Complex Queries and Complex Data: Challenges in Similarity Search

Johannes Niedermayer

München 2015

Complex Queries and Complex Data: Challenges in Similarity Search

Johannes Niedermayer

Dissertation
an der
Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Johannes Niedermayer
aus München

München, den 21.07.2015

Erstgutachter: PD Dr. Peer Kröger
Zweitgutachter: Prof. Dr. Michael Gertz
Tag der mündlichen Prüfung: 30.10.2015

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Name, Vorname

....

Ort, Datum Unterschrift Doktorand/in

Formular 3.2

To my future wife.
To my parents.
To my brother.

Contents

List of Figures

List of Tables

Table of Notations

General Notations

Symbol	Description
$\mathbb{M}$	A metric space
$\mathbb{R}^d$	The d-dimensional vector space
$\mathbb{B}^d$	The space of d-dimensional binary vectors
D	A database of objects
q	A query object
$o \in D$	A database object
$dist(x, y)$	A distance function, usually a metric
$\epsilon - range(q, D, \epsilon)$	ϵ-range query
$kNN(q, D, k)$	kNN query
$RkNN(q, D, k)$	RkNN query
$MINDIST(A, B)$	Minimum distance between objects A and B
$MAXDIST(A, B)$	Maximum distance between object A and B
$Dom(A, B, C)$	Spatial domination function of the objects A, B, and C
$\mathcal{H}$	A family of hash functions
$\mathcal{G}$	A function family
Q	A queue

Part II: The RkNN Join

Symbol	Description
R	A set of objects, if not denoted otherwise $R \subset \mathbb{R}^d$. Corresponds to the left (*query*) set of the join.
S	A set of objects, usually $S \subset \mathbb{R}^d$. Corresponds to the right (*database*) set of the join.
r	Element from R, $r \in R$
s	Element from S, $s \in S$
$\mathcal{R}, \mathcal{S}$	Index of set R and S, respectively

Part III: Nearest Neighbor Queries on Uncertain Spatio-Temporal Data

Symbol	Description
$\mathbb{T}$	A time domain $\mathbb{T} = \{0, \ldots, n\}$
T	A time interval
$t \in \mathbb{T}$	A single point in time
$\mathbb{S}$	A spatial domain
$s_i \in \mathbb{S}$	A spatial location, i.e. a state
$\theta_i \in \mathbb{S}$	A spatial observation
Θ^o	The representation of object o, i.e. a set of location-time tuples, w.l.o.g. ordered by the timesteps t_i $\Theta^o = \{\langle t_1^o, \theta_1^o\rangle, \langle t_2^o, \theta_2^o\rangle, \ldots, \langle t_{\lvert\Theta^o\rvert}^o, \theta_{\lvert\Theta^o\rvert}^o\rangle\}$
$o(t)$	The realization of the random variable representing object o at time t
$M^o(t)$	The transition matrix of object o at time t
$\vec{s^o}(t)$	The probability distribution over spatial locations of object o at time t
τ	Probability threshold
l	A literal ($\exists$ query)
c	A clause ($\exists$ query)
x	A variable ($\exists$ query)
$F(t), R(t)$	Forward and backward model for sampling

Part IV: kNN Queries for Image Retrieval

Symbol	Description
I_j	An image $I_j \in D$
$p_j^i \in I_j$	A keypoint, $p_j^i = (v_j^i, x_j^i, y_j^i, s_j^i, r_j^i, \sigma_j^i, A_j^i)$
v_j^i	Feature vector of feature i in image j
x_j^i, y_j^i	Interest point location of feature i in image j
s_j^i	Scale of feature i in image j
σ_j^i	Response of feature i in image j
A_j^i	Affine matrix of feature i in image j
$m(x)$	An arbitrary matching function
$\Psi \subseteq I_j$	A subset of image features
M	A set of matches
n	Bound on the number of matching queries
Q	A set of query objects
$\delta_{xy}, \delta_s, \delta_{d_v}, \delta_\alpha, \delta_r, \delta_{d_{x,y}}$	Thresholds for match expansion

Summary

With the widespread availability of wearable computers, equipped with sensors such as GPS or cameras, and with the ubiquitous presence of microblogging platforms, social media sites and digital marketplaces, data can be collected and shared on a massive scale. A necessary building block for taking advantage from this vast amount of information are efficient and effective similarity search algorithms that are able to find objects in a database which are similar to a query object. Due to the general applicability of similarity search over different data types and applications, the formalization of this concept and the development of strategies for evaluating similarity queries has evolved to an important field of research in the database community, spatio-temporal database community, and others, such as information retrieval and computer vision. This thesis concentrates on a special instance of similarity queries, namely k-Nearest Neighbor (kNN) Queries and their close relative, Reverse k-Nearest Neighbor (RkNN) Queries.

As a first contribution we provide an in-depth analysis of the RkNN join. While the problem of reverse nearest neighbor queries has received a vast amount of research interest, the problem of performing such queries in a bulk has not seen an in-depth analysis so far. We first formalize the RkNN join, identifying its monochromatic and bichromatic versions and their self-join variants. After pinpointing the monochromatic RkNN join as an important and interesting instance, we develop solutions for this class, including a self-pruning and a mutual pruning algorithm. We then evaluate these algorithms extensively on a variety of synthetic and real datasets.

From this starting point of similarity queries on certain data we shift our focus to uncertain data, addressing nearest neighbor queries in uncertain spatio-temporal databases. Starting from the traditional definition of nearest neighbor queries and a data model for uncertain spatio-temporal data, we develop efficient query mechanisms that consider temporal dependencies during query evaluation. We define intuitive query semantics, aiming not only at returning the objects closest to the query but also their probability of being a nearest neighbor. After theoretically evaluating these query

predicates we develop efficient querying algorithms for the proposed query predicates. Given the findings of this research on nearest neighbor queries, we extend these results to reverse nearest neighbor queries.

Finally we address the problem of querying large datasets containing set-based objects, namely image databases, where images are represented by (multi-)sets of vectors and additional metadata describing the position of features in the image. We aim at reducing the number of kNN queries performed during query processing and evaluate a modified pipeline that aims at optimizing the query accuracy at a small number of kNN queries. Additionally, as feature representations in object recognition are moving more and more from the real-valued domain to the binary domain, we evaluate efficient indexing techniques for binary feature vectors.

Zusammenfassung

Nicht nur durch die Verbreitung von tragbaren Computern, die mit einer Vielzahl von Sensoren wie GPS oder Kameras ausgestattet sind, sondern auch durch die breite Nutzung von Microblogging-Plattformen, Social-Media Websites und digitale Marktplätze wie Amazon und Ebay wird durch die User eine gigantische Menge an Daten veröffentlicht. Um aus diesen Daten einen Mehrwert erzeugen zu können bedarf es effizienter und effektiver Algorithmen zur Ähnlichkeitssuche, die zu einem gegebenen Anfrageobjekt ähnliche Objekte in einer Datenbank identifiziert. Durch die Allgemeinheit dieses Konzeptes der Ähnlichkeit über unterschiedliche Datentypen und Anwendungen hinweg hat sich die Ähnlichkeitssuche zu einem wichtigen Forschungsfeld, nicht nur im Datenbankumfeld oder im Bereich raum-zeitlicher Datenbanken, sondern auch in anderen Forschungsgebieten wie dem Information Retrieval oder dem Maschinellen Sehen entwickelt. In der vorliegenden Arbeit beschäftigen wir uns mit einem speziellen Anfrageprädikat im Bereich der Ähnlichkeitsanfragen, mit k-nächste Nachbarn (kNN) Anfragen und ihrem Verwandten, den Revers k-nächsten Nachbarn (RkNN) Anfragen.

In einem ersten Beitrag analysieren wir den RkNN Join. Obwohl das Problem von reverse nächsten Nachbar Anfragen in den letzten Jahren eine breite Aufmerksamkeit in der Forschungsgemeinschaft erfahren hat, wurde das Problem eine Menge von RkNN Anfragen gleichzeitig auszuführen nicht ausreichend analysiert. Aus diesem Grund formalisieren wir das Problem des RkNN Joins mit seinen monochromatischen und bichromatischen Varianten. Wir identifizieren den monochromatischen RkNN Join als einen wichtigen und interessanten Fall und entwickeln entsprechende Anfragealgorithmen. In einer detaillierten Evaluation vergleichen wir die ausgearbeiteten Verfahren auf einer Vielzahl von synthetischen und realen Datensätzen.

Nach diesem Kapitel über Ähnlichkeitssuche auf sicheren Daten konzentrieren wir uns auf unsichere Daten, speziell im Bereich raum-zeitlicher Datenbanken. Ausgehend von der traditionellen Definition von Nachbarschaftsanfragen und einem Datenmodell für unsichere raum-zeitliche Daten entwickeln wir effiziente Anfrageverfahren, die zeitliche Abhängigkeiten bei der Anfrage-

bearbeitung beachten. Zu diesem Zweck definieren wir Anfrageprädikate die nicht nur die Objekte zurückzugeben, die dem Anfrageobjekt am nächsten sind, sondern auch die Wahrscheinlichkeit mit der sie ein nächster Nachbar sind. Wir evaluieren die definierten Anfrageprädikate theoretisch und entwickeln effiziente Anfragestrategien, die eine Anfragebearbeitung zu vertretbaren Laufzeiten gewährleisten. Ausgehend von den Ergebnissen für Nachbarschaftsanfragen erweitern wir unsere Ergebnisse auf Reverse Nachbarschaftsanfragen.

Zuletzt behandeln wir das Problem der Anfragebearbeitung bei Mengenbasierten Objekten, die zum Beispiel in Bilddatenbanken Verwendung finden: Oft werden Bilder durch eine Menge von Merkmalsvektoren und zusätzliche Metadaten (zum Beispiel die Position der Merkmale im Bild) dargestellt. Wir evaluieren eine modifizierte Pipeline, die darauf abzielt, die Anfragegenauigkeit bei einer kleinen Anzahl an kNN-Anfragen zu maximieren. Da reellwertige Merkmalsvektoren im Bereich der Objekterkennung immer öfter durch Bitvektoren ersetzt werden, die sich durch einen geringeren Speicherplatzbedarf und höhere Laufzeiteffizienz auszeichnen, evaluieren wir außerdem Indexierungsverfahren für Binärvektoren.

Part I

Preliminaries

Chapter 1

Introduction

With the widespread availability of wearable computers, not only smartphones, but also smart watches[1] and Google Glass[2], equipped with sensors such as GPS, accelerometers, gyroscopes and cameras, data can be collected on a massive scale and shared on the internet such as with *Flickr*, *Instagram*, and *YouTube* for visual data[3] or *Endomondo*[4] for spatio-temporal data. In addition to this ubiquitous presence of sensor devices in our everyday life, micro-blogging platforms such as Twitter, social media sites including Facebook and digital marketplaces such as Amazon allow users to publish news, opinions and reviews without having in-depth technical knowledge.[5] In the future, the general spread of the *internet of things* will give rise to an even bigger flood of data that have to be stored, queried and mined. While in the year 2013 20 billion *things* were connected to the internet, this number is estimated to reach 32 billion in 2020 [1].

A necessary building block for taking advantage from this vast amount of information available on the web, for example for recommendation purposes or data mining applications, is an efficient and effective search framework. Many approaches of extracting useful information from large amounts of data, in the area of data mining (e.g. clustering, outlier detection, and classification) but also in the context of search engines, take advantage from a notion of similarity between different data instances. Similarity is a broad concept of describing the relation between object instances. In the context of spatial data, similarity can be based on the spatial proximity of objects, assessing close objects as more similar than objects further away from another.

[1] http://www.apple.com/watch/
[2] http://www.google.com/glass/
[3] http://www.flickr.com, http://www.instagram.com, http://www.youtube.com
[4] http://www.endomondo.com
[5] http://www.twitter.com, http://www.facebook.com, http://www.amazon.com

In the context of spatio-temporal data, similarity could be measured by comparing the spatial proximity of two trajectories over time. The similarity of textual data can be assessed by comparing the words contained in different document and clues for visual similarity can be derived from color, gradient or texture patterns. Due to this widespread applicability of similarity, the formalization of this concept and the development of strategies for evaluating similarity queries has not only evolved to an important field of research in the database community and spatio-temporal database community, but also in a wide range of other communities, e.g. in the area of information retrieval and computer vision.

Before diving deeper into the contents of this thesis, this first part provides an overview of fundamental principles essential for its understanding. Chapter 2 aims at defining the concept of similarity search, summarizes relevant query predicates and provides an overview of general processing strategies relevant in the context of this thesis, such as feature extraction and indexing techniques. It further provides an overview over application areas of similarity queries and current challenges in this field of research. In Chapter 3, an overview of the structure of this thesis is given, relating its scientific contributions to the challenges mentioned in Chapter 2. Finally, Chapter 4 outlines the papers published in the context of this thesis and summarizes the contributions of the author.

Chapter 2

Similarity Search

2.1 Mathematical Definitions

Despite its wide range of applications in query processing and data mining, the idea of similarity can be boiled down to only a few mathematical concepts. We measure the similarity of two objects with a metric [2]:

Definition 1 (Metric)**.** *Let* $\mathbb{M}$ *be a set. A function* $dist : \mathbb{M} \times \mathbb{M} \to \mathbb{R}$ *mapping pairs of objects from* $\mathbb{M}$ *to a real value, is called a metric on* $\mathbb{M}$ *if, for all* $x, y, z \in \mathbb{M}$ *the following holds:*

1. *Non-negativity:* $dist(x, y) \geq 0$
2. *Separation:* $dist(x, y) = 0$ *if and only if* $x = y$
3. *Symmetry:* $dist(x, y) = dist(y, x)$
4. *Triangle Inequality:* $dist(x, y) \leq dist(x, z) + dist(z, y)$

Given this concept of a metric, data is interpreted as a set of elements from a metric space [2]:

Definition 2 (Metric Space)**.** *A metric space* $(\mathbb{M}, dist)$ *is a set* $\mathbb{M}$ *equipped with a metric* $dist$*.*

A distance is referred to as a superset of metrics [2], providing only the properties non-negativity, symmetry and $dist(x, x) = 0$. We will however use the general term distance function and distance synonymously, as we will only consider metric spaces in this thesis.

In computer science, a special and important instance of metric spaces is $(\mathbb{R}^d, ||x-y||_p)$, $d \in \mathbb{N}$ over the real-valued d-dimensional vector space $\mathbb{R}^d$ and the function $||x-y||_p$, $x, y \in \mathbb{R}^d, p \in \mathbb{N}$ derived from the l_p-norm[2]

$$||x||_p := \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}}$$

Relevant materializations of such norms include the Manhattan norm $||x||_1$, the max-norm $||x||_\infty = \lim_{p \to \infty} \sqrt[p]{\sum_{i=1}^{d} |x_i|^p}$, and the Euclidean norm $||x||_2$, which is an intuitive basis for calculating distances in our real world. The metrics derived from these norms can be computed efficiently, and – especially in the case of low-dimensional spaces – efficient indexing techniques can be employed to speed up query evaluation. As a consequence, feature extraction techniques are usually designed to generate real-valued features, as e.g. in the area of image retrieval [3]. Another interesting metric space is the domain of binary vectors $\mathbb{B}^d = \{0,1\}^d$ in combination with the Hamming distance $dist_H(x,y) = popcnt(x \oplus y)$ (see Part IV) where $\oplus$ denotes the XOR operation and $popcnt(x)$ returns the number of ones in a bit-string; it can be computed even faster than the Euclidean distance. While there exist definitions for the real vector space as well, in the special case of binary strings, the Hamming distance coincides with the l_1-distance [2].

2.2 Query Types

Given these mathematical means of calculating the similarity between objects, by exploiting the distance function *dist*, similarity queries can be defined. The classic queries include ε-range and k-nearest neighbor (kNN) queries, however there have also more complex query predicates been defined, such as reverse k-nearest neighbor (RkNN) and skyline queries. Let us first consider one of the simplest similarity queries, the ϵ-range query.

Definition 3 (ε-range query)**.** *Given a database D and a query object q , an ε-range query returns all objects $o \in D$ having a distance of at most ε to q:*

$$\varepsilon - range(q, D, \varepsilon) = \{o \in D | dist(o,q) \leq \varepsilon\}$$

An example of ε-range queries in a spatial context would be the query "Give me all supermarkets within a radius of 500 meters around my current position". The German real estate intermediate ImmoScout[1] allows users to

[1] https://www.immobilienscout24.de

search for the apartments closest to a predefined location. In other scenarios however, the limitations of this simple query type hinder its applicability, as the range ε has to be known in advance. Otherwise, chances are high that an ε-range query returns either an empty result set if the range is chosen too small, or a very large result set if the range is chosen too large. Still, range queries are an important building block of data mining applications such as DBSCAN[4], which uses a simple heuristic to determine ε. This heuristic is based on k-nearest neighbor (kNN) queries:

Definition 4 (k-nearest neighbor (kNN) query)**.** *Given a database D and a query object q, a k-nearest neighbor query returns the k objects $o \in D$ with smallest distance $dist(o, q)$ to q:*

$$kNN(q, D, k) = \{o \in D || \{o' \in D | dist(o', q) < dist(o, q)\}| < k\}$$

In a spatial context, such a query (in this case $k = 1$) might be similar to the following: "Where is the supermarket closest to my current position?". Google provides similar features with Google Maps[2]: By searching "Bookstore near University, Munich", a list of bookstores closest to the university is provided to the user. The concept of nearest neighbors is also a building block for data mining algorithms, e.g. classification (kNN classifiers [5]), clustering (k-means [5]) or outlier detection (e.g. LOF [6]). In kNN classification, an object is labeled by predicting its class based on the k-nearest neighbors of the query in training set. In k-means clustering, the assignment of an object to its corresponding cluster is based on a 1NN query over the set of cluster means. For LOF, the neighborship of a feature is evaluated to compute an outlier score of the reference vector.

The fundamentals of kNN queries have been considered, for example, in [7, 8, 9]; recent research focuses on solutions for specialized applications, such as in the context of image databases [10, 11, 12] and uncertain data [13, 14]. Another query predicate derived from the kNN query is the reverse k-nearest neighbor (RkNN) query [15], which is especially of interest in the area of data mining and spatial query processing. There exist two instances of RkNN queries, the monochromatic and bichromatic version. While the monochromatic version is more relevant in the context of this thesis (and we will refer to the monochromatic case as RkNN query), we will introduce both of them for the sake of completeness:

Definition 5 (Monochromatic reverse k-nearest neighbor (MRkNN) query)**.** *Given a database D and a query object q, a reverse k-nearest neighbor query*

[2] https://www.google.de/maps/

returns the objects $o \in D$ having q as one of their k-nearest neighbors:

$$RkNN(q, D, k) = \{o \in D | q \in kNN(o, D \setminus \{o\} \cup \{q\}, k)\}$$

In the slightly different bichromatic RkNN query, two sets R (corresponding to the query set) and S (corresponding to the database set $S = D$) are given. The goal of the bichromatic RkNN query is then to compute for a query point $q \in R$ all points from S for which q is one of the k closest points from R[16]:

Definition 6 (Bichromatic reverse k-nearest neighbor (BRkNN) query)**.** *Given two sets R and S and a query object $q \in R$, a bichromatic reverse k-nearest neighbor query returns the objects $o \in S$ having q as one of their k-nearest neighbors from R:*

$$BRkNN(q, R, S, k) = \{s \in S | q \in kNN(s, R, k)\}$$

These two variants of th RkNN query, the monochromatic and the bichromatic case, vary in the data set on which the kNN query is posed. For a monochromatic RkNN query, the kNN query is run on D (i.e. on the database set), while for the bichromatic case it is run on R (i.e. the set of objects the query is taken from).

The latter two query predicates, the kNN query and the RkNN query will be the focus of this thesis. In addition to these common query types there clearly exist other important query predicates in the context of similarity queries, such as the skyline query [17] or spatial skyline query [18], which will not be considered in this thesis and shall therefore be omitted.

2.3 Pipeline

As processing similarity queries can be boiled down to simple mathematical concepts, the query processing pipeline is –despite some data-specific adaptions especially for complex data types– largely independent of the given data. An overview over a standard query processing pipeline is provided in Figure 2.1. During a first preprocessing step, feature extraction (see Section 2.3.1), object properties relevant for assessing the similarity of objects are extracted from each entry stored in the database. These features can be simple real-valued feature vectors or more complex objects such as polygons or sets of vectors. In a second preprocessing step, this feature representation is further processed in order to enable efficient query processing (see Section 2.3.2). This can be achieved for example by indexing the features or by computing approximations such as minimum bounding axis-parallel

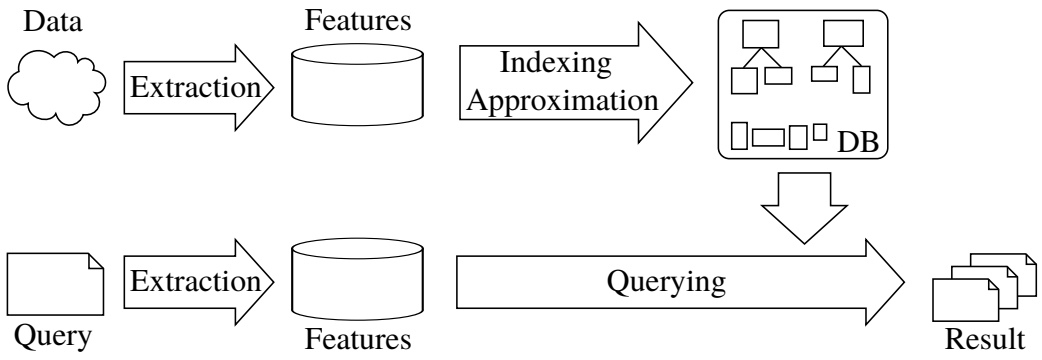


Figure 2.1: Query Processing Pipeline

rectangles (MBRs) for polygons or sets of vectors. The features and the corresponding data structures are then stored in a database. During query processing, a feature representation is extracted from the query object and the precomputed data structures in the database are used to efficiently compute the query result. If necessary, approximations of the query object are created during query processing to enable fast comparisons with approximations stored in the database. The steps of this query processing pipeline will be discussed in the following.

2.3.1 Feature Extraction

Feature extraction describes the process of extracting relevant information from collected data (such as images, texts or trajectories), in order to make further efficient automatic processing of the data possible. As the term *extraction* states, this process is usually lossy and discards information not relevant for query processing and mining. As a result, there is often a trade-off between descriptiveness of the extracted features, and the efficiency of the consecutive query processing. Closely entangled to selecting a given feature representation is the choice of a distance function where efficiency and effectiveness have to be considered as well. As a result, the choice of a feature extraction technique in combination with the corresponding distance measure defines the metric space $(\mathbb{M}, dist)$. If possible, the metric space $(\mathbb{R}^d, ||x - y||_p)$ is preferred to general metric data, as index structures and distance calculations for vectorial data are often more efficient than for general metric data, especially on low-dimensional data.

In the context of *spatial* and *spatio-temporal* query processing, similarity search can often be boiled down to proximity search where the spatial distance between objects provides a measure of similarity. The most straight-

forward solution would be to consider the position of static or moving objects as features from $\mathbb{R}^2$ and an l_p norm metric, resulting in $(\mathbb{R}^2, ||x-y||_p)$. While this disregards the spherical nature of earth, it is still a valid choice when distortions can be neglected due to the small spatial distribution of a dataset. A better approximation of earth distances comes from spherical geometry, where the underlying domain is $\mathbb{G} \subset \mathbb{R}^2$, consisting of tuples (θ, ϕ) denoting longitude and latitude with $\forall(\theta, \phi) \in \mathbb{G} : \theta \in [-\pi, \pi], \phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, where the great circle distance [2] can be used to measure similarity. Metric space representations of spatial data in form of a graph are common as well. In graphs, nodes can for example represent street crossings and edges between nodes street segments; the weight of an edge might represent arbitrary quality criteria such as the spatial distance between nodes or their travel time. Given this graph-based representation, the weighted path metric [2], i.e. the length of the shortest path between two objects, can be used as a distance measure. Note however that the metric property is not always fulfilled and depends on the underlying graph.

While the mathematical representation of objects in the context of geospatial data has an intuitive interpretation in the real world, feature representations for similarity search in general are often more abstract. *Textual data* can be searched using the vector space model (see e.g. [19]). Given a dictionary of words present in the database, a vector representation of a text file consist of a (sparse) vector counting (and probably weighting) the occurrences of each word contained in the dictionary in the corresponding text file. As a distance function, the cosine distance $dist_C = 1 - \frac{<x,y>}{||x||_2 * ||y||_2}$ (with $< x, y >$ denoting the dot product) can be used [2], which is however not a metric, as for a given vector x there exist many possible vectors αy with equivalent cosine distance to x, but different distance to the origin. Textual similarity search is ubiquitous on the internet: a well-known publicly available search engine for textual data is Lucene[3]; in October 2010 Twitter announced that their real-time search was moved to Lucene as well [20].

A simple means of assessing the similarity of *images* could be achieved by extracting a histogram of the color distribution of an image, resulting for example in peaks for blueish and greenish colors for nature photography. Another type of features for image data are histograms over gradient orientations within the image[21] or textural features such as Haralick features [22]. More complex representations are based on local features such as SIFT[3] and the Bag of Visual Words (BoVW) [23] approach. A variety of search engines on the web such as TinEye are devoted to similarity search on images. By generating digital signatures from images stored on the web and searching

[3] http://lucene.apache.org/

for (partial) signatures in their databases, it allows finding scaled, edited or cropped images [24].

The most straightforward way of comparing the similarity of *time series* would be interpreting both time series as vectors and computing the Euclidean distance between them, possibly handling different lengths of time series in a suitable way. More sophisticated distance functions such as Dynamic Time Warping (DTW) which is however not a metric [25] and the Levenstein metric [2] aim at comparing possibly unaligned or temporally stretched time series.

2.3.2 Indexing and Query Processing

A trivial solution to solving the queries defined earlier is a linear scan. In the context of ε-range queries, for each object o the distance to the query has to be computed. Based on this distance, o can be accepted as a query result, or it can be discarded, resulting in a complexity of $O(|D|)$. The trivial solution for kNN queries is to sort all objects in the database by their distance to q and return the first k objects from this list, resulting in a complexity of $O(|D| \log(|D|))$ for the trivial solution. For the RkNN query, the trivial solution of the kNN query can be extended; for each $o \in D$, a kNN query could be processed on $D \setminus \{o\} \cup \{q\}$, if $q \in kNN(o, D \setminus \{o\} \cup \{q\}, k)$, o would be returned as a result. In this case, the runtime complexity would be $O(|D|^2 \log(|D|))$. Unfortunately, algorithms with such a complexity can not handle large datasets, such that a variety of techniques has been developed in the past aiming at increasing the computational efficiency of evaluating the query predicates mentioned earlier. Two of these, and most likely the most important ones, are the use of *index structures* and the *filter-refinement* paradigm [26].

2.3.2.1 Index Structures

To speed up query processing, index structures provide a re-ordering of the data that allow to identify candidates satisfying a given query predicate without scanning all entries of the database. As this thesis concentrates on vectorial data, multidimensional index structures are most important for this work. Two important types of index structures used within the context of this thesis are tree-structured indices (such as R-trees [27]) and hash-based index structures (such as LSH [7]) which will be introduced in the following.

Tree-structured Indices Tree-structured indices divide the dataset recursively in smaller (usually distinct) subsets of objects. In every level of the

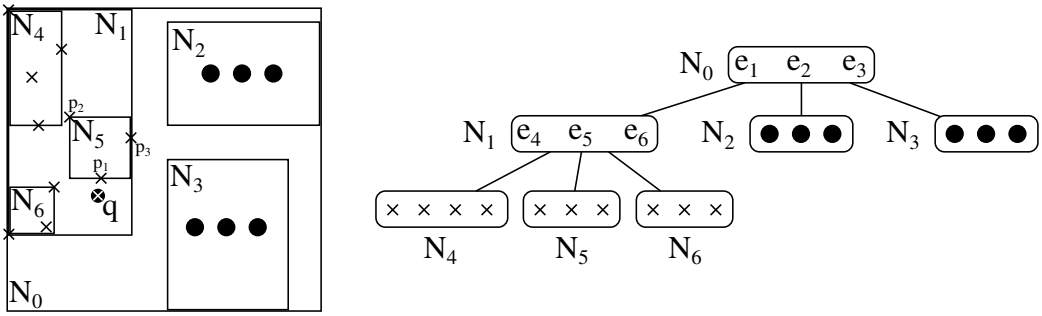


Figure 2.2: The R-Tree

tree, each of these subsets is further split, resulting in a tree-like structure. An index structure supports the efficient evaluation of a query predicate if the predicate can be checked already in intermediate nodes of the tree. If, based on an intermediate approximation, a query predicate cannot be fulfilled for any data contained in the intermediate nodes' subtree, independent of the data's actual value, the whole subtree can be pruned without further investigation, pruning large parts of the search space.

An in the context of this thesis important instance of tree-structured indices is the R-tree [27] (and its variants such as the R*-tree [28]), which will be used in Part II and Part III of this thesis. R-trees are balanced trees specifically designed for handling multidimensional vectorial data. While the R-tree has been developed for indexing complex objects [27], it is also often considered useful for indexing simple vectors [28]. An exemplary R-tree indexing two-dimensional points is shown in Figure 2.2 where the subtrees rooted in node N_2 and N_3 have been cut off for the sake of simplicity. In R-trees, sets of points in d-dimensional space are summarized by Minimally Bounding axis-parallel Rectangles (MBRs). An MBR A is described by its minimum and maximum coordinates A^{min} and A^{max} with $\underset{i=1..d}{\forall} A_i^{min} \leq A_i^{max}$. The tree consists of a set of nodes corresponding to pages in secondary memory if the index is disk resident. A leaf node (nodes N_4, N_5, N_6 in the figure) consists of a set of tuples (o, MBR) with o denoting data objects [28], or of a set of tuples (o_{ref}, MBR) pointing to data objects [27, 28]. Intermediate nodes (N_1, N_2, N_3) contain entries $({}^*N, MBR)$ pointing to leaf nodes, with MBRs minimally bounding the points of the leaf node the MBR is referencing. In higher levels of the index, intermediate nodes contain entries whose MBRs are minimally bounding the set of MBRs of the corresponding child nodes, such as the root node N_0 in Figure 2.2. R-trees are generated (and updated) in a way that during a query whole subtrees can be pruned

at an early stage of tree traversal, reducing the computational complexity of a query. For this reason, Guttman proposed to minimize the total area of entries within a node during insertion and split [27]. Improved optimization criteria have been proposed later for the R*-tree [28].

Given such an index based on spatial approximations, efficient similarity search can be achieved using the concept of distance bounds, helping to decide which pages have to be resolved during query evaluation. For example, an ε-range query around a query point q does not have to consider MBRs where any possible point $p \in MBR$ is further away from the query than ε. This distance bound is called the $MINDIST$ [9]. It was initially defined for a d-dimensional query point q and an MBR A for the Euclidean distance:

$$MINDIST(q, A) = \sqrt{\sum_{i=1}^{d} |q_i - t_i|^2} \text{ with } t_i = \begin{cases} A_i^{min} & \text{if } q_i < A_i^{min} \\ A_i^{max} & \text{if } q_i > A_i^{max} \\ q_i & \text{else} \end{cases}$$

It can be further extended to the case of two MBRs and general l_p distances[29]:

$$MINDIST(A, B) = \sqrt[p]{\sum_{i=1}^{d} \begin{cases} |A_i^{min} - B_i^{max}|^p & \text{if } A_i^{min} > B_i^{max} \\ |B_i^{min} - A_i^{max}|^p & \text{if } B_i^{min} > A_i^{max} \\ 0 & \text{else} \end{cases}}$$

On the other hand, given a large ε, pages bounded by an MBR where *any* point lies within the range ε of the query can be accepted completely without further consideration; this concept is described by the $MAXDIST$, which is however less often used as the pruning power of the $MINDIST$ is much higher than the probability of accepting a match based on the $MAXDIST$. The $MINDIST$ and $MAXDIST$ can however be used in combination to check for spatial domination, e.g. in the context of RkNN queries [29], see Part II and Part III. Let $X_k^{mid} = \frac{X_k^{min} + X_k^{max}}{2}$, then [29]:

$$MAXDIST(A, B) = \sqrt[p]{\sum_{i=1}^{d} \begin{cases} |A_i^{max} - B_i^{min}|^p & \text{if } A_i^{mid} \geq B_i^{mid} \\ |B_i^{max} - A_i^{min}|^p & \text{if } B_i^{mid} > A_i^{mid} \end{cases}}$$

Many algorithms for query processing on vectors stored in R-trees are based on these distance approximations. One of them is the Hjaltason-Samet algorithm for kNN queries [8], which we will introduce briefly based on the example R-tree in Figure 2.2. The Hjaltason-Samet algorithm is based on a priority queue Q containing entries or data objects ordered by their MINDIST to the query q, aiming at resolving pages first that have a higher probability

of containing a kNN query result. First, the root entry r (an entry representing the root node N_0) is inserted into the queue, $Q = [r]$. Then, the first element (i.e. r) is taken from the queue, the corresponding node N_0 is resolved and the entries it contains (e_1, e_2, e_3, see Figure 2.2 right) are inserted into the queue by increasing $MINDIST$ to q, resulting in $Q = [e_1, e_3, e_2]$. During the next step, as e_1 is closest to the query, the node N_1 corresponding to this entry is resolved and its children added to the queue, resulting in $Q = [e_5, e_6, e_3, e_4, e_2]$. The next step, resolving e_5, leads to the queue $Q = [p_1, e_6, p_3, e_3, e_4, p_2, e_2]$. Finally the first nearest neighbor, p_1 can be returned. This procedure has the advantage that k does not even have to be known in advance, as the algorithm's state (i.e. the priority queue) can be stored and reused later to return the second nearest neighbor and so on. Therefore, the algorithm is not only able to return kNN results, but also to be used for nearest neighbor rankings.

Hash-based Indices A technique for speeding up query processing conceptually different to the tree-based index structures discussed earlier is hash-based indexing which will be a building block in Part IV of this thesis. While there exist hash-based index structures for most likely any data type, as this thesis concentrates on vectorial data, this paragraph will also concentrate on hashing techniques on such data. In vector spaces the concept of Locality Sensitive Hashing (LSH), initially proposed by Indyk and Motwani [7] and improved By Gionis et al. in [30], has turned out to be a good solution over the years. LSH generates hashes from the data such that, given a specific distance function, points closer to an arbitrary reference point have a higher probability of being assigned the same hash as the reference than points further away. Feature vectors are inserted into a set of hash tables based on a set of such locality-sensitive hash functions. During query processing, candidates with hash values equivalent to the query are retrieved from the tables. Concatenating the resulting candidates of all tables leads to the candidate set. During refinement, the actual distance between the query and each candidate is computed and the query predicate is evaluated on this reduced set, speeding up the evaluation. There exists a variety of hash functions for different data types, including real-valued data under l_p norms [31] and the binary space $\mathbb{B}^d$ [7]. However, as there is no guarantee that two objects being close to each other generate the same hash, results provided by LSH are usually approximate. The quality of approximation can be increased by increasing the number of hash tables or by probing not only the hash cell of the query object, but also neighboring cells [32].

2.3.2.2 Filter-Refinement

Another technique for speeding up the evaluation of similarity queries, basically generalizing the index structures defined above, is filter-refinement (sometimes also called multi-step) query processing [33]. Filter-refinement is based on a cascade of filters with increasing complexity, where each filter reduces the size of the candidate set, pruning *true drops* and possibly accepting *true hits*. In the beginning, all objects stored in the database are considered as candidates. As the first filters have to evaluate a large number of candidate objects, filters are ordered by increasing complexity; the first filters are usually rather cheap, as especially the first filter can be sped up using conventional indexing techniques [33]. Later in the filter cascade, when the number of candidates has already been reduced by prior filter steps, filter complexity increases in order to prune or accept more candidate objects. A visualization of the filter-refinement pipeline is shown in Figure 2.3. First, the whole database is filtered by an initial filter f_1, identifying and removing objects from the candidate list that do definitely not satisfy the query predicate, i.e. true drops ($\not{\checkmark}_1$). On the other hand it is sometimes possible to accept some objects as part of the result without further investigating them by the remaining filters, leading to a set of true positives ($\checkmark_1$); these true positives are added to the result without further refinement. The remaining database objects that were neither accepted nor discarded ($?_1$) are fed into the next filter. This process continues for all remaining filters. Finally, the remaining candidates $?_n$ have to be refined by testing the actual (expensive) query predicate, however the number of objects in $?_n$ is usually much smaller than the number of objects stored in the database, significantly speeding up query evaluation.

A variety of research on this pipeline has been published in the past, including [34, 35, 36]. Furthermore this pipeline will be utilized in Part III of this thesis: In the context of uncertain spatial and spatio-temporal databases, the position of an object can be described by a set of points that describe the possible positions (with non-zero probability) of an object at a given point in time. Each of these points has a probability assigned, describing the probability distribution of the object's position. Evaluating a query predicate on these probability distributions can be very expensive, however the filter-refinement technique makes it possible to prune large parts of the search space without even considering probabilities. Let us consider for example the case of ϵ-range queries. A filter would disregard probability information and simply bound the position of an object by an MBR. Pruning of true drops could then be achieved by employing the $MINDIST$ introduced earlier in this section and an R-tree could be used to further speed up this first filter

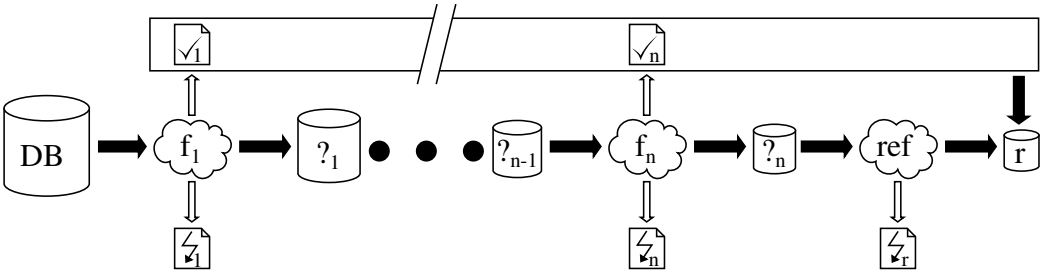


Figure 2.3: The Filter-Refinement approach

step. In a second filter step, the actual positions or tighter approximations could be considered, however without involving probabilities, yet. During refinement the result probabilities are computed.

2.4 Challenges

Over the last decades, a considerable amount of research has been pushed into solving and speeding up similarity queries. However, due to the increasing amount of data stored, due to the development of new hardware (such as Google Glass[4]) and the widespread availability of sensing devices (such as GPS), a variety of challenges remain – or emerge with new applications. In this section we aim at pointing out some of these challenges; while there definitely exist more than the ones addressed, we will only consider those relevant in the context of this thesis.

2.4.1 Complex Data

The ubiquitous availability of sensor devices, the advancement in computational power, the digitalization of our everyday life, and the spread of the internet of things has led to a proliferation of complex data, outpacing traditional (relational) data such as booking information. Sources of complexity can be diverse, starting from the temporal variability of data over the uncertainty of sensor devices and ranging to set-based data where objects are described by a combination of more or less primitive sub-objects.

