$,

  $$\gamma < \frac{2(1-\nu)}{(\frac{\alpha}{c})^2-1-2\log\frac{\alpha}{c}} < (1-\nu)\Big(\frac{2c^2}{c^2-\alpha^2}\Big)^2 = \mathcal{O}\Big(\frac{1-\nu}{\epsilon^2}\Big).$$

  Consequently, we reduce the problem to $\mathcal{O}(d/(\gamma\log n))$ instances of $\mathsf{NN}_{\text{wfn}}(\frac{c+1}{2},\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log n))$. By [1], each of these instances is solved with the pre-processing time $\mathcal{O}\big(\frac{1}{\epsilon}\big)^{\frac{1-\nu}{\epsilon^2}\log n} = n^{\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon})}$ and with the query time $\mathcal{O}(\gamma\log n)$.

- $c\geq 2$:

  After setting α to any constant value such that $\log 2 > \log\alpha + 1/2$ in Corollary 1.3.1, we reduce the problem to $\mathcal{O}(d/(\gamma\log n))$ instances of $\mathsf{NN}_{\text{wfn}}(\mathcal{O}(1),\mathcal{O}(\gamma\log n))$. We have:

  $$\gamma < \frac{2(1-\nu)}{(\frac{\alpha}{c})^2-1-2\log\frac{\alpha}{c}} < \frac{1-\nu}{\log c-\log\alpha-\frac{1}{2}} = \mathcal{O}\Big(\frac{1-\nu}{\log c}\Big).$$

  Each of these instances is solved with the pre-processing time $n\mathcal{O}(1)^{\frac{(1-\nu)\log n}{\log c}} = n^{1+\mathcal{O}(\frac{1-\nu}{\log c})}$ and with the query time $\mathcal{O}(\gamma\log n)$.

This ends the proof. □

4.1.2 Fast pre-processing

In this section, we prove Theorem 4.0.2. To achieve the algorithm with the fast, linear pre-processing, we will store all input points in the hash-map. During the query phase, we will ask the hash-map for all of $\mathcal{O}(1/\epsilon)^d$ of points which are close to this query point. This way, most of the computation is moved from the pre-processing to the query phase. The dimension reduction is used in a similar manner as in Section 4.1.1. We skip the computation steps which are analogical to the corresponding ones in the previous section.

Theorem 4.0.2. *The $\mathsf{NN}_{\mathrm{wfn}}(c, d)$ in ℓ_2 can be solved with the pre-processing time $\mathcal{O}(d^{\omega-1}n)$ and the amortized query time:*

- $\mathcal{O}(d^{\omega-1} + dn^{\frac{1}{1+\mathcal{O}(\epsilon^2 \log^{-1}\frac{1}{\epsilon})}})$ *for any $c < 2$,*
- $\mathcal{O}(d^{\omega-1} + dn^{\mathcal{O}(\frac{1}{\log c})})$ *for any $c \geq 2$,*

where $\epsilon = c - 1$.

Proof. Consider two cases:

- For $c < 2$, after setting $\alpha = c/2 + 1/2$ in Corollary 1.3.1 we get an algorithm with:
  - the query time: $\mathcal{O}\big(d^{\omega-1} + n^{\nu} + d/(\gamma \log n)\ \mathrm{query}(\frac{c+1}{2}, \gamma \log n)\big) = \mathcal{O}\big(n^{\nu} + n^{\mathcal{O}(\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon})}\big)$,
  - the pre-processing time: $\mathcal{O}\big(d^{\omega-1}n + d/(\gamma \log n)\ \mathrm{preproc}(\frac{c+1}{2}, \gamma \log n)\big) = \mathcal{O}(d^{\omega-1}n)$.

  Let us assume that the query time is $\mathcal{O}\big(n^{\nu} + n^{D\frac{1-\nu}{\epsilon^2}\log\frac{1}{\epsilon}}\big)$ for some constant D. After setting

  $$\nu = \frac{D\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}}{D\frac{1}{\epsilon^2}\log\frac{1}{\epsilon} + 1},$$

  we get the query time complexity equal to $\mathcal{O}(n^{\frac{1}{\epsilon^2 D^{-1}\log^{-1}\frac{1}{\epsilon}+1}})$.

- For $c \geq 2$, after setting α to any constant value such that $\log 2 > \log\alpha + 1/2$ and $\nu = 0$ in Corollary 1.3.1, we get the algorithm with:

- the query time: $\mathcal{O}\big(d^{\omega-1} + n^{\nu} + d/(\gamma \log n)\ \text{query}(\alpha, \gamma \log n)\big) = \mathcal{O}(dn^{\mathcal{O}(\frac{1}{\log c})})$
- the pre-processing time: $\mathcal{O}\big(d^{\omega-1}n + d/(\gamma \log n)\ \text{preproc}(\alpha, \gamma \log n)\big) = \mathcal{O}(d^{\omega-1}n)$.

This ends the proof. $\square$

One can produce the Monte Carlo version of Theorems 4.0.1 and 4.0.2, which have only slightly better complexities (some factors of d would be removed), because the dimension reduction is simpler in this case.

Part II

Nearest Neighbors in Social Networks

Chapter 5

Introduction

In this part of the thesis, we study the structure of the neighborhood in social networks. In general, social networks consists of users who disseminate information in a directed graph of connections. We construct a neighborhood with the emphasis on the information dissemination, i.e., roughly speaking the "weight" of the connection between two users is related to the volume of information passed between these nodes. Unlike the models presented in the first part of the thesis, this relation is asymmetric. It is often the case that, the information flow is unidirectional, e.g., the relation between a social media influencer and her fan in which only fan responds to the influencer's content and not the other way around.

More precisely, we say that a node Q is a neighbor of a node P if there is a large chance that the information produced by P will be spread by Q. The decision of spreading of the news depends on many factors like the content of the message, the personal relationship between P and Q, the time of the day, the current mood of the receiver etc. In this work, we consider the probabilistic model in which we treat the impact of all unobserved variables as a random factor. We assume that there exists a directed graph, such that the information might be spread only between nodes which are connected in the graph. Since the information spread process is probabilistic, there are many realizations of this process. A realization of this process: the starting node together with all other nodes which passed this information is called a cascade.

The crucial mechanisms describing the neighborhood are the structure of the graph and information dissemination model. In this part of the thesis, we study both of these aspects. In Chapter 6, we consider retweet graph in the Twitter network and we propose models for information dissemination in this graph. We discuss existing models, show their drawbacks and provide new models which overcome the existing issues. We provide an approach which enables meaningful comparison of different models of information dissemination with the real cascade distribution.

In Chapter 7, we study the performance of the basic information dissemination model on random graphs. We consider the directed acyclic version of Erdős–Rényi graphs. We provide the characteristics of these graphs and show that the considered model satisfy the basic properties of the cascade distribution. It is known that the cascades distribution can be approximated by power–law distribution. We show that this is the case for the Erdős–Rényi graphs.

5.1 Preliminaries

For two series x_n, y_n, we will say that $x_n \sim y_n$ iff $\lim_{n\to\infty} \frac{x_n}{y_n} = 1$. The same notation is used when a random variable X satisfies a distribution $Dist$: $X \sim Dist$.