2.4.1.1 Multi-Instance data

While simple objects can often be described by a single feature vector, more complex objects can consist of sets of those feature representations. One ex-

[4]https://www.google.com/glass/start/

ample that we will address within this thesis is image data. In keypoint-based object recognition [3], an image is described by a set of feature vectors. These vectors that are generated around interesting regions in the image, so-called interest points or keypoints. To find an image, not a single kNN query has to be posed on the database, but rather a bulk of queries has to be processed, sometimes over a thousand. Together with other inherent properties of image features, e.g. their high-dimensional nature, this data becomes challenging to query; current solutions involve the Bag of Visual Words pipeline [23] or specialized approximate index structures [12] for nearest neighbor processing. Generally other types of data such as check-ins of users in geosocial networks or simple time series could be interpreted as multi-instance data as well, however we will consider these primarily as, probably multidimensional, temporal data.

2.4.1.2 Temporal Data

Traditional time series data has received a vast amount of research interest (see e.g. [37] for a tutorial on this topic) over decades due to its ubiquitous presence in our everyday lives, be it stock market price, the development of temperature over time or medical time series such as ECG. The universal availability of appropriate sensing devices such as GPS, and the users interest in publicly sharing this data, has pushed temporal data into the geospatial research community, encouraging the development of new techniques for managing and querying spatio-temporal data, including new index structures and query predicates [38, 39, 40, 41]. As we will see in Part III of this thesis, the introduction of temporal aspects on spatial data can make the query processing significantly more difficult, especially if uncertainty is involved.

2.4.1.3 Uncertain Data

A large fraction of today's data is user-generated and publicly shared using Facebook, Flickr, Instagram, YouTube, Endomondo or other service providers. As user-generated data is usually incomplete and sometimes incorrect, data scientists more and more face the challenge of handling incomplete, incorrect and generally uncertain data. Furthermore, physical actors and sensors are error-prone, introducing uncertainty into applications not directly involving users into the data generation process. A good example for uncertain data comes from the spatio-temporal domain where data is collected by RFID or GPS sensors. RFID sensors are placed at fixed positions and aim at identifying persons or objects entering the sensors range. When an object leaves a sensor's range and does not directly enter the range of an-

other sensor, its position is unknown and has to be derived from its previous, and possibly future positions (if the query is historic rather than predictive). GPS receivers can only detect their position up to a certain accuracy which depends on their surrounding environment; the accuracy in an urban environment close to high buildings is lower than in an open field. If the trajectory of an object is determined based on check-ins such as in geo-social networks, the position in the time interval between two check-ins is inherently unknown and has to be derived from the available check-in data. As a consequence of this pervasive presence of uncertainty, uncertain data processing has gained a significant amount of research interest. See for example [42] for a recent overview on uncertainty in spatial and spatio-temporal data.

2.4.2 Complex Query Predicates

While most fundamental query predicates such as range queries and kNN queries have been proposed decades ago, due to the increasing digitalization of our world and the resulting demand for solving sophisticated problems computationally, the formalization of new query predicates has not found an end in the new millennium. Examples of such new query predicates include the RkNN query [15] or the Skyline query [17]. In addition to these new query predicates for traditional vectorial data, the fusion of domains (such as spatial and temporal aspects as well as uncertainty) makes the formalization of new query predicates necessary in order to cover the new semantics of the query. New query predicates usually adapt the existing ones, e.g. ε-range or kNN queries, to a new data type, such as time-parameterized queries [38] in the context of spatio-temporal data. This adaption of existing predicates becomes relevant especially in the context of uncertain data. While queries on such data could simply consider the expected value of an object, this would disrespect most of the value of uncertainty: Given a probability distribution over object properties, it is possible to provide measures of confidence with the query result, making the interpretation of the result easier for users. To enable such augmented results, the semantics of a query on uncertain data has to be defined. An example of such a work in the context of spatio-temporal data is [41]. In this work the authors proposed a new query predicate extending window queries – a generalization of ε-range queries – on uncertain spatio-temporal data, allowing to compute the probability of an object being part of the query result. The definition of new query predicates clearly enforces the development of specialized query processing techniques, efficient index structures, and query optimization strategies. For example, uncertain window queries can be supported by a specialized index structure called the UST-tree [40].

2.4.3 Large Volumes

Similar to the past, companies still experience a high growth in the amount of data stored, and there is no end in sight. In the year 2014, Facebook stored 300PB of Hive Data in their data warehouse, increasing by a daily rate of 600TB and multiplying by a factor of three within a year [43]. YouTube reports a hundred hours of newly uploaded videos every minute [44]. EMC^2 estimated the size of the digital universe to about 4.4 zettabytes in 2014, increasing to 44 zettabytes in 2020 [1]. As a result, handling large volumes of data remains a challenging task, despite the advances in computational power and storage technology. While this problem also holds for large datasets of well-researched types such as text, it becomes even more challenging in the context of more complex data, e.g. temporal and/or uncertain data where query complexity is already high on relatively small datasets. One approach for reducing the memory footprint of large volumes is for example compression [43]. Speeding up the query evaluation of complex queries can be achieved by sophisticated approximation and indexing techniques such as in e.g. [40] for uncertain spatio-temporal queries. The efficiency of mining on and summarizing such data sets can be improved by employing techniques such as MapReduce [45].

Chapter 3

Thesis Overview and Contributions

In this thesis, we will focus on the challenges identified in the previous chapter in a variety of scenarios. Part II addresses an in-depth analysis of the RkNN join, a complex query predicate building upon RkNN queries. Part III aims at efficiently processing similarity queries on uncertain spatio-temporal data, namely addressing the challenges of uncertainty and temporal variability on large data volumes. Finally the focus of Part IV will be the processing of set-based similarity queries.

In Part II we provide an in-depth analysis of the RkNN join. While the problem of reverse nearest neighbor queries has received a vast amount of research interest over the last 15 years, the problem of performing such queries in a bulk has not seen an in-depth analysis so far, despite its possible applications. To find efficient solutions for RkNN join processing, we first formalize the RkNN join, identifying its monochromatic and bichromatic versions and their self-join variants. The bichromatic RkNN join can be transformed to a standard kNN join, leading to an intuitive solution for bichromatic join processing by falling back to well-researched solutions for kNN join processing. However, such a transformation is not possible for the monochromatic case. As a consequence, we develop solutions for this variant of the RkNN join, based on different strategies to identify objects that do not qualify as a result. We then evaluate these solutions in depth on a variety of synthetic and real datasets.

Part III of this thesis aims at extending nearest neighbor queries to uncertain spatio-temporal data. Starting from the traditional definition of kNN queries and a Markov-chain based approach for modeling uncertain trajectories, the goal of this part is to develop efficient querying mechanisms for nearest neighbor queries on uncertain trajectories. In contrast to previous

research on similarity search on uncertain spatio-temporal data we aim at considering temporal dependencies during query evaluation, as in Markov chains the position of an object at a given time instance affects the future position of the object. To achieve this goal, we first extend nearest neighbor queries to Markov-based object representations in an intuitive way, aiming not only at returning the objects closest to the query but also their probability of being a nearest neighbor. Then, we theoretically evaluate the runtime of these query predicates. Based on these results we develop efficient querying algorithms for the proposed query predicates based on efficient and effective sampling techniques and an index-based filter-refinement approach. Given the findings of this research on nearest neighbor queries, we extend these results to reverse nearest neighbor queries, providing efficient and effective solutions for their evaluation. Our experimental analysis evaluates the proposed approaches on synthetic data and on a real dataset of taxi trajectories in the city of Beijing.

Finally, in Part III of this thesis, we address the problem of querying large datasets containing set-based objects, namely image databases. In keypoint based object recognition, images are represented by (multi-)sets of vectors, usually describing intensity patches around keypoints in the image, and additional metadata such as the position and scale of these keypoints. As a first contribution, we aim at reducing the number of keypoints considered during query evaluation, as such a reduction can considerably reduce retrieval times. However, such a reduction also reduces the quality of the query result, as the number of keypoints queried reduces both recall and mean average precision (MAP) of the query. To mitigate these problems, we evaluate a modified pipeline involving the expansion of keypoint matches on the image level for increasing MAP and the excessive use of kNN queries, where k can be used to further boost retrieval performance by optimizing recall. On a variety of well-known benchmark datasets we show that such a pipeline can achieve competitive results compared to approaches considering the complete set of keypoints during matching. As a second contribution, as local image features are more and more shifting from the real-valued domain to the binary domain due to the excellent storage and matching efficiency of such approaches, we evaluate efficient indexing techniques for binary feature vectors based on the Locality Sensitive Hashing scheme.

To summarize, the further structure of this thesis is as follows. The following Part II addresses our research on the RkNN join. In Part III we address the challenge of kNN and RkNN queries on uncertain spatio-temporal data. The last practical part of this thesis, Part IV, addresses similarity queries on set-based objects. Part V concludes this thesis.

Chapter 4

Incorporated Publications and Coauthorship

Publications that have been incorporated into this thesis, especially from Part II and Part III have been previously published as a result of a cooperation between researchers from the Database Systems Group of the LMU Munich (Part II), sometimes together with external scientists (Part III). This section aims at providing an outline of the contributions of the author of this thesis to these papers. The chapters in this thesis corresponding to these papers consist of these papers' texts, with some modifications and extensions applied by the author of this thesis. The papers' texts have been written in a cooperation between all authors of the papers including the author of this thesis.

RkNN Join Processing. Part II of this thesis addresses the RkNN join which has been researched in [46, 47, 48] as a cooperation between Tobias Emrich, Hans-Peter Kriegel, Peer Kröger, Johannes Niedermayer, Matthias Renz and Andreas Züfle[1]. The theoretic definition of RkNN join variants, classification of existing RkNN join approaches, and the extension of the proposed algorithms to metric space has been developed by the author of this thesis completely. The self-pruning approach has been developed, refined, implemented and evaluated by the author of this thesis based on valuable discussions with the coauthors of these papers. The mutual pruning approach was completed and experimentally evaluated by the author of this thesis after some preliminary work of the coauthors.

Nearest Neighbor Queries in Uncertain Databases. The next part of this thesis, Part III, addresses uncertain nearest neighbor and reverse nearest neighbor queries on moving object trajectories.

[1] Authors of these papers are listed in alphabetical order

The work on uncertain nearest neighbor queries has been previously published in [13], results of this research have also been included in [49, 50]. The publication resulted from a cooperation with two external researchers, Nikos Mamoulis and Lei Chen in addition to Andreas Züfle, Tobias Emrich, Matthias Renz and Hans-Peter Kriegel. The theoretical analysis of the ∃NN query, including its proof, has been developed by the author of this thesis with some valuable input of Tobias Emrich. The theoretical evaluation of the ∀NN query has been developed by the author of this thesis together with Andreas Züfle: the author of this thesis provided the proof that the ∀NN query cannot be solved efficiently by model adaption and significantly reworked the remaining proofs from the section in the context of this thesis. The sampling approach has been developed by the author of this thesis based on valuable discussions with the coauthors; its theoretical evaluation has been drafted by Andreas Züfle and has been refined and completed together with the author of this thesis. The corresponding proof has seen a variety of improvements for the inclusion in this thesis, aiming at making it more complete and easier to understand. The author of this thesis also realized the relation of the sampling approach to the Baum-Welch algorithm, resulting in a new section addressing this issue. The implementation (which has been achieved by extending an existing framework) and evaluation of the proposed techniques has been conducted by the author of this thesis, building upon valuable discussions from the coauthors of this paper.

Our research on uncertain RkNN queries on moving object trajectories has been published in [51][2], including mostly the coauthors from the predecessor paper on uncertain kNN queries. The algorithms for RNN queries on uncertain spatio-temporal data have been developed by the author of this thesis together with Tobias Emrich, and implemented and evaluated by the author of this thesis.

Some graphics included in the corresponding chapter have been taken from our publication [52][3]; the server-side SVG visualization code where these figures are derived from has been implemented by the author of this thesis as well.

Nearest Neighbor Queries in Image Databases. Part IV, including the corresponding publications [53, 54] coauthored by Peer Kröger, has been developed, implemented and evaluated by the author of this thesis who is grateful to Tobias Emrich, Markus Mauder and Peer Kröger for their valuable discussions that helped improving this research.

[2] Authors of this paper are listed in alphabetical order

[3] Authors of this paper are listed in alphabetical order

Part II

The RkNN Join

Chapter 5

Introduction

The increasing digitalization of our world in both industrial and personal environments demands efficient and effective techniques for querying and mining the collected data. As a consequence, researchers in industry and academics are constantly experiencing the limitations of existing query predicates. Therefore, while standard queries such as ε-range queries or kNN queries have been proposed decades ago, the formalization of new query predicates has not found an end until now. While sometimes such new query predicates appear out of nowhere as a solution for new research questions emerging from new technologies and new data sources, they often develop gradually as a result of constant research that finally results in a formalization of a given problem. The following part aims at formalizing such a new query predicate, which has been spooking around for quite some time, however without any in-depth formalization and evaluation: the RkNN Join.

Since the proposal of Reverse k-Nearest Neighbor (RkNN) queries in the beginning of the new millennium[15] considerable research effort has been put into the development of efficient RkNN query strategies, where the Reverse k-Nearest Neighbors of a query object r have to be retrieved from a set of database features S. However, the problem of solving complex RkNN queries where the query consists of a set R of query objects, i. e. a *bulk* or *group* query or simply a *join*, has not seen a thorough formalization and evaluation so far.

The problem of RkNN joins can for example arise in the strategic decision making process of companies that supply products to clients which are typically shops (cf. Figure 5.1). Consider a supplier (e.g. supplying video stores) that has a set of products R (e.g. videos each described by a given set of features like genre, length, etc.). Each client (e.g. video store) also has a portfolio S (e.g. a set of videos) which typically include different groups of products (videos) satisfying different preferences and, thus, different groups

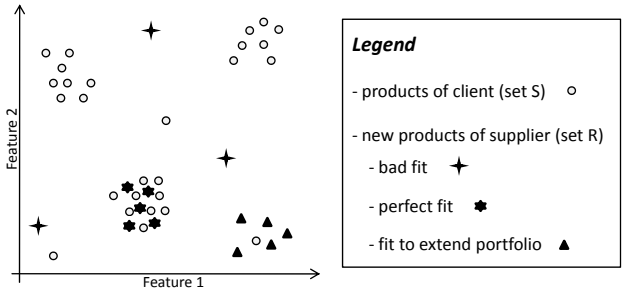


Figure 5.1: Application of RkNN join between two sets of products R and S for product (set) recommendation.

of customers. In order to judge which products should be offered by the supplier to a given client, the supplier needs information about which objects in R fit to the characteristics of the client's portfolio S and/or would be a good supplement to extend this portfolio. From the supplier's point of view it is particularly important to know about the data characteristics. Thus the following process can be employed in order to recommend updates for clients: First, for each product s in S the kNNs are computed. Second, for each product r in R, it is examined whether or not r is amongst the kNN of which product s. In other words, for each r, an RkNN query in S is launched. If r has a lot of RkNNs in S, this indicates that r fits well to a corresponding group within the data distribution of s (although this usually needs additional inspection). If r has no RkNNs in S, r obviously does not fit well.

Analogously, RkNN joins can also be employed for solving *inverse* queries [55] where the task is to find for a given set of query objects the set of database objects, having all (/most) query objects in their kNN set. Furthermore, the RkNN join operation plays a role in updating patterns derived by a variety of data mining algorithms that rely on kNN information after changes to the database, e.g. shared-neighbor clustering [56, 57] and kNN-based outlier detection [58, 59].

For evaluating single RkNN queries, two groups of algorithms have evolved over time. *Self pruning* approaches (e.g. [15, 60, 61]) have to perform costly precomputations in order to materialize kNN-spheres for all database objects. These kNN-spheres are used for pruning candidates during query execution. In contrast, *mutual pruning* approaches (e.g. [62, 63, 64]) do not perform any precomputations. This results in more flexibility in terms of updates and the choice of k because materialized results need not to be updated each time the database changes. Furthermore, in contrast to self pruning approaches, mutual pruning approaches do not require the parameter k to be known prior

to index generation. However, mutual pruning introduces costly refinement of candidates, resulting in higher overall cost.

In Part II of this thesis, we discuss how self and mutual pruning approaches adapt to RkNN joins and we compare their performance to existing solutions for single-point RkNN queries in a join setting. We show that the overhead of performing a traditional RkNN-query for each point in the query set R separately cannot be justified, even if R is small. Additionally, we find that with increasing the size of R self pruning approaches that compute kNN spheres on the fly become more useful than approaches based on mutual pruning. The key contributions of this part are as follows:

- We provide a formal classification of variants of the RkNN join.
- We integrate a variety of existing RkNN join algorithms into our classification.
- We propose self- and mutual-pruning algorithms for solving RkNN joins that do not rely on index structures specialized on RkNN queries or RkNN join operations and are therefore independent of the surrounding database environment.
- Both of these algorithms are extended to metric spaces in order to allow processing a larger variety of data sets.
- A systematic experimental evaluation compares the performance of the proposed join algorithms to classic algorithms for single RkNN queries and provides recommendations on which solutions fit best for specific scenarios.

This research on RkNN join processing has been previously published in [46, 47, 48]; for an overview over the contributions of the author of this thesis to this work we refer to Chapter 4.

In addition to this short introduction, Part II of this thesis is divided into four chapters. The next chapter 6 dives into the preliminaries of RkNN join processing: In Section 6.1 we provide a formal problem definition while Section 6.2 summarizes related work on RkNN query processing. A classification of existing RkNN joins is provided in Section 6.3. In Chapter 7 we propose the mutual pruning algorithm (Section 7.1) and the self-pruning approach (Section 7.2) for RkNN join processing. We then extend our work to general metric spaces (Section 7.3) in a short excursus. Finally, in Chapter 8, we provide an extensive performance comparison of the proposed self- and mutual-pruning approaches. The final chapter 9 concludes Part II of the thesis.

Chapter 6

Preliminaries

6.1 Problem Definition

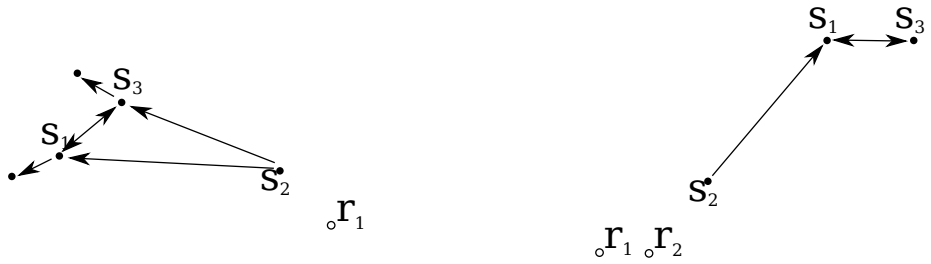
Recall from the introduction (Definition 4) that, given a finite multidimensional data set $S \subset \mathbb{R}^d$ ($s_i \in \mathbb{R}^d$) and a query point $r \in \mathbb{R}^d$, a k-nearest neighbor (kNN) query returns the k points from S closest to r. In contrast, a monochromatic RkNN query (Definition 5) returns all points $s_i \in S$ that would have r as one of its nearest neighbors. In Figure 6.1a an R2NN query is shown. Arrows denote a subset of the 2NN relationships between points from S; for a given point, arrows point in the direction of the nearest neighbors. Since r_1 is closer to s_2 than its 2NN s_1, the result set of an R2NN query with query point r_1 is $\{s_2\}$. s_3 is not a result of the query since its 2NN s_1 is closer than r_1. Note that the RkNN query is not symmetric to the kNN query, i.e. the kNN result $k\text{NN}(r_1,S) \neq \text{R}k\text{NN}(r_1, S)$, because the 2NNs of r_1 are s_2 and s_3. Therefore the result of an RkNN(r_1,S) query cannot be directly inferred from the result of a kNN query kNN(r_1,S).

In this part, we address the problem of RkNN joins. Given two sets R and S, the goal of a monochromatic RkNN join is to compute, for each point $r \in R$ its monochromatic RkNNs in S.

Definition 7 (Monochromatic RkNN join)**.** *Given finite sets $S \subset \mathbb{R}^d$ and $R \subset \mathbb{R}^d$, the monochromatic RkNN join $R \overset{MRkNN}{\bowtie} S$ returns a set of pairs containing for each $r \in R$ its RkNN from S:*

$$R \overset{MRkNN}{\bowtie} S = \{(r, s) | r \in R \wedge s \in S \wedge s \in RkNN(r, S, k)\}$$

An example for $k = 1$ can be found in Figure 6.1b. The result for both objects from R in this example is $\text{R1NN}(r_1) = \text{R1NN}(r_2) = \{s_2\}$, i.e.



(a) Monochromatic R2NN Query (b) Monochromatic R1NN Join

Figure 6.1: Visualization of the Monochromatic RkNN Query and the Monochromatic RkNN Join

$R \overset{\text{MRkNN}}{\bowtie} S = \{(r_1, s_2), (r_2, s_2)\}$. Note that the elements r_1 and r_2 from R do not influence each other, i.e., r_1 cannot be a result object of r_2 and vice versa. This follows directly from the definition of the MRkNN join. A variant of the RkNN join is the bichromatic join $R \overset{\text{BRkNN}}{\bowtie} S$ where for all $r \in R$ a bichromatic RkNN query is performed:

Definition 8 (Bichromatic RkNN join)**.** *Given two finite sets $S \subset \mathbb{R}^d$ and $R \subset \mathbb{R}^d$, the bichromatic RkNN join $R \overset{BRkNN}{\bowtie} S$ returns a set of pairs containing for each $r \in R$ its BRkNN from S:*

$$R \overset{BRkNN}{\bowtie} S = \{(r, s) | r \in R \wedge s \in S \wedge s \in BRkNN(r, R, S, k)\}$$

A BRkNN join can be expressed with a kNN join. Based on the definition of the BRkNN query, the BRkNN join can be converted:

$$R \overset{\text{BRkNN}}{\bowtie} S = \{(r, s) | r \in R \wedge s \in S \wedge r \in kNN(s, R, k)\}$$

The kNN join can be defined as:

$$S \overset{\text{kNN}}{\bowtie} R = \{(s, r) | s \in S \wedge r \in R \wedge r \in kNN(s, R, k)\}$$

$$\Leftrightarrow R \overset{\text{BRkNN}}{\bowtie} S = S \overset{\text{kNN}}{\bowtie} R$$

Hence, a BRkNN join can be performed by reusing the techniques from kNN research, such as [65, 66]. Furthermore, this problem has already been addressed in [67] and [68]. In Section 6.3 we will closely examine these publications.

Last but not least let us analyze the special case where $R = S$. The monochromatic RkNN-self-join can be defined as follows:

Definition 9 (Monochromatic RkNN Self Join)**.** *Given a finite set $S \subset \mathbb{R}^d$, the monochromatic RkNN self join is defined as $S \overset{MRkNN}{\bowtie} S$.*

Performing a monochromatic RkNN-self-join is trivial, since it is possible to perform a self-kNN-join on S:

$$S \overset{\text{MRkNN}}{\bowtie} S = \{(r,s) | r \in S \wedge s \in S \wedge r \in (k+1)NN(s, S \cup \{r\}, k+1)\}$$
$$= \{(r,s) | r \in S \wedge s \in S \wedge r \in kNN(s, S, k)\}$$

The resulting pairs (r, s) of the kNN-join just have to be inverted to produce the result of the RkNN join. Notice that with a self RkNN join, a query point always returns itself as one of its RkNNs.

In summary, we observe that the monochromatic RkNN join where $R \neq S$ cannot be matched to existing well-addressed problems. Therefore, we will address the processing of this problem in the following.

6.2 Related Work

The problem of efficiently supporting RkNN queries has been studied extensively in the past years. Existing approaches for Euclidean RkNN search can be classified as self pruning approaches or mutual pruning approaches. Since these approaches are the foundations of RkNN join processing, we review them in the following.

Self pruning approaches like the RNN-Tree [15] and the RdNN-Tree [60] are usually designed on top of a hierarchically organized tree-like index structure. They try to conservatively or exactly estimate the kNN distance of each index entry e. If this estimate is smaller than the distance of e to the query q, then e can be pruned. Thereby, self pruning approaches do not usually consider other entries (database points or index nodes) in order to estimate the kNN distance of an entry e, but simply precompute kNN distances of database points and propagate these distances to higher level index nodes. The major limitation of these approaches is that the precomputation and update (in case of database changes) of kNN distances is time consuming, the storage of these distances wastes memory and, thus, these methods are usually limited to one specific or a few values of k. Approaches like [61, 69] try to overcome these limitations by using approximations of kNN distances but this in turn yields an additional refinement overhead during query processing – or only approximate results.

Mutual pruning approaches such as [62, 63, 64, 70] use other points to prune a given index entry e. The most well-known approach called TPL is

presented in [64]. It uses any hierarchical tree-based index structure such as an R-Tree to compute a nearest neighbor ranking of the query point q. The key idea is to iteratively construct Voronoi hyper-planes around q w.r.t. to the points from the ranking. Points and index entries that are beyond k Voronoi hyper-planes w.r.t. q can be pruned and do not have to be considered for Voronoi construction anymore. Such RkNN query approaches can generally be applied to a join setting, however without additional adaptions they scale linear in the query set R. We will show this behavior in the experimental section of this paper exemplarily with the TPL algorithm.

A *combination of self- and mutual pruning* is presented in [71]. It obtains conservative and progressive distance approximations between a query point and arbitrarily approximated regions of a metric index structure. A further specialization of this approach to Euclidean data is proposed in [72] exploiting geometric properties to achieve a higher pruning power.

Beside solutions for Euclidean data, there exist solutions for *general metric spaces* (e.g. [61, 69]). Typically, metric approaches are less efficient than the approaches tailored for Euclidean data because they cannot make use of the Euclidean geometry.

Furthermore, there exist approximate solutions for the RkNN query problem that aim at reducing the query execution time at the cost of accuracy (e.g. [63, 73]).

In the following section, we will concentrate on related work on RkNN join processing and analyze these approaches more deeply.

6.3 Classification of Existing RkNN Joins

There exist a few other publications mentioning the problem of RkNN joins, however without providing an in-depth formalization of the problem including its variants. For the sake of completeness, we hereby theoretically evaluate the existing publications and convert their definitions of the RkNN join problem to our classification.

Bichromatic RkNN Joins as a by-product of incremental kNN Queries The authors of [67] aim at computing high-dimensional kNN joins, providing incremental updates to the user. As a by-product however, the authors provide means of performing RkNN joins on the database. They define the RkNN join as follows: $R \overset{[67]}{\ltimes} S =$

$$\{(q, RkNN(q)) | q \in S \wedge RkNN(q) \subset R \wedge \{\forall p \in RkNN(q) : p \in R \wedge q \in kNN(p)\}\}$$

We first split the result set $RkNN(q)$ into the elements $RkNN(q)_i$ of the corresponding set. This allows to simplify the above formula:

$$R \overset{[67]}{\bowtie} S = \{(q, RkNN(q)_i) | q \in S \wedge RkNN(q)_i \in R \wedge q \in kNN(RkNN(q)_i)\}\}$$

Substituting $q = s, RkNN(q)_i = r$ leads to:

$$R \overset{[67]}{\bowtie} S = \{(s, r) | s \in S \wedge r \in R \wedge s \in kNN(r)\}$$

As the query result s relates to the set S, we can bring $kNN(r)$ to our notation $kNN(r, S, k)$, leading to

$$R \overset{[67]}{\bowtie} S = \{(s, r) | s \in S \wedge r \in R \wedge s \in kNN(r, S, k)\}$$

which relates to our definition of the *bichromatic* RkNN join by simply swapping the sets R and S:

$$R \overset{[67]}{\bowtie} S = \{(r, s) | r \in R \wedge s \in S \wedge r \in kNN(s, R, k)\} = R \overset{\text{BRkNN}}{\bowtie} S$$

The RkNN Join from Venkateswaran Another definition of the RkNN join has been provided in [68]. The author defines the RkNN join as

$$R \overset{[68]}{\bowtie} S = \{U | U \subseteq S \wedge \forall u \in U : \exists v \in R : dist(v, u) \leq dist(v, t) \forall t \in S\}$$

By splitting the set U into single objects, we get

$$R \overset{[68]}{\bowtie} S = \{u | u \in S \wedge \exists v \in R : dist(v, u) \leq dist(v, t) \forall t \in S\}$$

We also add v to the resulting values and remove the $\exists$-quantifier (it is already implicitly contained in the definition of the join set), getting tuples

$$R \overset{[68]}{\bowtie} S = \{(u, v) | u \in S \wedge v \in R \wedge \forall t \in S : dist(v, u) \leq dist(v, t)\}$$

Substitution ($u = s$, $v = r$) yields:

$$R \overset{[68]}{\bowtie} S = \{(s, r) | s \in S \wedge r \in R \wedge \forall t \in S : dist(r, s) \leq dist(r, t)\}$$

The last factor is simply a nearest neighbor query, yielding

$$R \overset{[68]}{\bowtie} S = \{(s, r) | s \in S \wedge r \in R \wedge s \in kNN(r, S, 1)\}$$

Therefore, a generalization of this definition to kNN (which has already been done in [68]) and swapping the semantic of R and S yields our definition of the bichromatic RkNN join:

$$R \overset{[68]}{\bowtie} S = \{(r, s) | r \in R \wedge s \in S \wedge r \in kNN(s, R)\} = R \overset{\text{BRkNN}}{\bowtie} S$$

High-Dimensional Monochromatic RkNN Join The authors of the paper [74] have proposed an algorithm for monochromatic RkNN join processing if a specialized index structure such as an RdkNN tree is given; in contrast, our approaches do not need such specialized index structures, as kNN distance computations are included in the query evaluation step. However still, such an algorithm can be useful if the set S is relatively stable. The algorithm performs a parallel tree traversal; pruning of irrelevant subtrees is possible by employing the kNN distance stored in nodes of the tree. As the paper is written in Chinese language, we were not able to deeply analyze this paper and therefore will not further consider it in our research.

Bichromatic RkNN queries Although bichromatic RkNN queries are by definition not an RkNN join, we would like to investigate a simple solution to solve this problem due to its algorithmic structure which is similar to our self-pruning approach proposed in Section 7.2. The solution has been proposed in [15]. To solve a bichromatic RkNN query $BRkNN(r, R, S)$, it is possible to materialize the kNN-spheres of every point $s \in S$ over all points $r \in R$, i.e. compute the nearest neighbors of points s in set R. As a second step, we would have to report the points s that have r as one of their nearest neighbors by checking if r is contained in the kNN sphere of s. Although the algorithmic structure of this approach is similar to our self-pruning approach, the semantic is different. First, for the bichromatic RkNN query, the kNN join is performed between the sets S and R. For our RkNN join, the kNN join is a self-join. Second, to compute the actual result, the bichromatic RkNN query employs a simple point inclusion test of r over the computed kNN spheres. In contrast, we have to perform a varying range join, see Section 7.2.

Chapter 7

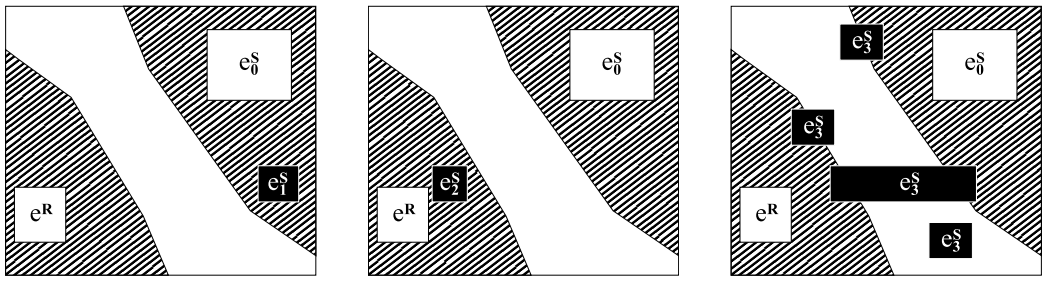
Algorithms

7.1 The Mutual Pruning Algorithm

Mutual pruning approaches such as TPL [64] are state-of-the-art solutions for single RkNN queries. In this paper we aim at analyzing whether this assumption still holds for an RkNN join setting. Therefore, in this section, we propose an algorithm for processing RkNN joins based on a strategy similar to TPL; we call this solution the *mutual pruning approach* or the *UL* approach, as it is based on *Update Lists*. We assume that both sets R and S are indexed by an aggregated hierarchical tree-like access structure such as the aR^*-tree [75]. An aR^*-Tree is equivalent to an R^*-Tree but stores an additional integer value (often called weight) within each entry, corresponding to the number of objects contained in the subtree. The indexes are denoted by $\mathcal{R}$ and $\mathcal{S}$, respectively.

7.1.1 General Idea

The proposed algorithm is based on a solution for Ranking-RkNN queries, initially suggested in [76]. Unlike TPL, which can only use leaf entries (points) to prune other leaf entries and intermediate entries (MBRs), the technique of [76] further permits to use intermediate entries for pruning, thus, allowing to prune entries while traversing the tree, without having to wait for k leaf entries to be refined first. The algorithm of [76] uses the MAXDIST-MINDIST-approach as a simple method for mutual pruning using rectangles. This approach exploits that, for three rectangles R, A, B, it holds that A must be closer to R than B, if $MAXDIST(A, R) < MINDIST(B, R)$. The



(a) Case 1 (b) Case 2 (c) Case 3

Figure 7.1: Spatial Domination.

algorithm that we use in this work, will augment the algorithm of [76] by replacing the MAXDIST-MINDIST-approach by the spatial pruning approach proposed in [29] which has been shown to be more selective. In the following, the base algorithm of [76], enhanced by [29] will be extended to process joins.

The mutual pruning approach introduced in this section is based on an idea which is often used for efficient spatial join processing: Both indexes $\mathcal{R}$ and $\mathcal{S}$ are traversed in parallel, result candidates for points $r \in R$ of the outer set are collected and for each point $r \in R$ irrelevant subtrees of the inner index $\mathcal{S}$ are pruned; we will evaluate if this approach is also useful for RkNN joins during performance analysis. Thus, at some point of traversing both trees, we will need to identify pairs of entries ($e^R \in \mathcal{R}$, $e^S \in \mathcal{S}$) for which we can already decide, that for any pair of points ($r \in e^R$, $s \in e^S$) it must/must not hold that s is a RkNN of r. To make this decision without accessing the exact positions of children of e^R and e^S, we will use the concept of spatial domination ([29]): If an entry e^R is (spatially) dominated by at least k entries in $\mathcal{S}$ with respect to e^S, then no point in e^S can possibly have any point of e^R as one of its k-nearest neighbors. Due to the spatial extent of MBRs, this decision is not always definite. We have to distinct several cases, as illustrated in Figure 7.1. The subfigures visualize two pages e^R and e_0^S, and one of the additional pages e_1^S, e_2^S, e_3^S. The striped areas in the picture denote the set of points on which a closer decision can definitely be made. This means, no matter which points from the rectangle e^R and e_0^S are chosen, the point in the striped area is always closer to the point from e^R (e_0^S) than it is to the point from e_0^S (e^R). Therefore, in the first case (a), e_0^S is definitely closer to e_1^S than e^R is to e_1^S. In the second case (b), e^R is definitely closer to e_2^S than e_0^S is to e_2^S. In the third case (c), in all of the four subcases, no decision can be made.

More formally, in the first case, we can decide that an entry is pruned by another entry. For example, in Figure 7.1a, entry e^R is dominated by

entry e_0^S with respect to entry e_1^S, since for all possible triples of points $(s_0 \in e_0^S, s_1 \in e_1^S, r \in e^R)$ it holds that s_0 must be closer to s_1 than r is to s_1. This domination relation can be used to prune e_1^S: If the number of objects contained in e_0^S is at least k, then we can safely conclude that at least k objects must be closer to any point in e_1^S than e_1^S is to e^R, and, thus, e_1^S and all its child entries can be pruned. To efficiently decide if an entry e_0^S dominates an entry e^R with respect to an entry e_1^S (all entries can be points or rectangles), we utilize the decision criterion $Dom(e_0^S, e^R, e_1^S)$ proposed in [29] which prevents us from doing a costly materialization of the pruning regions like the striped areas in Figure 7.1. Materialization here means the exact computation of the polygonal areas that allow pruning a page.

In the second case, we can decide that neither an entry, nor its children can possibly be pruned by another entry. In Figure 7.1b, consider entry e_2^S. It holds that for any triple of points $(s_0 \in e_0^S, s_2 \in e_2^S, r \in e^R)$, that r is closer to s_2 than s_0 is to s_2. Although, in this case, we cannot prune e_2^S, we can safely avoid further domination tests of children of the tested entries, as e_0^S and its children will never prune e_2^S. We can efficiently perform this test by evaluating the aforementioned criterion $Dom(e^R, e_0^S, e_2^S)$.

Finally, in the third case, both predicates, i.e. $Dom(e_0^S, e^R, e_3^S)$ and $Dom(e^R, e_0^S, e_3^S)$, do not hold for any entry e_3^S in Figure 7.1c. In this case, some points in e_3^S may be closer to some points e_0^S than some points in e^R are to the points from e_3^S, while other points may not. Thus, we have to refine at least some of the entries e_0^S, e_3^S or e^R. The reason for the inability to make a decision here, is that the pruning region between two rectangles is not a single line, but a whole region (called *tube* here, cf. Figure 7.1). For objects that fall into the tube, no decision can be made.

At any time of the execution of the algorithm only one entry e^R of the outer set is considered. For e^R, we minimize the number of domination checks that have to be performed. Therefore, we keep track of pairs of entries in $\mathcal{S}$, for which case three holds, because only in this case, refinement of entries may allow to prune further result pairs. This is achieved by managing, for each entry $e^S \in \mathcal{S}$, two lists $e^S.update1 \subset \mathcal{S}$ and $e^S.update2 \subset \mathcal{S}$: List $e^S.update1$ contains the set of entries with respect to which e^S may dominate e^R but does not dominate e^R for sure. Essentially, any entry in $e^S.update1$ may be pruned if e^S is refined. List $e^S.update2$ contains the set of entries, which may dominate e^R with respect to e^S, but which do not dominate e^R for sure. Thus, $e^S.update2$ contains the set of entries, whose children may potentially cause e^S to be pruned.

Algorithm 1 joinEntry(Entry e^R, Queue Q^S)

1: **for all** $e_i^S \in Q^S$ **do**
2: {Update domination count (lower bound) of all e_i^S}
3: **for all** $e_j^S \in e_i^S$.update2 **do**
4: **if** Dom(e_j^S, e^R, e_i^S) **then**
5: {definite decision possible, e_j^S prunes e_i^S}
6: e_i^S.dominationCount += e_j^S.weight
7: **else if** Dom(e^R, e_j^S, e_i^S) **then**
8: {e_i^S can definitely not be pruned by e_j^S}
9: e_i^S.update2.remove(e_j^S)
10: e_j^S.update1.remove(e_i^S)
11: **end if**
12: **end for**
13: **if** e_i^S.dominationCount $\geq k$ **then**
14: {no point in e_i^S can be an RkNN of a point in e^R}
15: delete(Q^S, e_i^S))
16: **end if**
17: **end for**
18: {in the following, resolve $\mathcal{S}$}
19: Queue $Q_c^S = \emptyset$
20: **while** ($e_i^S = Q^S$.poll()) $\neq$ NULL **do**
21: Go to line 20 if e_i^S.dominationCount $\geq k$ {e_i^S does not contain result candidates}
22: **if** Vol(e_i^S) > Vol(e^R) **then**
23: {go one level down in the subtree of e_i^S and add child pages to Q^S}
24: Q^S.add(resolve(e_i^S, e^R))
25: **else if** isLeaf(e_i^S) $\wedge$ isLeaf(e^R) **then**
26: {if no further refinement is possible, results still have to be verified}
27: **if** $e^R \in kNN(e_i^S)$ **then**
28: reportResult($< e^R, e_i^S >$)
29: **end if**
30: **else**
31: {put pages e_i^S into Q_c^S if they could neither be pruned nor reported as result}
32: Q_c^S.add(e_i^S)
33: **end if**
34: **end while**
35: {in the following, resolve e^R}
36: **if** $\neg$isLeaf(e^R) **then**
37: {finally, refine e_i^R by recursively calling *joinEntry* with Q_c^S}
38: **for all** $e_i^R \in e^R$.children **do**
39: joinEntry(e_i^R, clone(Q_c^S))
40: **end for**
41: **end if**

7.1.2 The Algorithm *joinEntry*

In order to implement these ideas, we use the recursive function shown in Algorithm 1, $joinEntry(Entry\, e^R, Queue\, Q^S)$. It receives an entry $e^R \in \mathcal{R}$ that represents the currently processed entry from the index of the outer set R, which can be a point, a leaf node containing several points, or an intermediate node. Q^S represents a set of entries from $\mathcal{S}$ sorted decreasingly in the number $|e^S.update1|$ of objects that an entry $e^S \in \mathcal{S}$ is able to prune. The reason is that resolving nodes with a large $update1$ list potentially allows pruning many other nodes.