Zipf distribution is a discrete distribution on set $\{1, \ldots, n\}$, such that:

$$\mathbb{P}\left[Z = k\right] = \frac{1}{k^s H_{n,s}} \text{ for } k = 1, \ldots, n,$$

where variable Z has Zipf distribution. $H_{n,s}$ is a normalizing constant – the n'th generalized harmonic number. If the variable satisfies Zipf's distribution, we would interchangeably say that it satisfy power law. Also, satisfying power law would usually mean that the distribution can be approximated with small error by Zipf's distribution.

Binomial distribution $Bin(n, p)$ is a discrete distribution on set $\{0, \ldots, n\}$ which is number of successes in a sequence of n experiments, where each of experiment has probability of success equal to p. We have:

$$\mathbb{P}\left[Z = k\right] = \binom{n}{k} p^k (1-p)^{n-k},$$

where variable $Z \sim Bin(n, p)$.

Negative binomial distribution $NB(r, p)$ is a discrete distribution on set of natural numbers which is the number of experiments done until the rth

failure. We have:

$$\mathbb{P}\left[Z = k\right] = \binom{k+r-1}{k} p^k (1-p)^r,$$

where variable $Z \sim NB(r, p)$.

It is well known that the cumulative distribution functions (cdf's) of binomial and negative binomial are connected

Fact 5.1.1. *Let $X \sim Bin(n, p)$ be binomial random variable and let $Y \sim NB(r, 1-p)$ be negative binomial random variable, then:*

$$\mathbb{P}\left[X < r\right] = \mathbb{P}\left[Y > n\right].$$

Poisson distribution $Pois(\lambda)$ is a discrete distribution on set of natural numbers such that:

$$\mathbb{P}\left[Z = k\right] = \frac{\lambda^k e^{\lambda}}{k!},$$

where variable $Z \sim Pois(\lambda)$.

Let us state the well–known Chernoff's bound:

Theorem 5.1.2. *Let X be random variable with binomial distribution: $Bin(n, p)$, then for any $\epsilon > 0$*

$$\Pr(X \geq (1+\epsilon)pn) \leq e^{-\frac{\epsilon^2 pn}{2+\epsilon}}, \qquad 0 \leq \epsilon,$$

and

$$\Pr(X \leq (1-\epsilon)pn) \leq e^{-\frac{\epsilon^2 pn}{2}}, \qquad 0 \leq \epsilon \leq 1.$$

5.2 Related Work

In this section we discuss previous studies on the structure of information dissemination in social networks and the structure of the social network itself. There are different aspects which were taken into consideration while modeling the information cascades:

- What is the model of large cascades? [34, 35, 36]
- What is the distribution of cascade sizes? [37]

- How to optimally choose nodes of the social networks in order to inform the largest number of nodes in the network? [38, 39]

Most of the work was devoted to study large cascades what is motivated by optimizing the choice of influencer in viral marketing. In our work, we focus on the whole distribution of cascades including small ones. In fact, the vast majority of cascades is very small. For more references on modeling cascades see [40].

5.2.1 Information Dissemination Models

There are different models of information dissemination used in context of rumor spreading, viral marketing, infection expansion or distributed computing. One of the most popular is SIR model in which each node spreads the information with fixed probability p to any other user [41]. The first models made unrealistic assumption that each node is connected with each other node. This problem can be avoided when we assume that the information is disseminated only along the edges of a given fixed graph (see e.g., [42, 43]). One can see that setting one parameter p for all nodes might be to restricted, thus researchers consider model with a distinct probability $p_{v,w}$ for all nodes' pairs v, w (see e.g. [44]). Unfortunately, this model might be strongly over–fitted, because there is not enough information for most of the nodes pairs. It might be interesting to introduce small number of latent variables and affiliate each user with a distribution of these latent variables. It could strongly reduce the number of variables while distinguishing different types of users. Moreover, recent research on the power law distributions show that approximation of such distributions might be generated by a mixing a number of latent non power law distributions [45, 46]. Unfortunately, we are not aware of any prior work on such models, thus in this thesis we assume that models are parametrized by some small number of parameters.

The phantom observed in SIR model is that the very large cascades are overrepresented [47]. The real cascades distribution is a heavy tail distribution for which this phantom does not occur [48]. In our work, we propose enhancements of SIR model which improves the distribution of the cascades and avoids mentioned blow–up effect. Up to our knowledge, the only work in which similar models were considered is [37]. However, [37] did not contain any meticulous measurements of goodness of the fit to the real world data.

Another method for correcting the cascade distribution is adding *stiffler* effect [49]. The stifflers do not spread the information. Moreover, stiffler can turn its susceptible neighbors into stifflers. Unfortunately, it is not clear how to justify such a model in social networks. We can assume that stifflers are entities who received the information from outside of a given social network and also disseminate this information outside the network. The models proposed in this work are more direct and interpretable.

Watts at al. [50] considered threshold based information spreading mechanism. A given node spreads the information if the number of its graph neighbors who spread the news is larger than certain threshold. Although, it is shown that such information dissemination mechanism produces proper cascades distribution for large random graphs, there is no evidence that such a mechanism applies to real social networks.

5.2.2 Graph Models

There are different models of social networks starting from the network itself and ending with randomly generated graphs. The natural approach is to use the considered network e.g.: Twiter, Flicktr or Digg [35, 36, 51]. The simulations presented in this work were done using the Twitter network. There are various way of defining the graph structure. One might use the follower–followee relationship. At twitter one can retweet the message written by user who is not followed by us. Moreover, some of the followers never disseminate the information produced by the user they follow. Thus, one might consider a retweet graph in favor of folower-folowee graph as a valid medium of information dissemination [52]. Formally, the retweet graph is a directed graph in which there is an arc between two users if and only if we observe any retweet or any replay between these users. In our simulations we use retweet graph. Another issue that needs to be taken into account is the fact that the social network can evolve over time [53]. Moreover, retweet graph depends on the volume of events in our dataset. In Section 6.2, we describe experiment which shows, that presented models do not depend on the specific graph.

A certain properties are satisfied by social networks. The in and out degrees of the nodes follow the Zipf distribution, i.e., the satisfy power law. There are similar characteristics related to number of messages sent by a user, the lengths of threads, number of links between the origin and the node which is furthest away (exploration depth) etc. [49, 54, 55]. The graphs which are known to satisfy such constraints are Kronecker graphs [56]. The Kronecker graphs are build in a recursive manner: computing the Kronecker product of two adjacency matrices gives a matrix for new graph.

In our work, we follow the similar approach, i.e., we show the theoretical properties of the information dissemination on radnom acyclic Erdős–Rényi graph [11]. We show, that for directed acyclic Erdős–Rényi graphs and the SIR model the cascade sizes follow the power law. Such graph were previously used in the research on the properties of large cascades in [57].

5.3 Our Results

Let us outline the results presented in this part of the thesis. We propose a model of neighborhood in the social networks. The key ingredient of these models is the process of information dissemination. We study this process on graphs obtained from real networks but also on graph generated by a random process.

In Chapter 6, we discuss information dissemination process on Twitter network [10]. We show a meticulous way of comparing different models. We propose two new models *exp*–SIR and multi–source. The summary of KS-test results are presented in Table 6.1. The *exp*–SIR model takes into account the effect of suppressing the interest in new information over time. From another perspective, the *exp*–SIR model takes into account the fact that close relationship with the followee increases the chance of spreading information. In the multi–source model, we assume that information might be spread outside of our network: by the word of mouth, television radio, newspapers, phone etc. We show that both models fix problems which were present in the previous research and that the obtained distribution of the cascades is much closer to the real one. In particular, presented models suppress blow up effect (see Figure 5.1) and strongly improve KS-test score which is proposed to measure the distance between the real cascades distribution and the distribution generated by the information dissemination model.

In Chapter 7, we analyze the basic model for information dissemination on directed acyclic Erdős–Rényi graphs. We show that these models satisfy the basic property of real cascades: the distribution of cascade sizes satisfies power law. Moreover, we show the fundamental properties of considered random graphs. In Theorem 7.1.1 (page 84), we show that the distribution of the degrees in the graph is approximately uniform. The distribution of the

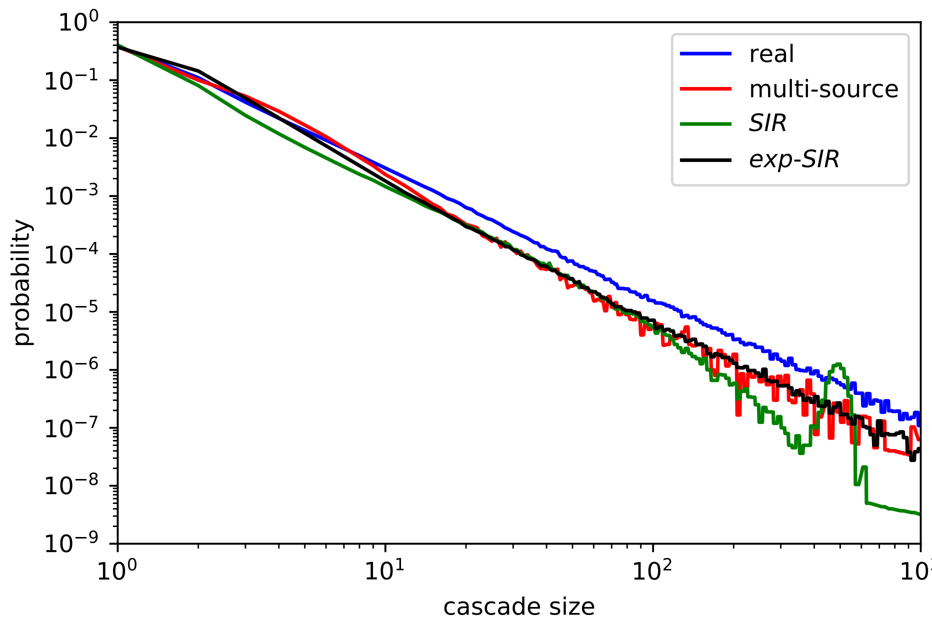

Figure 5.1: The comparison of the distributions of all presented models and the original distribution. The plot for SIR model shows large overrepresentation of cascades larger than 1000. This is not present for the enhanced exp–SIR and multi–source models.

degrees is the crucial property that characterizes the graph. In the Theorem 7.2.1 (page 87) we show the the most important result of this chapter, i.e. we show that the probability of a cascade of size k in random directed acyclic Erdős–Rényi graph with n vertices is $(n(1-(\alpha p))^k)^{-1}$, where p is a parameter of the graph and α is the parameter of the SIR model. For small αp the above formula approximately equals $(n\alpha pk)^{-1}$ which implies power law distribution of cascades.

Chapter 6

Infromation Dissemination on Retweet Graph

In this chapter, we study the structure of neighborhood in social networks by the example of Twitter network [10]. There are many possible concepts of neighborhood in the social network with the most trivial being the one given by the network itself: the neighbors of a given person are people who follow her in Twitter. This definition does not fully capture the hidden information dissemination network, because large part of the communication is done through retweets (retweet is done not necessarily by the follower) and some followers might never actually respond to the messages written by the followee. The other commonly used graph structure is so–called retweet graph [52]. In this graph, the user A is connected with user B, if and only if the user B retweetted or responded to the message which was produced by A. The retweet graph might be favored when the main goal is to study the dissemination of information.

Having the network which links the users and the information dissemination process, we define the neighborhood of a given user by all people who propagate the information produced by this user. Similarly, the neighborhood might have been defined as all people who received the information while have not actively disseminated it. Is this a proper definition of neighborhood? Unfortunately the answer is no, the decision of "retweetting" is not deterministic. Clearly, it can depend on the message itself but also, the current mood of the user, the time of the day etc. Thus, the proper definition of the neighborhood for a given user would be, all users who propagate the information produced by this user with at least probability p. We consider a random process of information dissemination, in which information is propagated with certain probability. The given realization of this process – a concrete series of retweets and replays is called cascade. The process of information dissemination is sometimes called rumor spreading.

As mentioned before, often considered models assume that each user who received the information propagates it to her neighbors with constant probability p [41]. In the SIR model each of node is in one of the three states: Susceptible (S), Infected (I) and Recovered (R). In the beginning all nodes are in state S, besides one initially infected node (seed) which is in state I. Next, each infected node will propagate the information to all of its susceptible neighbors with probability p, changing their state from S to I. After propagating the information the state of the infected node is changed to recovered.

Algorithm 5: The SIR algorithm.

```
Data: $p$ – the probability of spreading the infromation
Result: $size$ – the size of the cascade
$size = 0$;
/*Mark all nodes as susceptible*/
for $n \in all\_nodes()$ do
    $n.state = S$ ;
end
/*Mark one random node $i$ as infected*/
$i = random\_node()$;
$i.state = I$;
while there is node with state $I$ do
    $n = get\_infected()$;
    $n.state = R$ ;
    if $bernoulli(p) == 1$ then
        for $m \in neigbors(n)$ do
            if $m.state == S$ then
                $m.state = I$ ;
                $size = size + 1$ ;
            end
        end
    end
end
```

There are many variants of the SIR model. Often, the infected node infect each of its neighbor independently with probability p. We will refer to this model as $\overline{SIR}$ [41]. For the Twitter network the SIR model seems to be more relevant than $\overline{SIR}$, because usually in the Twitter, unlike the other social networks, the information is published to all people at once. Nevertheless, the results presented in this thesis work for both SIR and $\overline{SIR}$ models.