In each call of $joinEntry()$, a lower bound of the number of objects dominating e^R with respect to e_i^S is updated for each entry $e_i^S \in Q^S$. This lower bound is denoted as *domination count*. Clearly, if for any entry e_i^S, it holds that the domination count $\geq k$, then the pair $< e^R, e_i^S >$ can be safely pruned. Note that using the notion of domination count, the list $e_i^S.update1$ can be interpreted as the list of entries e_j^S, for which the domination count of e_j^S may be increased by refinement of e_i^S. The list $e_i^S.update2$ can be interpreted as the list of entries whose refinement may increase the domination count of e_i^S. In Line 4 of Algorithm 1, the domination count of e_i^S is updated by calling $Dom(e_j^S, e_R, e_i^S)$ for each entry e_j^S in the list $e_i^S.update2$. If $Dom(e_j^S, e_R, e_i^S)$ holds, then the domination count of e_i^S is increased by the number of objects in e_j^S. The number of leaf entries is stored in each intermediate entry of the index. Otherwise, i.e., if e_j^S does not dominate e^R w.r.t. e_i^S, we check if it is still possible that any point in e_j^S dominates points in e^R with respect to any point in e_i^S. If that is not the case, then e_j^S is removed from the list of $e_i^S.update2$, and e_i^S is removed from the list of entries $e_j^S.update1$ (Lines 9-10). If these checks have increased the domination count of e_i^S to k or more, we can safely prune e_i^S in Line 15 and remove all its references from the $update1$ lists of other entries; this is achieved by the *delete* function.

Now that we have updated domination count values of all $e_i^S \in Q^S$, we start our refinement round in Line 20. Here, we have to decide which entry to refine. We can refine the outer entry e^R, or we can refine some, or all entries in the queue of inner entries Q^S. A heuristic that has shown good results in practice, is to try to keep, at each stage of the algorithm, both inner and outer entries at about the same volume. Using this heuristics, we first refine all inner entries $e_i^S \in Q^S$ which have a larger volume than the outer entry e^R in line 24. The corresponding algorithm is introduced in the next section.

After refining entries e_i^S, we next check in Line 25 if both inner entry e_i^S and outer entry e^R currently considered are both point entries. If that is the case, clearly, neither entry can be further refined, and we perform a kNN query using e_i^S as query object to decide whether e^R is a kNN of e_i^S, and, if

so, return the pair $< e^R, e_i^S >$ as a result. Finally, all entries e_i^S which could neither be pruned nor returned as a result, are stored in a new queue Q_C^S. This queue is then used to refine the outer entry e^R: For each child of e^R, the algorithm *joinEntry* is called recursively, using Q_C^S as inner queue.

7.1.3 Refinement: The *resolve*-Routine

Our algorithm for refinement of an inner entry e^S is shown in Algorithm 2 and works as follows: We first consider the set of entries $e^S.update1$ of other inner entries e_j^S whose domination count may be increased by children e_i^S of e^S. For each of these entries, we first remove e^S from its list $e_j^S.update2$, since e^S will be replaced by its children later on. Although e^S does not dominate e^R w.r.t. e_j^S, the children of e^S may do. Thus, for each child e_i^S of e^S, we now test if e_i^S dominates e^R w.r.t. e_j^S in Line 6 of Algorithm 2. If this is the case, then the domination count of e_j^S is incremented according to the number of objects in e_i^S.[1] Otherwise, we check if it is possible for e_i^S to dominate e^R w.r.t. e_j^S, and, if that is the case, then e_j^S is added to the list $e_i^S.update1$ of entries which e_i^S may affect, and e_i^S is added to the list $e_j^S.update2$ of entries which may affect e_j^S. Now that we have checked which objects the children e_i^S of e^S may affect, we next check which other entries may affect a child e_i^S. Thus, we check the list $e^S.update2$ of entries which may affect the domination count of e^S. For each such entry e_j^S and for each child e_i^S, we check if e_j^S dominates e^R w.r.t. e_i^S. If that is the case, the domination count of e_i^S is adjusted accordingly. Otherwise, if e_j^S can possibly dominate e^R w.r.t. e_i^S, then we add e_j^S to the list of entries $e_i^S.update2$, and we add e_i^S to the list $e_j^S.update1$. Finally, all child entries of e^S are returned, except those child entries, for which their corresponding domination count already reaches k.

7.2 A Self Pruning Approach

7.2.1 General Idea

We also developed a *self pruning approach* that does not rely on materialized information as existing self pruning techniques but computes kNN distances on the fly (therefore it will also be referred to as the *kNN* approach); the idea is based on the research results from [15]. For single RkNN queries,

[1] The check, whether the new domination count of e_j^S exceeds k will be performed in Line 21 of Algorithm 1

Algorithm 2 resolve(Entry e^S, Entry e^R)

```
1: LIST l
2: {(1) check which objects the children e_i^S of e^S may affect}
3: for all e_j^S ∈ e^S.update1 do
4:    e_j^S.update2.remove(e^S) {remove, children of e^S are now relevant instead
      of e^S}
5:    for all e_i^S ∈ e^S.children do
6:       if Dom(e_i^S, e^R, e_j^S) then
7:          {definite decision possible, e_i^S prunes e_j^S}
8:          e_j^S.dominationCount += e_i^S.weight
9:       else if ¬ Dom(e^R, e_i^S, e_j^S) then
10:         {no definite decision possible, e_i^S might prune e_j^S}
11:         e_j^S.update2.add(e_i^S)
12:         e_i^S.update1.add(e_j^S)
13:      end if
14:   end for
15: end for
16: {(2) check which other entries may affect a child e_i^S}
17: for all e_i^S ∈ e^S.children do
18:   for all e_j^S ∈ e^S.update2 do
19:      if Dom(e_j^S, e^R, e_i^S) then
20:         {definite decision possible, e_j^S prunes e_i^S}
21:         e_i^S.dominationCount += e_j^S.weight
22:      else if ¬ Dom(e^R, e_j^S, e_i^S) then
23:         {no definite decision possible, e_j^S might prune e_i^S}
24:         e_i^S.update2.add(e_j^S)
25:         e_j^S.update1.add(e_i^S)
26:      end if
27:   end for
28:   if e_i^S.dominationCount < k then
29:      {only return relevant entries that can not be pruned, yet}
30:      l.add(e_i^S)
31:   end if
32: end for
33: return l
```

Algorithm 3 $RkNN_{selfPruning}(R, S)$

```
   T = ∅
   {Perform self-join on S}
3: X = {(s, d(s)) | s ∈ S ∧ d(s) = max_{x∈kNN(s,S,k+1))} dist(x, s)}
   {Perform varying-range join on R}
   for all (s, d(s)) ∈ X do
6:   for all r ∈ R do
       if dist(s, r) < d(s) then
         T = T ∪ {(r, s)}
9:     end if
     end for
   end for
12: return T
```

such a self pruning approach obviously suffers from this overhead. However, intuitively, these on the fly computations may amortize for a large number of queries, i.e., a large outer set R (and we will see in the experiments that the break even point is surprisingly small). Thus, the idea of the following solution provides a dedicated algorithm to perform an RkNN join using the techniques of self pruning. In the following, we still assume that $\mathcal{R}$ and $\mathcal{S}$ are aR^*-tree representations of R and S, respectively. Let us first decompose the definition of the RkNN join into smaller pieces (see Algorithm 3). The RkNN query returns all pairs of points $(r \in R, s \in S)$ such that r is located in the kNN-spheres of s. Therefore, as a first step we simply compute the kNN-spheres of all points in S. This can be done by performing a self-(k+1)NN-join on S (see Line 3).[2] Then, for each of the resulting kNN-spheres, the set of points in R that is enclosed by this sphere has to be returned (see Line 5). This can be done by performing an ε-range query for each kNN-sphere on the set R. Note that the later query does not correspond to an ε-range-join, since the kNN-spheres of each $s \in S$ will usually have different radii. Rather, a "varying-range-join" needs to be applied. This introduces some interesting possibilities of optimization when pruning subtrees in $\mathcal{R}$. We will address this problem in Section 7.2.3. To summarize, an RkNN join can be performed by combining a self-(k+1)NN-join on S with a varying-range-join, a generalization of the ε-range-join. In the following, we will refine the algorithm sketch described in Algorithm 3. For this purpose, we now describe algorithms for computing the kNN-join and the varying-range-join.

[2]Each point will have itself in its kNN set, thus we need a $k + 1$NN-join to find the kNNs of each point in S.

7.2.2 Implementing the Self-kNN-Join

A variety of researchers addressed the problem of efficiently computing the result of a kNN-join. In our implementation we decided to evaluate two different approaches. One of them is a sequential second-order nested-loop join, that does not directly facilitate the structure of the underlying index, however it exploits the spatial proximity of points in leaf nodes of the tree. The other one is a hierarchical join that fully utilizes the index to reduce the number of unnecessary comparisons. Both of them shall be introduced in this section. In Chapter 8 we will also investigate which of the suggested algorithms fits best for specific data sets. For the sake of clarity, we assume to have two virtual copies $\mathcal{S}_l$ and $\mathcal{S}_r$ of $\mathcal{S}$ in the following. An entry from $\mathcal{S}_l$ is denoted by e_l^S and an entry from $\mathcal{S}_r$ as e_r^S.

7.2.2.1 Sequential Self-kNN-Join

For the sequential self-kNN-join we implemented a cache-aware nested-block loop join that takes the specific properties of the self-join, and the spatial proximity of values in the leaf nodes of a tree-structured index into account. It is based on the implementation in the ELKI-framework [77]. The algorithm basically takes a leaf from the index $\mathcal{S}$, performs a self-join within the same leaf first in order to initialize its maximum kNNDist ($MaxKNNDist_{temp}$) as its pruning distance. The pruning distance is used to prune whole leaf pages later on without processing the points contained in them. Then it sequentially accesses all leaf nodes of the index and joins them with the currently processed node. This version is not only easy to implement, it also enables to return partial results at each time where a page e_l^S has been processed such that memory can be freed after directly performing a varying-range-join on the partial result.

For each leaf node e_l^S from $\mathcal{S}_l$, the algorithm proceeds as follows. First, in order to initialize the $MaxKNNDist_{temp}(e_l^S)$ of e_l^S, a sequential self-kNN-join of e_l^S is performed. The kNN-join basically collects for each point in $p \in e_l^S$ the k points from e_r^S that are closest to p. Note that the actual kNN's are not relevant, just the distances of current kNN-candidates have to be tracked and hence the memory consumption can be reduced significantly compared to traditional kNN join processing. In a second step, each leaf $e_r^S \neq e_l^S$ of $\mathcal{S}_r$ is traversed in arbitrary, but deterministic, order. The page e_r^S can be pruned if $MINDIST(e_r^S, e_l^S) > MaxKNNDist_{temp}(e_l^S)$. If the page cannot be pruned, each point from e_l^S is joined with each point from e_r^S. After joining the contained points, the pruning distance $MaxKNNDist_{temp}(e_l^S)$ is updated when necessary. After finishing the join of e_l^S, the kNN-distances

of points in this page can be returned as a partial result. The join of the following leaf node $e_{l'}^S$ of $\mathcal{S}_l$ is again joined with itself first, however the join of remaining nodes is performed in opposite order. The intention of this proceeding is that some nodes from the last traversal are still stored in the page cache such that they do not have to be reloaded, reducing the number of page accesses.

7.2.2.2 Hierarchical Self-kNN-Join

The hierarchical kNN-Join is based on the best-first kNN search algorithm from [66], however with some minor adaptions to fit the special properties of our setting, a self-kNN-join. Actually, this join is only semi-hierarchical since it does not join two trees but rather joins each leaf from one tree with another tree, yielding an average complexity of $O(|S| \log(|S|))$ for a self join in contrast to $O(|S|^2)$ for the sequential self-join described in the section before. In contrast to a fully hierarchical approach this algorithm enables the possibility to return partial results each time a page e_l^S has been processed; we will exploit this property when processing the varying-range-join.

For each leaf node e_l^S taken from $\mathcal{S}_l$, the tree $\mathcal{S}_r$ is traversed starting with the root node in best-first order according to the $MINDIST$ of the current group and the currently processed subtree. For this purpose, the root entry of $\mathcal{S}_r$ is inserted into a priority queue. The priority queue is ordered by increasing $MINDIST$ to e_l^S. $MINDIST$ between two overlapping entries is always zero, however a bigger overlap is usually better than a smaller one since this can reduce the $MaxKNNDist_{temp}$ of the currently processed page more effectively. Therefore e_r^S that overlap e_l^S more than other pages are preferred. The algorithm pops the first entry e_r^S from the priority queue and checks if $MINDIST(e_r^S, e_l^S) \leq MaxKNNDist_{temp}$. Entries with $MINDIST(e_r^S, e_l^S) > MaxKNNDist_{temp}$ are pruned as in the sequential approach. If an entry cannot be pruned, the corresponding node has to be accessed. If the node is an intermediate node, all its children that cannot be pruned are inserted into the priority queue, i.e., the page is resolved. If the node is a leaf node, the page is joined. Joining is similar to the proceeding of the sequential join, however it employs some additional improvements from [66]: points $r_i \in e_r^S$ with $MINDIST(e_l^S, r_i) > MaxKNNDist_{temp}(e_l^S)$ and all points $l_i \in e_l^S$ with $NNDist(l_i) < MINDIST(l_i, e_r^S)$ can be pruned during an initial scan of the processed pages. This preprocessing reduces the quadratic cost of comparing all entries from one node with all entries from the other node. After finishing the join $MaxKNNDist_{temp}$ is updated when necessary.

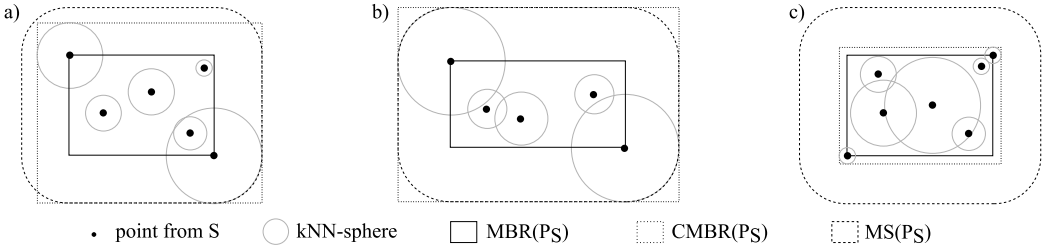


Figure 7.2: Comparison of $MS(e^S)$ and $CMBR(e^S)$ in an (a) average, (b) worst, and (c) best case.

7.2.3 Implementing the Varying-Range-Join

A trivial solution for performing a varying-range-join would be to sequentially search the set R for each $s \in S$ to find $\{r \in R | dist(r,s) < kNNDist(s)\}$. This operation returns the subset of R that would contain s as one of its RkNN-results. As a first improvement, the tree structure of $\mathcal{R}$ can be exploited to reduce the asymptotic complexity to an (on average) logarithmic one by performing a depth-first traversal: Only nodes e^R of $\mathcal{R}$ that intersect the kNN-sphere of s need to be considered, i.e., subtrees with $MINDIST(e^R, s) > kNNDist(s)$ can be pruned. For reducing the number of disc accesses, e.g. if a slow HDD is used instead of a fast SSD, it would be a good idea to traverse $\mathcal{R}$ less often with a larger subset of S. Since both self-kNN-join algorithms join pages e^S of objects, partial results show spatial proximity. This spatial proximity can be used to prune subtrees from $\mathcal{R}$ that cannot contain result candidates for any point in e^S, greatly speeding up the range query. This technique can even be extended. The straightforward approach for computing an RkNN join-result would be to materialize all kNN-spheres first and then perform a range query for each of the spheres on $\mathcal{R}$. However, in order to keep the memory consumption for storing kNN-spheres low, a readily kNN-joined page can be directly range-joined with $\mathcal{R}$ such that the corresponding kNN-spheres of the page do not have to be stored further.

This varying-range-join is implemented in form of a depth-first index traversal. It receives a set e^S containing points from S in combination with their kNN-distances and $\mathcal{R}$. The algorithm first checks each child of the root node of $\mathcal{R}$ whether or not it has to be accessed, i.e., if this page could contain a varying-range-join result of any point in e^S. If the child has to be accessed, it proceeds in a depth-first order, accessing its children and testing them recursively. The pseudo code is depicted in Algorithm 4. Checking if any child from e^R might have a point from e^S as an RkNN is evaluated in the function PossibleCandidate(e^S, e_1^R). Checking can be achieved in two

Algorithm 4 VaryingRangeJoin(Entry e^R, Set e^S)

```
if e^R points to a LeafNode then
  for all r ∈ e^R do
    for all s ∈ e^S do
      if dist(r,s) ≤ kNNDist(s) then
        report (r,s) as a result tuple
      end if
    end for
  end for
else
  for all e_1^R ∈ e^R do
    if PossibleCandidate(e^S,e_1^R) then
      VaryingRangeJoin(e_1^R,e^S)
    end if
  end for
end if
```

ways, by employing the Minkowski sum, or by laying an MBR around all kNN-spheres contained in e^S:

- Use the Minkowski sum $MS(e^S)$, following the idea of [60]: Check whether the MBR of $e^{\mathcal{R}}$ has a distance of less than $\max_{s \in e^S} kNNDist(s)$ to the MBR of the points in e^S. This can be evaluated by checking whether $MINDIST(MBR(e_1^R), MBR(e^S))$ is less than or equal to $\max_{s \in e^S} kNNDist(s)$
- Build a bounding box around all kNN-spheres, following [15]: Let $CMBR(e^S) = MBR(\bigcup_{s \in e^S} kNNSphere(s))$. e_1^R has to be evaluated if $CMBR(e^S) \cap MBR(e_1^R) \neq \emptyset$. The MBR $CMBR(e^S)$ can be calculated as $(0 \leq i \leq d)$:

$$\begin{aligned} CMBR(e^S).min_i &= \min_{p_j \in e^S} \{p_j[i] - kNNDist(p_j)\} \\ CMBR(e^S).max_i &= \max_{p_j \in e^S} \{p_j[i] + kNNDist(p_j)\} \end{aligned}$$

While the first check is quite simple to perform and for traditional range-joins even very restrictive, the second approach can be better when the diameter of kNN-spheres differs. An example for both bounding boxes is shown in Figure 7.2 (a). However, note that none of the checks is the best solution for

all scenarios. There exist special cases where the Minkowski sum fits the contained spheres tighter than $CMBR(e^S)$. For example if $|e^S| = 1$, the volume covered by a Minkowski sum is smaller than $CMBR(e^S)$ (see also Figure 7.2 (b)). More general, the worst case for $CMBR(e^S)$ happens if at each face of $MBR(e^S)$ there is a point p with $kNNDist(p) = \max_{s \in e^S} kNNDist(s)$. In this case there is $MS(e^S) \subset CMBR(e^S)$ and hence the Minkowski sum shows higher pruning power. The best case (Figure 7.2 (c)) happens if for each point $p \in e^S$ there is $kNNSphere(p) \setminus MBR(e^S) = \emptyset$, i.e., points with large kNN-spheres are close to the center of $MBR(e^S)$. In this scenario, $CMBR(e^S)$ performs better than the Minkowski sum approach.

7.3 Extension to Metric Spaces

In this short excursus we sketch how to extend the theoretical results from the previous sections to general metric spaces. For a definition of metric spaces, we refer to Definition 1 in Part I. As a vector space that we addressed in the remainder of this work offers some additional properties to general metric spaces, pruning in a vector space is usually more efficient and not directly applicable to general metric spaces. Exemplarily, the TPL algorithm uses the perpendicular bisector between two points in space for pruning that only exists under the l_2 norm. Therefore, in the following we will generalize the results from previous sections and discuss how RkNN joins can be performed in general metric spaces.

In the following we again assume that the data sets R and S are indexed by some index structure, however in this section a *metric* index structure such as the M-tree [78]. An M-tree is a balanced tree that can be dynamically updated. Inner nodes contain entries e storing an anchor object (or routing object) $e.a$, a covering radius $e.r$ and a pointer $e.c$ to the corresponding node that contains the children entries. All children of e and their descendants have a distance of at most $e.r$ to $e.a$. Furthermore, a routing entry contains a field $e.p$ denoting the distance to the parent routing object $P(e).a$. Although $e.p$ could be computed during query processing, it is stored in the entries as distance calculations in metric spaces can be costly. The data objects of an M-tree are referenced via data entries in the leaf nodes; these entries store the actual object value, an object id and a field $e.p$ which is defined in the same way as for the inner entries.

As a consequence of the information stored in the index structure, pruning irrelevant subtrees of an M-tree can be achieved by considering an entry's diameter and the stored anchor object. For this purpose we can employ the (general) $MINDIST(e_1, e_2)$ between two entries in the tree that assumes

two data elements to be as close to each other as possible; these distance bounds follow directly from [78]:

$$MINDIST(e1, e2) = max(dist(e_1.a, e_2.a) - e_1.r - e_2.r, 0)$$

Similarly, we can define the $MAXDIST(e_1, e_2)$ that assumes two data elements to be as far away of each other as possible by:

$$MAXDIST(e1, e2) = dist(e_1.a, e_2.a) + e_1.r + e_2.r$$

In the case where e_1 (e_2) becomes a data object, the corresponding radius $e_1.r$ ($e_2.r$) becomes zero.

Unfortunately these distance approximations make it necessary that the distance between two entries is computed. Assuming that the distance between the parents of a node are already known (and therefore cached), the number of distance calculations can often be reduced [78] (the case where the distance between $P(e_1)$ and e_2 is already known or vice-versa can be constructed equivalently):

$$MINDIST'(e1, e2) = max(d(P(e_1).a, P(e_2).a) - e_1.p - e_2.p - e_1.r - e_2.r, 0)$$

In this case, the distance between the parents is known such that for this $MINDIST$, distance computations can be avoided, however at the cost of accuracy and space. For a rough estimate of the $MAXDIST$, $MAXDIST'$ can be derived equivalently.

7.3.1 Adaptions of the Update List Approach

The algorithm based on update lists (Section 7.1) employs the spatial pruning from [29] that has been specifically designed for vector spaces under l_p norms and therefore cannot be used for general metric spaces. Therefore, the decision criterion has to be replaced by the weaker $MAXDIST$-$MINDIST$-criterion [29]:

$$Dom(A, B, R) := MAXDIST(A, R) < MINDIST(B, R)$$

The domination criterion can be easily extended by evaluating based on $MINDIST'$ and $MAXDIST'$ first. The $MAXDIST$-$MINDIST$ approach can also be used with vectorial data [76], for example in R-trees, however, it shows a lower pruning power than the improved domination criterion we used [29]. Furthermore, if vectorial data is stored in index structures with spherical representations, special spatial pruning criteria on spheres can be employed, such as [49].

The heuristic for deciding whether to resolve pages from R or from S must also be adapted, as general metric spaces do not have any notion of volume. Therefore, instead of employing the volume, the diameter $e.d$ could be used to come to a resolvement decision: In order to decide about the refinement of a node, we would have to check whether $Vol(e_1) > Vol(e_2)$. Note that this would be equivalent to a decision based on the volume of intermediate entries if an M-tree would be used to index vectorial data.

7.3.2 Adaptions of the kNN-Based Approach

The kNN-join based approaches can be adapted in a similar fashion. Let us first consider the first phase of the algorithm, the kNN join. Both kNN variants used throughout the paper can be used in the general metric case without major modifications; just the $MINDIST$ and $MAXDIST$ functions have to be modified. $MAXDIST'$ and $MINDIST'$ could be used as well. For the second phase, the varying-range join, we have to find the surrounding sphere of the kNN spheres of a data page e^S. The diameter of the surrounding sphere can be simply computed as

$$MS(e^S) = e^S.d + \max_{s \in e^S}\{kNNDist(s^S)\}$$

This solution conforms to the Minkowski sum approach from Section 7.2. In order to realize the $CMBR$ approach, we would have to find a new center point for the surrounding sphere, which is prohibitively expensive. The query processing for the varying range join can also be extended by the cached versions of distance estimates.

Chapter 8

Evaluation

We evaluate the mutual pruning approach (referred to as UL due to its use of update lists), and the self pruning variants (kNN$_*$) in comparison to the single RkNN query processor TPL in an RkNN join setting within the Java-based KDD-framework ELKI [77]. As performance indicators we chose the CPU time and the number of page accesses. Note that we did not evaluate existing self-pruning techniques such as [60] since these are only applicable if the value of k is fixed over all performed RkNN queries.

It should be noted that, for estimating the *CPU time* of a particular algorithm, we measured the thread time of the corresponding Java thread and performed dry runs before executing the actual simulations in order to keep the impact of garbage collection and Just-In-Time compilation on the results low. A test run was aborted if it lasted at least 20 times longer than executing a self pruning based RkNN join with the same set of variables. For measuring the number of *page accesses*, we assumed that a given number of pages fit into a dedicated cache. If a page has to be accessed but is not contained in the page cache, it has to be reloaded. If the cache is already full and a new page has to be loaded, an old page is kicked out in LRU manner. The page cache only manages data pages from secondary storage, remaining data structures have to be stored in main memory. In order to avoid everything being stored in memory, we employed a restrictive approach, assuming that only node- and entry-IDs can be stored in main memory. Therefore, each time information about an entry, e.g. an MBR or a data point, has to be accessed, the corresponding data page has to be reloaded into the cache. We chose this restrictive strategy because all algorithms employ different methods of processing data. While a sequential kNN-join processes only two data pages at once, mutual pruning based approaches like TPL and UL process the whole tree at a time by facilitating lists of entries, e.g. the TPL entry heap. These data structures are principally unbounded,

such that storing whole entries in memory could lead to a situation where the whole tree can be accessed without reloading any page from disk. This would bound the number of page accesses to the tree size, which is unrealistic in large database environments. We set the cache size to fit about 5% of all nodes in the default setting.

We chose the underlying synthetic data sets from R and S to be normally distributed with equivalent mean and a standard deviation of 0.15. We set the default size of R to $|R| = 0.01|S|$, since the performance of TPL degenerates with increasing $|R|$. For each of the analyzed algorithms we used exactly the same data set given a specific set of input variables in order to reduce skewed results. As an index structure for querying we employed an aggregated R*-tree (aR*-Tree [75]). During performance analysis, we analyzed the impact of k, the number of data points in R and S, the dimensionality d, the overlap o between the data sets R and S, the page size p and the cache size c on the performance of the evaluated algorithms keeping all but one variable at a fixed default value while varying a single independent variable. Input values for each of the analyzed independent variables can be found in Table 8.1. Furthermore, we investigated empirically for which sizes of the sets R and S TPL and $k\text{NN}_*$-algorithms show equivalent performance. This experiment is of great practical relevance since it gives hints on which specific algorithm to use in a specific setting.

Table 8.1: Values for the evaluated independent variables. Default values are denoted in bold.

Variable	Values	Unit
k	5, **10**, 100, 500, 1000	points
d	2, **4**, 6, 8, 10	dimensions
$\|R\|$	10, **100**, 1000, 10000, 20000, 40000	points
$\|S\|$	10, 1000, **10000**, 20000, 40000, 80000	points
o	**0.0**, 0.2, 0.4	$\|\mu_S - \mu_R\|$
p	512, **1024**, 2048, 4096, 8192	bytes
c	4096, 16364, **32768**, 65536, 131072	bytes

TPL was implemented as suggested in [64], however we did not incorporate the clipping step since this technique increased the computational cost of the algorithm in preliminary experiments and had only a marginal effect on the page accesses (especially when $d > 2$). Instead we implemented the decision criterion from [29].

Concerning the nomenclature of the algorithms we use the following notation. UL is the mutual pruning based algorithm. The additional subscript

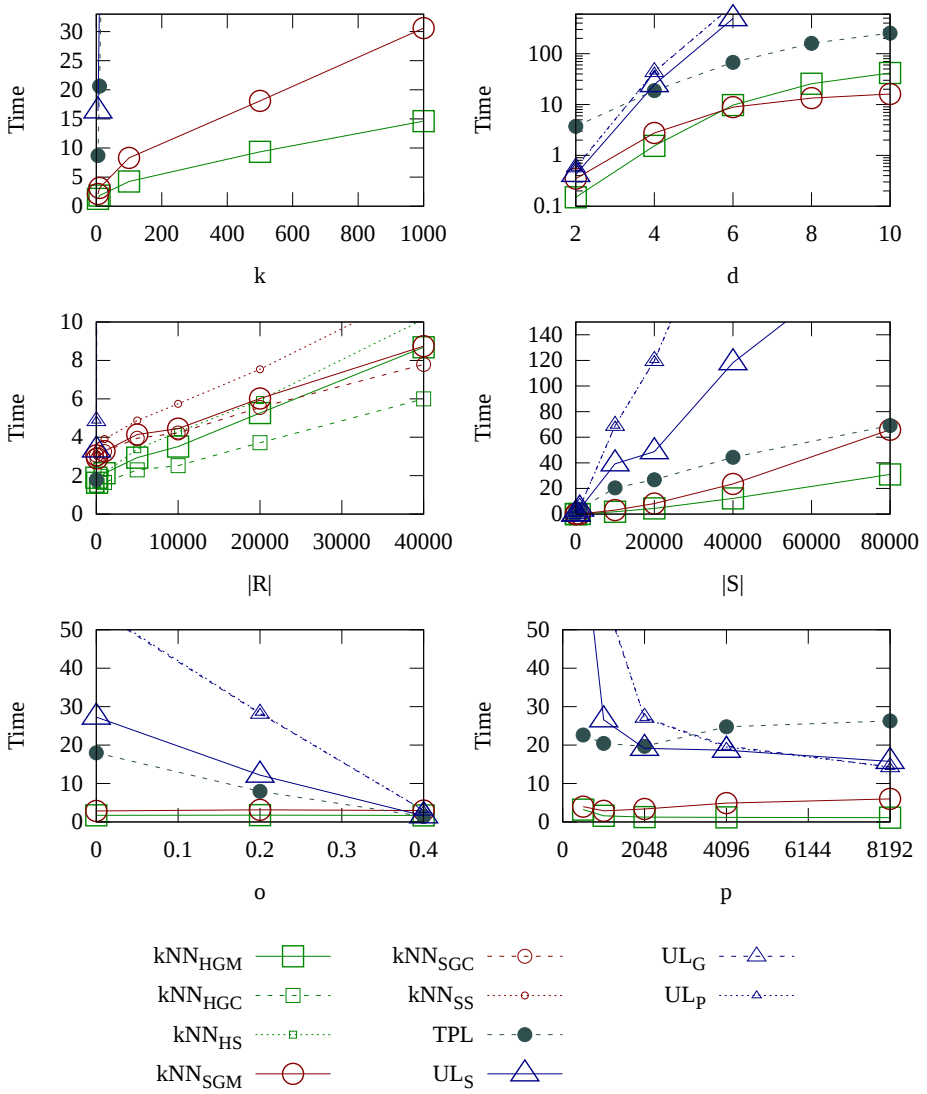


Figure 8.1: Performance (CPU time), synthetic dataset. Time is measured in seconds.

S (Single) means that every *single* point of R was queried on its own. With $\mathrm{UL_G}$ (Group), a whole set of points, a leaf page, was queried at once. $\mathrm{UL_P}$ (Parallel) traversed both indexes for R and S in parallel. For the kNN algorithms we employ a similar nomenclature $k\mathrm{NN}_{\mathrm{ABC}}$:

- $A \in \{H, S\}$ denotes whether a hierarchical or sequential nearest neighbor join is used (compare Section 7.2).
- $B \in \{S, G\}$: S denotes whether the whole kNN join is processed first

and then each resulting kNN sphere is used to perform a *single* range query on R. G (*group*) denotes that after the kNNs of a single page from S is computed, the corresponding page is used to perform a varying-range-query on the tree of R. The second method can be employed to keep the number of intermediate results low and avoid swapping these intermediate results to disk. Furthermore both methods vary in the algorithm used for the range-join, i.e., whether each point is queried separately or groups of points are queried together.

- $C \in \{C, M\}$ indicates how the MBR is computed when performing a group range-join (C: CMBR, M: Minkowski sum, compare Section 7.2).

Note that, in order to avoid clutter, if a group of algorithms (i.e. the different UL variants, the different hierarchical kNN variants or the different sequential kNN variants) had similar performance, we replaced them by a representative solution from this group. The complete plots can be found in [46, 47].

8.1 Experiments on Synthetic Data

Varying k. In a first series of experiments, we varied the parameter k. While the execution time of the self pruning based $k\text{NN}_*$ increases moderately with k, the remaining mutual pruning approaches become already unusable with low values for k (cf. Figure 8.1, top left). The runtime of TPL increases considerably fast. The reason for this is that not only the number of result candidates but also the number of objects which are necessary in order to confirm (or prune) these candidates increase super-linear in k. For the UL approaches, the runtime behaviour is similar. The main problem with this family of algorithms is their use of update lists. Each time a page is resolved, the corresponding update lists have to be partially recomputed. This leads to an increase of cost with larger k since on the one hand side, more pages have to be resolved, and on the other hand the length of the update lists of an entry increases and therefore more distance calculations are necessary.

The runtime behaviour of all kNN-based algorithms increases more moderate with increasing k. The performance of the hierarchical join is more stable w.r.t. different values of k, on the one hand side because the hierarchical version of the join can prune whole subtrees on the index level, and on the other hand because the hierarchical join employs further optimizations when joining leaf pages. Furthermore, note that the effect of performing a single/group varying-range join and using different MBRs is quite small. The

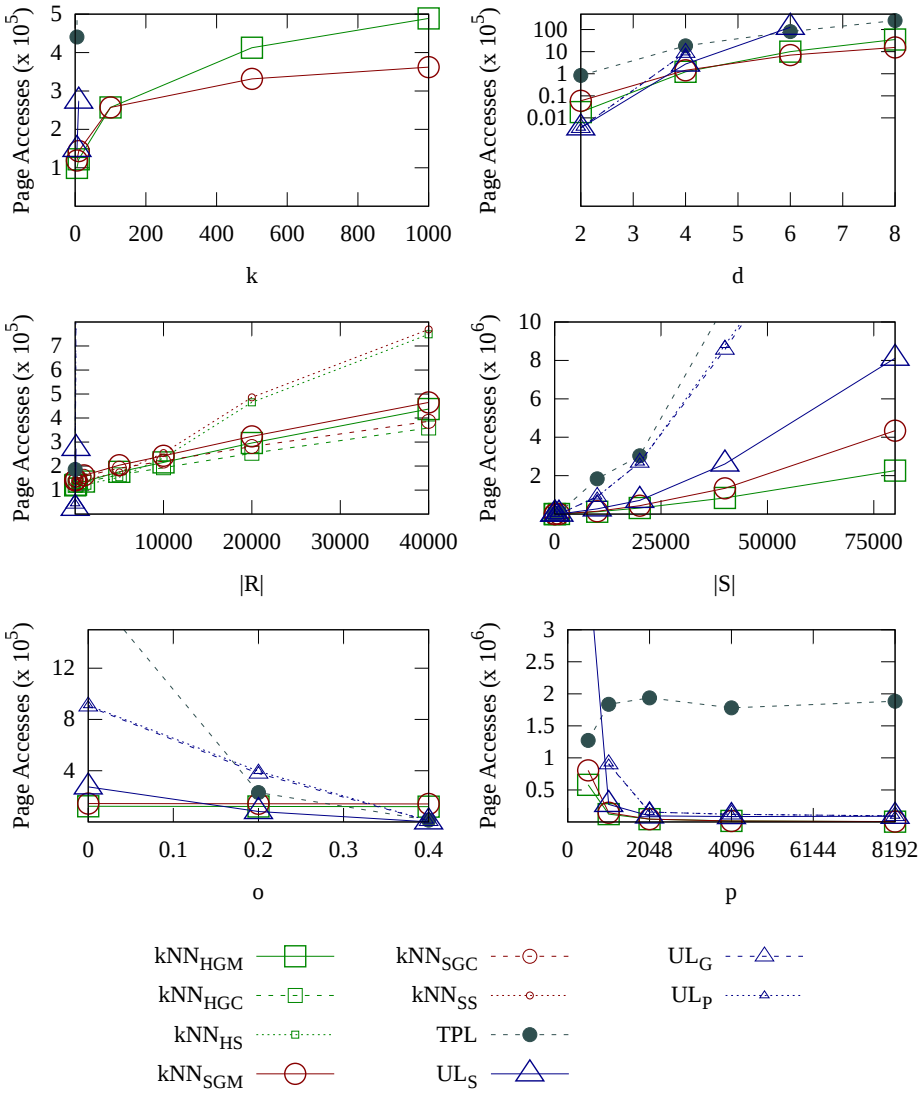


Figure 8.2: Performance (page accesses), synthetic dataset.

more sophisticated approach $k\text{NN}_{HGC}$ is indeed the best one, however the difference in execution time compared to the simpler solutions $k\text{NN}_{HGM}$ and $k\text{NN}_{HS}$ is quite low because in this setting $|R|$ is too small.

Concerning the number of page accesses, the picture is quite similar (cf. Figure 8.2, top left). TPL and UL show a worse performance than the kNN-based solutions. However, interestingly the curves of the sequential and hierarchical kNN-join intersect if k becomes rather large, making the sequential join a better choice. This shows that with large values of k the

pruning of whole subtrees of the index-based approach fails such that the overhead for traversing the index nodes of the tree cannot be justified any more.

Varying the Dimensionality (d). Taking a look at the performance of the different algorithms with varying dimensionality offers other interesting results. Note that the scale of these graphs is logarithmic (cf. Figure 8.1 and 8.2, top right). The UL approaches scale worse than the other approaches, because the pruning power of index-level pruning decreases with increasing dimensionality. However, with a dimensionality of 2 and 3, the UL approaches perform better than TPL concerning the execution time of the algorithms. Beginning with a dimensionality of 4, the UL approaches scale worse than TPL because the pruning power of index-level pruning decreases with increasing dimensionality. With increasing d, the number of entries in an update list increases super-linearly. Therefore, much more entries have to be checked each time an intermediate node is resolved, leading to a significant drop in performance. Note that $\mathrm{UL_G}$ and $\mathrm{UL_P}$ perform very similar to $\mathrm{UL_S}$, which is an interesting observation, since for kNN joins parallel tree traversals usually show a higher gain in performance than in an RkNN setting. TPL and the hierarchical join variants scale similarly, however, in this specific setting, the TPL based join performs by over a magnitude worse than the hierarchical join. Comparing sequential and hierarchical join, the performance of the index used by the hierarchical join degenerates with higher dimensionality, such that the sequential join is able to outperform all remaining approaches if the number of dimensions becomes higher than 6. This behaviour is most likely well explained by the "curse of dimensionality" and the degradation of spatial index structures in high dimensions.