Algorithm 6: The $\overline{SIR}$ algorithm.

```
Data: $p$ – the probability of spreading the infromation
Result: $size$ – the size of the cascade
$size = 0$;
/*Mark all nodes as susceptible*/
for $n \in all\_nodes()$ do
    $n.state = S$ ;
end
/*Mark one random node $i$ as infected*/
$i = random\_node()$;
$i.state = I$;
while there is node with state $I$ do
    $n = get\_infected()$;
    $n.state = R$ ;
    for $m \in neigbors(n)$ do
        if $m.state == S$ and $bernoulli(p) == 1$ then
            $m.state = I$ ;
            $size = size + 1$ ;
        end
    end
end
```

In this section, we show that both models mentioned before are incorrect – they suffer from the “blow up” effect. Roughly speaking, if the cascade becomes large enough, it is likely that it will cover large part of the graph. This phenomenon does not occur in Twitter network. In this chapter, we propose the way of measuring how well the model fits the real world data. We propose models which avoid the blow up factors and show that these models improve with the respect to the proposed objective functions. Having correct information spreading model, we can properly compute the neighborhood of a given node.

The proposed models are modifications of SIR and $\overline{SIR}$ models. In further sections, we will usually describe this modification for SIR model, the $\overline{SIR}$ model can be modified in an analogous way.

6.1 Measuring the correctness of the model

Many information–dissemination processes, satisfy the power law, i.e., the sizes of the cascades are distributed according to the Zipf distribution (see Section 5.1). The cascades in Twitter network also follow this pattern.

We show that cascades generated using SIR and $\overline{SIR}$ do not follow this pattern (see Figure 6.1). As mentioned before, these models suffer from the blow up effect.

We propose Kolmogorov–Smirnov test (KS-Test) to measure how well the model fits the data. The KS test is a nonparametric method designed to compare two distributions based on samples, which is an established method for fitting power–law distributions [58]. We start with computing empirical cumulative distribution function (*cdf*). Then, the result of KS test for two samples S_1 and S_2 is $\|cdf_{S_1} - cdf_{S_2}\|_\infty$, where cdf_S is empirical *cdf* computed from sample S. In other words, the KS test is the maximal difference between values of empirical *cdf*'s on the same argument.

All of the parameters are estimated by grid search. In Figure 6.2 we present dependence between the probability of spreading the information and the KS test in SIR model.

6.2 Dataset

Our dataset is obtained from 10% sample of the Twitter network gathered between May 19 to May 30, 2013 [10]. It contained 500 million tweets. We constructed retweet graph in the following way: we put an edge from user v to user w if and only if user v replied, retweeted, or mentioned user w. We obtained a graph with roughly 71 million vertices and 230 million edges.

In order to obtain cascades, we considered all tweets with the same hashtag. We were interested only in new cascades, thus we keep only these hashtags which did not occur on the first day in our dataset. This should not affect the real tweet distribution, because most of the cascades last only few days [59].

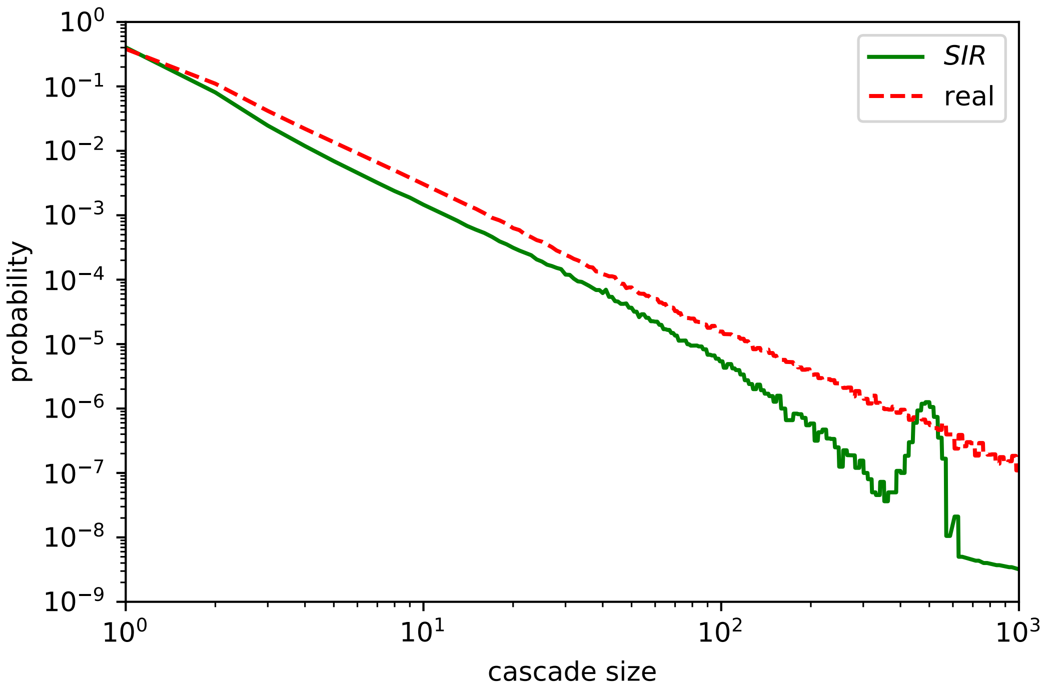

Figure 6.1: The distributions for cascade sizes obtained from SIR simulations and real distribution of cascade sizes. In the SIR model we observe the over–representation of very large cascades.

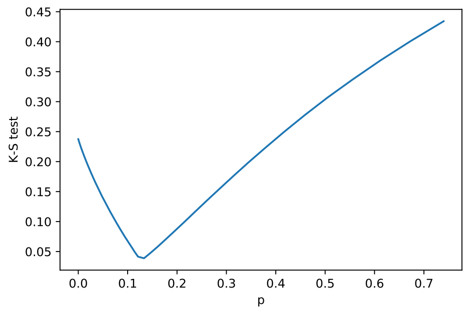

Figure 6.2: The K-S test values for cascades size distribution obtained from SIR simulations for different parameters p in SIR.

After applying this filtering, we obtained 7.7 million new, distinct hashtags.

The obtained retweet graph depends on the volume of the dataset. If the data set is larger, the graph becomes denser. All parameters presented in this work might be invalid for a graph obtained from a different dataset. Nevertheless, we claim that the presented models are general and do not depend on the specific graph. In order to prove that our models are not overfitted, we divided our dataset into two equal sets both containing a number of consecutive days. The training set was used to compute parameters and these parameters were used for the test set. We compute KS Test values for all presented models: SIR, *exp–*SIR, and multi–source, and we discovered that results for the training and the test set are equal up to 0.001 error. Thus, we claim that we do not overfit the dataset. All of the subsequent results were obtained using the whole dataset.

In order to repeat or extend our study but also to enable researchers to work on new models, the used data was made publicly available [60]. The data was anonymized in order to satisfy Twitter rules regarding public sharing.

6.3 Exponential Decay

We start with the following observation: people are willing to disseminate information more often, when they are more related to the author of the information. The information might be more attractive, when it is produced by our friend, than when it is produced by unknown person and our friend only passed it through [61].

The above idea is formalized in the following way. We introduce *exp–*SIR (*exp–*$\overline{SIR}$) model. Assume, that given information has been passed k times. The chance, that it will be passed again equals p^k, for some constant p. This way, probability of passing the information decays in each iteration of the cascade which avoid blow to occur. In Figure 6.4 we present dependence between the probability of spreading the information and the KS test in *exp–*SIR model. The test result for *exp–*SIR model with comparison to the real distribution is 0.0207. This improves over the SIR model for which the test result with comparison to the real distribution is 0.0447.