The results in terms of the number of disk accesses look very similar, therefore they shall not be further investigated. However note that the UL approaches show much better performance in terms of the number of disk accesses if the dimensionality is low. Therefore, we recommend to use UL for spatial applications (i.e. 2D data).

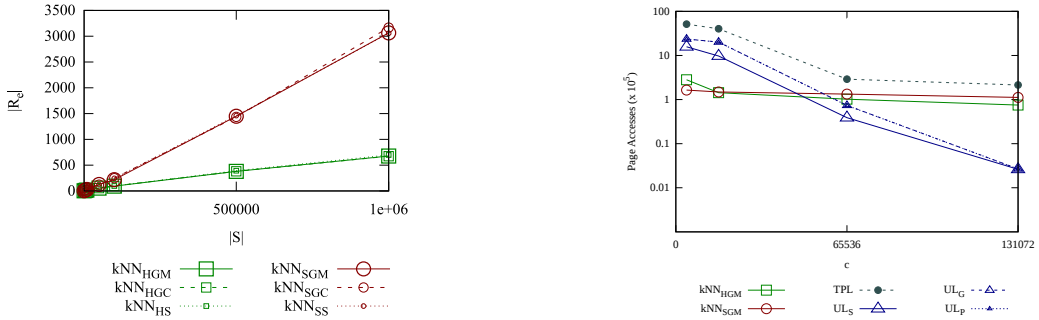
Varying the Size of R ($|R|$). Varying $|R|$ shows a negative effect on the mutual pruning approaches TPL and UL (cf Figure 8.1 center left). Especially TPL and UL_S scale linearly with $|R|$, as increasing $|R|$ simply increases the number of RkNN queries. Although the execution time of the join-based algorithms grows with $|R|$ as well, this increase is moderate. Note that with a larger $|R|$ the difference between the different range-join algorithms becomes more obvious. If $|R|$ is very large, the $CMBR$ method become by about 30% better than the more simple Minkowsky MBRs. Furthermore, the epsilon-range-join where each kNN-sphere is queried on its own is outperformed by its group-based counterparts. Recall that we set the default size of R to

$|R| = 100$. In this case even linearly scanning R for a kNN-sphere from S would not lead to a significant loss in performance. However, if $|R|$ becomes larger, the logarithmic performance of a hierarchical join becomes visible, and pruning subtrees earlier can further speed up the query such that the choice of an MBR falls clearly on the $CMBR$s.

Taking a look at the number of disk accesses (illustrated in Figure 8.2 center left) shows that performing an epsilon-range query for each kNN-sphere from S is not a good idea if R becomes large. In this scenario, many of the pages from R have to be reloaded during each of the $|S|$ queries, putting high load on the secondary storage. However, if neighboured values are queried in groups as done in the HGM and HGC algorithms, the cache can be used more efficiently even though these approaches mutually load R and S into the cache.

Varying the Size of S ($|S|$). Next we analyzed the effect of different values for $|S|$ regarding the CPU time (cf Figure 8.1 center right). Again, the hierarchical join variants perform best, followed directly by the sequential join. On the other hand, the UL variants perform worst. However, the UL variant that queries a single point from R during each iteration performs best since this enables highest pruning power. Most importantly the shape of the TPL algorithm looks totally different. It increases faster than the curve of the kNN-join variants first, but then intersects them for higher values of $|S|$, indicating the different asymptotic complexity of the different algorithms. Not taking into account the epsilon-range join, the asymptotic complexity of the sequential join is $O(|S|^2)$ while the complexity of the hierarchical join is on average $O(|S| \log(|S|))$ which is in accordance to the empirical results. To the best of our knowledge the theoretical runtime complexity of the TPL algorithm has not been analyzed so far, however the structure of the algorithm suggests a logarithmic average complexity for a single query point.

Thinking further, the point of intersection between TPL and the kNN-join variants is determined by the size of R since a new value in R introduces a new TPL-query – which is expensive – but only a single more point as a possible range-query result for the kNN algorithms – which is cheap. Specifically given a fixed set $|S|$, there exists a set R_e for which TPL and the hierarchical join algorithms show similar performance. In a practical setting the size of $|R_e|$ for a given set $|S|$ is of great value, since this knowledge can be used to decide whether to use TPL or a kNN join for performing the query. Therefore we ran an additional experiment showing the size of R_e for a given size of $|S|$ where both algorithms have an equal runtime. The results can be found in Figure 8.3a. Due to the quadratic complexity of the sequential join, the size of R_e increases (slightly) super-linear with the size of S. In contrast the size



(a) Equilibrium (b) Varying the cache size.

Figure 8.3: Given a set S of size $|S|$, the left figure visualizes the size of set R_e for which TPL and the kNN-based joins have the same computational performance on our test sets R and S (note that this might vary with other datasets). The right figure visualizes the results for varying cache size. Note the logarithmic scale on the y-Axis.

of R_e increases (slightly) sub-linear for the hierarchical kNN-join, suggesting this algorithm for large databases. Given a database with 1 million points, for the hierarchical join there is $R_e \approx 0.0007|S|$ and for the sequential join $R_e \approx 0.003|S|$. Thus as a rule of thumb it is strongly recommendable to use a self pruning based approach if $|R| > 0.01 \cdot |S|$.

Varying the Overlap Between R and S (o). Until now we assumed that the normally distributed sets of values R and S overlap completely, i.e. both sets have the same mean. This assumption is quite intuitive for example if we assume that R and S are drawn from the same distribution. In this experiment however, we aim at analyzing what happens if R and S are taken from different distributions. We do so by varying the mean of R and S, i.e. by decreasing the overlap of the two sets ($o = mean(R) - mean(S)$). First of all, the runtime performance of the join-based algorithms stays pretty much constant (cf Figure 8.1 bottom left). Although this might seem counter-intuitive in the beginning, recall that the kNN-join is only performed on the set S. Therefore the distance between R and S does not affect this cost of the RkNN join. Furthermore, the set R is small in our setting, hence the cost for the varying-range-join can be neglected. However only this varying-range-join is affected by a different overlap between R and S. All remaining variants can take severe profit from lower overlap between the sets R and S. All of them employ pruning to avoid descending into subtrees that do not have to be taken into account to answer the query. If the overlap decreases, subtrees can be pruned earlier (because the MINDIST between a subtree

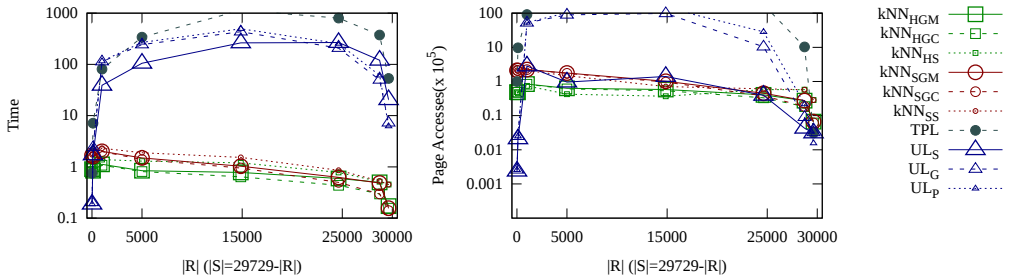


Figure 8.4: Performance (CPU time in seconds, page accesses), real dataset (HSV).

and the query point increases), reducing the CPU-time and number of page accesses (cf Figure 8.2 bottom left). Furthermore note that the approaches UL_P and UL_G consistently perform worse than UL_S over all values of o, showing that the overhead for a parallel tree traversal outweighs the power of early pruning in our setting.

Varying the Cache Size (c). For analyzing the cache size (see Figure 8.3b) we only provide an insight into the impact on the number of page accesses since the cache does not affect the CPU time. Both kNN-join variants are barely affected by the size of the cache, the gain of all of these approaches is usually less than 30%. In contrast, the mutual pruning based algorithms behave by over an order of magnitude (TPL) and two orders of magnitude (UL) better for larger cache sizes. The results show that in a scenario where the accessible memory is large (about 10% of the index), the UL algorithms can be a useful choice. Although they show a higher computational complexity, this disadvantage is compensated by the low number of disc accesses performed by this group of algorithms. If the cache size is relatively small, e.g. due to a multi-user environment, the kNN-based algorithms are the matter of choice.

Note that in order to evaluate the performance of the approaches for larger datasets, we conducted the same experiment again, however with 10^6 points in the database, a page size of 8192, and for cache sizes in range $[2^{15}, 2^{18}]$. Although the overall number of page access clearly increases with larger datasets, the relative performance of the different groups of algorithms and their behaviour did not change significantly compared to Figure 8.3b.

Varying the Page Size (p). The effect of an increasing page size is twofold. While the UL and hierarchical kNN-join approaches take mainly profit from a larger page size, the graph of the TPL and sequential kNN algorithms show a more or less increasing trend regarding the CPU cost (see Figure 8.1 bottom right). For the sequential join, if the page size grows too

big, many pages have to be visited since they fall into the kNN-circles of a query point. Concerning the number of disk accesses (see Figure 8.2 bottom right), the kNN-join variants take profit from larger page sizes. If the page size is small, the number of points processed during one iteration of the join is small as well, since one iteration computes the kNN for one leaf from S. Therefore many iterations are necessary for computing the whole join result, and each of these iterations involves reloading pages into the cache.

8.2 Real Data Experiments

HSV Dataset. Now let us take a look at experiments driven with real data. As an input, we employed 3D HSV color feature vectors extracted from the CalTech data set[1]. We split the input data set containing 29639 feature vectors into two sets R and S such that $|R| + |S| = 29639$, varying the size of R. The results can be found in Figure 8.4. Notice the two different behaviours of the self- and mutual pruning techniques. Concerning the CPU time, the self pruning approaches show better performance if R is large and S is small. The reason for this behaviour is that a self-join is always a quite expensive operation. For the hierarchical kNN self-join the complexity in the best case is about $O(|S| \log(|S|))$ while for the sequential join the complexity is $O(|S|^2)$. Combined with the varying-range-join we have an overall average complexity of $O(|S| \log(|S|) + |S| \log(|R|))$ or $O(|S|^2 + |S| \log(|R|))$. Now if $|S|$ is large (and $|R|$ can be neglected), this leads to a complexity of $O(|S| \log(|S|))$ and $O(|S|^2)$, respectively. On the other hand, if $|S|$ can be neglected, we have a complexity of $O(\log(|R|))$ in both cases. This also explains why the hierarchical and sequential techniques show the same performance if $|S|$ becomes small. For TPL, the behaviour is different: its complexity in $|S|$ is sub-linear but for each $r \in R$ a separate query has to be performed, such that its complexity increases linear with $|R|$. Therefore the performance of TPL is better for small R (and large S), but worse for large R (and small S). For the UL approaches the results can be explained equivalently. Further note that the the UL_S approach in this scenario shows a significantly lower number of page accesses than UL_G and UL_P

Post Office Dataset. As a second real dataset we employed a set of 123,593 post offices in the north-eastern united states.[2] The set is clustered in the metropolitan areas, containing further noise in the rural areas. We included this dataset as it evaluates the applicability of the proposed

[1] http://www.vision.caltech.edu/Image_Datasets/Caltech256/

[2] www.rtreeportal.org/

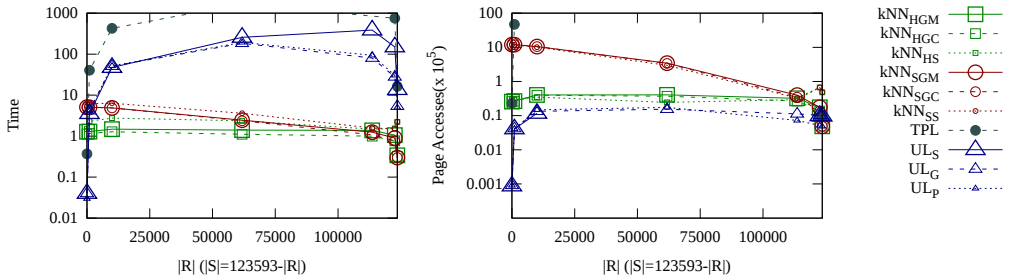


Figure 8.5: Performance (CPU time, page accesses), real dataset (post office).

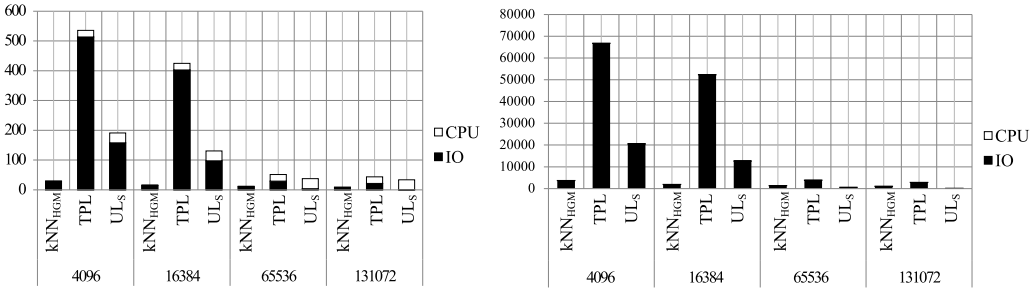


Figure 8.6: CPU and IO time in seconds when varying the cache size. Left: SSD, right: HDD

approaches for geospatial applications. Although the overall behaviour of all algorithms on this two-dimensional dataset is similar to the behaviour on the mentioned three-dimensional data (see Figure 8.5), the UL algorithms perform better in this special case concerning the number of page accesses. The observation that the UL approaches can outperform other algorithms on low-dimensional data has already been made when analyzing the synthetic data.

8.3 Comparing CPU-Cost and IO-Cost

Last but not least let us shortly analyze whether the techniques are IO- or CPU-bound. For this purpose, we reuse the results from the experiment where we varied the cache size. From the number of IO operations we computed the resulting IO time for both a solid state drive and a hard disk drive. We assumed an SSD with page access time of $0.1ms$, for a HDD we assumed a combined seek time and latency of 13ms. These values have been taken from [79]. The graph in Figure 8.6 shows the amount of IO time and CPU time for sample approaches of the self- and mutual pruning approaches for

both solid state (SSD) and hard disk (HDD) drives. Note that in most of the evaluated scenarios all of the algorithms are IO-bound, i.e. the IO-time is larger than the CPU time needed for evaluating a join. However there are small differences: in the case of an SSD the UL approaches (which have been optimized to greatly reduce the number of page accesses) become CPU-bound if the cache size becomes very large. In the case of a hard disk, all approaches are IO-bound even for very large cache sizes.

Chapter 9

Conclusion

In this work, we addressed the problem of running multiple RkNN-queries at a time, i.e. a RkNN join. For this purpose, we formally classified variants of the RkNN join, including monochromatic and bichromatic scenarios as well as self joins. Furthermore, we proposed algorithms for RkNN join queries based on the well-known self and mutual pruning paradigms for single RkNN queries. We have shown that, in several scenarios, classic algorithms for performing single RkNN queries do not yield the expected performance, and that our newly proposed algorithms usually lead to better results. In addition, according to our experiments, RkNN join algorithms based on the self pruning paradigm show better results than algorithms following the mutual pruning paradigm. Our experiments further indicate that not all scenarios ask for using RkNN join algorithms. To summarize the contribution of our experiments, we suggest different scenarios for RkNN query processing:

- If the database is relatively static and many RkNN queries are run expecting low latency, preprocessing as used in [15, 60] is a useful choice since self pruning is usually more selective than mutual pruning. This avoids computing the kNN-spheres each time a query is performed as done with our join algorithms.

- If the database is dynamic and RkNN queries are performed in a bulk, our proposed join algorithms, especially the self pruning variant clearly performs best. The technique is also preferable if an RkNN join of intermediate query results has to be computed. Furthermore, our self-pruning algorithm could be easily adapted to performing RkNN joins when each of the objects in R has another value of k. For low-dimensional data, such as 2D geo-spatial data, our mutual pruning RkNN join algorithm is the matter of choice.

- If the database is highly dynamic and single RkNN-queries have to be performed immediately, single RkNN queries based on mutual pruning like TPL [64] are the method of choice. This is also the case where the database is static but the self join of the whole set S cannot be justified by a low number of RkNN queries.

Part III

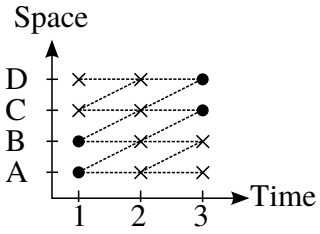
Nearest Neighbor Queries on Uncertain Spatio-Temporal Data

Chapter 10

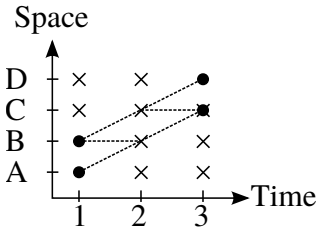
Introduction

In the following part of this thesis we shift our focus from similarity queries on certain data to an uncertain setting, aiming at providing efficient solutions for similarity search on uncertain spatio-temporal data. The challenges in this context are manifold, necessitating the development of suitable query algorithms and index structures. First, we have to address the combination of both spatial and temporal data dimensions. This problem is further complicated by a second challenge, the uncertainty of data, which becomes even more difficult in the context of temporal data. The combination of this *spatio-temporal* and *uncertain* nature of data imposes a third research challenge, namely defining query predicates, models and algorithms that capture the temporal *dependencies* of data during query processing, providing accurate results to the user under uncertainty.

To illuminate the problem of temporal dependencies in the context of uncertain spatio-temporal data, consider Figure 10.1 that shows a state space of one-dimensional locations (y-axis) over time (x-axis). If we have information on the spatial position of an object at a given point in time, this knowledge also affects preceding and succeeding timesteps, at least in our physical world. The object in Figure 10.1 is either in state A or B at time 1 (denoted by black circles, Figure 10.1a), but it is unknown where exactly the object is. At time 2, we assume the position of the object to be unknown, and at time 3, we assume that the object is in state C or D. Given additional knowledge on the underlying state space (dashed lines), namely which states can be reached from another state at a given timestep, we can see that the object's position depends on the assumption on its previous location (Figure 10.1b): While it is generally possible that the object is either in state C or D, given that we know that it was in state A previously, it cannot be located in state D. Such restrictions on the motion patterns of objects are inherent to spatio-temporal problems, for example, but not solely,



(a) Observations and Transition Probabilites.



(b) Merging Knowledge from State and Transitions.

Figure 10.1: Uncertainty in a Spatio-Temporal Context.

due to physical constraints: Real-world objects can only travel with limited speed and therefore the spatial position of an object at consecutive points in time is highly correlated. Therefore we are facing not only the problem that the probability distribution of an object in the past affects the probability distribution of an object in the future and vice versa, but also that the probability distribution in the future is conditioned to the objects' state in the past. This even holds for very simple uncertainty models such as the Markov model. Therefore, during query processing we cannot simply consider each timestep separately, as this would neglect such temporal dependencies.

The necessity of considering uncertainty during query evaluation originates from the physical limitations of the sensing devices or limitations of the data collection process. Specifically, it is usually not possible to *continuously* capture the position of an object for each point of time. In an indoor tracking environment where the movement of a person is captured using static RFID sensors, the position of the people in-between two successive tracking events is not available ([80]). The same holds for geosocial network (GSN) applications, where users have recently been enabled to publicly share trajectories, such as bike routes[1], tourist routes[2] and GPS trajectories[3]. In many applications, the frequency of data collection is decreased to save resources such as battery power and wireless network traffic. Examples of trajectories with a relatively low frequency can be found on Bikely. Furthermore, traditional check-in data of GSN users often shows a frequency high enough to allow inference of a user's position in between discrete check-ins. Incomplete (location, time) data is also collected in mobile object tracking applications. For example, in the T-Drive dataset ([81]) which consists of GPS-logs of

[1]http://www.bikely.com/
[2]http://www.everytrail.com/
[3]http://www.gpsxchange.com, http://www.gpsshare.com/

taxis in Beijing, the time between two successive GPS measurements ranges from two seconds up to several minutes. In the GeoLife dataset ([82]), GPS observations of mobile users are logged frequently, usually every 1-5 seconds per point, while some observations still have a lower sampling rate. All these datasets create a common challenge of interpolating the position of a user in-between discrete observations. In-between these observations the exact values are not explicitly stored in the database and are thus uncertain from the database perspective.

The efficient management of spatio-temporal data is of great interest in a plethora of application domains: from structural and environmental monitoring and weather forecasting, through disaster/rescue management and remediation, to Geographic Information Systems (GIS) and traffic control and information systems.

In this work, we consider a database D of uncertain moving object trajectories, where for each trajectory there is a set of observations for only some of the history timestamps. Thus, the entire trajectory of an object is described by a time-dependent random variable, i.e., a stochastic process. Based on this definition of uncertain trajectories, we address the problems of probabilistic nearest neighbor (PNN) queries and probabilistic reverse nearest neighbor (PRNN) queries. Research results from this section have been previously published in [13, 51, 52], and have also been included in [49, 50]. For an overview over the contributions of the author of this thesis to this work we refer to Chapter 4.

Probabilistic Nearest Neighbor Queries Given a reference state or trajectory q and a time interval T, we define *probabilistic* nearest neighbor (PNN) query semantics, which are extensions of nearest neighbor queries in trajectory databases [83, 84, 85, 86]. Specifically, a P$\exists$NNQ (P$\forall$NNQ) query retrieves all objects in D, which have sufficiently high probability to be the NN of q at one time (at the entire set of times) in T; a probabilistic *continuous* NN (PCNNQ) query finds for each object $o \in D$ the time subsets T_i of T, wherein o has high enough probability to be the NN of q at the entire set of times in T_i. Note that to the best of our knowledge this is the first approach that tackles the PNN query problem correctly in consideration of Possible Worlds Semantics (PWS).

PNN queries find several applications in analyzing historical trajectory data. For example, consider a geosocial network where users can publish their current spatial position at any time by so-called check-ins. For a historical event, users might want to find their nearest friends during this event, e.g. to share pictures and experiences. As another application example, consider

GPS-tracked taxi cars as given in the T-Drive dataset [81] where PNN queries can be used for analysis tasks like the assessment of taxi-client assignment procedures.

The main contributions of our work in the context of probabilistic nearest neighbor queries on uncertain spatio-temporal data are as follows:

- The proposal of P∃NNQ, P∀NNQ and PCNNQ predicates aiming at providing query results for uncertain spatio-temporal nearest neighbor queries in accordance to PWS.
- A thorough theoretical complexity analysis for variants of probabilistic NN query problems. Specifically we show that the proposed query semantics are computationally difficult to solve and therefore approximate solutions are mandatory to apply these technique in real-world settings.
- A sampling-based approximate solution for all of the proposed PNN problems based on Bayesian inference.
- A thorough experimental evaluation of the proposed concepts on real and synthetic data.

Probabilistic Reverse Nearest Neighbor Queries Additionally we address the problem of performing Probabilistic Reverse Nearest Neighbor (PRNN) queries on uncertain spatio-temporal data by extending our research results from PNN queries. Given a query q, a reverse nearest neighbor query returns the objects in the database having q as one of its nearest neighbors. Xu et al. [87] were the first researchers to address RNN queries on uncertain spatio-temporal data under the Markov model and showed how to answer an "interval reverse nearest neighbor query" [87]. However, as we will see later, the solution presented in [87] does not consider possible worlds semantics. In this work we fill this research gap by proposing algorithms to answer reverse nearest neighbour queries according to PWS. The contributions of this work on probabilistic reverse nearest neighbor queries can be summarized as follows:

- We introduce two query definitions for the reverse nearest neighbor problem on uncertain trajectory data, the P∃RNNQ and P∀RNNQ queries which are consistent with our previously defined P∃NNQ and P∀NNQ queries.
- We demonstrate solutions to answer the queries we defined efficiently and, most importantly, according to possible worlds semantics.

- We provide an extensive experimental evaluation of the proposed methods both on synthetic and real world datasets.

Structure of this Part Part III of this thesis is structured as follows. First, in Chapter 11, we address the preliminaries of PNN and PRNN queries, including a formal problem definition (Section 11.1) and an overview of related work (Section 11.2). The following Chapter 12 dives deeper into probabilistic nearest neighbor queries. A complexity analysis, approximate solutions and pruning techniques of the proposed query semantics are provided in Sections 12.1-12.3. An extensive experimental evaluation of the proposed techniques is presented in Section 12.4. Chapter 13 concentrates on reverse nearest neighbor queries. Section 13.1 introduces algorithms for the queries proposed in the problem definition. An extensive experimental evaluation follows in Section 13.2. Chapter 14 concludes this part.

Chapter 11

Preliminaries

11.1 Problem Definition

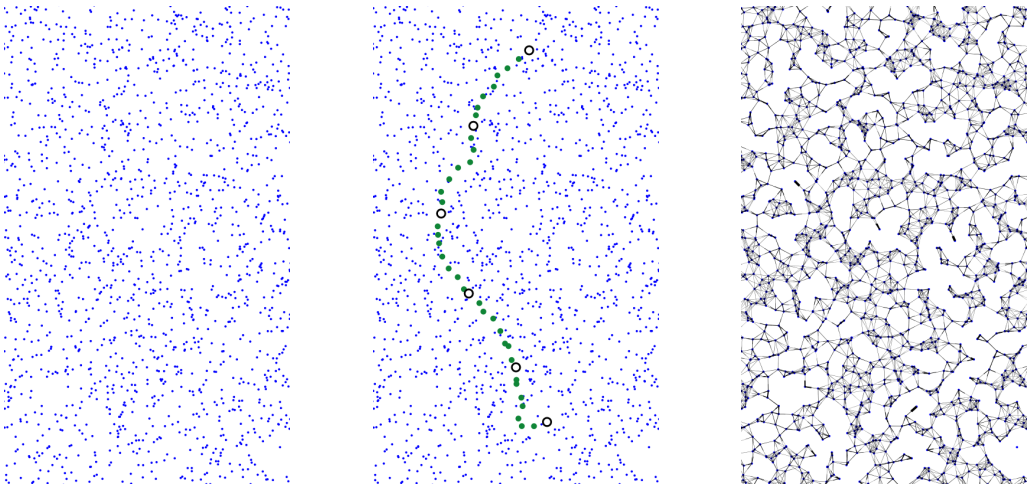
A spatio-temporal database D stores triples (o_i, *time*, *location*), where o_i is a unique object identifier, *time* $\in \mathbb{T}$ is a point in time and *location* $\in \mathbb{S}$ is a position in space. Semantically, each such triple corresponds to an *observation* that object o_i has been seen at some *location* at some *time*. In D, an object o_i can be described by a function $o_i(t) : \mathbb{T} \rightarrow \mathbb{S}$ that maps each point in time to a location in space; this function is called *trajectory*.

In this work, we assume a discrete time domain $\mathbb{T} = \{0, \ldots, n\}$. Thus, a trajectory becomes a sequence, i.e., a function on a discrete and ordinal scaled domain. Furthermore, we assume a discrete state space of possible locations (*states*): $\mathbb{S} = \{s_1, ..., s_{|\mathbb{S}|}\} \subset \mathbb{R}^d$, i.e., we use a finite alphabet of possible locations in a d-dimensional space. The way of discretizing space is application-dependent: for example, in traffic applications we may use road crossings, in indoor tracking applications we may use the positions of RFID trackers and rooms, and for free-space movement we may use a simple grid or a random distribution of states for discretization. Figure 11.1a visualizes such a randomly distributed state space with $s_i \in \mathbb{S}$ shown as blue dots.

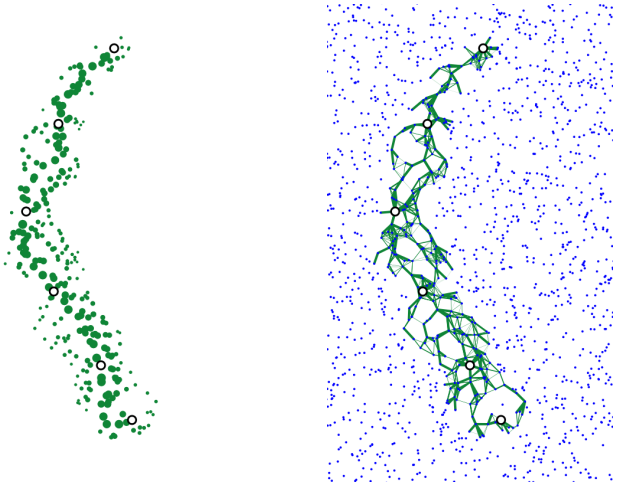
11.1.1 Uncertain Trajectory Model

Let D be a database containing the trajectories of $|D|$ uncertain moving objects $\{o_1, ..., o_{|D|}\}$. For each object o in D we store a set of observations $\Theta^o = \{\langle t_1^o, \theta_1^o\rangle, \langle t_2^o, \theta_2^o\rangle, \ldots, \langle t_{|\Theta^o|}^o, \theta_{|\Theta^o|}^o\rangle\}$ where $t_i^o \in \mathbb{T}$ denotes the time and $\theta_i^o \in \mathbb{S}$ the location of observation Θ_i^o. W.l.o.g. let $t_1^o < t_2^o < \ldots < t_{|\Theta^o|}^o$. Note that the location of an observation is assumed to be certain, while the location of an object between two observations is uncertain. A certain trajectory of a database object o is shown in Figure 11.1b where the (unknown) ground

truth is visualized as green dots (the time dimension is not shown for the sake of simplicity), and observations θ_i are visualized as white circles with black boundary.



(a) State Space (b) Ground Truth (c) A-priori Transitions



(d) State Probabilities (e) Transition Probabilities

Figure 11.1: Model Visualization (best viewed in color)

According to [41], we can interpret the location of an uncertain moving object o at time t as a realization of a random variable $o(t)$. Given a time interval $[t_s, t_e]$, the sequence of uncertain locations of an object is a family of

correlated random variables, i.e., a stochastic process.

This definition allows us to assess the probability of a possible trajectory, i.e., the realization of the corresponding stochastic process. In this work we follow the approaches from [41, 40, 87] and employ the first-order Markov chain model as a specific instance of a stochastic process. The state space of the model is the spatial domain $\mathbb{S}$. State transitions are defined over the time domain $\mathbb{T}$. In addition, the Markov chain model is based on the assumption that the position $o(t+1)$ of an uncertain object o at time $t+1$ only depends on the position $o(t)$ of o at time t. Clearly, the assumption is overly restrictive, as for example vehicles on a road network will never follow a first-order Markov chain. Such vehicles generally follow a best path (e.g. the shortest path or the path having the most beautiful landscape, etc.). Nevertheless, such a simplified model can, as we will see in our experimental evaluation, accurately model the set of possible trajectories that a vehicle may have taken between two discrete observations. Theoretically, this high accuracy can be explained by combining both observation information and the Markov model into a new model.

The probability $M_{ij}^{o}(t) := P(o(t+1) = s_j | o(t) = s_i)$ is the *transition probability* of a given object o from state s_i to state s_j at a given time t. Transition probabilities are stored in a matrix $M^o(t)$, called *transition matrix of object o at time t*. In general, every object o might have a different transition matrix, and the transition matrix of an object might vary over time. We can imagine the transition probabilities of an uncertain object as a (directed) graph, with weights on an edge between s_i and s_j denoting the probability of an object o transitioning from state s_i to state s_j at a given point in time, given it was previously in state s_i. Figure 11.1c visualizes the transition probabilities of our example, with the thickness of an edge denoting the probability of a state change.

Further, let $\vec{s}^{o}(t) = (s_1, \ldots, s_{|\mathbb{S}|})^T$ be the distribution vector of a given single object o at time t, where $\vec{s}_i^{o}(t) = P(o(t) = s_i)$, i.e. each element of the vector describes o's probability of visiting the state s_i at time t. Without any further knowledge (from observations) the distribution vector $\vec{s}^{o}(t+1)$ can be inferred from $\vec{s}^{o}(t)$ by applying the following formula: $\vec{s}^{o}(t+1) = M^o(t)^T \cdot \vec{s}^{o}(t)$.

The traditional Markov model [41] uses forward probabilities only. In Section 12.2, we use a Bayesian inference approach, to condition this *a-priori Markov chain* to an adapted *a-posteriori Markov chain* which also considers all observations of an object. Figure 11.1d visualizes the state probabilities $\vec{s}_t$ of our running example considering both observations in the past and in the future, and aggregating over all points in time t (always drawing the maximum probability for a given state). In this visualization, larger dots

represent higher probabilities for an object being located in a given state. It is clearly visible that the states on the closest path between consecutive observation have highest probability after integrating observations into the a priori Markov model, while states further away from this closest path have lower probabilities of being accessed. An additional aggregation of the transition probabilities of the same object, given all observations of the object, is shown in Figure 11.1e. This a-posteriori model, visualized by its state and transition probabilities, combines the information from the objects observations (Figure 11.1b) and the global a priori transition matrix (Figure 11.1c).

11.1.2 Nearest Neighbor Queries

In Chapter 12 of this work we consider three types of time-parameterized nearest neighbor queries that take as input a certain reference state or trajectory q and a set of timesteps T. All of the proposed query predicates, i.e. P∀NNQ, P∃NNQ and PCNNQ, are inspired by corresponding nearest neighbor semantics on certain trajectories, as defined in [83, 88, 86]. The query definitions also follow the definition of probabilistic window queries from [41]. Note that q can be both a state or a trajectory, since a query state is simply a trivial query trajectory.

Definition 10 (P∃NN Query)**.** *A probabilistic ∃ nearest neighbor query retrieves all objects $o \in D$ which have a sufficiently high probability to be the nearest neighbor of q for at least one point of time $t \in T$, formally:*

$$P\exists NNQ(q, D, T, \tau) = \{o \in D : P\exists NN(o, q, D, T) \geq \tau\}$$

where

$$P\exists NN(o, q, D, T) = P(\exists t \in T : \forall o' \in D \backslash o : dist(q(t), o(t)) \leq dist(q(t), o'(t)))$$

and $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a distance function, typically the Euclidean distance.

This definition is an extension of the spatio-temporal query proposed in [83] to the case of uncertainty. Given the application for mining taxi trajectories from the introduction, a $P\exists NNQ(q, D, T, \tau)$ query returns all taxis having a probability of at least τ of having been the closest cab to a given query for at least one query timestep.

In addition to the $P\exists NNQ$, we consider NN queries with the ∀ quantifier, which have also been proposed in [83] for crisp trajectory data.

Definition 11 (P∀NN Query)**.** *A probabilistic ∀ nearest neighbor query retrieves all objects $o \in D$ which have a sufficiently high probability ($P\forall NN$) to be the nearest neighbor of q for the entire set of timestamps T, formally:*

$$P\forall NNQ(q, D, T, \tau) = \{o \in D : P\forall NN(o, q, D, T) \geq \tau\}$$

where

$$P\forall NN(o, q, D, T) = P(\forall t \in T : \forall o' \in D \backslash o : dist(q(t), o(t)) \leq dist(q(t), o'(t)))$$

and $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *a distance function, typically the Euclidean distance.*

In the running taxi-tracking application $P\forall NNQ(q, D, T, \tau)$ returns all taxis having a probability of at least τ of having been the closest cab to the query for all query timesteps. The main difference between Definition 10 and Definition 11 is that a $P\exists NNQ(q, D, T, \tau)$ requires a candidate object to be the nearest neighbor of q for at least one point of time in T to qualify as a result, while $P\forall NNQ(o, q, D, T)$ requires a candidate object to remain the nearest neighbor for the whole duration of T.

In addition to these semantics for probabilistic nearest neighbor queries we now introduce a *continuous* query type which intuitively extends the spatio-temporal continuous nearest neighbor query [88, 86] to apply on uncertain trajectories.

Definition 12 (PCNN Query)**.** *A probabilistic continuous nearest neighbor query retrieves all objects* $o \in D$ *together with the set of timestep sets* $\{T_i\}$ *where in each* T_i *the object has a sufficiently high probability to be always the nearest neighbor of* $q(t)$*, formally:*

$$PCNNQ(q, D, T, \tau) = \{(o, T_i) : o \in D, T_i \subseteq T, P\forall NN(o, q, D, T_i) \geq \tau\}.$$

Analogously to the CNN query definition [88, 86], in order to reduce redundant answers it makes sense to redefine the PCNN Query where we focus on results that maximize $|T_i|$*, formally:*

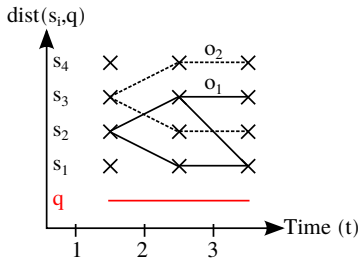
$$PCNNQ(q, D, T, \tau) =$$

$$\{(o, T_i) : o \in D, T_i \subseteq T, P\forall NN(o, q, D, T_i) \geq \tau$$

$$\wedge \forall T_j \supset T_i : P\forall NN(o, q, D, T_j) < \tau\}$$

Note that according to this definition, result sets $T_i \subseteq T$ do not have to be connected.

Example 1. *To illustrate the three query types, consider the scenario shown in Figure 11.2a consisting of a query trajectory and two uncertain database objects* $D = \{o_1, o_2\}$ *in a discretized space and time domain. For simplicity, whenever an object has two alternatives for choosing a possible state transition, each transition is assumed to have a probability of 0.5. These probabilities define the Markov chains of* o_1 *and* o_2*. Thus* o_1 *has three possible trajectories and* o_2 *has two possible trajectories, the probabilities of which are shown*



(a) Uncertain Trajectories.

object	trajectory	P(tr)
o_1 —	$tr_{1,1} = s_2, s_1, s_1$	0.5
o_1 —	$tr_{1,2} = s_2, s_3, s_1$	0.25
o_1 —	$tr_{1,3} = s_2, s_3, s_3$	0.25
o_2 ···	$tr_{2,1} = s_3, s_2, s_2$	0.5
o_2 ···	$tr_{2,2} = s_3, s_4, s_4$	0.5

(b) Possible Worlds.

Figure 11.2: Example Setup for PNN Queries

in Figure 11.2b. Using possible worlds semantics, any PNN query can naively be computed by considering all six possible combinations $(tr_{1,i}, tr_{2,j})$, $i \in \{1,2,3\}$, $j \in \{1,2\}$, *called possible worlds, of possible trajectories of objects* o_1 *and* o_2. *The total probability of all possible worlds where* o_2 *is closer to* q *than* o_1 *at any time, by definition, equals the probability* $P\exists NN(o_2, q, D, \{1,2,3\})$. *For this example these possible worlds are* $(tr_{1,2}, tr_{2,1})$ *and* $(tr_{1,3}, tr_{2,1})$. *Assuming object independence,* $P(tr_{1,i}, tr_{2,j})$ *of a possible world is given by the product* $P(tr_{1,i}) \cdot P(tr_{2,j})$ *yielding* $P\exists NN(o_2, q, D, \{1,2,3\}) = P(tr_{1,2}) \cdot P(tr_{2,1}) + P(tr_{1,3}) \cdot P(tr_{2,1}) = 0.25 \cdot 0.5 + 0.25 \cdot 0.5 = 0.25$. *Accordingly the probability* $P\forall NN(o_1, q, D, \{1,2,3\}) = 0.75$ *can be computed by the sum of the probabilities* $P(tr_{1,1}, tr_{2,1})$, $P(tr_{1,1}, tr_{2,2}), P(tr_{1,2}, tr_{2,2})$ *and* $P(tr_{1,3}, tr_{2,2})$ *of worlds where* o_1 *is always closer to* q *than* o_2. *A* $PCNNQ(q, D, \{1,2,3\}, 0.1)$ *will return the object* o_1 *together with the interval* $\{1,2,3\}$ *and* o_2 *together with the interval* $\{2,3\}$, *as in these intervals, the respective objects have a probability of at least* 0.1 *to be closest to* q.

In this example, exact probabilities have been computed by explicit consideration of all possible worlds. However, since the number of possible trajectories grows exponentially large in the number of time transitions, and the total number of possible worlds is furthermore exponential in the number of objects, the challenge of this work is to find a more efficient approach to approximate the same nearest neighbor probabilities without enumeration of all possible worlds.

11.1.3 Probabilistic Reverse Nearest Neighbor Queries

In the following we also define two types of probabilistic reverse nearest neighbor queries. These definitions conceptually follow the ones of probabilistic nearest neighbor queries defined previously. Again we assume that the RNN query takes as input a set of timestamps T and either a single state or a

(certain) query trajectory q.

Definition 13 (P∃RNN Query)**.** *A probabilistic ∃ reverse nearest neighbor query retrieves all objects $o \in D$ having a sufficiently high probability ($P\exists RNN$) to be the reverse nearest neighbor of q for at least one point of time $t \in T$, formally:*

$$P\exists RNNQ(q, D, T, \tau) = \{o \in D : P\exists RNN(o, q, D, T) \geq \tau\}$$

where

$$P\exists RNN(o, q, D, T) = P(\exists t \in T : \forall o' \in D \backslash o : dist(o(t), q(t)) \leq dist(o(t), o'(t)))$$

and $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a distance function, typically the Euclidean distance.

This query returns all objects from the database having a probability greater τ to have q as their probabilistic ∃ nearest neighbor. In addition to this ∃ query, we consider RNN queries with the ∀ quantifier:

Definition 14 (P∀RNN Query)**.** *A probabilistic ∀ reverse nearest neighbor query retrieves all objects $o \in D$ having a sufficiently high probability ($P\forall RNN$) to be the reverse nearest neighbor of q for the entire set of timestamps T, formally:*

$$P\forall RNNQ(q, D, T, \tau) = \{o \in D : P\forall RNN(o, q, D, T) \geq \tau\}$$

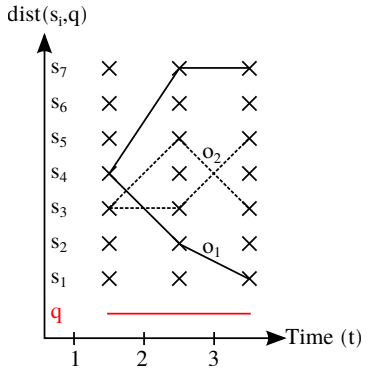
where

$$P\forall RNN(o, q, D, T) = P(\forall t \in T : \forall o' \in D \setminus o : d(o(t), q(t)) \leq d(o(t), o'(t)))$$

and $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a distance function, typically the Euclidean distance.

The above definition returns all objects from the database which have a probability greater τ to have q as their probabilistic ∀ nearest neighbor.

Example 2. *To illustrate the differences between the proposed RkNN query predicates consider the example in Figure 11.3. Here the query consists of a static query point and the two objects o_1 and o_2 each have 2 possible trajectories. o_1 (solid line) follows the lower trajectory ($tr_{1,1}$) with a probability of 0.4 and the upper trajectory ($tr_{1,2}$) with a probability of 0.6. o_2 (dotted line) follows trajectory $tr_{2,1}$ with a probability of 0.2 and trajectory $tr_{2,2}$ with a probability of 0.8. For query object q and the query interval $T = [2, 3]$, we can compute the probability for each object to be probabilistic reverse nearest neighbor of q. Specifically for o_1 the probability $P\exists RNN(o_1, q, D, T) = 0.4$ since whenever o_1 follows $tr_{1,1}$ then at least at $t = 3$, o_1 is RNN of q. The probability for $P\forall RNN(o_1, q, D, T)$ in contrast is 0.32 since it has to hold*



(a) Uncertain Trajectories.

object	trajectory	P(tr)
o_1 —	$tr_{1,1} = s_4, s_2, s_1$	0.4
o_1 —	$tr_{1,2} = s_4, s_7, s_7$	0.6
o_2 ···	$tr_{2,1} = s_3, s_3, s_5$	0.2
o_2 ···	$tr_{2,2} = s_3, s_5, s_3$	0.8

(b) Possible Worlds.

Figure 11.3: Example Setup for PRNN Queries

that o_1 follows $tr_{1,1}$ (this event has a probability of 0.4) and o_2 has to follow $tr_{2,2}$ (this event has a probability of 0.8). Since both events are mutually independent we can just multiply the probabilities to obtain the final result probability. Regarding object o_2 we can find no possible world (combination of possible trajectories of the two objects) where o_2 is always ($T = [2, 3]$) RNN, thus $P\forall RNN(o_2, q, D, T) = 0$. However $P\exists RNN(o_2, q, D, T) = 0.6$ since whenever o_1 follows $tr_{1,2}$ then o_2 is RNN either at $t = 2$ or at $t = 3$.

An important observation is that it is not possible to compute these probabilities by just considering the snapshot RNN probabilities for each query time stamp individually. For example the probability for o_2 to be RNN at time $t = 2$ is 0.12 (the possible world where objects follow $tr_{1,2}$ and $tr_{2,1}$) and the probability at $t = 3$ is 0.48 (the possible world where objects follow $tr_{1,2}$ and $tr_{2,2}$). However these events are not mutually independent and thus we cannot just multiply the probabilities to obtain the probability that object o_2 is RNN at both points of time (note that the true probability of this event is $P\forall RNN(o_2, q, D, T) = 0$).

11.2 Related Work

Within the last decade, a considerable amount of research effort has been put into query processing in trajectory databases (e.g. [86, 89, 90, 91, 84]). In these works, the trajectories have been assumed to be certain, by employing linear [86] or more complex [89] types of interpolation to supplement sparse observational data. However, employing linear interpolation between consecutive observations might create impossible patterns of movement, such as

cars travelling through lakes or similar impossible-to-cross terrain. Furthermore, treating the data as uncertain and answering probabilistic queries over them offers better insights[1].

Uncertain Trajectory Modeling. In the past, several models of uncertainty paired with appropriate query evaluation techniques have been proposed for moving object trajectories (e.g. [92, 93, 14, 41]). Many of these techniques aim at providing conservative bounds for the positions of uncertain objects. This can be achieved by employing geometric objects such as cylinders [93, 14] or beads [94] as trajectory approximations. While such approaches allow to answer queries such as "is it possible for object o to intersect a query window q", they are not able to assign probabilities to these events conforming to possible worlds semantics.

Other approaches use independent probability density functions (pdf) at each point of time to model the uncertain positions of an object [95, 14, 92]. However, as shown in [41], this may produce wrong results (not in accordance with possible world semantics) for queries referring to a time interval because they ignore the temporal dependence between consecutive object positions in time. To capture such dependencies, recent approaches model the uncertain movement of objects based on stochastic processes. In particular, in [96, 41, 80, 87], trajectories are modeled based on Markov chains. This approach permits correct consideration of possible world semantics in the trajectory domain.

Nearest Neighbor Queries. In the context of certain trajectory databases there is not a common definition of nearest neighbor queries, but rather a set of different interpretations. In [83], given a query trajectory (or spatial point) q and a time interval T, a NN query returns either the trajectory from the database which is closest to q during T or for each $t \in T$ the trajectory which is closest to q. The latter problem has also been addressed in [84]. Similarly, in [97], all trajectories which are nearest neighbors to q for at least one point of time t are computed.

Other approaches consider *continuous* nearest neighbor (CNN) semantics, definition of this query varies between publications [85, 88, 86]. CNN have also been addressed for objects with uncertain velocity and direction in [98]; the solutions proposed only find possible results, but not result probabilities. Solutions for road network data were also proposed for the case where the velocities of objects are unknown [99]. Furthermore, [14, 100] extended the problem of continuous kNN queries (on historical search) to an uncertain setting, serving as important preliminary work, however, based on a model which is not capable to return answers according to possible world semantics.

[1] http://infoblog.stanford.edu/2008/07/why-uncertainty-in-data-is-great-posted.html

Reverse Nearest Neighbour Queries. For an overview over general research on (certain) reverse nearest neighbor queries we refer to Section 6.2 in Part II. Recently, probabilistic reverse nearest neighbor queries have gained significant attention [101, 102, 103]. The solution proposed by Chen et al. [101] aims at processing PRNN queries on uncertain objects represented by continuous probability density functions (PDFs). In contrast, Cheema et al. [102] provided solutions for the discrete case. Regarding reverse nearest neighbor processing using the Markov model, to the best of our knowledge, there exists only one work so far which addresses interval reverse nearest neighbor queries [87]. The approach basically computes for each point of time in the query interval separately the probability for each object $o \in D$ to be the RNN to the query object. Then for each object o, the number of times where o has the highest probability to be RNN is counted. The object with the highest count is returned. Upon investigation, this approach has certain drawbacks. First, the proposed algorithm is not in accordance with possible worlds semantics, since successive points of time are considered independently. Second, the paper does not show how to incorporate additional observations (besides the first appearance of an object). In our research we aim filling this research gap and solving the two said issues by proposing algorithms following possible world semantics that allow incorporating observations.

Chapter 12

Nearest Neighbor Queries

12.1 Theoretical Analysis

This section theoretically studies the runtime complexity of the P∃NNQ, P∀NNQ and PCNNQ queries.

12.1.1 The P∃NN Query

In a P∃NNQ query, for any candidate object $o \in D$, Definition 10 requires the probability $P\exists NN(o, q, D, T)$. However, the following lemma shows that this probability is hard to compute.

Lemma 1. *The computation of $P\exists NN(o, q, D, T)$ is NP-hard.*

Proof. $P\exists NN(o, q, D, T)$ is equal to $1-P(\neg\exists t \in T, \forall o' \in D\backslash o : d(q(t), o(t)) \leq d(q(t), o'(t)))$. We will show that deciding if there exists a possible world for which the expression:

$$\neg\exists t \in T, \forall o' \in D \setminus o : d(q(t), o(t)) \leq d(q(t), o'(t)) \tag{12.1}$$

is satisfied is an NP-hard problem. (Note that this is a much easier problem than computing the actual probability.) Specifically, we will reduce the well-known NP-hard k-SAT problem to the problem of deciding on the existence of a possible world for which Expression 12.1 holds.

For this purpose, we provide a mapping to convert a Boolean formula in conjunctive normal form to a Markov chain modeling the decision problem of Expression 12.1 in polynomial time. Thus, if the decision problem could be computed in PTIME, then k-SAT could also be solved in PTIME, which would only be possible if P=NP. A k-SAT expression E is based on a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. The *literal* l_i of a variable x_i is either

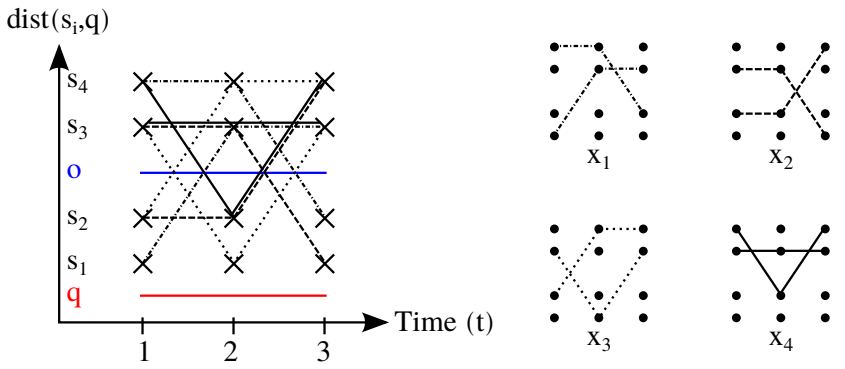


Figure 12.1: An example instance of our mapping of the 3-SAT problem to a set of Markov chains.

x_i or $\neg x_i$ and a *clause* $c = \bigvee_{x_i \in \mathbb{C}} l_i$ is a disjunction of literals where $\mathbb{C} \subseteq X$ and $|\mathbb{C}| < k$. Then E is defined as a conjunction of clauses: $E = c_1 \wedge c_2 \wedge \ldots \wedge c_m$.

For our mapping, we will consider a simplified version of the P∃NN problem, specifically (1) q is a certain point, (2) o is a certain point and (3) the state space $\mathbb{S}$ of possible locations only includes 4 states. As illustrated in Figure 12.1, compared to o, states s_1 and s_2 are closer to q and states s_3 and s_4 are further from q.[1] Therefore, if an uncertain object is at states s_1 or s_2 then o is not the nearest neighbor of q.

In our mapping, each variable $x_i \in X$ is equivalent to one uncertain object $o'_i \in D \setminus o$. Furthermore each disjunctive clause c_j is interpreted as an event happening at time $t = j$, i.e., the event c_1 happens at time $t = 1$, c_2 happens at time $t = 2$ etc. Each clause c_j can be seen as a disjunctive event that at least one object o'_i at time $t = j$ is closer to q than o (in this case, c_j is *true*). Therefore, the conjunction of all these events, i.e. expression $E = \bigwedge_{1 \leq j \leq m} c_j$, becomes true if the set of variables is chosen in a way that at each point in time, compared to o, at least one object is closer to q; this directly represents Expression 12.1.

Let l_i^j be the literal of variable x_i in clause c_j. Based on the above discussion, we are able to construct for each object o'_i two possible trajectories (worlds). The first one, based on the assumption that x_i is true, transitions between states s_2 (if l_i^j = true) and s_4 (if l_i^j = false). The second one, based on the assumption that x_i is set to false, transitions between states s_1 (if l_i^j = true) and s_3 (if l_i^j = false). Since these two trajectories can never be in the same state it is straightforward to construct a time-inhomogeneous Markov

[1] The states of o and q are omitted for the sake of simplicity.

chain $M^o(t)$ for each object o_i' and each timestamp j. This construction is, however, not complete: in k-SAT, not every variable x_i (corresponding to o_i') is contained in each term c_j which does not correspond to our setting, since an uncertain object has to be *somewhere* at each point in time. The interpretation of an absent variable in a given clause is simply that object o_i' is definitely not closer to q than o at time t (independent of its value x_i), we therefore can easily integrate this case by making sure that object o_i' is definitely further away from q than o, i.e. by moving o_i' into the states s_3 or s_4 depending on the initial value of x_i.

After the Markov chains for each uncertain object o_i' in D have been determined, we would just have to traverse them and compute the probability $P\exists NN(o, q, D, T)$. If this probability is < 1, there would exist a solution to the corresponding k-SAT formula. However it is not possible to achieve this efficiently in the general case as long as $P \neq NP$. Therefore computing $P\exists NN$ in subexponential time is impossible. □

Example: Consider a set of Boolean variables $X = \{x_1, \ldots, x_4\}$ and the following formula:

$$E = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2)$$

Therefore, we have

$$c_1 = (\neg x_1 \vee x_2 \vee x_3), c_2 = (x_2 \vee \neg x_3 \vee x_4) \text{ and } c_3 = (x_1 \vee \neg x_2)$$

By employing the mapping discussed above, we get the four inhomogeneous Markov chains illustrated in Figure 12.1. For instance, under the condition that x_1 is set to $true$, the value of the literal $\neg x_1$ is false at $t = 1$ (in clause c_1) such that o_1' starts in the state s_4. On the other hand, if x_1 is set to $false$, then o_1' starts in the state s_1.

In the second clause c_2, since $x_1 \notin \mathbb{C}_2$, the position of o_1' must not affect the result. Therefore, for both cases $x_1 = false$ and $x_1 = true$, o_1' must be behind o. In the last clause c_3, if $x_1 = true$ the object moves to state s_2. On the other hand, if $x_1 = false$, the object moves to state s_3.

12.1.2 The P∀NN Query

In the following section we address the complexity of the P∀NN query. In this context we provide several insights. First, we show that computing the probability that an object is closer to the query than another uncertain object can be computed efficiently in sub-exponential time. To analyze the general case of more than two objects, we concentrate on a specific class of algorithms

that aims at integrating *dominance observations* into the Markov model of a given object. By integrating these dominance observations, the set of possible worlds of the object is reduced to those that have not yet been pruned by another object. We show that the adaption of transition matrices of the considered object cannot ba achieved without losing the Markov property, resulting in exponential runtimes, as the joint Markov chain of a set of objects has a size exponential in the number of objects considered. The following proof is an extension of the proof from our paper [13] (and the accompanying technical report) and has been derived from these publications independently of [50]. Before diving deeper into the topic, let us first recap three very basic general lemmas that allow working with conditional probabilities; the first simple lemma extends the Bayes Theorem:

Lemma 2 (Conditional Bayes (CB))**.**

$$P(A|B,C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}$$

Proof. We have

$$P(A \wedge B \wedge C) = P(C)P(B|C)P(A|B,C)$$

and

$$P(A \wedge B \wedge C) = P(C)P(A|C)P(B|A,C)$$

hence

$$P(C)P(A|C)P(B|A,C) = P(C)P(B|C)P(A|B,C)$$

division by $P(C)$ and $P(B|C)$ yields the proof of Lemma 2 (note that both of these probabilities have to be greater than zero). □

The second lemma extends the law of conditional probability:

Lemma 3 (Conditioned Conditional Probability (CCP))**.**

$$P(A|B,C) * P(B|C) = P(A \wedge B|C)$$

Proof. From the left hand side of the equation we get

$$P(A|B,C) * P(B|C) \stackrel{CP}{=} \frac{P(A \wedge B \wedge C)}{P(B \wedge C)} * \frac{P(B \wedge C)}{P(C)} = \frac{P(A \wedge B \wedge C)}{P(C)}$$

From the right hand side we get

$$P(A \wedge B|C) \stackrel{CP}{=} \frac{P(A \wedge B \wedge C)}{P(C)}$$

It follows that the above statement holds (note that $P(C)$ and $P(B \wedge C)$ must be greater than zero). □

Lemma 4 (Conditional Total Probability (CTP))**.**

$$P(A|B) = \sum_j P(A|B, C_j)P(C_j|B)$$

Proof.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{\sum_j P(A \wedge B|C_j)P(C_j)}{P(B)} = \frac{\sum_j P(A \wedge B \wedge C_j)}{P(B)}$$

$$\stackrel{CP}{=} \frac{\sum_j P(A|B, C_j)P(B \wedge C_j)}{P(B)} = \frac{\sum_j P(A|B, C_j)P(C_j|B)P(B)}{P(B)}$$

$$= \sum_j P(A|B, C_j)P(C_j|B)$$

($P(B)$ and $P(B \wedge C_j)$ must be greater than zero) □

12.1.2.1 Dominance: Two-Object Case

Let $o \prec_q^T o_a$ denote the random predicate that is true iff object o is closer to q than object $o_a \in D$ during the query time $T = [t_{start}, t_{end}]$, i.e. $\forall t \in T : dist(o(t), q(t)) \leq dist(o_a(t), q(t))$. If $o \prec_q^T o_a$ holds, we say that o *dominates* o_a with respect to q during T. If the parameters q and T are clear from the context, we simply say that o dominates o_a. This section addresses the calculation of the single object probability $P\forall NN(o, q, \{o_a\}, T)$. This probability can be rewritten as $P(o \prec_q^T o_a|\mathcal{A}, \Omega)$. For our proof, the following information is given:

1. The transition probabilities of o conditioned to an arbitrary condition Ω, i.e.
$$P(o(t) = s_j|o(t-1) = s_i, \Omega)$$

2. The transition probabilities of o_a conditioned to an arbitrary condition $\mathcal{A}$, i.e.
$$P(o_a(t) = s_j|o_a(t-1) = s_i, \mathcal{A})$$

3. the initial distribution of o (o_a) given the condition Ω ($\mathcal{A}$), i.e.
$$P(o(t_0) = s_i|\Omega) \text{ and } P(o_a(t_0) = s_i|\mathcal{A})$$

4. The condition Ω is independent to o_a and the condition $\mathcal{A}$ is independent to o. $\mathcal{A}, \Omega$ can be an arbitrary condition, e.g. observations of an object.

5. Both objects fulfill the Markov property.

6. Both objects are independent of each other.

Lemma 5. *Given $\Xi = \Omega, \mathcal{A}$ the probability $P(o \prec_q^T o_a|\Xi)$ that o dominates o_a can be computed in PTIME.*

Proof. We will show this inductively. Recall that we aim at computing $P\forall NN(o, q, \{o_a\}, T) = P(\forall_{t\in T} : dist(o(t), q(t)) \leq dist(o_a(t), q(t))) \stackrel{Def.}{=} P(o \prec_q^T o_a)$.

Induction Basis. Let $|T| = 1$, i.e. we want to compute $P(o \prec_q^T o_a|\Xi)$ only for time t_0. Then we have

$$P(o \prec_q^T o_a|\Xi) = P(d(o(t), q(t)) \leq d(o_a(t), q(t))|\Xi)$$

$$\stackrel{CTP.}{=} \sum_{s_i,s_j,s_q} P(o \prec_q^T o_a|o(t_0) = s_i, o_a(t_0) = s_j, q(t_0) = s_q, \Xi)$$

$$*P(o(t_0) = s_i \wedge o_a(t_0) = s_j \wedge q(t_0) = s_q|\Xi)$$

There is only a single state possible for $q(t_0)$ since q is a certain trajectory. Because the query is certain, we now only consider the case where $q(t_0) = s_q$, the remaining probabilities are zero.

$$\sum_{s_i,s_j} P(o \prec_q^T o_a|o(t_0) = s_i, o_a(t_0) = s_j, q(t_0) = s_q, \Xi)$$

$$*P(o(t_0) = s_i \wedge o_a(t_0) = s_j \wedge true|\Xi)$$

$$= \sum_{s_i,s_j} P(o \prec_q^T o_a|o(t_0) = s_i, o_a(t_0) = s_j, q(t_0) = s_q, \Xi)$$

$$*P(o(t_0) = s_i \wedge o_a(t_0) = s_j|\Xi)$$

The second factor can be split up by employing the independence assumptions of o and o_a:

$$P(A \wedge B|C, D) \stackrel{Ind.A,B}{=} P(A|C, D) * P(B|C, D) \stackrel{Ind.A,D/B,C}{=} P(A|C) * P(B|D)$$

By employing the independence of both objects we get

$$= \sum_{s_i,s_j} P(o \prec_q^T o_a|o(t_0) = s_i, o_a(t_0) = s_j, \Omega, \mathcal{A}) P(o(t_0) = s_i|\Omega) P(o_a(t_0) = s_j|\mathcal{A})$$

Further, the dominance property is certain if the state of both objects and the query is given (it is independent from $\mathcal{A}$ and Ω). We thus get

$$\sum_{s_i,s_j} P(o \prec_q^T o_a | o(t_0) = s_i, o_a(t_0) = s_j, q(t_0) = s_q)$$

$$* P(o(t_0) = s_i | \Omega) P(o_a(t_0) = s_j | \mathcal{A})$$

Now the first expression is always 1 or 0 given the current state of o and o_a. We can store this simply in a mask matrix $C(t)$ with $C_{ij}(t)$ set to one if o is closer to q than o_a at time t. The above expression can be rewritten as $\vec{s}^{o}(t_0) \cdot \vec{s}^{o_a}(t_0)^T \bullet C(t_0)$, with $\bullet$ denoting the element-wise multiplication of two matrices of equal size. As a result, the runtime complexity of this induction base is $O(|\mathbb{S}|^2)$. Furthermore note that the above is, due to Lemma 3, also equivalent to

$$\sum_{s_i,s_j} P(o \prec_q^T o_a \wedge o(t_0) = s_i \wedge o_a(t_0) = s_j | \Xi)$$

Inductive Step. Let $P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t_n) = s_i \wedge o_a(t_n) = s_j | \Xi)$ be given (induction basis). We now aim at computing $P(o \prec_q^{[t_0,t_{n+1}]} o_a \wedge o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j | \Xi)$. As the position of q is already given by the dominance predicate, we can add it redundantly (its probability is always 1 as q is a certain trajectory):

$$= P(o \prec_q^{[t_0,t_{n+1}]} o_a \wedge o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j \wedge q(t_{n+1}) = s_{q1} | \Xi) \overset{CCP}{=}$$

$$P(o \prec_q^{\{t_{n+1}\}} o_a | o(t_{n+1}) = s_i, o_a(t_{n+1}) = s_j, q(t_{n+1}) = s_{q1}, o \prec_q^{[t_0,t_n]} o_a, \Xi)$$

$$* P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j \wedge q(t_{n+1}) = s_{q1} | \Xi)$$

Note that the first probability is only defined in the case where the second is greater than zero; this is however not a limitation as during the execution of the algorithm we will have to compute the second probability first nonetheless. In the case where the first probability is undefined (and the second probability is zero), the whole probability is actually zero which can be directly seen from the left-hand side of the equation. If the first factor is defined, it equals the probabilities $C_{ij}(t_{n+1})$; it is independent of dominance in previous timesteps and Ξ if the positions of all objects are fixed. To compute the second factor, we first drop the trivial predicate $q(t_{n+1}) = s_{q1}$ (which is always true) and then apply the law of total probability:

$$C_{ij}(t_{n+1}) * \sum_{s_k,s_l,b \in \mathbb{B}} (P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j | \ldots$$

$$\ldots|o(t_n) = s_k, o_a(t_n) = s_l, \Xi, (o \prec_q^{[t_0,t_n]} o_a) = b)$$

$$*P(o \prec_q^{[t_0,t_n]} o_a = b \wedge o(t) = s_k \wedge o_a(t) = s_l|\Xi)$$

If $b = false$ we have a contradiction in the first factor of the sum and thus the resulting probability is 0, therefore we have, since there are only two Boolean values:

$$C_{ij}(t_{n+1})$$

$$*\sum_{s_k,s_l} P(o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j|o(t_n) = s_k, o_a(t_n) = s_l, o \prec_q^{[t_0,t_n]} o_a, \Xi)$$

$$*P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t) = s_k \wedge o_a(t) = s_l|\Xi))$$

For the first factor (the joint transition probability) without additional conditions $o \prec_q^{[t_0,t_n]} o_a$, the Markov property holds. This follows directly from the independence of the two objects o and o_a. As a result, we can discard the past, i.e. $o \prec_q^{[t_0,t_{n-1}]} o_a$. We further can discard $o \prec_q^{[t_n,t_n]} o_a$, because given the exact state of o and o_a, this condition does not provide any additional information. Hence we get

$$C_{ij}(t_{n+1}) * \sum_{s_k,s_l} P(o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j|o(t_n) = s_k, o_a(t_n) = s_l, \Xi)$$

$$*P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t) = s_k \wedge o_a(t) = s_l|\Xi))$$

Applying the property of independence between o and o_a, we get:

$$C_{ij}(t_{n+1}) * \sum_{s_k,s_l} P(o(t_{n+1}) = s_i|o(t_n) = s_k, \Omega)P(o_a(t_{n+1}) = s_j|o_a(t_n) = s_l, \mathcal{A})$$

$$*P(o \prec_q^{[t_0,t_n]} o_a \wedge o(t) = s_k \wedge o_a(t) = s_l|\Xi))$$

To summarize, this corresponds to the probability $P(o \prec_q^{[t_0,t_{n+1}]} o_a \wedge o(t_{n+1}) = s_i \wedge o_a(t_{n+1}) = s_j|\Xi)$.

By employing the law of total probability once again, we can compute $P(o \prec_q^{[t_0,t_{n+1}]} o_a|\Xi)$ from these probabilities, summing over all possible states s_i and s_j

Clearly, the complexity of this lemma is polynomial. For each point in time (t), for each state of o_a and o, we have to compute the above probability. Computing a single probability has quadratic $|\mathbb{S}|^2$ complexity. Computing all $|\mathbb{S}|^2$ probabilities has therefore complexity $|\mathbb{S}|^4$. Therefore we get the complexity $O(t * |\mathbb{S}|^4)$ for the two-object case. □

12.1.2.2 Dominance: Many-Object Case

Given that the dominance probabilities can be computed in polynomial time in the case of two uncertain objects and a (certain) query, the question is now if such dominance probabilities can also be computed in the case of non-trivial databases containing more than two objects. In our investigation we concentrate on a class of algorithms that aims at integrating the dominance of a competing object into the model of the considered object: possible worlds where an object is further away from the query than another object are pruned from the model, adapting the object's transition matrix to *dominance observations*. These dominance observations are similar to spatio-temporal observations: Spatio-temporal observations prune possible worlds of an object that are not in accordance to an observation at a given point in time, and can be integrated into a model efficiently – we will address this case in the next chapter. However, dominance observations arise from the interaction of multiple uncertain objects, making the problem computationally difficult. To demonstrate the basic idea of such algorithms integrating observations into a model, let us first show that given that we *could* integrate such observations into a model, we could compute nearest neighbor probabilities efficiently; this proof is straightforward:

Lemma 6. *Given that a model* $(M_{i,j}^{posterior}(t)) = P(o(t+1) = s_i | o(t) = s_j, o \prec_q^T o_a, \Omega, \mathcal{A})$ *can be computed in PTIME, the probability* $P\forall NN(o, q, D, T)$ *can be computed in PTIME.*

Proof. First let us note that, following the law of conditional probability, we have

$$P\forall NN(o,q,D,T) = P(\forall_{o_i \in D} : o \prec_q^T o_i) = P(\bigwedge_{o_i \in D} o \prec_q^T o_i)$$

$$= \prod_{1 \leq a \leq D} P(o \prec_q^T o_a | \bigwedge_{j<a} o \prec_q^T o_j)$$

Given that, the theorem can be shown inductively.

Induction Base. Given an object o_1, we have to compute $P(o \prec_q^T o_1 | \mathcal{A}, \Omega)$ which can be computed directly by employing Lemma 5. To allow the inductive step, we further compute $P(o(t) = s_i | o(t-1) = s_j, \mathcal{A}, \Omega, o \prec_q^T o_1)$, which is given in the theorem. For the inductive step, we set $\Omega_1 := \mathcal{A}, \Omega, o \prec_q^T o_1$ and note that Ω_1 is independent of the remaining objects, that have not yet been included into probability calculation.

Inductive Step. In the inductive step we assume $P(o(t) = s_i | o(t-1) = s_j, \Omega, \bigwedge_{i \leq n}(o \prec_q^T o_i \wedge \mathcal{A}_i)) = P(o(t) = s_i | o(t-1) = s_j, \Omega_n)$ as given. The goal is to compute $P(o \prec_q^T o_{n+1} | \mathcal{A}_n, \Omega_n)$. This probability can be easily computed by employing Lemma 5. □

12.1.2.3 Violation of the Markov Property.

Lemma 7. *Let o and o_a be uncertain spatio-temporal objects. Models resulting from adapting the Markov model of o to dominance observations violate the Markov property.*

Proof. The idea of the model adaption is to define a joint stochastic process with transition probabilities

$$P(\omega(t) = \sigma_i | \omega(t-1) = \sigma_j, \mathcal{A}, \Omega)$$

$$= P(o(t) = s_i \wedge o_a(t) = s_j | o(t-1) = s_k, o_a(t-1) = s_l, \mathcal{A}, \Omega)$$

and an initial state

$$P(\omega(t_0) = \sigma_i, \mathcal{A}, \Omega) = P(o(t_0) = s_i \wedge o_a(t_0) = s_j | \mathcal{A}, \Omega)$$

The goal is to incorporate dominance observations into this joint stochastic process and then reduce this joint process to a stochastic process only modelling the movement of o. If we would not reduce the process after incorporating observations, we would be exponential in $|D|$, resulting in an exponential algorithm. If a solution for model adaption would be correct, the adapted model of o would have to fulfill the Markov property, i.e.

$$P(o(t) = s_i | o(t-1) = s_k, o \prec_q^T o_a, \mathcal{A}, \Omega)$$

$$= P(o(t) = s_i | o(t-1) = s_k, \ldots, o(t-n) = s_{x_{t-n}}, o \prec_q^T o_a, \mathcal{A}, \Omega)$$

Figure 12.2 illustrates the inequality of these two cases. The example shows two uncertain objects o (blue) and o_a (red), their transition probabilities and the set of possible worlds. In the transparent and strike-through worlds, o does not dominate o_a because o is in a higher state than o_a. Therefore, we have $P(o \prec_q^T o_a) = 0.5$. We aim at calculating the dominance for the whole time interval $T = [t-3, t]$

Now we compute both

$$P(o(t) = 0 | o(t-1) = 0, o \prec_q^T o_a) = \frac{P(o(t) = 0 \wedge o(t-1) = 0 \wedge o \prec_q^T o_a)}{P(o(t-1) = 0 \wedge o \prec_q^T o_a))}$$

and

$$P(o(t) = 0 | o(t-1) = 0, \ldots, o(t-3) = 0, o \prec_q^T o_a)$$

$$= \frac{P(o(t) = 0 \wedge o(t-1) = 0 \wedge \ldots \wedge o(t-3) = 0 \wedge o \prec_q^T o_a)}{P(o(t-1) = 0 \wedge o(t-2) = 0 \wedge o(t-3) = 0 \wedge o \prec_q^T o_a))}$$

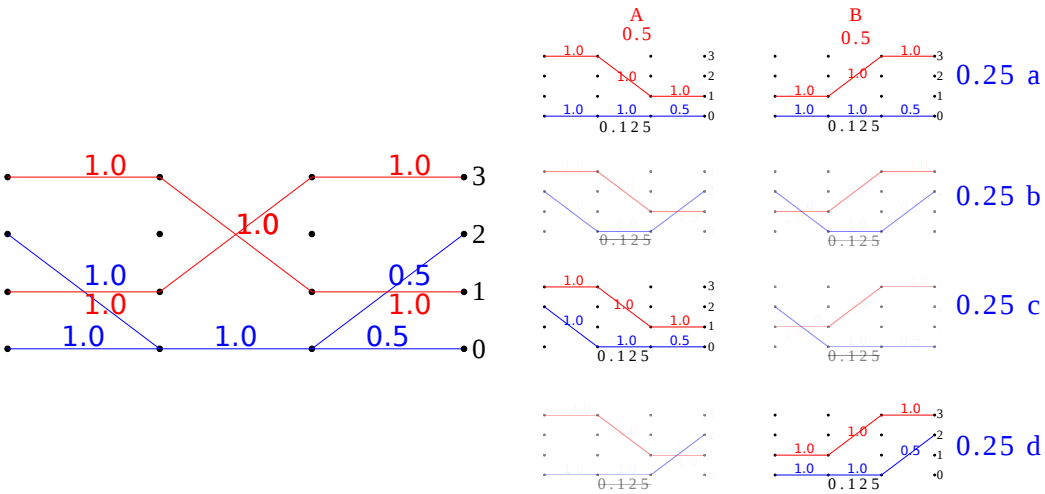


Figure 12.2: P∀NN: Violation of the Markov assumption

The probabilities in the fractions can be easily computed by summing the probabilities of possible worlds for which the conditions hold. We get:

$$P(o(t) = 0 \wedge o(t-1) = 0 \wedge o \prec_q^T o_a) = 0.375 \text{ (Aa, Ac, Ba)}$$

$$P(o(t-1) = 0 \wedge o \prec_q^T o_a) = 0.5 \text{ (Aa, Ac, Ba, Bd)}$$

resulting in $P(o(t) = 0 | o(t-1) = 0, o \prec_q^T o_a) = 0.75$. When considering more states in the past, we get the following probabilities:

$$P(o(t) = 0 \wedge o(t-1) = 0 \wedge \ldots \wedge o(t-3) = 0 \wedge o \prec o_a) = 0.25 \text{ (Aa, Ba)}$$

$$P(o(t-1) = 0 \wedge P(o(t-2) = 0 \wedge o \wedge o(t-3) = 0 \prec o_a) = 0.375 \text{ (Aa, Ba, Bd)}$$

Thus we get a probability of 0.66. This shows that the Markov assumption does not hold any more if we incorporate dominance observations into the model of o.

□

12.1.3 The PCNN Query

The traditional CNN query [88, 86], retrieves the nearest neighbor of every point on a given query trajectory in a time interval T. This basic definition usually returns $m << |T|$ time intervals together having the same nearest neighbor. The main issue when considering uncertain trajectories and extending the query definition is the possibly large number of results due to highly overlapping and alternating result intervals. In particular, considering

Definition 12, a PCNN result may have an exponential number of elements when τ becomes small. This is because in the worst case each $T_i \subseteq T$ can be associated with an object o for which the probability $P\forall NN(o, q, D, T_i) \geq \tau$, i.e., $2^{|T|}$ different T_i's occur in the result set.

To alleviate (but not solve) this issue, in the following we propose a technique based on Apriori pattern mining to return the subsets of T that have a probability greater than τ. This algorithm requires to compute a $P\forall NN$ probability in each validation step; we assume that this is achieved by employing the sampling approach proposed in Section 12.2. Since each subset of T may have a probability greater than τ (especially when τ is chosen too small), a worst-case of $O(2^{|T|})$ validations may have to be performed.

Algorithm. Algorithm 5 shows how to compute, for a query trajectory q, a time interval T, a probability threshold τ, and an uncertain trajectory $o \in D$ all $T_i \subseteq T$ for which o is the nearest neighbor to q at all timestamps in T_i with probability of at least τ, and the corresponding probabilities.

Algorithm 5 $PC_\tau NN(q, o, D, \mathrm{T}, \tau)$

1: $L_1 = \{(\{t\}, P) | t \in T \wedge P = P\forall NN(o, q, D \setminus \{o\}, \{t\}) \wedge P \geq \tau\}$
2: **for** $\kappa = 2; L_{\kappa-1} \neq \emptyset; \kappa++$ **do**
3: $\quad X^\kappa = \{T_\kappa \subseteq T | |T_\kappa| = \kappa \wedge \forall T'_{\kappa-1} \subset T_\kappa \exists (T'_{\kappa-1}, P) \in L_{\kappa-1}\}$
4: $\quad L_\kappa = \{(T_\kappa, P) | T_\kappa \in X^\kappa \wedge P = P\forall NN(o, q, D \setminus \{o\}, T_\kappa) \geq \tau\}$
5: **end for**
6: **return** $\bigcup_\kappa L_\kappa$

We take advantage of the anti-monotonicity property that for a T_i to qualify as a result of the PCNNQ query, all proper subsets of T_i must also satisfy this query. In other words if o is the P$\forall$NN of q in T_i with probability at least τ, then for all $T_j \subset T_i$ o must be the P$\forall$NN of q in T_j with probability at least τ. Exploiting this property, we adapt the Apriori pattern-mining approach from [104] to solve the problem as follows. We start by computing the probabilities of all single points of time to be query results (line 1). Then, we iteratively consider the set X^κ of all timestamp sets with κ points of time by extending timestamp sets $T_{\kappa-1}$ with an additional point of time $t \in T \setminus T_{\kappa-1}$, such that all $T'_{\kappa-1} \subset T_\kappa$ have qualified at the previous iteration, i.e., we have $P\forall NN(o, q, D \setminus \{o\}, T'_{\kappa-1})) \geq \tau$ (line 3).