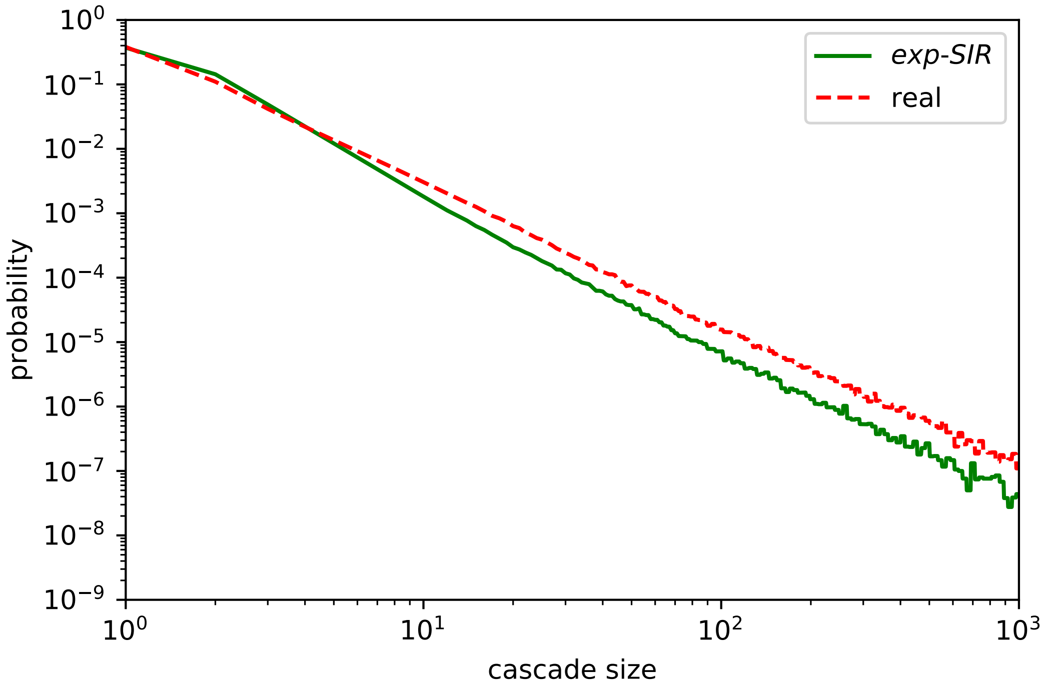

Figure 6.3: The distribution of cascade sizes obtained from exp–SIR simulations and real distribution of cascade sizes. In the exp–SIR model we do not observe the over–representation of very large cascades which was present in SIR model.

6.4 Multi source model

The exponential decay improved our model but it is still imperfect. In real life data, we observe, that many rumors do not have one source in the network. In deed, we observe only some part of the underlying information dissemination networks. Information is passed through many other channels including other social networks but also television or "mouth to mouth" communication. In this model, we assume that each node can become a source independently with some fixed probability p. The number of nodes (n) in the graph is very large, thus the probability (p) of becoming a source is very low. The number of sources is the binomial distribution: $Bin(n, p)$. By the law of rare events the binomial distribution can be approximated by Poisson distribution: $Pois(np)$.

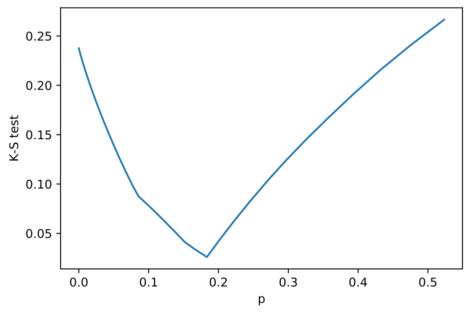

Figure 6.4: The K-S test values for cascades size distribution obtained from SIR simulations for different parameters p in exp–SIR.

The process is divided into rounds. In each round, all active nodes propagate information using the Twitter network with exponential model and also some new sources are informed by some unknown medium. We propose the following random cascade generation process:

- Draw single origin point and start simulation.
- In each round of the propagation:
    - draw number of new origins N,
    - draw N new origins uniformly from the network and start simulation with exponential decay from each of these points.

Although it might happen, that cascades from many sources will have non-empty intersection, the probability of such event is very low. The following pseudo–code describes the computation of the size of the cascade:

Algorithm 7: The multi source algorithm

```
Data: $p$ – the parameter of the $exp$–$SIR$ model, $\lambda$ – parameter of
      the Poisson distribution
Result: $size$ – the size of the cascade
$rounds = 1$;
$size = 1$;
for $round \in \{1..rounds\}$ do
    $N = Pois(\lambda).sample()$;
    for $i \in \{1..N\}$ do
        $cascade\_size, cascade\_rounds = exp$–$SIR(p)$;
        $rounds = max(rounds, round + cascade\_rounds)$;
        $size = size + cascade\_size$;
    end
end
```

Where $Pois(\lambda).sample()$ returns sample from Poisson distribution with mean λ and *exp*–SIR returns the size and the number of rounds for cascade in *exp*–SIR model with parameter p. The test result for multi-source model with comparison to the real distribution is 0.0116.

Table 6.1: The K-S test comparison of the discussed models with the real cascade size distribution.

Model	K-S test
SIR	0.0447
exp–SIR	0.0207
multi-source	0.0116

6.5 Computing Neighborhood

Given a model of information dissemination, one can compute the neighborhood of a given node. Let us recall that a neighborhood of a given node is defined as all nodes who would spread the information produced by this user with probability not smaller than some fixed value. One can use the standard Monte Carlo methods, i.e., we sample a number of cascades starting