The P$\forall$NN probability is monotonically decreasing with the number of points in time considered, i.e. if $T_\kappa \subset T_{\kappa+1}$ there is $\mathrm{P}\forall\mathrm{NN}(o, q, D \setminus \{o\}, T_\kappa) \geq \mathrm{P}\forall\mathrm{NN}(o, q, D \setminus \{o\}, T_{\kappa+1})$. Therefore we do not have to further consider the set of points of time T_κ that do not qualify for the next iterations during the iterative construction of sets of time points. Based on the sets of timesteps T_κ constructed in each iteration we compute $P\forall NN(o, q, D \setminus \{o\}, T_\kappa)$ to

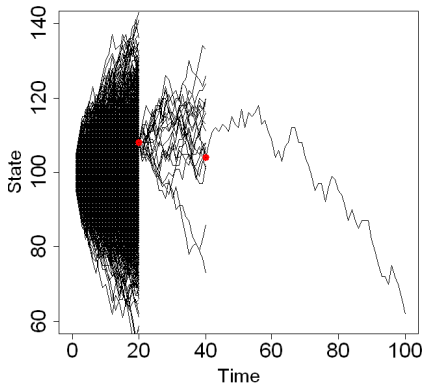


Figure 12.3: Traditional MC-Sampling.

build the set of results of length κ (line 4) that are finally collected and reported as result in line 6. The basic algorithm can be sped up by employing the property that given $\mathrm{P}\forall\mathrm{NN}(o, q, D \setminus \{o\}, T_1) = 1$ the probability of $\mathrm{P}\forall\mathrm{NN}(o, q, D \setminus \{o\}, T_1 \cup T_2) = \mathrm{P}\forall\mathrm{NN}(o, q, D \setminus \{o\}, T_2)$.

Based on Algorithm 5 it is possible to define a straightforward algorithm for processing PCNNQ queries (by considering each object o' from the database). Again this approach can be improved by the use of an appropriate index-structure (cf. Section 12.3).

12.2 Sampling Possible Trajectories

Based on the discussion in the previous sections, it is clear that answering probabilistic queries over uncertain trajectory databases has high run-time cost. Therefore, like previous work [105], we now study sampling-based approximate solutions to improve query efficiency.

12.2.1 Traditional Sampling

To sample possible trajectories of an object, a traditional Monte-Carlo approach would start by taking the first observation of the object, and then perform forward transitions using the a-priori transition matrix. This approach however, cannot directly account for additional observations for latter timestamps. Figure 12.3 illustrates a total of 1000 samples drawn in a one-dimensional space. Starting at the first observation time $t = 0$, transitions are performed using the a-priori Markov chain. At the second observation at time $t = 20$, the great majority of trajectories becomes inconsistent. Such

impossible trajectories have to be dropped. At time $t = 40$, even more trajectories become invalid; After this observation, only one out of a thousand samples remains possible and useful.

Clearly, the number of trajectory generations required to obtain a single valid trajectory sample increases exponentially in the number of observations of an object, making this traditional Monte-Carlo approach inappropriate in obtaining a sufficient number of valid samples within acceptable time.

12.2.2 Efficient and Appropriate Sampling

To tackle the disadvantages of traditional sampling, we now introduce an optimized approach af drawing samples. On these samples, traditional nearest neighbor algorithms for (certain) trajectories ([83, 84, 97, 85, 88, 86]) can be used to estimate nearest neighbor probabilities.

In a nutshell, our approach starts with the initial observation θ_1^o at time t_1^o, and performs transitions for object o using the a-priori Markov chain of o until the final observation $\theta_{|\Theta^o|}^o$ at time $t_{|\Theta^o|}$ is reached. During this *Forward*-run phase, Bayesian inference is used to construct a time-reversed Markov model $R^o(t)$ of o at time t given observations in the past, i.e., a model that describes the probability

$$R_{ij}^o(t) := P(o(t-1) = s_j | o(t) = s_i, \{\theta_i^o | t_i^o < t\})$$

of coming from a state s_j at time $t-1$, given being at state s_i at time t and the observations in the past. Then, in a second *Backward*-run phase, our approach traverses time backwards, from time $t_{|\Theta^o|}$ to t_1, by employing the time-reversed Markov model $R^o(t)$ constructed in the forward phase. Again, Bayesian inference is used to construct a new Markov model $F^o(t-1)$ that is further adapted to incorporate knowledge about observations in the future. This new Markov model contains the transition probabilities

$$F_{ij}^o(t-1) := P(o(t) = s_j | o(t-1) = s_i, \Theta^o). \tag{12.2}$$

for each point of time t, given all observations, i.e., in the past, the present and the future.

As an illustration, Figure 12.4(left) shows the initial model given by the a-priori Markov chain, using the first observation only. In this case, a large set of (*time*, *location*) pairs can be reached with a probability greater than zero.

The adapted model after the forward phase (given by the a-priori Markov chain and all observations), depicted in Figure 12.4(center), significantly reduces the space of reachable (*time*, *location*) pairs and adapts respective

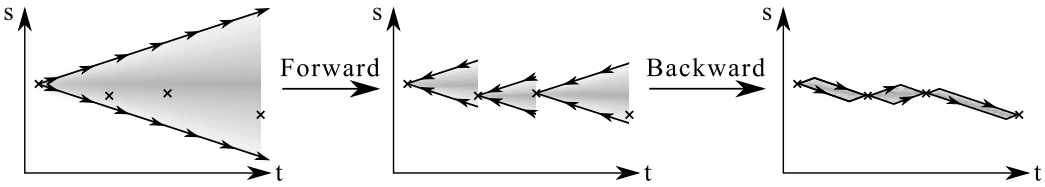


Figure 12.4: An overview over our forward-backward-algorithm.

probabilities. The main goal of the forward-phase is to construct the necessary data structures for efficient implementation of the backward-phase, i.e., $R^o(t)$. Figure 12.4(right) shows the resulting model after the backward phase. In the following section, both phases are elaborated in detail.

Note that the Baum-Welch algorithm for hidden Markov models (HMMs) is similar to the proposed algorithm. This algorithm aims at estimating *time-invariant* transition matrices and emission probabilities of a hidden Markov model. In contrast, we assume this underlying model to be given, however we aim at adapting it by computing *time-variant* transition matrices. Despite these differences, the above algorithm could also be proven by showing that our model is a special case of a HMM and deducting the algorithm from the Baum-Welch algorithm [106]. We will address the similarities to [106] later in this section. Related to our algorithm is also the Forward-Backward-Algorithm for HMMs that aims at computing the *state distribution* of a HMM for each point in time. In contrast, we aim at computing transition matrices for each point in time, given a set of observations.

Again, the following proof is an extension of the proof from [13] (and the accompanying technical report) and has been derived from these publications independently of [50].

12.2.2.1 Forward Phase

Let $past^o(t) := \{\theta_i^o | t_i^o < t\}$ denote the set of observations temporally preceding t. During the forward phase we aim at computing the **backward probabilities** $R_{ij}(t)$:

$$R_{ij}(t) = P(o(t-1) = s_j | o(t) = s_i, past(t))$$

Precondition 1. *All observations θ_i^o of an object are in accordance to the transition model: it is possible to reach a descending observation from a preceding one given the a-priori transition model of o.*

For computing the backward probabilities, we distinguish three cases:
Case 1: $o(t) = s_i$ **contradicts** $past(t)$. In *all* cases where $o(t) = s_i$ is not

in conformance to $past(t)$, transition probabilities are undefined, as we have a contradiction in the condition of the probabilities $R_{ij}(t)$.
Case 2: $o(t-1) = s_j$ **contradicts** $past(t)$. If s_j is not reachable given $past(t)$, we have $R_{ij}(t) = 0$.
Case 3: Neither $o(t) = s_i$ **nor** $o(t-1) = s_j$ **contradict** $past(t)$. In this case, in order to compute the backward probabilities, we apply Conditional Bayes:

$$R_{ij}(t) = P(o(t-1) = s_j | o(t) = s_i, past(t))$$

$$\stackrel{CB}{=} \frac{P(o(t) = s_i | o(t-1) = s_j, past(t)) P(o(t-1) = s_j | past(t))}{P(o(t) = s_i | past(t))} \tag{12.3}$$

As neither $o(t) = s_i$ nor $o(t-1) = s_j$ contradict the past (these cases are addressed in Case 1 and Case 2), the expression is always defined. Given that, these three factors can be computed as follows:

- $P(o(t-1) = s_j | past(t))$ can be computed as follows. Given that we have an observation at time $t-1$, the probability is 1. Given that there is no observation at time $t-1$, we can compute this probability by traversing the Markov chain starting from the last observation before t (exploiting the Markov property) with the basic Markov transition rule. The probability of reaching state s_j at time $t-1$ is then the wanted probability.

- For the probability $P(o(t) = s_i | past(t))$ the same as for the upper probability holds, however, in contrast to the upper probability, the observation at time t is not considered.

- $P(o(t) = s_i | o(t-1) = s_j, past(t))$ are the transition probabilities $P(o(t) = s_i | o(t-1) = s_j)$. $past(t-1)$ can be removed due to the Markov property. If there is an observation $present(t-1)$, this observation must be state s_j, as otherwise we would have a contradiction (which then would contradict the assumption of Case 3). As in this case the condition is redundant, $present(t-1)$ can be removed as well.

Markov Property. We still have to show that $R_{ij}(t)$ follows the Markov assumption, given that a state sequence can be generated by the initial model:

$$R_{ij}(t) = P(o(t-1) = s_j | o(t) = s_i, past(t))$$

$$\stackrel{?}{=} P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n) = s_{x_{t+n}}, past(t))$$

Precondition 2. *States in the condition, i.e.* $o(t) = s_i, \ldots, o(t+n) = s_{x_{t+n}}$ *are in accordance to the initial transition model and* $past(t)$.

Case 1: $o(t-1) = s_j$ does not contradict the previous states or $past(t)$. In this first case, the resulting probabilities are non-zero.

$$P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n) = s_{x_{t+n}}, past(t))$$

$$\stackrel{CB}{=} P(o(t+n) = s_{x_{t+n}} | o(t-1), \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t)) *$$
$$\frac{P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t))}{P(o(t+n) = s_{x_{t+n}} | o(t) = s_i, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t))}$$

Following Precondition 2, the denominator of this expression will be greater than zero; according to the precondition of Case 1, the first factor of the numerator is defined as well. By employing the Markov assumption of the initial model, the first factor of the numerator and the denominator can be rewritten.

$$P(o(t+n) = s_{x_{t+n}} | o(t+n-1) = s_{x_{t+n-1}}) *$$
$$\frac{P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t))}{P(o(t+n) = s_{x_{t+n}} | o(t+n-1) = s_{x_{t+n-1}})}$$

After cancelling, we get

$$P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t))$$

which equals the input formula reduced by one timestep. Applying this trick iteratively, yields $P(o(t-1) = s_j | o(t) = s_i, past(t))$.

Case 2: $o(t-1) = s_j$ contradicts a previous state or past(t). We prove the Markov assumption in this case by contradiction.

As a **first** instance, assume $P(o(t-1) = s_j | o(t) = s_i, past(t)) \neq 0$ (the contradiction follows from a state at $t+k, k > 0$). Further recall that if this probability is $\neq 0$, then there exists also a possible trajectory resembling this probability for the initial model and the adapted one[2]. In this first case, we can generate $P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n) = s_{x_{t+n}}, past(t))$ by reversely applying the proof from Case 1. Let us assume we are at iteration n of the proof and $P(o(t-1) = s_j | o(t) = s_i, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t)) = P(o(t-1) = s_j | o(t) = s_i, past(t)) \neq 0$ and therefore $P_t = P(o(t-1) = s_j \wedge o(t) = s_i \wedge \ldots \wedge o(t+n-1) = s_{x_{t+n-1}}, past(t))) \neq 0$. We expand the probability by $P(o(t+n) = s_{x_{t+n}} | o(t+n-1) = s_{x_{t+n-1}})$ in both the numerator and the denominator; according to Precondition 2, these probabilities are non-zero. By applying the Markov assumption in forward direction we get

$$P(o(t+n) = s_{x_{t+n}} | o(t-1) = s_j, \ldots, o(t+n-1) = s_{x_{t+n-1}}, past(t)) *$$

[2] $P(o(t-1) = s_j | o(t) = s_i, past(t)) \neq 0 \Rightarrow \frac{P(o(t-1) = s_j \wedge o(t) = s_i \wedge past(t))}{P(o(t) = s_i \wedge past(t))} \neq 0$
$\Rightarrow P(o(t-1) = s_j \wedge o(t) = s_i \wedge past(t)) \neq 0 \Rightarrow P(o(t-1) = s_j \wedge o(t) = s_i) \neq 0$

$$\frac{P(o(t-1)=s_j|o(t)=s_i,\ldots,o(t+n-1)=s_{x_{t+n-1}},past(t))}{P(o(t+n)=s_{x_{t+n}}|o(t)=s_i,\ldots,o(t+n-1)=s_{x_{t+n-1}},past(t))}$$

The denominator of this expression is greater than zero (and also defined) due to Precondition 2. The second term of the numerator is non-zero as well; we know this from the last iteration. Concerning the first term in the numerator, the condition is defined, we know this from P_t. Given that $o(t+n) = s_{x_{t+n}}$ contradicts the condition (and therefore $o(t-1) = s_j$), the first probability in the numerator would have to be zero, which however contradicts the Markov assumption of the initial model. Furthermore note that the resulting probability is exactly equivalent to the one from the last iteration due to the application of the Markov property. To conclude the inductive step, we have to show that given that this derived probability is non-zero, the corresponding trajectory of the initial model has a non-zero probability as well (as in this case the first numerator probability in the next iteration would again allow application of the Markov assumption); this can be shown equivalently to footnote 2.

This leads to a contradiction, as it shows that $o(t-1) = s_j$ is in accordance to all other states in the condition.

In the **second** case, we assume $P(o(t-1) = s_j|o(t) = s_i, past(t)) = 0$. We assume that the Markov property does not hold and therefore $P(o(t-1) = s_j|o(t) = s_i,\ldots,o(t+n) = s_{x_{t+n}}, past(t)) \neq 0$. This is, however, mathematically impossible[3].

This concludes the forward phase of the algorithm.

12.2.2.2 Backward Phase

We are now considering the **backward phase**. In the following, let t_e denote the time of the last observation of the currently considered object. For this phase we show how to compute the forward transition probabilities $F_{ij}(t) = P(o(t) = s_j|o(t-1) = s_i, past(t_e+1)))$ that condition the movement of the stochastic process to observations at all relevant timesteps.

Case 1: $o(t-1) = s_i$ **contradicts** $past(t_e+1)$**.** In the case where $o(t-1) = s_i$ is not in accordance to either past or future observations, F_{ij} is undefined. It can however be easily avoided by starting at an observation in the future (ensuring conformance to future observations), and transitioning backward using the reverse transition matrices computed during the forward phase (ensuring conformance to past observations, as these have already been integrated into the model).

[3] $P(A|B) = 0 \Rightarrow P(A \wedge B) = 0$ and $P(A|B,C) \neq 0 \Rightarrow P(A \wedge B \wedge C) \neq 0$. Given that $P(A \wedge B \wedge C) \neq 0$ it is impossible that $P(A \wedge B) = 0$

Case 2: $o(t) = s_j$ contradicts $past(t_e + 1)$. If s_j is not in conformance to $past(t_e + 1)$, its probability becomes zero, as in this case the event $o(t) = s_j$ is impossible to happen given $past(t_e + 1)$; this follows directly from $F_{ij}(t)$.
Case 3: Neither $o(t-1) = s_i$ nor $o(t) = s_j$ contradict $past(t_e + 1)$. In the third case we can compute F_{ij} by applying conditional Bayes:

$$F_{ij}(t) = P(o(t) = s_j | o(t-1) = s_i, past(t_e + 1)))$$

$$\stackrel{CB}{=} \frac{P(o(t-1) = s_i | o(t) = s_j, past(t_e + 1)) P(o(t) = s_j | past(t_e + 1))}{P(o(t-1) = s_i | past(t_e + 1))}$$

Similar to the forward phase we have to compute these three factors to compute $F_{ij}(t)$:

- $P(o(t-1) = s_i | o(t) = s_j, past(t_e + 1)) \stackrel{Rev.Markov}{=} P(o(t-1) = s_i | o(t) = s_j, past(t + 1))$. $past(t + 1)$ can be reduced to $past(t)$ by considering that in this third case we assume that $o(t) = s_j$ does not contradict $past(t+1)$, and in this case both of these conditions become equivalent. The resulting probability $P(o(t-1) = s_i | o(t) = s_j, past(t))$ equals exactly the backward transition probability $R_{ji}(t)$ from the forward phase

- $P(o(t) = s_j | past(t_e + 1))$ can be computed inductively. $P(o(t_e) = s_i | past(t_e + 1))$ is given as a result of the forward phase, and is one if s_i corresponds to the last observation and zero otherwise. Now let $P(o(t) = s_j | past(t_e + 1))$ be given. We are looking for $P(o(t-1) = s_i | past(t_e + 1)) =$

  $$\sum_{s_j} P(o(t-1) = s_i | o(t) = s_j, past(t_e + 1)) P(o(t) = s_j | past(t_e + 1))$$

  where the first factor is again the element from the transition matrix $R_{ji}(t)$. The second factor is known given the induction basis. Note that, as we build a sum over all s_j, some of the transition probabilities are undefined from the forward phase; this happens when $o(t) = s_j$ contradicts $past(t_e + 1)$. By applying CCP to the product in this formula, we get $P(o(t-1) = s_i \wedge o(t) = s_j | past(t_e + 1))$. This probability is clearly zero, as in such a case $P(o(t) = s_j | past(t_e + 1))$ is zero as well.

- $P(o(t) = s_j | past(t_e + 1))$ can be computed equivalently.

After the backward phase, we have $F_{ij}(t) = P(o(t) = s_j | o(t-1) = s_i, past(t_e + 1)))$ which directly corresponds to the transition probabilities of the process given all observations $past(t_e + 1)$.

Markov Property. Again, we have to show that for $F_{ij}(t)$ the Markov assumption holds:

$$P(o(t) = s_j | o(t-1) = s_i, past(t_e + 1))$$

$$= P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n) = s_{x_{t-n}}, past(t_e + 1))$$

Precondition 3. *States in the condition, i.e.* $o(t-1) = s_i, \ldots, o(t-n) = s_{x_{t-n}}$ *are in accordance to the initial transition model and* $past(t_e + 1)$.

For this part of the proof let $future(t)$ denote the set of observations temporally descending time t.
Case 1: $o(t) = s_j$ **does not contradict the previous states or** $past(t_e + 1)$**.** In this case the transition probability is non-zero. Applying conditional Bayes yields

$$P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n) = s_{x_{t-n}}, past(t_e + 1))$$

$$\stackrel{CB}{=} P(o(t-n) = s_{x_{t-n}} | o(t-n+1) = s_{x_{t-n+1}}, \ldots, o(t) = s_j, past(t_e + 1)) *$$
$$\frac{P(o(t) = s_j | o(t-n+1) = s_{x_{t-n+1}}, \ldots, o(t-1) = s_i, past(t_e + 1))}{P(o(t-n) = s_{x_{t-n}} | o(t-n+1) = s_{x_{t-n+1}}, \ldots, o(t-1) = s_i, past(t_e + 1))}$$

We can split up $past(t_e + 1)$ as follows: $past(t - n + 1)$ is contained in the Markov chain $R_{ij}(t-n+1)$. $future(t-n+1)$ can be discarded by employing the reverse Markov property that holds for R_{ij}. For $present(t - n + 1)$ we employ Precondition 3. Given this assumption, this predicate does not provide any additional information, and thus can be discarded. Therefore and according to the reverse Markov property the left part of the numerator equals the denominator. As a result, after cancelling, we get

$$P(o(t) = s_j | o(t-n+1) = s_{x_{t-n+1}}, \ldots, o(t-1) = s_i, past(t_e + 1))$$

which is our input probability with the last timestep removed. Iterative application of conditional Bayes, yields the Markov property for F_{ij}.
Case 2: $o(t) = s_j$ **contradicts a previous state or** $past(t_e + 1)$**.** For this case we prove the Markov assumption by contradiction.

As a **first** instance, assume $P(o(t) = s_j | o(t-1) = s_i, past(t_e + 1)) \neq 0$ (the contradiction follows at time $t - k, k > 1$, and the corresponding trajectory of the reverse and initial model have non-zero probability as well, see footnote 2). In this case, we can generate $P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n) = s_{x_{t-n}}, past(t_e + 1))$ by reversely applying the proof from Case 1. Let us assume we are in iteration n of the proof and $P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n+1) = s_{x_{t-n+1}}, past(t_e + 1)) \neq 0$ (all states

are still in conformance and the corresponding trajectory of the reverse and initial model have non-zero probability). We expand the probability by $P(o(t-n) = s_{x_{t-n}} | o(t-n+1) = s_{x_{t-n+1}}, past(t-n+1))$ in both the numerator and the denominator; according to Precondition 3, both of these probabilities are non-zero. By applying the Markov assumption in backward direction we get

$$P(o(t-n) = s_{x_{t-n}} | o(t) = s_j, \ldots, o(t-n+1) = s_{x_{t-n+1}}, past(t_e+1)) *$$

$$\frac{P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n+1) = s_{x_{t-n+1}}, past(t_e+1))}{P(o(t-n) = s_{x_{t-n}} | o(t-1) = s_i, \ldots, o(t-n+1) = s_{x_{t-n+1}}, past(t_e+1))}$$

The denominator of this expression is greater than zero due to Precondition 3. The second term of the numerator is non-zero as well; we know this from the last iteration. Concerning the first term in the numerator, recall that the condition is non-contradictory, we know this from the second factor in the numerator. As we also know that the second probability resembles a valid trajectory according to the initial model fulfilling Precondition 2, the application of the Markov assumption in backward direction for the first factor is valid. Given that $o(t-n) = s_{x_{t-n}}$ contradicts the condition (and therefore $o(t) = s_j$), the first probability in the numerator would have to be zero, which however contradicts the reverse Markov assumption. Furthermore note that the resulting probability is exactly equivalent to the one from the last iteration due to the application of the Markov assumption. This leads to a contradiction, as it shows that $o(t) = s_j$ is in accordance to all previous states including $o(t-n) = s_{x_{t-n}}$.

By employing footnote2 we can finally show the new probability extended by a point of time resembles again a valid trajectory according to the initial model, concluding the proof.

In the **second** case, we assume $P(o(t) = s_j | o(t-1) = s_i, past(t_e+1)) = 0$. We assume that the Markov property does not hold and therefore $P(o(t) = s_j | o(t-1) = s_i, \ldots, o(t-n) = s_{x_{t-n}}, past(t_e+1)) \neq 0$, which is however impossible given the assumption.

12.2.2.3 Sampling Process

Algorithm 6 summarizes the construction of the transition model for a given object o. In the forward phase, the new distribution vector $\vec{s}^{o}(t)$ of o at time t and backward probability matrix $R^o(t)$ at time t can be efficiently derived from the temporary matrix $X'(t)$, computed in Line 4. The equation is equivalent to a simple transition at time t, except that the state vector is converted to a diagonal matrix first. This trick allows to obtain a matrix

Algorithm 6 AdaptTransitionMatrices(o)

1: {Forward-Phase}
2: $\vec{s}^{\,o}(t_1^o) = \theta_1^o$
3: **for** $t = t_1^o + 1;\ t \leq t^o_{|\Theta^o|};\ t{+}{+}$ **do**
4: $\quad X'(t) = M^o(t-1)^T \cdot diag(\vec{s}^{\,o}(t-1))$
5: $\quad \forall i \in \{1 \dots |\mathbb{S}|\} : \vec{s}^{\,o}(t)_i = \sum_{j=1}^{|S|} X'_{ij}(t)$
6: $\quad \forall i,j \in \{1 \dots |\mathbb{S}|\} : R^o(t)_{ij} = \frac{X'_{ij}(t)}{\vec{s}^{\,o}(t)_i}$
7: $\quad$ **if** $t \in \Theta^o$ **then**
8: $\quad\quad \vec{s}^{\,o}(t) = \theta_t^o$ {Incorporate observation}
9: $\quad$ **end if**
10: **end for**
11: {Backward-Phase}
12: **for** $t = t^o_{|\Theta^o|} - 1; t \geq t_1^o; t$-- **do**
13: $\quad X'(t) = R^o(t+1)^T \cdot diag(\vec{s}^{\,o}(t+1))$
14: $\quad \forall i \in \{1 \dots |\mathbb{S}|\} : \vec{s}^{\,o}(t)_i = \sum_{j=1}^{|S|} X'_{ij}(t)$
15: $\quad \forall i,j \in \{1 \dots |\mathbb{S}|\} : F^o(t)_{ij} = \frac{X'_{ij}(t)}{\vec{s}^{\,o}(t)_i}$
16: **end for**
17: **return** F^o

describing the joint distribution of the position of o at time $t-1$ and t. Formally, each entry $X'(t)_{i,j}$ corresponds to the probability $P(o(t-1) = s_j \wedge o(t) = s_i | past^o(t))$ which is equivalent to the *numerator* of Equation 12.3. To obtain the denominator of Eq. 12.3 we first compute the row-wise sum of $X'(t)$ in Line 5. The resulting vector directly corresponds to $\vec{s}^{\,o}(t)$, since for any matrix A and vector x it holds that $A \cdot x = rowsum(A \cdot diag(x))$. By employing this *rowsum* operation, only one matrix multiplication is required for computing $R^o(t)$ and $\vec{s}^{\,o}(t)$.

Next, the elements of the temporary matrix $X'(t)$ and the elements of $\vec{s}^{\,o}(t)$ are normalized in Equation 12.3, as shown in Line 6 of the algorithm. Here, some probabilities might be undefined if the normalizer becomes zero (see Case 2 of our proof for the forward phase); we can simply set these to zero as this will avoid special cases during the backward phase.

Finally, possible observations at time t are integrated in Line 8. In Lines 12 to 15, the same procedure is followed in time-reversed direction, using the backward transition matrix $R^o(t)$ to compute the a-posteriori matrix $F^o(t)$.

The overall complexity of this algorithm is $O(|T| \cdot |\mathbb{S}|^2)$. The initial matrix multiplication requires $|\mathbb{S}|^2$ multiplications. While the complexity of a matrix

multiplication is in $O(|\mathbb{S}|^3)$, the multiplication of a matrix with a diagonal matrix, i.e., $M^T \cdot s$ can be rewritten as $M_i^T \cdot s_{ii}$, which is actually a multiplication of a vector with a scalar, resulting in an overall complexity of $O(|\mathbb{S}|^2)$. Re-diagonalization needs $|\mathbb{S}|^2$ additions as well, such as re-normalizing the transition matrix, yielding $3 \cdot |T| \cdot |\mathbb{S}|^2$ for the forward phase. The backward phase has the same complexity as the forward phase, leading to an overall complexity of $O(|T| \cdot |\mathbb{S}|^2)$.

Once the transition matrices $F^o(t)$ for each point of time t have been computed, the actual sampling process is simple: For each object o, each sampling iteration starts at the initial position θ_1^o at time t_1^o. Then, random transitions are performed, using $F^o(t)$ until the final observation of o is reached. Doing this for each object $o \in D$, yields a (certain) trajectory database, on which exact NN-queries can be answered using previous work. Since the event that an object o is a $\forall$NN ($\exists$NN) of q is a binomial distributed random variable, we can use methods from statistics, such as the Hoeffding's inequality ([107]) to give a bound of the estimation error, for a given number of samples.

12.2.2.4 Relationship to Hidden Markov Models

The research from [41, 40, 13, 87] concentrates on simple Markov Models enhanced by certain observations. In this section we draw a connection between the model from these papers and another, more general class of stochastic processes, namely the *Hidden Markov Model*. Based on these Hidden Markov Models we can approach the sampling algorithm from a different point of view, drawing connections to existing algorithms such as the Baum-Welch-Algorithm or the Forward-Backward algorithm. Both of these have been developed for the case of Hidden Markov Models. According to [108], a Hidden Markov Model is a combined stochastic process. The hidden process cannot be observed directly but is rather distorted by a second process:

Definition 15 (Hidden Markov Model [108])**.** *Given a time-invariant transition matrix M_{ij}, a time-invariant emission matrix B_{ij}, and an initial state distribution π, a hidden Markov model is a stochastic process $\lambda = (M, B, \pi)$.*

To train the model parameters of a Hidden Markov model (M, B, and π) based on a (long enough) sequence of observations $\theta_1, ..., \theta_n$, the Baum-Welch algorithm [108] can be employed that estimates these parameters based on the expectation-maximization technique iteratively and approximately. To keep the complexity of estimating model parameters low, M and B are usually assumed to be time-homogeneous. In our scenario assumptions are slightly different: M is not necessarily time-invariant, but B_{ij} is trivial, i.e.

$b_{ii} = 1$ and $b_{ij} = 0, i \neq j$. We also assume the a-priori transition matrices that model the *general* movement patterns of objects to be given. We simply want to adapt the model to the known movement of a *specific* object. These different conditions allow us to estimate the transition matrix M accurately as shown previously.

Despite these differences, the algorithm from the last section can be seen as a modification of the Baum-Welch algorithm[108]. The Baum-Welch algorithm receives a list of observations $\theta_1, ..., \theta_n$, an initial estimate of the transition matrix M, an estimate of the initial distribution π and an estimate of the emission probabilities B. During the first iteration the parameters B, M, and π can be arbitrary. Based on the observation sequence, the parameters of the transition matrix, emission probabilities and initial distribution are refined iteratively. During each iteration, the error of the posterior estimates B', M', and π' decreases. In our scenario B and π are given and fixed, we therefore concentrate on the computation of M'. The iteration formula of the adapted matrix in the Baum-Welch algorithm is as follows:

$$m'_{ij} = \frac{\sum_{t=1}^{n} P(o(t) = s_i, o(t+1) = s_j | \Theta, \lambda)}{\sum_{t=1}^{n} P(o(t) = s_i | \Theta, \lambda)}$$

As the algorithm works with time-homogeneous transition matrices, each observation can be assumed to be taken at any time. Therefore, the Baum-Welch algorithm can take the average over all transition probabilities at all points in time; it has been shown that each iteration of the formula decreases the error of the parameter estimates. As we assume time-inhomogeneous transition matrices, averaging is impossible. However, the numerator of this formula without building the sum is semantically equivalent to the numerator of our algorithm during the backward phase; the same holds for the denominator. Therefore our algorithm can be seen as a relative of the Baum-Welch algorithm in the context of a Markov models, given certain observations and time-inhomogeneous transition matrices with B and π fixed.

12.3 Spatial Pruning

Pruning objects in probabilistic NN search can be achieved by employing appropriate index structures available for querying uncertain spatio-temporal data. In this work, we use the *UST-tree* [40]. In this section, we briefly summarize the index and show how it can be employed to efficiently prune irrelevant database objects, identify result candidates, and find influence objects that might affect the ∀NN probability of a candidate object.

The UST-Tree. Given an uncertain spatio-temporal object o, the main idea of the UST-tree is to conservatively approximate the set of possible (*location*, *time*) pairs that o could have possibly visited, given its observations Θ^o. In a first approximation step, these (*location*, *time*) pairs, as well as the possible (*location*, *time*) pairs defined by θ_i^o and θ_{i+1}^o are minimally bounded by rectangles. Such a rectangle, for observations θ_i^o and θ_{i+1}^o is defined by the time interval $[t_i^o, t_{i+1}^o]$, as well as the minimal and maximal longitude and latitude values of all reachable states. The second approximation step bounds the (*location*, *time*) pairs at a given point in time by a parameterized spatio-temporal diamond, providing better pruning power than the rectangles, however at a higher computational complexity.

Example 3. *Consider Figure 12.5, where four objects objects A, B, C and D are given by three observations at time* 0, 5 *and* 10. *For each object, the set of possible states in the corresponding time intervals* $[0, 5]$ *and* $[5, 10]$ *is approximated by two minimum bounding rectangles. For illustration, the set of possible states at each point of time is also depicted by dashed rectangles; these dashed rectangles represent the spatio-temporal diamond.*

The UST-tree indexes the resulting rectangles and diamonds using an R^*-tree ([28]). We now discuss how such an index structure can be used for the evaluation of P∀NNQ and P∃NNQ queries.

Pruning candidates of P∀NNQ queries. For a P∀NNQ query, an object must have a non-zero probability of being the closest object to q, for all timestamps in the query interval. As a consequence, to find candidate objects for the P∀NNQ query, we have to consider for all objects $o \in D$ whether for each $t \in q.T$ there does not exist an object $o' \in D$ such that $MINDIST(o(t), q(t)) > MAXDIST(o'(t), q(t))$. Here, $MINDIST(o(t), q(t))$ ($MAXDIST(o(t), q(t))$) denotes the minimum (maximum) distance between the possible states of $o(t)$ and $q(t)$. Thus, the set of candidates $C_\forall(q)$ of a P∀NNQ is defined as

$$C_\forall(q) =$$

$$\{o \in D | \forall t \in q.T : MINDIST(o(t), q(t)) \leq min_{o' \in D} MAXDIST(o'(t), q(t))\}$$

Applying spatial pruning on the leaf level of the UST-tree, we have to apply the $MINDIST$ and $MAXDIST$ distance computations on the minimum bounding rectangles on the leaf level in consideration of the time intervals associated with these leaf entries. In our example, given the query point q with $q.T = [2, 8]$, only object A is a candidate, since $MINDIST(q(t), A(t)) \leq MAXDIST(q(t), o(t))$ for all $o \in D$ in the time intervals [0,5] and [5,10], both together covering $q.T$. Objects B, C and D can be safely pruned.

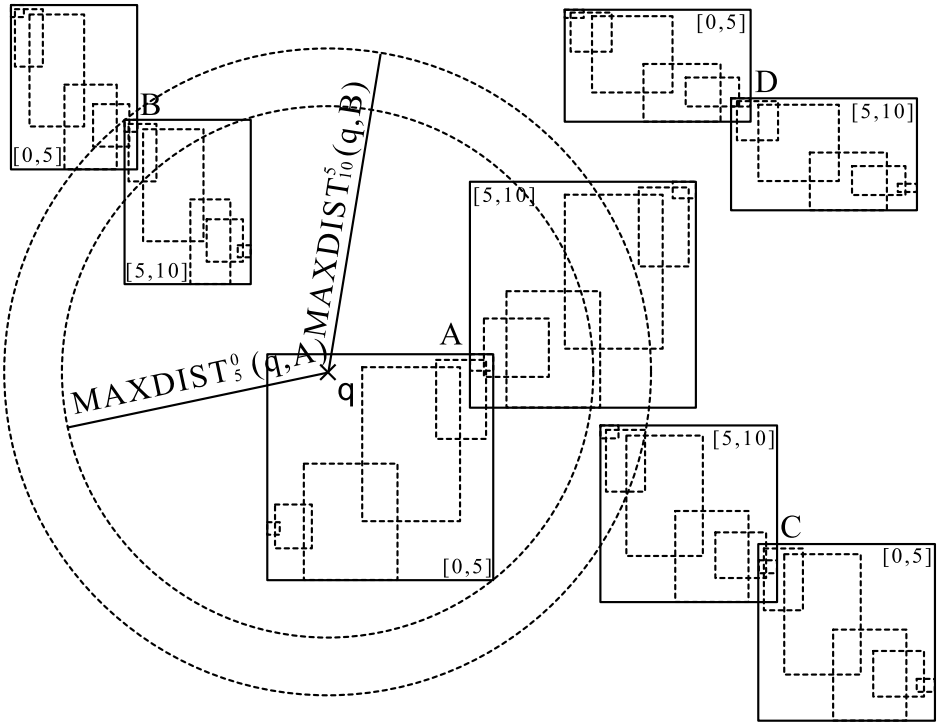


Figure 12.5: Spatio-Temporal Pruning Example.

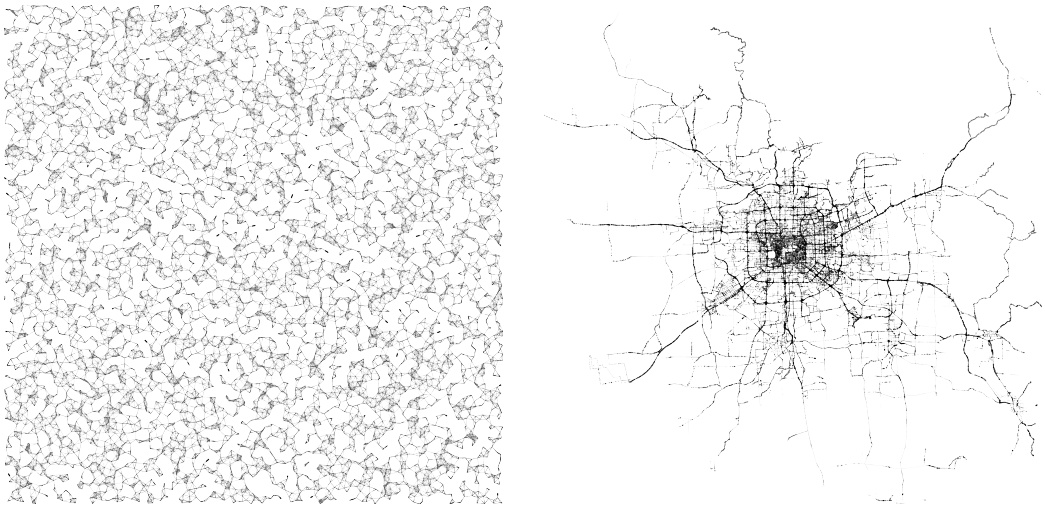
It is important to note that pruned objects, i.e., objects not contained in $C_\forall(q)$ may still affect the ∀NN probability of other objects and even may prune other objects. For example, though object B is not a candidate, it affects the ∀NN probability of all other objects and contributes to prune possible worlds of object A, because $MAXDIST(q(t), A(t)) > MINDIST(q(t), B(t))\ \forall t \in [5, 10]$. All objects having at at least one timestamp $t \in q.T$ a non-zero probability being the NN of q may influence the ∀NN probability of other objects. Since we need these objects for the verification step, we have to maintain them in an additional list $I_\forall(q)$:

$$I_\forall(q) =$$

$$\{o \in D | \exists t \in T : MINDIST(o(t), q(t)) \leq min_{o' \in D} MAXDIST(o'(t), q(t))\}$$

To perform spatial pruning at the non-leaf level of the UST-tree, we can analogously apply $MINDIST$ and $MAXDIST$ on the MBRs of the non-leaf level.

Pruning for the P∃NNQ query. Pruning for the P∃NNQ query is very similar to that for the P∀NNQ query. However, we have to consider that an object being the nearest neighbor for a single point in time is already a



(a) State Space (Synthetic Data) (b) State Space (Real Data)

Figure 12.6: Examples of the models used for synthetic and real data. Black lines denote transition probabilities. Thicker lines denote higher probabilities, thinner lines lower probabilities. The synthetic model consists of 10k states.

valid query result. Therefore, no distinction is made between *candidate*s and *influence object*s. Every pruner can be a valid result of the P∃NNQ query, such that each object with a $MINDIST$ smaller than the pruning distance has to be refined. The remaining procedure of the P∃NNQ-algorithm is equivalent to P∀NNQ-pruning.

12.4 Experimental Evaluation

Setup Our experimental evaluation focuses on the efficiency and effectiveness of sampling-based P∀NNQ, P∃NNQ and PCNNQ queries. We conducted a set of experiments to verify both the effectiveness and efficiency of the proposed solutions, using a desktop computer having an Intel i7-870 CPU at 2.93 GHz and 8GB of RAM. All algorithms were implemented in C++.

Artificial Data. Artificial data for our experiments was created in three steps: state space generation, transition matrix construction and object creation. First, the data generator constructs a two-dimensional Euclidean *state space*, consisting of N states. Each of these states is drawn uniformly from the $[0, 1]^2$ square. In order to construct a transition matrix, we derive a graph by introducing edges between any point p and its neighbors having a

Table 12.1: Parameters varied during our experimental evaluation (synthetic data). Differing parameters for continuous experiments are denoted by a superscript c. Default values are denoted in bold.

Variable	Values	Unit
$\|D\|$	1k, **10k**, 20k	objects
$N = \|\mathbb{S}\|$	10k, **100k**, 500k	states
τ	**0**,**0.1**^{c}, 0.5^c, 0.9^c	probability
b	6, **8**, 10	edges per node

distance less than $r = \sqrt{\frac{b}{N*\pi}}$ with b denoting the average branching factor of the underlying network. This parameter ensures that the degree of a node does not depend on the number of states in the network. Each edge in the resulting network represents a non-zero entry in the transition matrix. The transition probability of this entry is indirectly proportional to the distance between the two vertices. An example of such a model can be found in 12.6a.

To create observations of an object o, we sample a sequence of states and compute the shortest paths between them, modeling the motion of o during its whole lifetime (which we set to 100 steps by default). To add uncertainty to the resulting path, every l^{th} node, $l = i * v$, $v \in [0, 1]$, of this trajectory is used as an observed state. i denotes the time between consecutive observations and v denotes a lag parameter describing the extra time that o requires due to deviation from the shortest path; the smaller v, the more lag is introduced to o's motion. With, for example, $v = 0.8$, the object moves only with 80% of its maximum speed. The resulting uncertain trajectories were distributed over the database time horizon (default: 1000 timestamps) and indexed by a UST-tree [40]. As a pruning step for query evaluation, we employed the UST-tree's MBR filtering approach described in Section 12.3. Our experiments concentrate on evaluating nearest neighbor queries given a certain query state. These states were uniformly drawn from the underlying state space.

Real Data. We also generated a data set from a set of GPS trajectories of taxis in the city of Beijing [109] using map matching; the code for map matching was provided by one of our students in the context of a diploma thesis. First, trajectories from the dataset below a given GPS-frequency were filtered out since these trajectories are not fine-granular enough to provide useful information during the training step. The remaining trajectories were interpolated to obtain measurements with a frequency of 1Hz. These trajectories were then map matched to a reduced Beijing-graph obtained from OpenStreetMap (OSM). Due to the sparsity of data, we assume that

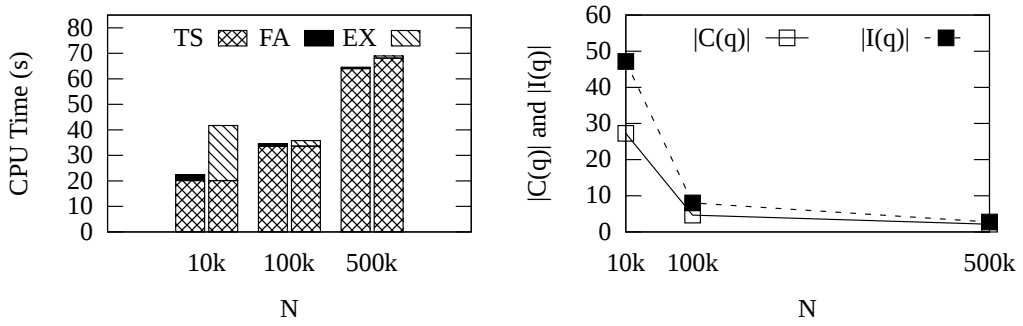


Figure 12.7: Varying the Number of States N

a-priori, all objects utilize the same Markov model M. The time domain is discretized to one tic every 10 seconds. From the map matched trajectories, the transition matrix was extracted by aggregating the turning probabilities at crossroads. OSM-nodes with no hits in the underlying training data were filtered out. The state space was then formed by the remaining nodes of the OSM graph, all in all 68902 states. The resulting model is visualized in Figure 12.6b. Certain trajectories of cars were taken directly from the map matched trajectories, but in order to ensure comparability to the artificial data have been capped at a length of 100 tics and distributed in the database horizon. The certain trajectories were then made uncertain by taking every l-th GPS measurement as an observation; the discarded GPS measurements serve as ground truth for effectiveness experiments. For the real data experiment varying the number of objects, we set $l = 8$.

12.4.1 Evaluation: P∀NNQ and P∃NNQ

For performance analysis, the sampling approach (Section 12.2) is divided into two phases. In the first phase the trajectory sampler (TS) is initialized (the adapted transition matrices are computed according to Algorithm 6). This phase can be performed once and used for all queries. In the second phase, the actual sampling of 10k trajectories (per object) for the approximate P∀NNQ (FA) and P∃NNQ (EX) queries is performed. An overview over the evaluated parameters can be found in Table 12.1. These parameters lead to a total of $110k$ observations (11 per object) and $100k$ diamonds for the UST-index.

Varying N. In the first experiment (Figure 12.7) we investigate the effect of an increasing state space size N, while keeping a constant average branching factor of network nodes. This effect corresponds to expanding the underlying state space, e.g., from a single country to a whole continent. In

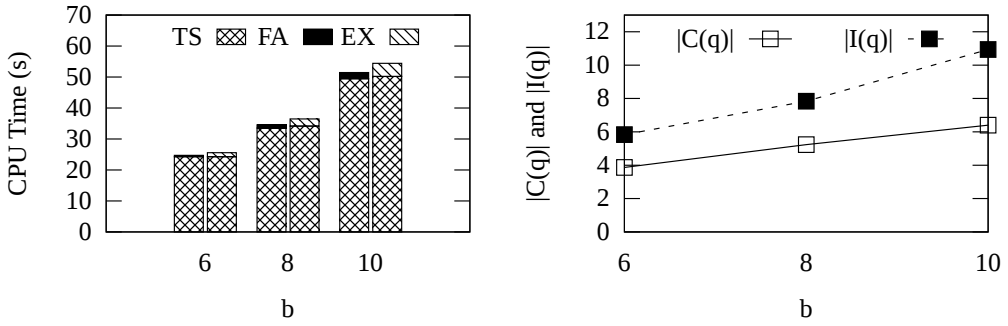


Figure 12.8: Varying the Branching Factor b

Figure 12.7 (left) we can see that increasing N leads to a sub-linear increase in the run-time of the sampling approaches. This effect can be mostly explained by two aspects. First, the size of the a-priori model increases linearly with N, since the number of non-zero elements of the sparse matrix M increases linearly with N. This leads to an increase of the time complexity of matrix operations, and therefore makes adapting transition matrices more costly. At the same time, the number of candidates $|C(t)|$ and influence objects $|I(t)|$ (see Section 12.3) decreases significantly as seen in Figure 12.7 (right) because the degree of intersection between objects decreases with a higher number of states, making pruning more effective, and therefore reducing the actual cost for sampling.

The runtime difference among sampling the P∀NNQ and P∃NNQ query diminishes with increasing N because the size of the result set of the P∀NNQ increases with N while P∃NNQ produces less results with increasing N. The P∃NNQ runtime is also higher than the P∀NNQ runtime because for the P∃NNQ query not only candidate objects are possible results, but also influence objects.

Varying b. Figure 12.8 evaluates the branching factor b, i.e., the average degree of each network node. As expected, Figure 12.8 (left) shows that an increasing branching factor yields a higher run-time of all approaches due to a higher number of non-zero values in vectors and matrices, making computations more costly. Furthermore, in our setting, a larger branching factor also increases the number of influence objects, as shown in Figure 12.8 (right).

Varying $|D|$. The number of objects (Figure 12.9) leads to a decreasing performance as well. The more objects stored in a database with the same underlying motion model, the more candidates and influence objects are found during the filter step. This leads to an increasing number of probability

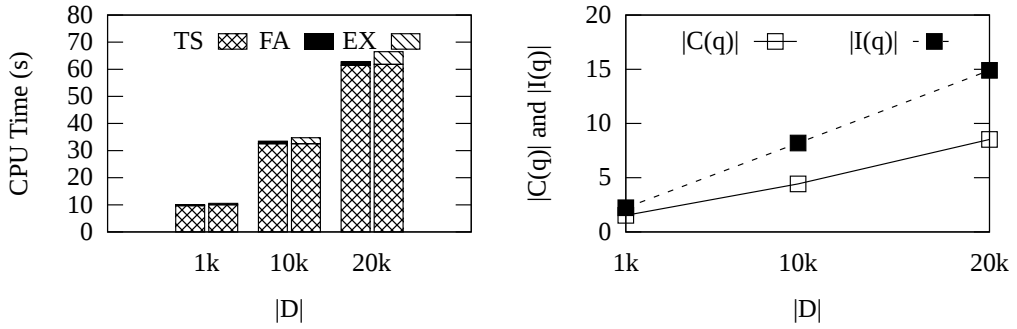


Figure 12.9: Varying the Number of Objects $|D|$

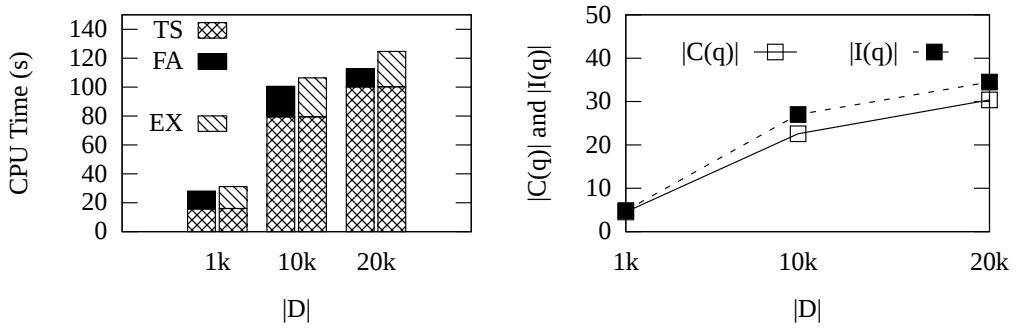


Figure 12.10: Real Data: Varying the Number of Objects

calculations during refinement, and hence a higher query cost.

Real Dataset. We conducted additional experiments to evaluate P∀NNQ and P∃NNQ queries on the taxi dataset (Figure 12.10). The underlying state space consisting of 68902 states is a bit smaller than the default synthetic dataset. Based on this dataset, we ran an experiment varying the number of objects between 1000 and 20000. The smaller size of the state space leads to a higher objects density, leading to a larger number of candidates and influence objects than the corresponding experiment on the artificial dataset. Additionally, the non-uniform distribution of taxis in the city is more dense close to the city center, making queries in this area more costly due to the higher number of candidates and pruners. Further note that in the real dataset, the motion patterns of objects are more diverse than on the synthetic data. There are taxis standing still, and taxis moving quite fast. Standing taxis have a larger area of uncertainty between observations, such that these objects reduce the performance of query evaluation.

Sampling Efficiency. In the next experiment we evaluate the overhead of the traditional sampling approach (using the a-priori Markov model

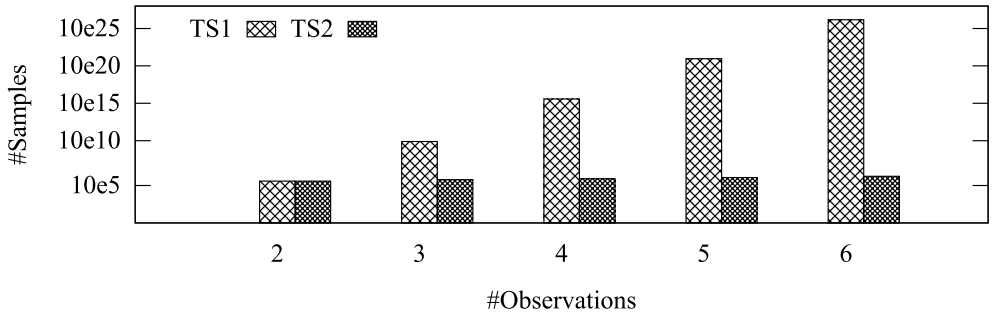
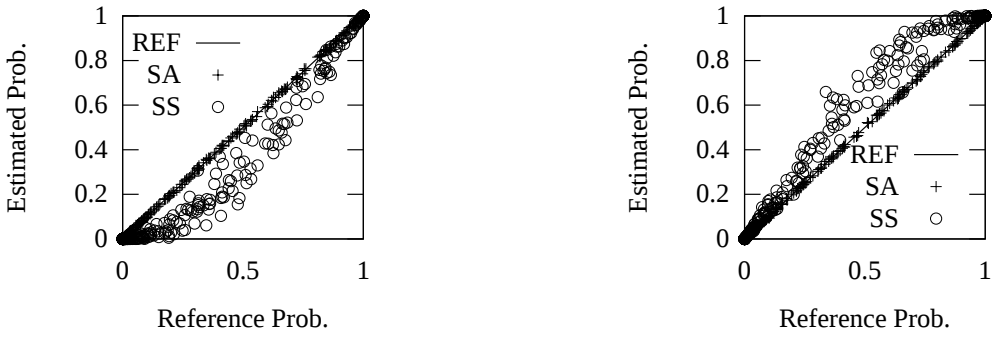


Figure 12.11: Efficiency of Sampling without Model Adaption.

only) compared to the approach presented in Section 12.2 which uses the a-posteriori model again based on the artificial dataset. The first, traditional approach (TS1, number of samples has been estimated) discards any trajectory not visiting all observations. As discussed in Section 12.2.1, the expected number of attempts required to draw one sample that hits all observations increases exponentially in the number of observations. This increase is shown in Figure 12.11, where the expected number of samples is depicted with respect to the number of observations. This approach can be improved, by segment-wise sampling between observations (TS2). Once the first observation is hit, the corresponding trajectory is memorized, and further samples from the current observation are drawn until the next observation is hit. The number of trajectories required to be drawn in order to obtain one possible trajectory, i.e., the trajectory hits all observations, is linear to the number of observations when using this approach. We note in Figure 12.11, that in either approach at least 100k samples are required even in the case of having only two observations. In contrast using the approach presented in Section 12.2, the number of trajectories that need to be sampled, in order to obtain a trajectory that hits all observations, is always *one*.

Sampling Precision and Effectiveness. Next, we evaluate the precision of our approximate P∀NNQ and P∃NNQ query and an aspect of a competitor approach [87]. The latter approach has been tailored for *reverse* NN queries, but can easily be adapted to NN query processing. Essentially, this approach performs a snapshot query $P\forall NNQ(q, D, \{t\}, \tau)$ for each $t \in T$. $P\forall NN(o, q, D, T)$ is estimated by $\prod_{t \in T} P\forall NN(o, q, D, \{t\})$. $P\exists NN(o, q, D, T)$ can be approximated by $1 - \prod_{t \in T}(1 - P\exists NN(o, q, D, \{t\}))$. The scatter plot in Figure 12.12 (left) illustrates a set of P∀NN probabilities on synthetic data ($v = 0.2$, $|T| = 5$). For each experiment, we estimate probabilities by our sampling approach (SA) (Section 12.2) with (10^4) sam-



(a) P∀NN (b) P∃NN

Figure 12.12: Effectiveness of Sampling, P∀NN and P∃NN

ples and by the adapted approach of [87] (SS). We approximated the exact approach (REF) by drawing a very high (10^6) number of samples.

We model each case as a (x,y) point, where x models the reference (REF) and y the estimated probability (SA or SS). For (REF) the results always lie on the diagonal identity function depicted by a straight line. Probabilities of SA are very close to the diagonal, showing that our sampling solution tightly approximates the results of the exact P∀NNQ query. Concerning the snapshot approach, a strong bias towards underestimating probabilities can be observed for the P∀NNQ query. The snapshot-based P∃NNQ-query overestimates the results. This bias is a result of treating points of time mutually independent. In reality, the position at time t must be in vicinity of the position at time $t-1$, due to maximum speed constraints. This positive correlation in space directly leads to a nearest neighbor correlation: If o is close to q at time $t-1$, then o is likely close to q at time t. And clearly, if o is more likely to be close to q at time t, then o is more likely to be the NN of q at time t. This correlation is ignored by snapshot approaches. It can be seen that the systematic error of such snapshot approaches is quite high.

The number of samples required to obtain an accurate approximation of the probability of a binomial distributed random event such as the event that o is the NN of q for each time $t \in T$ has been studied extensively in statistics [107]. Thus the required number of samples is not explicitly evaluated here.

Effectiveness of the Forward-Backward Model. We tested the effectiveness of the forward-backward model adaption in comparison to other approaches on the real dataset with a time interval between observations of 100 seconds. Figure 12.13 shows the mean error of these approaches, computed during each point of time, evaluated over a time interval of 30 tics (5 minutes). The mean error has been computed in leave-one-out manner, i.e.

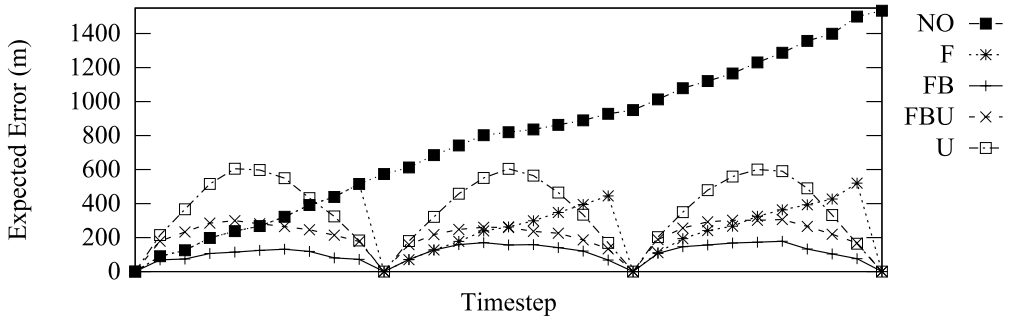


Figure 12.13: Real Data: Effectiveness of the Model Adaption

trajectories for computing the error have not been used to train the model in order to avoid overfitting. The figure visualizes the error of the a-priori model (NO) considering only the first observation, the model adapted by the forward phase only (F) and the forward-backward-adapted a-priori-model (FB) from this paper. We further implemented two additional approaches. The uniform approach (U), a competitor corresponding to [93, 94], discards all probability information of FB and, due to a lack of better knowledge, assumes all reachable states at a given time to have a uniform probability. The difference to the cylinders and beads approximation models presented in [93, 94] is that these models use conservative approximations that may include some (time, state) pairs actually having a zero probability for an object to be located at. Thus, our U approach is at least as good as the cylinders and beads approximation models in terms of effectiveness, regardless of the approximation type used. The approach FBU is equivalent to FB, however turning probabilities in the transition matrix are equally distributed instead of learning the exact transition probabilities from the underlying map data. First note that the approach not incorporating any observations (NO), yields significant errors compared to the remaining approaches. Clearly, observations can reduce errors and uncertainty during query evaluation. The forward-only approach (F) reduces this error, however the error is still high especially directly before an observation. This problem is solved by the forward-backward approach investigated in Section 12.2 (FB). Note that even if the Markov chain is assumed to be uniformly distributed (FBU), the results are still good, but worse than with the actual learned probabilities (FB). This is good news, as it shows that even a non-optimally learned Markov chain can lead to useful results, however with a slightly higher error. This good performance comes from the fact that with a uniform transition distribution the diamond-shaped space of possible time-state pairs still has

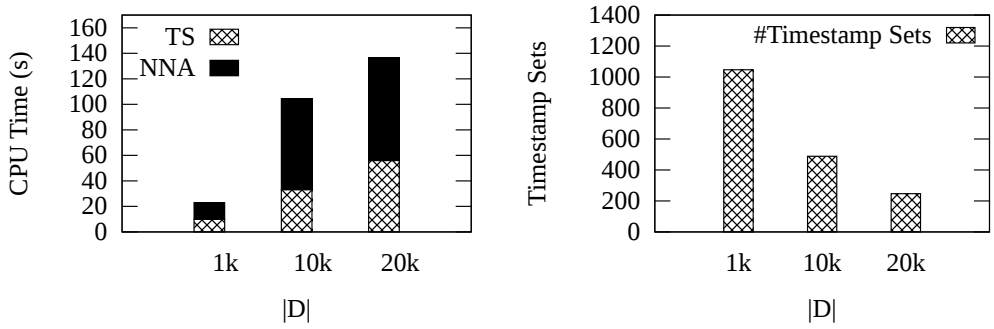


Figure 12.14: PCNN: Varying the Number of Objects

high probabilities in the center of the diamond, since trajectories near the center of the bead will have a higher likelihood than trajectories close to the beads boundary. This stands in contrast to the uniform approach (U) that models all states at the diamonds border to have the same probabilities as the states in the diamonds center; explaining why U performs worse than FBU. To conclude, combining observations with a sufficiently accurate transition matrix can produce the most accurate results.

12.4.2 Continuous Queries

In our experimental evaluation on continuous queries we compare the runtime and the size of the (unprocessed) result set for various database sizes and values of the threshold τ (default $\tau = 0.5$) using artificial data. After query evaluation, this result set can be further condensed, e.g. by removing all smaller sets of timestamps that are already implicitly contained in a larger set of timestamps.

Increasing the number of objects stored in the database leads to an increase in the time needed to compute the a-posteriori Markov model (TS) for each object (cf. Figure 12.14 (left)). This result is equivalent to the result for P∀NNQ queries, since a-posteriori models have to be computed for either query semantics. However, the time required to obtain a sufficient number of samples (SA) is much higher, since probabilities have to be estimated for a number of sets of time intervals, rather than for the single interval T. This increase in run-time is alleviated by the effect that the number of candidate time intervals obtained in the candidate time interval generation step of our Apriori-like algorithm decreases (Figure 12.14 (right)). This effect follows from the fact that more objects lead to more pruners, leading to smaller probabilities of time intervals, leading to fewer candidate time intervals.

The results of varying τ can be found in Figure 12.15. Clearly an in-

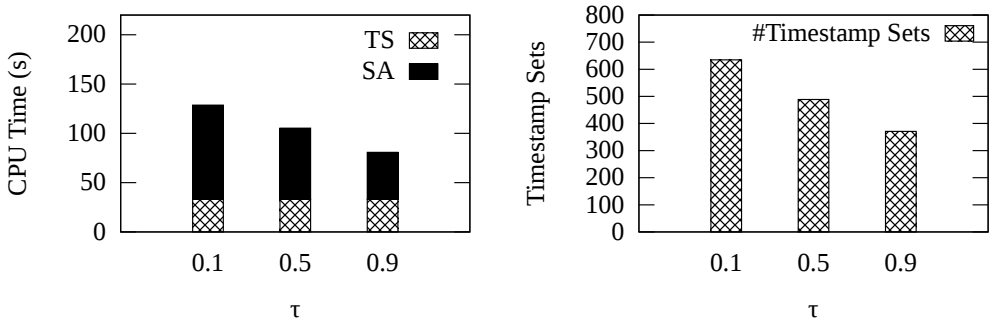


Figure 12.15: PCNN: Varying τ

creasing probability threshold decreases the average size of the result (Figure 12.15 (right)). Consequently, the computational complexity of the query decreases as fewer candidates are generated. Figure 12.15 (left) shows that the run-time of the sampling approach becomes very large for low values of τ, since samples have to be generated for each relevant candidate set. Similar to the Apriori-algorithm, the number of such candidates grows exponentially with T, if τ is small.

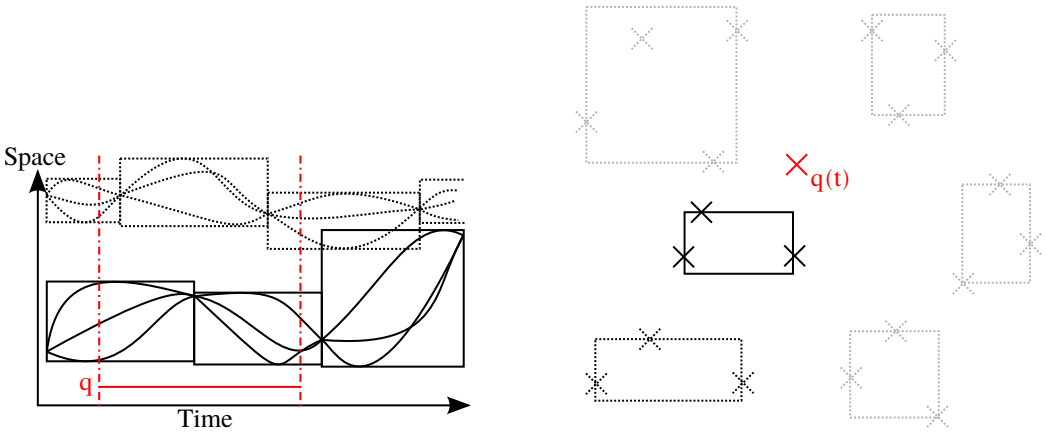
Chapter 13

Reverse Nearest Neighbor Queries

In the last chapter we addressed efficient probabilistic nearest neighbor query processing on spatio-temporal data. In this chapter we extend our results to reverse nearest neighbor queries, building upon the previously introduced sampling approach.

13.1 PRNN Query Processing

To process the two PRNN query types defined in Section 11.1, we proceed in a similar way as in the case of nearest neighbor queries. First, we perform a temporal and spatial filtering to quickly find candidates in the database and exclude as many objects as possible from further processing. In the second step we perform a verification of the remaining candidates to obtain the final result. Although different solutions to this problem are possible, we decided to describe an algorithm that splits the query involving several timesteps into a series of queries involving only a single point in time during the pruning phase. The interesting point in this algorithm is that it shows that spatial pruning *does not introduce errors* when disregarding temporal correlations. However, disregarding temporal correlation during the probability computation phase *does introduce errors*, as we have seen in Section 12.4.



(a) Uncertain trajectories in 1D space

(b) space (2D) at one point of time t

Figure 13.1: Spatio-temporal filtering (only leaf nodes are shown)

13.1.1 Temporal and Spatial Filtering

In the following we assume that the uncertain trajectory database D is indexed by an appropriate data structure, like the UST tree [40].[1] For the UST tree, the set of possible (location,time)-tuples between two observations of the same object is conservatively approximated by a minimum bounding rectangle (MBR) (cf Figure 13.1a). All these rectangles are then used as an input of an R*-Tree.

The pseudo code of the spatio-temporal filter is illustrated in Algorithm 7. The main idea is to (i) perform a *candidates search* for each timestamp t in the query interval T (cf. Figure 13.1a) separately and (ii) for each candidate find the set of objects which are needed for the verification step (influence objects S_{ifl}^{cnd}). For each t we consider the R-Tree $\mathcal{D}^t$ which results from intersecting the time-slice t with the R-Tree $\mathcal{D}$ (cf Figure 13.1b). This can be achieved efficiently during query processing by simply ignoring pages of the index that do not intersect with the value of t in the temporal domain. For each time t a reverse nearest neighbor candidate search [103] is performed.

Therefore, a priority queue Q is initialized, which organizes its entries by their minimum distance to the query object q. Q initially only contains the root node of $\mathcal{D}^t$. Additionally we initialize two empty sets. S_{cnd} contains all RNN candidates which are found during query processing and S_{prn} contains objects (leaf entries) or entries which have been verified not to contain can-

[1] Note that the techniques for pruning objects do not rely on this index and thus can also be applied in scenarios where there is no index present.

Algorithm 7 Spatio-Temporal Filter for the P∀RNN query

Require: q,T, $\mathcal{D}$
1: $\forall t \in T : S^t_{cnd} = \emptyset$
2: $\forall t \in T : S^{cnd,t}_{ifl} = \emptyset$
3: **for** each $t \in T$ **do**
4: initialize priority queue Q ordered by minimum distance to q
5: insert root entry of $\mathcal{D}^t$ into Q
6: $S_{prn} = \emptyset$
7: **while** Q is not empty **do**
8: retrieve first entry e from Q
9: **if** $\exists e_2 \in Q \cup S_{prn} \cup S^t_{cnd} : Dom(e_2, q, e)$ **then**
10: $S_{prn} = S_{prn} \cup \{e\}$
11: **else if** e is directory entry **then**
12: **for** each child ch in e **do**
13: insert ch in Q
14: **end for**
15: **else if** e is leaf entry **then**
16: $S^t_{cnd} = S^t_{cnd} \cup \{e\}$
17: **end if**
18: **end while**
19: **for** each $cnd \in S^t_{cnd}$ **do**
20: **if** $\exists le : Dom(le, q, e)$ **then**
21: discard candidate and continue;
22: **end if**
23: $S^{cnd,t}_{ifl} = \{le : \neg Dom(le, q, e) \wedge \neg Dom(q, le, e)\}$
24: **end for**
25: **end for**
26: $S_{ref} = \bigcap_{t \in T} S^t_{cnd}$
27: **for** each $cnd \in S_{ref}$ **do**
28: $S^{cnd}_{ifl} = \bigcup_{t \in T} S^{cnd,t}_{ifl}$
29: **end for**
30: **return** $\forall cnd \in S_{ref} : (cnd, S^{cnd}_{ifl})$

didates. Then, as long as there are entries in Q, a best-first traversal of $\mathcal{D}^t$ is performed. For each entry e which is de-heaped from Q, the algorithm checks whether e can be pruned (i.e., it cannot contain potential candidates) by another object or entry e_2 which has already been seen during processing. This is the case if e_2 dominates q w.r.t. e, i.e. e_2 is definitely closer to e than

q which implies that e cannot be RNN of q. To verify spatial domination Dom we adapt the technique proposed in [29], which we have already used in Section 7.1 of this thesis.

Algorithm 8 Verification for the P∀RNN query

Require: q,T, cnd, S_{ifl}^{cnd}, num_samples
1: num_satisfied = 0
2: **for** $0 \leq i \leq num\_samples$ **do**
3: num_satisfied = num_satisfied + 1
4: $cnds = sampleTrajectory(cnd)$
5: **for all** $o \in S_{ifl}^{cnd}$ **do**
6: $os = sampleTrajectory(o)$
7: **if** $\exists t \in T : dist(cnds(t), os(t)) < dist(cnds(t), q(t))$ **then**
8: num_satisfied = num_satisfied - 1
9: break
10: **end if**
11: **end for**
12: **end for**
13: **return** num_satisfied/num_samples

An entry which is pruned by this technique is moved to the S_{prn} set. If an entry cannot be pruned it is either moved to the candidate set if it is a leaf entry or put into the heap Q for further processing.

After the index traversal, for each candidate it can be checked if the candidate is pruned by another object. If the other object it is definitely closer to the candidate than the query, the candidate object can be discarded (see line 21). To find the set of objects that could possibly prune a candidate, we can again use the domination relation (see line 23). An object (leaf entry le) is necessary for the verification step if it might be closer to the candidate than the query, which is reflected by the statement in this line. A more detailed description of this step can be found in [103].

After performing this process for each timestamp t we have to merge the results for each point of time to obtain the final result. In the case of a P∀RNN we intersect the *candidate* sets for each point in time. The only difference for the P∃RNN query is that we have to unify the results in this step.

The *influencing* objects of each candidate have to be unified for each time $t \in T$ to obtain the final set of influencing objects. The algorithm ultimately returns a list of candidate objects together with their sets of influencing objects.

13.1.2 Verification

The objective of the verification step is to compute, for each candidate c, the probability $P\exists RNN(c, q, D, T)$ ($P\forall RNN(c, q, D, T)$) and compare this probability with the probability threshold τ. An interesting observation is, that we were able to prune objects based on the consideration of each point $t \in T$ separately, however as shown in Section 11.1.3 it is not possible to obtain the final probability value by just considering the single time probabilities. Thus our approach relies on sampling of possible trajectories for each candidate and the corresponding influence objects by falling back on the sampling approach from Section 12.2. On each sample (which then consists of a set of certain trajectories) we then are able to efficiently evaluate the query predicate. Repeating this step often enough we are able to approximate the true probability of $P\exists RNN(c, q, D, T)$ ($P\forall RNN(c, q, D, T)$) by the percentage of samples where the query predicate was satisfied. The algorithm for the verification of the P∀RNN query is given in Algorithm 8. Note, that it is possible to early terminate sampling of influence objects, when we find a time t where the candidate is not closer to q than to the object trajectory just sampled. For the P∃RNN query we can also implement this early termination whenever we can verify the above for each point of time.

13.2 Experiments

Our experimental evaluation of probabilistic RkNN queries reuses the experimental setup from the previous chapter. Again, we focus on testing the efficiency of our algorithm for P∀RNNQ and P∃RNNQ, by measuring (i) the number of candidates and influence objects remaining after pruning irrelevant objects based on their spatio-temporal MBRs, and (ii) the runtime of the refinement procedure, i.e. sampling. We split the sampling procedure into adapting the transition matrices (building the trajectory sampler) of the objects and the actual sampling process, as the adaption of transition matrices can be done as a preprocessing step, storing the adapted transition matrices of each object in the database. For our experiments we determined candidates based on MBR filtering, i.e. the MBR over all states that can be reached by an object between two consecutive observations. For our experiments we concentrate on the case of the query q being given as a query state.

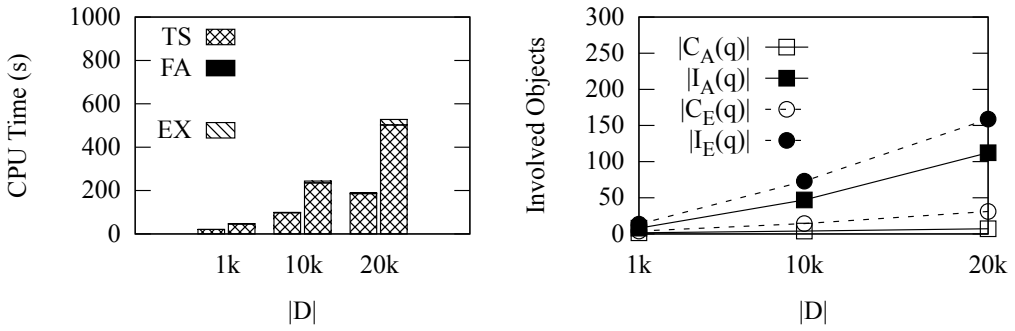


Figure 13.2: Synthetic Data, Varying $|D|$.

13.2.1 Evaluation: P∀RNNQ and P∃RNNQ

The default setting for our performance analysis is as follows: We set the number of states to $N = |\mathbb{S}| = 100$k, the database size (number of objects) to $|D| = 10k$, the average branching factor (synthetic data) to $b = 8$, probability threshold $\tau = 0$. The length of the query interval was set to $|T| = 10$. To compute a result probability, 10k possible worlds were sampled from the candidate objects.

In each experiment, the left plot shows a stacked histogram, visualizing the cost for building the trajectory sampler (TS) and the actual cost for sampling (EX for the $P\exists RNN$ query, FA for the $P\forall RNN$ query). The right plot first visualizes the number of candidates ($C_x(q)$) for the $P\exists RNN$ ($x = E$) and $P\forall RNN$ ($x = A$) query, i.e. the number of objects for which the nearest neighbor has to be computed. Second it visualizes the number of pruners or influence objects ($I_x(q)$), i.e. the number of objects that can prune candidates. Clearly the number of candidates and pruners is different for $P\forall RNN$ and $P\exists RNN$ queries: for the $P\exists RNN$ query objects not totally overlapping the query interval can be candidates, increasing the number of candidates. For the $P\forall RNN$ query, candidates can be definitely pruned if at least at one point of time another object prunes the candidate object.

Varying $|D|$. Let us first analyze the impact of the database size (see Figure 13.2), i.e. the number of uncertain objects on the runtime of a probabilistic reverse nearest neighbor query. First of all note that the number of candidates and influence objects increases if the database gets large. This is the case because with more objects, the density of objects increases and therefore more objects become possible results of the RNN query. Also, clearly, the number of influence objects increases due to the higher degree of intersection of MBRs. Second the probability computation of candidate objects during refinement is mostly determined by the computation of the

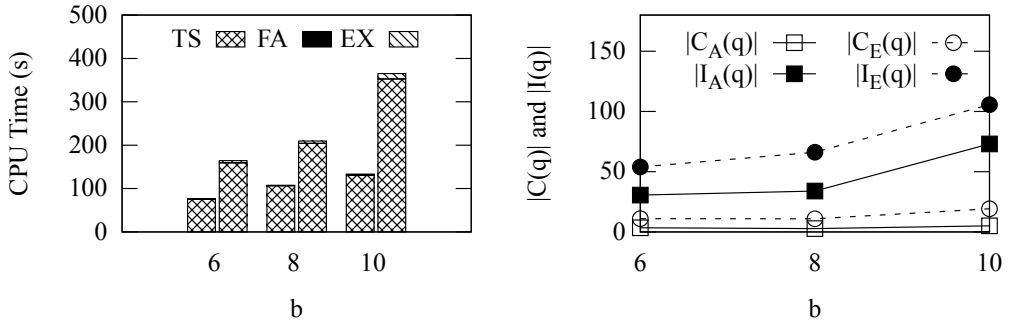


Figure 13.3: Synthetic Data, Varying b.

adapted transition matrices, i.e. building the trajectory sampler. This is actually good news, as this step can also be performed offline and the resulting transition matrices can be stored on disk. Last note that evaluating the $P\exists RNN$ query during the actual sampling process (EX and FA for $P\exists RNN$ and $P\forall RNN$ respectively) is also more expensive than the $P\forall RNN$ query, as more samples have to be drawn for each candidate object: possible worlds of $P\forall RNN$-candidates can be pruned if a single object at a single point in time prunes the candidate. For the $P\exists RNN$ query, all points in time have to be pruned which is much less probable. Additionally, more candidates than for the $P\forall RNN$ have to be evaluated, also increasing the complexity of the $P\exists RNN$ query.

Varying b. The effect of varying the branching factor b (see Figure 13.3) is similar but not as severe as varying the number of objects. Increasing the branching factor increases the number of states that can be reached during a single transition. In our setting, this also increases the uncertainty area of an uncertain object, making pruning less effective, and therefore increasing the number of objects that have to be considered during refinement. As a result, the computational complexity of the refinement phase increases with increasing branching factor.

Varying $N = |\mathbb{S}|$. Increasing the number of states in the network (see Figure 13.4) while keeping the branching factor b constant shows an opposite effect on the number of candidates: as the number of states increases, objects become more sparsely distributed such that pruning becomes more effective. Note that for small networks especially the sampling process of the $P\exists RNN$ query becomes very expensive. This effect diminishes with increasing size of the network. Although less objects are involved, building the adapted transition matrices becomes more expensive, as for example matrix operations become more expensive with larger state spaces.

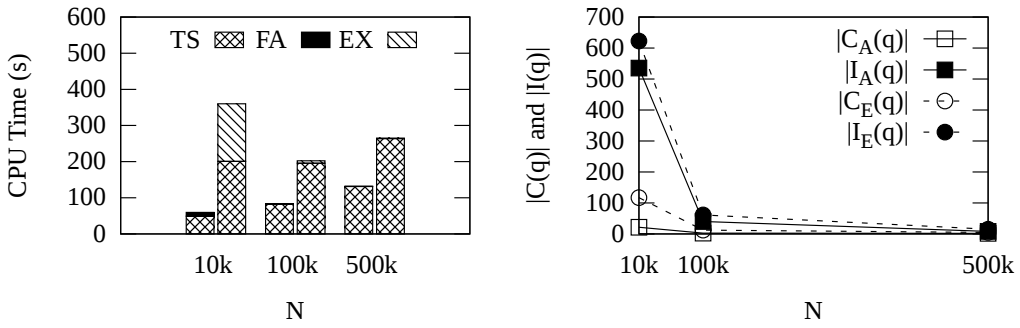


Figure 13.4: Synthetic Data, Varying $N = |\mathbb{S}|$.

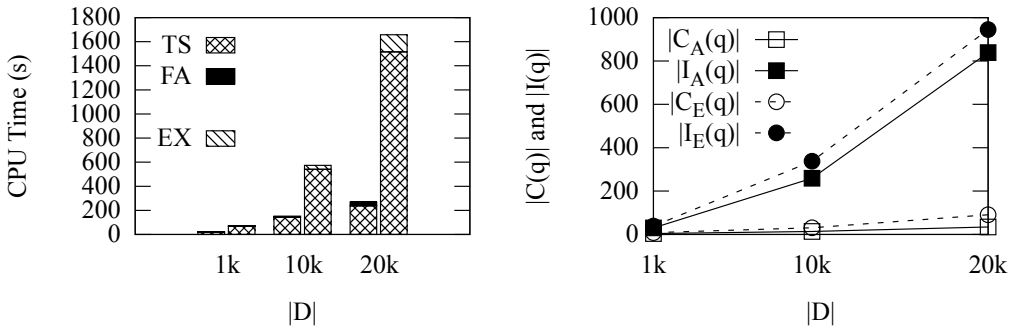


Figure 13.5: Real Data, Varying $|D|$.