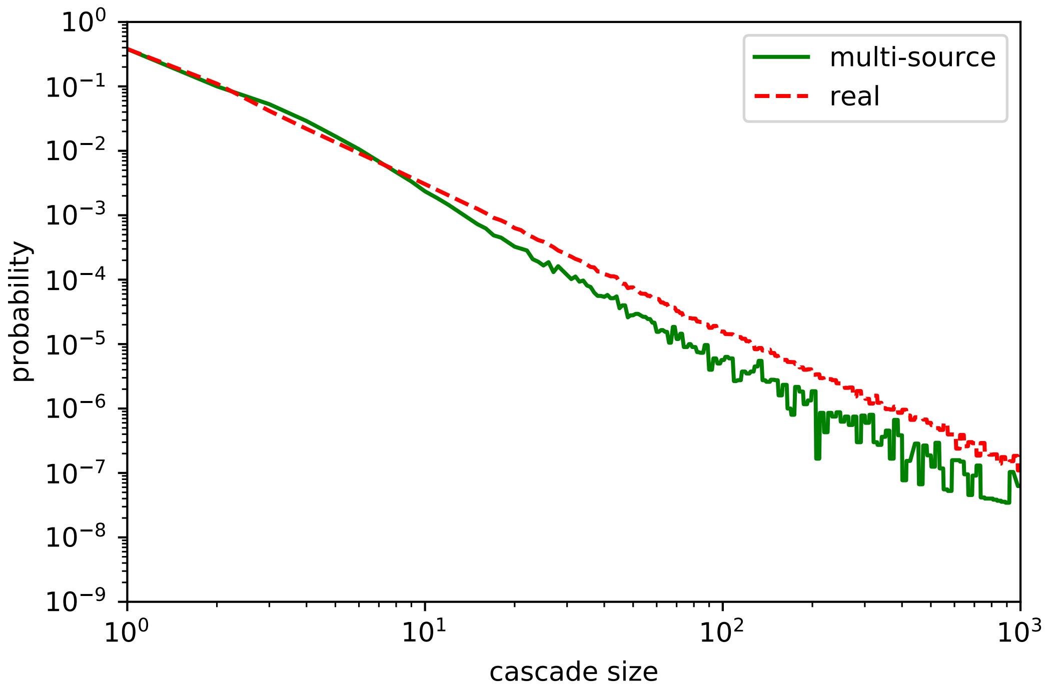

Figure 6.5: The distribution of cascade sizes obtained from multi–source model simulations and real distribution of cascade sizes.

from the node. For all nodes, we maintain the empirical probability that this node will spread the information. To compute this probability, we divide the number of times in which the node disseminated the information by the number of sampled cascades. The certainty of the empirical probability might be bounded using Chernoff's bound (Theorem 5.1.2). When the number of nodes is very large and the probability of dissemination is low, the effect of multi–source can be neglected in this simulation. This leads to pure *exp–SIR* model. The structure of the Twitter network is very hierarchical, i.e., it locally resembles the tree [43]. With this assumption, one can approximate the neighborhood directly from the model: if a node is at distance k from the seed, the probability is p^k.

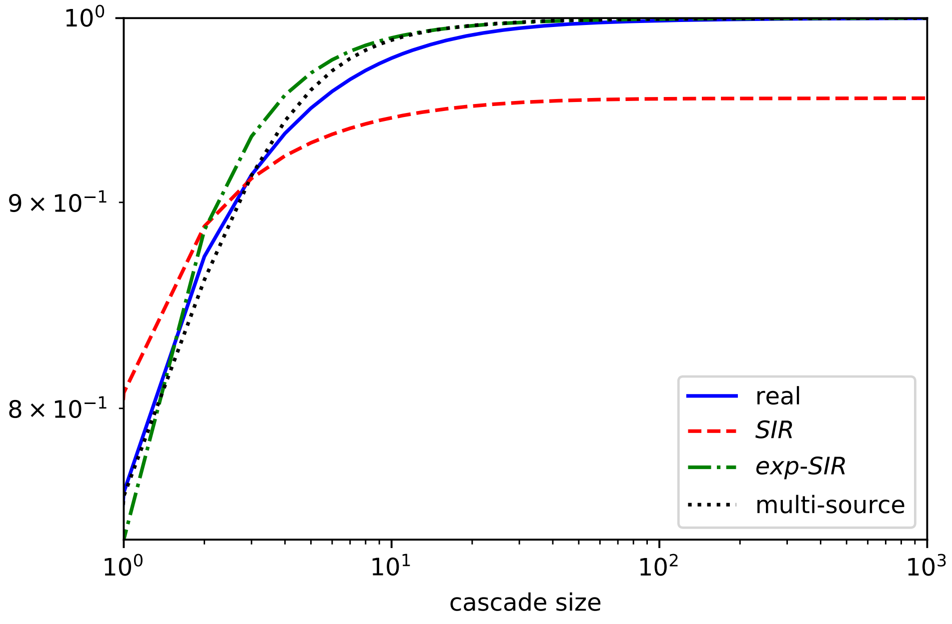

Figure 6.6: The comparison of cumulative distribution functions of all presented models and the original distribution.

Chapter 7

Theoretical Model of Information Dissemination

In this chapter, we study the theoretical models of dissemination which might lead to power–law distribution of cascades. This is, can we build a model of a network together with the dissemination process which would guarantee the power–law distribution of the cascade sizes? In this chapter, we show that this can be proven for the directed acyclic Erdős–Rényi graphs [11].

Definition 7.1.1. *Directed Erdős–Rényi with parameter p, n: $der(p, n)$ is n vertex directed acyclic graph. The vertices are labeled with numbers from 1 to n. For $i < j$, the probability of the edge from i to j equals p. All the edges are sampled independently.*

Let us observe that using this simple way of generating random graphs one can sample any directed acyclic graph. Let us discuss properties of $der(p, n)$ graphs. First, let us observe that distribution of nodes is roughly uniform. The expected degree of node i is $p(n - i - 1)$. In the following lemma we will show the further observations on the distribution of degrees in $der(p, n)$:

Theorem 7.1.1. *Assume that X be a distribution of degrees in $deg(p, n)$ graph, then the following properties hold:*

1. $\mathbb{P}[X = l] \leq \frac{1}{pn}$,
2. $\mathbb{P}[X = l] \geq \frac{1}{2pn}$ *for* $l < p(n - 1)$,
3. $\mathbb{P}[X = l] \leq \frac{\exp(-\frac{\epsilon^2}{2+\epsilon}(n-1)p)}{pn}$ *for* $l = (1 + \epsilon)p(n - 1)$ *and* $\epsilon > 0$,

4. $\mathbb{P}[X = l] \geq \frac{1-\exp(-\frac{\epsilon^2}{2}(n-1)p)}{pn}$ *for* $l = (1-\epsilon)p(n-1)$ *and* $1 > \epsilon > 0$.

Proof. Consider the node nr $n - l - j$ which can have potentially $l + j$ neighbors, then the probability that this node will have exactly l neighbors is $\binom{l+j}{l}p^l(1-p)^j$, thus:

$$\mathbb{P}[X = l] = \frac{1}{n}\sum_{j=0}^{n-l-1}\binom{l+j}{l}p^l(1-p)^j.$$

The above expression can be expressed as *cdf* of the random variable Z with negative binomial distribution $Z \sim NB(l+1, 1-p)$:

$$\mathbb{P}[X = l] = \frac{1}{pn}\mathbb{P}[Z \leq n-1].$$

Cdf of the negative binomial distribution can be expressed by the cdf of binomial distribution (see Fact 5.1.1):

$$\mathbb{P}[X = l] = \frac{1}{pn}\mathbb{P}[W \geq l+1],$$

where $W \sim Bin(n-1, p)$. Property 1, follows from the fact that probability of the event is bounded by 1. Property 2 follows from the fact that median of the $Bin(n-1, p)$ is at least $\lfloor p(n-1)\rfloor$. The last two properties follow directly from Chernoff's bound (Theorem 5.1.2). $\square$

Property 4 means that $\mathbb{P}[X = l]$ is very close to zero for $l > p(n-1)$ and l bounded from $p(n-1)$. Property 3 means that $\mathbb{P}[X = l]$ is very close to $\frac{1}{pn}$ for $l < p(n-1)$ and l bounded from $p(n-1)$. Property 2 means that $\mathbb{P}[X = l]$ is at least $\frac{1}{2pn}$ for any $l < p(n-1)$. Finally, property 1 means that $\mathbb{P}[X = l]$ is at most $\frac{1}{pn}$ for any l. Figure 7.1 contains the plot of real node degrees distribution for $der(p, n)$ graph.

7.1 Distribution of Cascade Sizes

In this section, we will formally analyze SIR and $\overline{SIR}$ models (see Section 6 for definitions). We are going to quantitatively describe the process of generating cascades in $der(p, n)$ by computing asymptotic distribution of

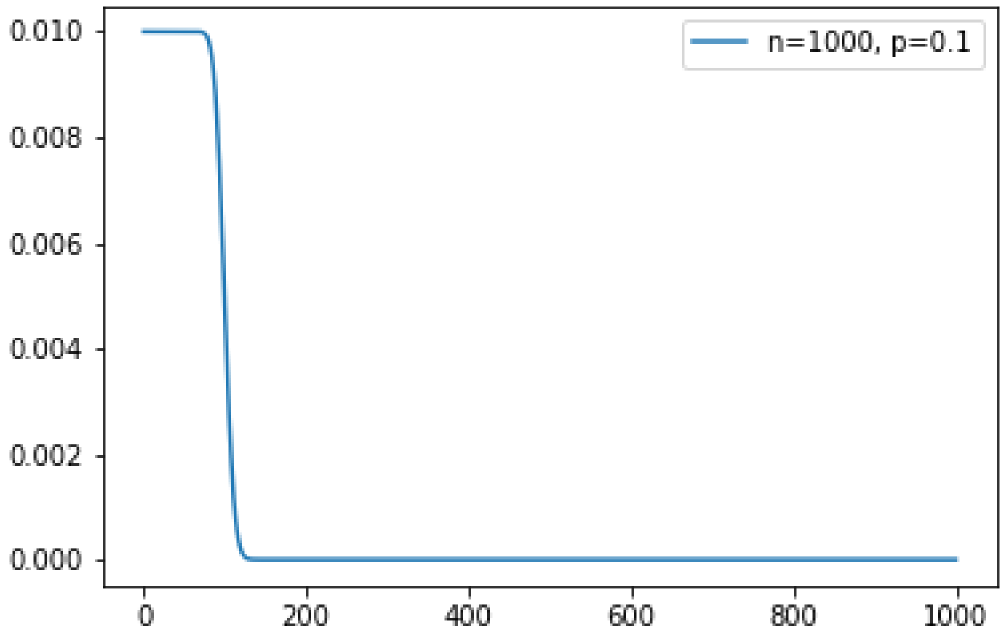

Figure 7.1: Real distribution of nodes for $der(p, n)$.

cascade sizes. The final cascade distribution is independent of the spreading model (SIR vs $\overline{SIR}$). We are going to show proof for SIR model, the proof for $\overline{SIR}$ model is analogical.

Let us set p to be the probability of the edge in the graph, α be the parameter of SIR model and $\beta = 1 - \alpha p$, which is the probability that, the informed user will not spread it through the given edge. Let $p_{n,k}$ be the probability that the cascade size is k when starting from vertex 1 in a graph with n vertices. We are going to produce recurrent dependencies. Clearly $p_{n,k} = 0$ for $k > n$, because cascade cannot be larger than the number of vertices and $p_{1,1} = 1$, since we assume that, the first vertex is always spreading. Note that by removing the nth vertex (the one who can potentially receive information from all other vertices) from $der(p, n)$ graph one get just the $der(n - 1, p)$ graph. For $1 \leq k \leq n$, we have

$$p_{n,k} = p_{n-1,k}\beta^k + p_{n-1,k-1}(1-\beta^{k-1}), \tag{7.1}$$

where β^k is the probability that none of the k informed vertices will spread the information to the last vertex and $(1-\beta^{k-1})$ is the probability that at least one the $k-1$ vertices will spread the information to the last vertex.

Let us notice, that starting from vertex no. i in the n elements graph is equivalent to starting from vertex no. 1 in $n-i+1$ elements graph. Finally, the distribution of cascades sizes is given by:

$$s_{n,k} = \frac{\sum_{i=1}^n p_{i,k}}{n}, \tag{7.2}$$

since we are starting from a random vertex. The following theorem gives asymptotic values of $s_{n,k}$ for large n.

Theorem 7.2.1. $s_{n,k} \sim (n(1-\beta^k))^{-1}$.

Proof. Denoting $\widetilde{s}_{n,k} = ns_{n,k}$, we can reformulate the observation thesis. Namely, it is enough to prove, that
$lim_{n\to\infty}\widetilde{s}_{n,k} = (1-\beta^k)^{-1}$. We will prove the thesis by induction by k. Let us start with $k=1$.

$$\widetilde{s}_{n,1} = \sum_{i=1}^{n} p_{i,1} = p_{1,1} + \sum_{i=1}^{n-1} p_{i+1,1} = 1 + \beta s_{n-1,1},$$

Finally, we have: $\widetilde{s}_{n,1} = \frac{1-\beta^n}{1-\beta}$. For $k>1$, we have

$$\widetilde{s}_{n,k} = \sum_{i=1}^{n} p_{i,k} = \sum_{i=1}^{n} \beta^k p_{i-1,k} + \sum_{i=1}^{n}(1-\beta^{k-1}) \cdot p_{i-1,k-1},$$

so $\widetilde{s}_{n,k}$ fulfills the same pattern as $p_{n,k}$, namely:

$$\widetilde{s}_{n,k} = \beta^k\widetilde{s}_{n-1,k} + (1-\beta^{k-1})\widetilde{s}_{n-1,k-1} \tag{7.3}$$

Easy induction reasoning shows that $\widetilde{s}_{n,k}$ is monotonic in n and bounded, so $a_k = \lim_{n\to\infty}\widetilde{s}_{n,k}$ exists. Taking limit on both sides of the equation 7.3 we get:

$$a_k = \beta^k a_k + (1-\beta^{k-1})a_{k-1} = \frac{1-\beta^{k-1}}{1-\beta^k}a_{k-1}$$

Finally, we have

$$\lim_{n\to\infty} \widetilde{s}_{n,k} = \frac{1}{1-\beta^k}$$

$\square$

We have $s_{n,k} \approx (n(1-(1-\epsilon)^k))^{-1}$, taking the Taylor's expansion of the function $(1-(1-\epsilon)^k)^{-1}$, we get $s_{n,k} \approx n^{-1}((k\epsilon)^{-1} + O(1))$. Since $(k\epsilon)^{-1} \gg 1$, we have $s_{n,k} \approx (nk\epsilon)^{-1}$. For given large n we can choose such α, that for small enough k, $s_{n,k}$ will follow the power law distribution. Note that the identical theorem is satisfied for the $\overline{SIR}$ model and the proof is analogous to the proof for SIR model.

Bibliography

[1] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[3] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Trans. Comput. Theory*, 6(1):5:1–5:13, March 2014.

[4] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 793–801, 2015.

[5] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, May 2009.

[6] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 47–66, Philadelphia, PA, USA, 2017.

[7] Piotr Indyk. On approximate nearest neighbors in non-euclidean spaces. In *39th Annual Symposium on Foundations of Computer Science, FOCS*

'98, November 8-11, 1998, Palo Alto, California, USA, pages 148–155, 1998.

[8] Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, and Piotr Wygocki. Locality-sensitive hashing without false negatives for ℓ_p. In *COCOON*, volume 9797 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 2016.

[9] Piotr Sankowski and Piotr Wygocki. Approximate nearest neighbors search without false negatives for ℓ_2 for $c > \sqrt{\log\log n}$. In *ISAAC*, volume 92 of *LIPIcs*, pages 63:1–63:12, 2017.

[10] twitter.com. Twitter api. `https://dev.twitter.com/rest/public`, 2016.

[11] P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the mathematical institute of the hungarian academy of sciences*, pages 17–61, 1960.

[12] Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, and Piotr Wygocki. There is something beyond the twitter network. In *HT*, pages 279–284. ACM, 2016.

[13] Karol Wegrzycki, Piotr Sankowski, Andrzej Pacuk, and Piotr Wygocki. Why do cascade sizes follow a power-law? In *WWW*, pages 569–576. ACM, 2017.

[14] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing).* The MIT Press, 2006.

[15] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2):357–365, December 2005.

[16] Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Computational Geometry*, 39(1):24 – 29, 2008.

[17] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor

searching in fixed dimensions. Technical report, College Park, MD, USA, 1995.

[18] Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 599–608, New York, NY, USA, 1997. ACM.

[19] Kenneth L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, pages 160–164, New York, NY, USA, 1994. ACM.

[20] Rasmus Pagh. Locality-sensitive hashing without false negatives. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1–9, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[21] T. D. Ahle. Optimal las vegas locality sensitive data structures. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, volume 00, pages 938–949, Oct. 2018.

[22] Piotr Indyk. Uncertainty principles, extractors, and explicit embeddings of l2 into l1. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 615–620, New York, NY, USA, 2007. ACM.

[23] Rafał Latała and Krzysztof Oleszkiewicz. Small ball probability estimates in terms of width. *STUDIA MATHEMATICA*, 169(3):305–314, 2005.

[24] D Cordero-Erausquin, M Fradelizi, and B Maurey. The (b) conjecture for the gaussian measure of dilates of symmetric convex sets and related problems. *Journal of Functional Analysis*, 214(2):410 – 427, 2004.

[25] A Carbery and J Wright. Distributional and l-q norm inequalities for polynomials over convex bodies in r-n. *Mathematical Research Letters*, 8(3):233–248, 5 2001.

[26] Sergey G. Bobkov and Gennadiy P. Chistyakov. On concentration functions of random variables. *Journal of Theoretical Probability*, 28(3):976–988, 2015.

[27] Keith Ball. Cube slicing in r¡sup¿n¡/sup¿. *Proceedings of the American Mathematical Society*, 97(3):465–473, 1986.

[28] William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Proceedings of the conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[29] Mark Veraar. On khintchine inequalities with a weight. *Proceedings of the American Mathematical Society*, 138:4119–4121, 2010.

[30] Uffe Haagerup. The best constants in the khintchine inequality. *Studia Mathematica*, 70(3):231–283, 1981.

[31] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[32] S. Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics and Statistics*, pages 4(1):66–70, 2011.

[33] Loïc Grenié and Giuseppe Molteni. Inequalities for the beta function. *Math. Inequal. Appl.*, 18(4):1427–1442, 2015.

[34] D.J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(9):5766, 2002.

[35] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 721–730, New York, NY, USA, 2009. ACM.

[36] Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone's an influencer: Quantifying influence on twitter. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 65–74, New York, NY, USA, 2011. ACM.

[37] Biru Cui, Shanchieh Jay Yang, and Christopher Homan. Non-independent cascade formation: Temporal and spatial effects. In Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang, editors, *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1923–1926. ACM, 2014.

[38] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[39] Sanjeev Goyal and Michael Kearns. Competitive contagion in networks. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 759–774, New York, NY, USA, 2012. ACM.

[40] Everett M. Rogers. *Diffusion of innovations.* Free Press, 2010.

[41] N. Bayley. The mathematical theory of epidemics. *Griffin, London*, 1975.

[42] Easley David and Kleinberg Jon. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, New York, NY, USA, 2010.

[43] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S. Glance, and Matthew Hurst. Patterns of cascading behavior in large blog graphs. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*, pages 551–556. SIAM, 2007.

[44] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part III*, KES '08, pages 67–75, Berlin, Heidelberg, 2008. Springer-Verlag.

[45] Laurence Aitchison, Nicola Corradi, and Peter E. Latham. Zipf's law arises naturally when there are underlying, unobserved variables. *PLoS Computational Biology*, 12(12), 2016.

[46] Thierry Mora and William Bialek. Are biological systems poised at criticality? *Journal of Statistical Physics*, 144(2):268–302, Jul 2011.

[47] Greg Ver Steeg, Rumi Ghosh, and Kristina Lerman. What stops social epidemics? In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*. The AAAI Press, 2011.

[48] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Trans. Web*, 1(1), May 2007.

[49] Marc Barthelemy, Alain Barrat, and Alessandro Vespignani. The role of geography and traffic in the structure of complex networks. *Advances in Complex Systems*, 10(1):5–28, 2007.

[50] Duncan J. Watts. A simple model of global cascades on random networks. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 99, pages 5766–5771, April 30 2002. http://www.jstor.org/view/00278424/sp020038/02x3936j/0.

[51] K. Lerman and R. Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. In *Proceedings of 4th International Conference on Weblogs and Social Media (ICWSM)*, 2010.

[52] David R. Bild, Yue Liu, Robert P. Dick, Zhuoqing Morley Mao, and Dan S. Wallach. Aggregate characterization of user behavior in twitter and analysis of the retweet graph. *ACM Trans. Internet Techn.*, 15(1):4:1–4:24, 2015.

[53] Christina Brandt and Jure Leskovec. Status and friendship: mechanisms of social network evolution. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 229–230. ACM, 2014.

[54] Rumi Ghosh and Bernardo A. Huberman. Ultrametricity of information cascades. *CoRR*, abs/1310.2619, 2013.

[55] José Luis Iribarren and Esteban Moro. Branching dynamics of viral information spreading. *CoRR*, abs/1110.1884, 2011.

[56] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, March 2010.

[57] Shlomo Havlin, N. A. M. Araujo, Sergey V. Buldyrev, C. S. Dias, Roni Parshani, G. Paul, and H. Eugene Stanley. Catastrophic cascade of failures in interdependent networks. *CoRR*, abs/1012.0206, 2010.

[58] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, November 2009.

[59] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[60] Andrzej Pacuk, Karol Wegrzycki, and Piotr Wygocki. Retweet graph. `http://social-networks.mimuw.edu.pl/data/`, 2016.

[61] Robert West, Ryen W. White, and Eric Horvitz. Here and there: goals, activities, and predictions about location from geotagged queries. In Gareth J. F. Jones, Paraic Sheridan, Diane Kelly, Maarten de Rijke, and Tetsuya Sakai, editors, *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 817–820. ACM, 2013.