Real Dataset. Last but not least we show results of the $P\forall RNN$ and $P\exists RNN$ queries on the real dataset (see Figure 13.5). We decided to vary the number of objects as the number of states and the branching factor is inherently given by the underlying model. The results are similar to the synthetic data, however more than twice as many objects have to be considered during refinement. We explain this exemplarily by the non-uniform distribution of taxis in the network. A part of this performance difference can also be explained by the slightly smaller network consisting of about 70k states instead of 100k states in the default setting.

Chapter 14

Conclusions

In Part III of this thesis, we addressed the problem of answering nearest neighbor and reverse nearest neighbor queries in uncertain spatio-temporal databases.

In the context of probabilistic nearest neighbor queries, we proposed three different query semantics: P∀NNQ queries, P∃NNQ queries and PCNN queries. We have first analyzed the complexity of these queries, showing that computing all of them has a high runtime complexity. These results provide general insights into the complexity of NN search over uncertain data in spatio-temporal databases since the Markov chain model is one of the simplest models that consider temporal dependencies. More complex models are expected to be at least as hard. To mitigate the problems of computational complexity, we used a sampling-based approach based on Bayesian inference. For the PCNNQ query we proposed to reduce the cardinality of the result set by means of an Apriori pattern mining approach. To cope with large trajectory databases, we introduced a pruning strategy to speed-up PNN queries exploiting the UST tree, an index for uncertain trajectory data. The experimental evaluation shows that our adapted a-posteriori model allows to effectively and efficiently answer probabilistic NN queries despite the strong a-priori Markov assumption.

Our second contribution in this part addressed probabilistic reverse nearest neighbor queries on uncertain spatio-temporal data. We defined two queries, the P∃RNNQ and the P∀RNNQ query and proposed pruning techniques to exclude irrelevant objects from costly probability computations. We then used the previously developed sampling technique to compute the actual P∃RNNQ and P∀RNNQ probabilities for the remaining candidate objects. During an extensive performance analysis on both synthetic and real data we empirically evaluated our theoretic results.

Part IV

kNN Queries for Image Retrieval

Chapter 15

Introduction

In the fourth part of this thesis we address applications of nearest neighbor queries in the area of computer vision, especially in the context of keypoint-based object recognition. In this field of research, images are described by a set of interest points, each of them represented by its spatial location, angle, characteristic scale and a feature vector describing the spatial neighborhood of the keypoint in the image. This rich description of images introduces a variety of challenges: Feature vectors are high-dimensional, making indexing difficult to achieve. At the same time, each image is described by a set of features, up to a few thousands, making an unoptimized linear scan over databases containing these features impossible even on relatively small datasets containing only a few thousand of images. Additionally, the set-based nature of such an object representation, describing a single image by a set of feature vectors, requires specialized querying techniques that do not only exploit information on feature vectors, but also geometric constraints on their corresponding keypoints' positions.

From a historical point of view, while the development of the SIFT-Descriptor [3] made effective object retrieval on a large scale feasible, its initial use of nearest neighbor queries lead to slow runtimes even on relatively small data sets. In 2003, the invention of the Bag of Visual Words (BoVW) technique [23] aimed at solving this issue by roughly approximating the matching step (which was by then relying on nearest neighbor queries) using quantization, initiating a whole new area of research. However soon the limitations of this rough approximation became obvious, enforcing the development of more accurate techniques for assigning query vectors to database features. Whilst initial approaches aiming at increasing the accuracy of the matching step such as soft assignment [110] were relatively close to the BoVW approach, the focus in recent years turned back more and more to approximate kNN queries [10, 111, 112, 11] due to their possible gain in matching

accuracy [113]: kNN queries provide an accurate ranking of the matching candidates and a measure of proximity between feature vectors and query vectors. This additional information can be exploited for weighting the scores of image matches, increasing retrieval accuracy considerably [113].

Current research on kNN processing in the image retrieval community focuses on maximizing accuracy, on minimizing the memory footprint of index structure and feature vectors, and on minimizing processing time. This goal is approached from different directions. In the area of real-valued features, such as SIFT [3], new indexing techniques have received a vast amount of interest even in the most prestigious computer vision conferences [10, 111, 112, 12]. Another important trend, aiming at memory reduction and at speeding up kNN queries, is the use of binary features: In the last years, a new kind of feature vector, not real-valued but rather binary-valued (e.g. [114, 115]) has emerged. In contrast to real-valued feature vectors, binary features are less redundant, often faster to compute and incur less storage overhead at the cost of a lower recognition rate. Binary features are, to the best of our knowledge, seldom queried with BoVW-based approaches [116, 117], but rather by more traditional approximate kNN techniques such as LSH-based approaches [7, 114, 11], but also based on quantization [118, 119].

In the last main part of this thesis, we build upon this current research on keypoint-based object recognition, aiming at increasing the usability of these new approaches in medium-scale to large-scale scenarios, addressing the following research issues:

Efficient Object Recognition on Set-based Image Representations. While advances in feature indexing resulted in a remarkable leap in performance concerning efficient and effective kNN query processing, with the vast amount of features that have to be matched during recognition (up to a few thousand), even very fast kNN indexing techniques that can provide approximate query results in under ten milliseconds (e.g. [10]) on large datasets containing millions of features would yield recognition runtimes of many seconds.

We argue that the use of kNN queries for object recognition in large-scale systems cannot be achieved by developing efficient indexing techniques alone. The problem of efficiency has to be approached from different research directions as well, such as the *number* of kNN queries posed on the system, as reducing the number of kNN queries linearly decreases the runtime of the matching step. In Chapter 17 we aim at addressing this problem. We evaluate an alternative recognition pipeline that ranks features extracted from the query image by assessing their matchability. Then, the most promising fea-

tures in this ranking are matched against the database using traditional kNN queries. However, despite gaining efficiency, the enforced reduction of kNN queries causes a reduction of feature matches, decreasing the quality of the query result. While recall can be increased by increasing k, to increase Mean Average Precision (MAP) we expand matches on the image level: Given a single seed feature match in a candidate image, this match is expanded by comparing its spatially neighboring keypoints. The idea of this additional step is to push load from the matching step (with complexity mostly determined by the underlying index structure) to an additional step that only has to consider the features stored in a single image pair. The resulting enriched set of matches can then be processed equivalently to techniques based on BoVW, e.g. by using query expansion [120] or geometric verification [121].

This work stands in contrast to research in the area of BoVW-based retrieval: Research involving the BoVW pipeline often assumes that the matching step is relatively cheap, especially if approximate cluster assignment techniques such as hierarchical k-means [122] or approximate k-means [121] are used. Therefore such research often focused on increasing Mean Average Precision (MAP) at a large number of query features. In contrast, Chapter 17 aims at maximizing MAP for a small number of processed features. This different optimization criterion is especially of interest as techniques that do not lead to significant gains in performance at a high number of features (where convergence to the maximum possible MAP has already been achieved by other techniques) can lead to a remarkably higher MAP when only a low number of features is queried. To summarize, the contribution of Chapter 17 is to provide a *simple* and *extensible* pipeline for *medium to large-scale* object retrieval based on *kNN queries* with all of the following properties:

- *Reduction of the number of keypoints queried* by a general keypoint ranking scheme in order to reduce matching times. The pipeline is not bound to a specific keypoint selection technique as long as keypoints can be ranked by their estimated quality.

- Acceleration of the pipeline by state-of-the art index structures such as (Locally Optimized) Product quantization [12] or Multi-Index-Hashing [11].

- *Geometric Match Expansion* to relieve the index structure and to increase query *MAP*.

- The use of many nearest neighbors ($k > 2$) to increase the number of seed hypotheses and therefore query *recall*.

- Consideration of *distances* between features during score generation to allow accurate scoring of image features by their similarity.

We further provide a thorough evaluation of this pipeline on a variety of well-known datasets, including Oxford5k, Oxford105k, Paris 6k, and INRIA Holidays, provide insights into advantages and disadvantages of the approach, and show that such match expansion techniques can lead to performance improvements. We also evaluate the effect of k in relation to the number of keypoints queried on the systems performance, and the pipeline's behaviour on different feature descriptors including real-valued (SIFT) and binary (BinBoost) features.

Indexing Binary Features for Object Recognition. As a second contribution we aim at shedding more light on the feature matching step in the context of binary features. During our evaluation in Chapter 17 we employ highly optimized *exact* indexing to find matching candidates of binary features. While these techniques are indeed astonishingly fast, they are still too slow for large scale applications requiring low response times. To address this issue, Chapter 18 provides an evaluation of approximate indexing techniques for binary features in image retrieval. Equivalently to the idea of Pauleve et al. [123] in the case of real-valued features, we reduce the matching step for binary features to the idea of LSH. Given this interpretation of the problem, all of the currently used approaches only differ in the computation of the hashes, which allows high comparability. During our experiments, we evaluate these hash functions under different conditions in the context of kNN queries and range queries, and investigate under which conditions each of these approaches performs best. We see the necessity of this research for the following reasons. First, query processing in binary space should consider its specific properties, such as its thick boundaries [118]. In practice, however, a variety of techniques known from real-valued features is applied to binary features, such as quantization, although only few experimental results, such as [118, 119], of these techniques in binary space are available; we will build upon and extend these results in Chapter 18.

Structure of this Part. This part of the thesis is structured as follows. First, in Chapter 16, we recap keypoint-based object recognition, provide a formal problem definition and an overview over related work. In Chapter 17 we address the problem of reducing the number of keypoints in set-based feature representations, considering a modified object recognition pipeline (Section 17.1) and providing an evaluation of this pipeline (Section 17.2) on different feature types and datasets. Chapter 18 is split into a review of

currently available techniques (Section 18.1) for querying binary features in image databases, and an in-depth analysis of the described techniques, complementing the results from [118, 119] (Section 18.2). Chapter 19 concludes this part.

This research on image retrieval has been previously published in [53, 54]; for an overview over the contributions of the author of this thesis to this work we refer to Chapter 4.

Chapter 16

Preliminaries

Many state-of-the-art solutions in the area of object recognition follow a common pipeline using the filter-refinement paradigm, see Figure 16.1. First, given an input image, keypoints are extracted at different image scales that aim at identifying interesting regions of the image. Then, for each of these keypoints, a *high-dimensional* feature vector is computed that describes the keypoint's surroundings in a scale-, rotation-, and translation-invariant way. Available keypoint descriptors include for example SIFT [3], BinBoost [124], and ORB [114]. For efficient query processing, these features have to be indexed together with meta-information such as the image ID where the feature was detected, and the corresponding keypoint's position, rotation, and scale. Indexing is often achieved with the Bag of Visual Words (BoVW) paradigm [23] which has been derived from text retrieval: With this approach, feature vectors are *quantized* using, for example, k-means. Each mean represents a single visual word and describes a set of visually similar features, as each database feature is mapped to its closest k-means centroid. During query processing, every query feature is assigned to the closest cluster center; database features falling into the same cluster center as the query vote for a given candidate image, filtering irrelevant results and allowing to rank candidates according to the number of features voting for them; usually additional weighting of correspondences is employed to increase te query precision. Finally, during a refinement step, geometric consistency between the matched features is checked, and a re-ranking based on the refinement

$$Image \xrightarrow{Extract} Feature\ Vectors \xrightarrow{Filter} Candidates\ Images \xrightarrow{Refine} Result$$

Figure 16.1: Object Recognition Pipeline

result is performed.

The roughness of binary decisions involved in the initial BoVW approach, either assigning or not assigning a query feature to a match candidate, has proven to be a major restriction, as for example the distance between tentatively corresponding features is not considered during matching. As a result this procedure has been softened more and more, for example with the appearance of soft assignment [110]. Recent approaches such as [10, 111, 112, 11] returned to approximate nearest neighbor queries, as including the closeness between a database feature and a query feature into the ranking can improve query performance. In the following we will formally summarize the previous informal description of keypoint-based object recognition and provide a formal problem definition.

16.1 Problem Definition

Let $D = \{I_0, ..., I_{|D|}\}$ denote a database of images I_j. Images are represented by a list of interest points and their corresponding feature vectors, i.e. $I_j = \{p_j^0, ..., p_j^{|I_j|}\}$ with $p_j^i = (v_j^i, x_j^i, y_j^i, s_j^i, r_j^i, \sigma_j^i)$ for affine-variant interest point descriptors and $p_j^i = (v_j^i, x_j^i, y_j^i, s_j^i, r_j^i, \sigma_j^i, A_j^i)$ for affine-invariant descriptors, with v_j^i a (real-valued or binary) feature vector, (x_j^i, y_j^i) the coordinate of the interest point in the image, s_j^i its scale, r_j^i its rotation, σ_j^i its response, and for affine-invariant descriptors A_j^i the parameters of the ellipse describing its affine shape, see [125].

Given a query image I_q containing an object o, we would like to retrieve all images $I_n \in D$ containing object o. This is usually achieved by a combination of *feature matching* and *scoring* [126]. During feature matching, we retrieve tuples of similar feature vectors $m(p_q^i) = \{(p_q^i, p_{x0}^{j0}), \ldots, (p_q^i, p_{xr}^{jr})\}$, $\{I_{x_0}, ..., I_{x_r}\} \subseteq D$ denoting that feature i of the query is visually similar to the features $\{p_{x0}^{j0}, ..., p_{xr}^{jr}\}$. This matching problem can for example be solved using the BoVW approach. In recent years however, as mentioned previously, there has been a shift away from BoVW towards more accurate, however less efficient kNN queries [10, 111, 112, 11], leading to $m(p_q^i) = \{(p_q^i, p_x^i) | p_x^i \in kNN(p_q^i, D, k)\}$. Now let $M = \cup_{i=0}^{|\Psi|} m(p_q^i)$, with $\Psi = \{p_q^0, \ldots, p_q^{|\Psi|}\} \subseteq I_q$ a subset of the query features. The score of database image $I_x \in D$ is computed as $\sum_{\{(p_q^i, p_y^j) \in M | x=y\}} score(p_q^i, p_y^j)$. The most trivial solution would be to increase the score of image I_x by one for each tentative match, resulting in $\sum_{\{(p_q^i, p_y^j) \in M | x=y\}} 1$. More sophisticated scoring approaches for kNN-based image retrieval can be found e.g. in [113].

Accurate kNN queries are, even after astonishing research efforts in the last years, still relatively expensive. For example, running a 100NN-query on 100 million binary features using Multi-Index Hashing (an index for binary features, see [127]) would take about 100ms, summing up to 100 seconds in a scenario where 1000 features are queried to retrieve a single image[1]. SIFT features are generally queried approximately, runtimes vary significantly with recall and are often between 8ms/query and 53 ms/query for a billion SIFT features at a recall below 0.5 [10].Generally, achieving good recall over 0.5 for 1NN queries with such techniques is very expensive.[2] Based on these observations we argue that in addition to indexing efficiency, other possibilities must be considered to reduce the complexity of the feature matching phase. Generally, to achieve this complexity reduction, different approaches are reasonable:

- Reduce the *dimensionality* of feature vectors. One well-known approach would be to apply PCA to SIFT features and drop the dimensions with least variance. Another more desirable option would be to directly extract lower-dimensional features.
- Reduce the *cost* of distance functions, for example by binarization [128, 129, 130, 131, 132] or by extracting binary features [124, 114] and using the Hamming distance.
- Reduce the *cost for querying*. A variety of (exact) indexing techniques have been proposed, e.g. Multi-index-hashing [11] for binary features.
- Reduce the *accuracy* of a matching query. This has been widely used in the past, e.g. BoVW [23] can be seen as an extreme case.
- Reduce the *number* of kNN queries, e.g. [133, 134, 135, 136].

In Chapter 17 of this thesis we focus on the last approach: Let a database of images, represented by sets of features describing the neighborhood around interest points, be given. Let n denote the upper bound on the number of matching queries, constraining the number of kNN queries. The goal of this research is to develop a retrieval algorithm that returns a list of images ranked by their visual similarity to the query. We aim at modifying the image recognition pipeline such that a given performance measure (in our case MAP) is maximized for a given n.

[1]Note that the number of features extracted from an image is sometimes even larger, see the dataset statistics in our experimental evaluation

[2]We are not aware of recall evaluations of these techniques for $k \neq 1$ although it was shown in [113] that a larger k can notably boost recognition performance.

The problem setting for reducing the number of keypoints is similar to BoVW-based approaches, however in such a context it is usually assumed that $n = |I_q|$. In our work we address the opposite case where $n << |I_q|$.

As an additional contribution, in Chapter 18 we provide an evaluation of indexing techniques for binary features, which is important when integrating binary features in the recognition pipeline above.

16.2 Related Work

This section, addressing related research, follows the organization of the image processing pipeline used in Section 17.1.

16.2.1 Keypoint Reduction

In order to reduce the number of extracted features that have to be matched, [133] aimed at predicting the matchability of features by interpreting the problem as a classification task. The authors have shown that this solution, based on a random forest of decision trees, leads to better matching results than if the same number of high-response features would be considered. Keypoint reduction can also be achieved by employing the Adaptive Non-Maxima suppression (ANMS) from Brown et al. [136]. Their approach aims at finding interest points that are sufficiently distributed across the whole image and is computationally relatively inexpensive. Hajebi and Zhang [134] proposed to keep track of the distribution of scores during query processing and stop the investigation of further features as soon as the score difference between the best-scored image and the average score becomes large enough. Other approaches to rank features are based on visual attention, e.g. [135]. In contrast to us, the authors query all features of higher scale levels to build a coarse-grained (32x32) top-down attention map and combine it with a bottom-up saliency map. Then, in an iterative fashion, the features in the most promising cells of these attention maps are queried. The authors perform some kind of geometric verification, but no match expansion. The approaches from Sattler et al. [137, 138], based on BoVW, consider features in ascending order of the length of inverted lists of the corresponding visual words. While we aim at reducing the number of query features with feature ranking, there exist also approaches aiming at reducing the number of database features [139, 140, 141]. Decreasing both database and query features can be a useful choice in the context of our pipeline, but is beyond the scope of this work.

16.2.2 kNN Indexing

Real-Valued Features. Exact kNN query processing on high-dimensional features often cannot significantly decrease runtimes compared to a linear scan due to the curse of dimensionality. Therefore indexing research in the image community concentrates on *approximate* nearest neighbor search. BoVW-based techniques [23, 122, 142, 121] can also be seen as a simple means of indexing. Some well-known approximate indexing techniques used in image retrieval are forests of randomized kD-trees[143, 144] and the k-means-tree [122, 144]. These techniques however suffer either from high storage complexity if the database descriptors are needed for refinement, or low-quality distance approximations in the case of BoVW-based techniques. Recent research in kNN indexing aims at providing low runtime and storage complexity while providing accurate distance approximations at the same time. One group of these techniques is based on the Product Quantization approach from Jégou et al. [12], a quantization-based approximate indexing technique distantly related to the BoVW paradigm. Recent extensions of this approach include [112, 111, 10]. A recent survey on approximate nearest neighbor queries in general has been provided in [145]. Another survey addressing high-dimensional indexing especially in the context of image retrieval has been published as well [146].

Binary Features. Another group of techniques aiming at efficient query processing is built on the idea of generating distance-preserving binary codes from real-valued features, sometimes referred to as *binarization*. *Binarization* can be seen as some sort of compression on real-valued image features. Binary codes do not only increase storage efficiency but also allow efficient query processing (usually) by using the Hamming distance, leading to fast distance computations. The generation of binary signatures corresponds to the problem of finding a function $f : \mathbb{R}^{d_1} \rightarrow \mathbb{B}^{d_2}$ where $\forall x_1, x_2 \in D : dist_{l_2}(x_1, x_2) \propto dist_H(f(x_1), f(x_2)))$ with $dist_{l_2}(x_1, x_2)$ denoting the Euclidean distance between feature vectors and $dist_H(f(x_1), f(x_2))$ corresponding to the Hamming distance between the transformed points. Recently developed binarization techniques include the approach from [128], Random Maximum Margin Hashing [129], Scalar Quantization [130], Spherical Hashing [131] and k-means hashing [132]. In contrast to binarization techniques, binary keypoint descriptors such as BinBoost and ORB [124, 114] can avoid the indirection of extracting real-valued (e.g. SIFT) features first and then binarizing them. The resulting binary codes can be queried by either retrieving features with equivalent codes from the database if the codes are relatively short (resulting in only a few look-up operations), by employ-

ing approximate LSH-based hashing [7] or using exact indexing [11, 127] and are relatively fast due to them employing the Hamming distance instead of the Euclidean distance. Additionally, Muja and Lowe [119] and Trzcinski et al. [118] proposed the use of a forest of random clustering trees for nearest neighbor search on binary features.

16.2.3 kNN-based Matching

kNN-based matching techniques have a long history in the context of Image retrieval. One of the most famous techniques using such approaches is Lowe's SIFT recognition pipeline [3]. Lowe retrieved, for each query feature, the two nearest neighbors from the database and accepted a feature as match if its distance ratio between 1NN and 2NN was above a given threshold. Jégou et al. [113] evaluated kNN-based matching based on local features, especially SIFT. They proposed a voting scheme optimized for kNN-based retrieval that aims at improving upon other voting schemes, such as binary and rank-based ones. A binary voting scheme would increase the score of a candidate image by 1 if a kNN-match between a query feature and a database feature is found. Rank-based schemes give higher scores to keypoints that have a low rank to the query feature. Their adaptive criterion basically scores matches relative to the distance of the k-th match. Furthermore, the authors analyzed normalization methods for the resulting votes in order to reduce the negative effect of favouring images with many features over those with only a few. Finally the authors investigated the impact of approximate nearest neighbor search on the quality of the results. They did however not consider reducing the number of query features. Qin et al. [147] proposed a normalization scheme for SIFT-features that locally reweighs their Euclidean distance, optimizing the separability of matching and non-matching features. Based on this normalization, the authors developed a new similarity function and scoring scheme based on thresholding rather than kNN query processing.

16.2.4 Match Expansion

As our technique aims at reducing the number of kNN queries during the matching step, the generation of a sufficient number of match hypotheses has to be achieved in a different fashion. We do so by applying a flood-filling approach using kNN matches as seed points. Match expansion has received quite some attention in the computer vision community [148, 149, 150, 23, 151, 152, 153, 139], and will most likely become more relevant again with the use of kNN-based matching techniques. One of the first techniques in this area of research has been proposed by Schmid and Mohr

[148]. They used the spatial neighbors of match candidates to increase the distinctiveness of features. A feature pair is only considered as a match if its feature-space distance is small enough *and* there exists a given fraction of features in the spatial neighborhood of these features whose feature-distance is small enough as well. They also considered the consistency of gradient angles between these features to reject false-positive matches, however they did not consider the combination of their approach with feature reduction. Sivic and Zisserman adapted the technique for Video Google [23]. We however do not address the problem of BoVW-based retrieval. Furthermore we do reject matches based on this technique but rather increase the score of a given image by considering neighboring features. Our work is also inspired by [150], where the authors used a region-growing approach for establishing correspondences in the context of multi-view matching. After establishing a set of initial matches in a traditional index-supported manner, an affine transformation is estimated that guides search of additional matches in a local neighborhood of the seed match. The authors, however, did not use this technique for reducing the number of queries in the matching step, but rather to increase the result quality. Ferrari et al. [151] developed another related technique in order to achieve high invariance to perspective distortion and non-rigid transformation; it further allowed to perform an accurate segmentation of objects during recognition. Their approach builds a dense grid of features over the image; in contrast we use the initially provided keypoints and descriptors that are stored in the database nonetheless, reducing computational overhead. A recent work related to this approach includes [153]. Guo and Cao [152] proposed to use Delaunay triangulation to improve geometric verification. Wu et al. [154] proposed to enrich visual words by their surrounding visual words, generating scores not only by the weight of a visual word, but also the neighboring features; the authors however did not consider keypoint reduction. Geometric min-Hashing [155], based on the BoVW-paradigm, considers neighboring features as well, however in the context of hashing: First, a reference feature is selected based on its min-hash c_1. Then from its spatial neighbors with similar scale, a description of the central feature's surroundings is generated based on min-hash, resulting in min-hashes $s_1, \ldots, s_n$. The tuple of min-hashes $(c_1, s_1, \ldots, s_n)$ is then used to query similar hashes in the database, compared to BoVW with the advantage that not only a single feature is considered when retrieving similar documents, but also its surroundings. The approach aims at increasing precision at the cost of recall, by dropping features that do not share a similar neighborhood. However, if we reduce the number of matching queries, one of the main concerns is recall, such that our approach aims at increasing MAP without negatively affecting recall. The authors of [137] combined keypoint

reduction and concepts similar to our match expansion, however in the context of 2D-to-3D matching and pose estimation, employing Lowe's SIFT ratio test without kNN-based scoring.

To summarize, while there exists a variety of techniques on feature reduction, match expansion, kNN query processing and distance-based scoring, to the best of our knowledge there does not exist a technique combining these techniques in the way proposed in the introduction.

Chapter 17

Minimizing the Number of Matching Queries for Object Retrieval

17.1 Pipeline

The general retrieval pipeline from this work follows the one used in the past for BoVW-based image retrieval, but in order to incorporate kNN queries and reduce the number of query features we applied some changes. In this section, we first provide a theoretic overview over the pipeline. Then, as implementing the pipeline in such a naive way would lead to unacceptable overhead in terms of memory and computational resources, we provide practical considerations about its implementation in a real-world setup.

17.1.1 Theory

We split our pipeline into the stages of feature detection and extraction, feature ranking, feature matching, match expansion, scoring, and re-ranking. The pipeline was designed with extensibility in mind such that each stage, e.g. keypoint reduction and match expansion, can be easily exchanged by different techniques.

1) Feature Extraction. During feature extraction, given the query image, we extract the set I_q of keypoints and descriptors. Possible features include floating point features such as SIFT [3] or binary features such as BinBoost and ORB [124, 114]. The cardinality of I_q depends on the used feature extractors and can range up to several thousand features.

2) Feature Ranking. The next stage, feature ranking, is based on the idea that some features in an image contain more information than others. For example, vegetation usually provides less information about a specific object contained in the image than the features of the object itself. We aim at ordering the extracted features by a given quality measure, as we would like to query the most promising features first, i.e. the features with the highest chance of providing good match hypotheses. There exist several techniques for feature ranking, and we will fall back to these instead of developing a new approach. The only criterion such a technique needs to fulfill in order to be integrated into the recognition pipeline is that it returns a quality score for each query feature. A simple baseline is a random ranking. Features can also be ranked by their response or size. More sophisticated techniques include Adaptive Non-Maximal Suppression [136] and the use of decision trees involving additional training [133], which has however neither been adapted to binary features nor to kNN-based matching, yet. The result of this feature ranking step is a feature list, ordered such that the most promising features appear first.

3) Feature Matching. The next step, feature matching, aims at finding match hypotheses for the highest ranked features found during the last step. For each of the first n features in the ranking, a kNN query is posed on the database. The selection of the parameter k of the kNN query is important for maximizing the quality of the query result [113]. On the one hand side, a large k decreases the quality of the query result, as this introduces a high number of erroneous correspondences which have to be filtered out during a verification step later in the pipeline. On the other hand, a small k also reduces the retrieval quality as many high-quality hypotheses are left unconsidered. Basically, k can be seen as a way to tweak *recall* at a given number of query features, as the number of images returned by the query is at most $n * k$. As a result, especially if a very small number of kNN queries is used for correspondence generation, it is possible that an even larger k increases effectiveness, as it allows for finding more initial correspondences (however of lower quality). We refer to Section 17.2 for an experimental analysis of this problem. The feature matching stage provides a list of tentative matches (tuples) (p_q^i, p_x^j).

4) Match expansion. The match expansion phase is tightly interleaved with the match generation phase. In our scenario where we want to pose a very small number of kNN queries on the system, we face the problem that even if we find some correspondences between the query and a database

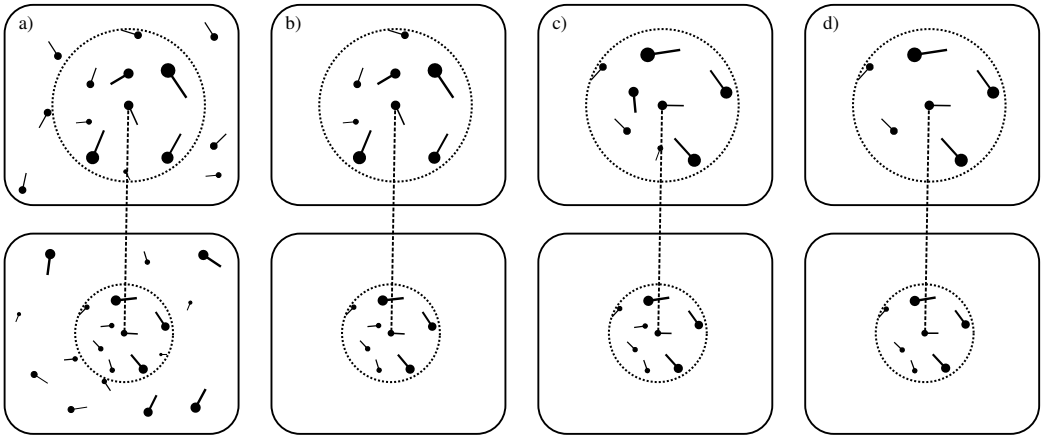


Figure 17.1: Generation of additional match hypotheses.

image, their number will be relatively low, increasing the probability that a good match is outranked by an image containing common random matches only. To resolve this problem, we shift the load of correspondence generation from the matching stage –that employs kNN queries– to an intermediate stage that avoids such queries. Match expansion aims at reducing the runtimes of generating additional matches, which usually depend on the underlying index structure, to runtimes depending on the features stored in a single image pair. When employing exhaustive search with product quantization for indexing, match expansion therefore avoids additional linear scans over the feature database; as non-exhaustive variants of product quantization only consider a fraction of features in the database, the gain of match expansion in this case depends on the desired recall of the index structure.

It is however important to realize that, while such a match expansion can find additional hypotheses for candidate images, i.e. increase MAP, it cannot retrieve any new candidates, i.e. increase recall. This stage therefore aims at compensating for the loss in MAP due to querying less features.

Match expansion exploits the keypoint information of the seed matches that provide scale, rotation, and possibly affine information. These properties can be used to identify spatially close keypoints, adapting the ideas of [150, 151, 155, 154]; we will use a modified version of [148] for expanding matches. Given that a match hypothesis is correct, not only the corresponding feature pair should match, but also its spatial neighborhood, as an object is usually not only described by a single but rather by multiple keypoints. The similarity of a match's neighborhood is evaluated using the procedure visualized in Figure 17.1. The figure shows an initial seed match, i.e. a kNN of a query feature, and keypoints surrounding the seed match. The scale of

each keypoint is represented by the keypoint's size, and the gradient direction is represented by a line anchored in the keypoint's center. The top row of this figure visualizes the features of the query image, while the bottom row visualizes the image features of a tentative match.

Starting point is an initial correspondence pair (p_q^i, p_D^j) established by kNN-search in feature space, see Figure 17.1 a). In a first step, features in a given spatial range are retrieved in the image I_q for p_q^i and in Image I_D for p_D^j, see Figure 17.1 b); the spatial range is visualized by a dotted circle. Given the constant δ_{xy}, the spatial range is given by $s_q^i \delta_{xy}$ for the query feature and $s_D^j \delta_{xy}$ for the matching database feature, achieving scale invariance. Spatially close keypoints with a significantly different scale (determined by the scale ratio threshold δ_s) than their reference feature are discarded (see the small features in the figure) similar to [155], resulting in two sets of features P_q and P_D. These remaining features are rotation-normalized using the reference keypoint's gradient orientation information r_q^i and r_D^i, rotating the set of keypoints and their corresponding gradient orientations, see Figure 17.1 c). Then the two lists of keypoints are traversed in parallel. If the rotation-normalized angle α to the reference feature, the rotation-normalized gradient angle r, and the feature-space distance of two features d_v are within a predefined threshold (δ_α, δ_r, and δ_{d_v} respectively) and the ratio of their scale-normalized spatial distance is within given bounds $\delta_{d_{xy}}$, the corresponding features are accepted as a matching pair (see Figure 17.1 d)). The remaining features are discarded. Note that, while the complexity of this step is $|P_q| * |P_D|$ in the worst case, it can be reduced by an efficient sweep-line implementation that sorts features by their angle α and traverses both lists in parallel.

This technique of finding neighboring keypoints assumes that two images are only distorted by similarity transforms. To mitigate the effects of non-similarity or even (small) non-rigid distortions, a recursive procedure (in our case with a maximum recursion depth of 2) can be chosen that performs the same procedure on each of the resulting pairs. Moreover, by choosing the Mahalanobis distance using the affinity matrices of the seed pair (A_q^i and A_D^i respectively) instead of Euclidean distances for finding spatially neighboring keypoints, the process can be extended to affine-invariant features. This technique returns features within an elliptical region around the seed points, reducing performance loss from affine distortions.

Result of the expansion phase is an extended list of match hypotheses.

5) Scoring. Scoring is again tightly interleaved with match generation. In this phase, based on the expanded list of matches, a score is computed for

every database image. In the simplest case, each hypothesis pair votes with a score of one for a given database image. This however, has shown to have a relatively low performance [113], as for example images containing many features would have higher scores than images containing only a few features. For this purpose, more sophisticated scoring techniques have been developed. We will adapt some of the techniques from [113], weighting scores based on the distance of the candidate feature to the query feature and the number of features in the image. For each matched feature from image I_x its score is increased by $\frac{\sqrt{d_{kNN}-d_{ref}}}{\sqrt{|I_q|}\sqrt{|I_x|}}$ with d_{kNN} the kNN distance of the seed feature, and d_{ref} the distance between the seed feature and its tentative match in the candidate image, i.e. features generated during match expansion are assigned the same score as their seed match. This score is similar to the scores from [113], however we have added additional square root weighting which further increased effectiveness of these scores. For scoring we implemented a simple burst removal [156] scheme after match expansion that allows only for one correspondence per query feature.

6) Re-Ranking. After building match hypotheses and scoring, the ranked list can be processed equivalently to BoVW-based approaches. Further steps can include geometric verification or query expansion techniques [120]. As these techniques are complementary to the remaining pipeline we will not further consider them in this chapter.

17.1.2 Practical Considerations

To enable efficient query processing using the pipeline summarized previously, three conditions must be fulfilled. First, it must be possible to efficiently retrieve the kNN features of a query feature and their corresponding keypoints from the database. Second, to enable match expansion, it must be possible to compute, given two keypoints, the distance of their corresponding feature vectors. Third, also concerning match expansion, it must be possible to pose a range query on all keypoints from a given image, retrieving spatially close keypoints. In the most basic case, the image database used for query processing can be seen of a list of tuples $(p_0^0, \ldots, p_0^{|I_0|}, \ldots, p_i^0, \ldots, p_i^{|I_i|}, \ldots)$ containing feature and keypoint information. The features in the list are ordered by their corresponding image to allow efficient match expansion. However, in order to enable usability of this approach in a practical setup, special care has to be taken concerning computational and memory efficiency and the thorough selection of parameters; we will address solutions for these challenges in the following section. Computational efficiency can be achieved using index-

ing techniques such as Product Quantization or Multi-Index Hashing, while the memory footprint of the image database can be reduced by compressing the feature vectors used during match expansion. Finally, the selection of parameters can be achieved using appropriate optimization techniques.

17.1.2.1 Indexing

In order to improve the performance of the pipeline in real-world applications, fast (approximate) indexing techniques optimized for high-dimensional data [10, 111, 112, 11, 12] can be employed. In this research we focused on (Locally Optimized) Product Quantization for real-valued features and Multi-Index Hashing for binary features; we will summarize these techniques in the following paragraphs for the sake of completeness.

Product Quantization. Approximate nearest neighbor search based on Product Quantization, initially proposed by Jégou et al. [12] and further optimized e.g. in [10, 111, 112], is an elegant solution for indexing high-dimensional real-valued features. During a training phase, features in the database are clustered using k-means and the database features are assigned to their closest cluster mean, partitioning the set of vectors into distinct cells, similar to Locality-Sensitive Hashing [7]. Then, for each feature vector, the residual to its corresponding cluster mean is computed and the resulting residuals are product quantized. Product quantization is achieved by splitting a vector into a small number of subvectors (e.g. 8) and quantizing each of these subvectors separately using a relatively small codebook of e.g. 256 centroids. Instead of storing the residuals themselves, only the cluster id of the closest residual is stored in the index for each subvector, resulting in a reduction in memory complexity. With product quantization using 8 subvectors of 256 cluster centers, a SIFT vector could be compressed from 128 bytes to 8 bytes, resulting in a compression of nearly 95%. The index itself consists mostly of a list of outer clusters and for each of these clusters an inverted list storing, for each feature assigned to this cluster mean, its list of quantized subvectors.

During query evaluation, the query is first assigned to the closest outer cluster mean (or possibly the closest c means in the case of multi-assignment). Then the inverted lists of these means are scanned, and a distance approximation is computed for each of the database vectors stored in this list: As vectors are represented as a list of their closest subvector-centroids, a distance approximation can be generated by summing the squared distances of the corresponding centroids which can be sped up with the use of look-up tables. The resulting distance approximations are then used to rank the feature

vectors. In the past, a variety of improvements of this approach have been proposed, for example the Inverted Multi-Index [112], Optimized Product Quantization [111], and Locally Optimized Product Quantization (LOPQ) [10]. For our experiments we will use the most recent of these approaches, namely LOPQ.

Multi-Index-Hashing. While Product Quantization has been developed to support efficient query processing on real-valued and high-dimensional feature vectors such as SIFT, Multi-Index Hashing (MIH) [11] has been specifically designed for binary features, such as ORB or BinBoost[124, 114]. It is based on the idea of Locality-Sensitive Hashing [7], however in contrast to this approach it aims at exact query processing. The idea behind MIH is, similar to Product Quantization, to split a binary vector into a set of subvectors. Each of these subvectors is indexed in a dedicated hash table with the subvectors' binary value directly representing the id of its hash cell: A single cell of the index contains all database vectors that contain a given subvector.

During query processing, the query is split into subvectors as well. These subvectors provide the hash cells that have to be looked up in order to find vectors with similar values. Bit-flipping the query subvectors and retrieving the corresponding hash cells allows retrieving features with similar, but not equivalent subvectors. To allow exact kNN processing, Norouzi et al. developed a retrieval strategy that enumerates all relevant bit-flip operations to retrieve an exact query result. In our experimental evaluation, we will use this index structure in combination with BinBoost[124] features to evaluate the pipeline from Section 17.1.1 on binary features.

17.1.2.2 Match Expansion

Concerning the expansion of initial matches we face two challenges. First, we have to find the best parameters for the expansion step. Second, memory consumption has to be minimized in order to store features in main memory and hence speed up query processing.

Parameter Selection. Unfortunately it is a tedious task to determine the thresholds of the flood-filling procedure for match expansion, namely δ_{d_v}, δ_α, δ_r, and $\delta_{d_{xy}}$, by hand. This problem can be solved by utilizing Nelder-Mead Simplex-Downhill optimization: After selecting the distance multiplier δ_{xy} and the maximum scale change ratio δ_s by considering runtime constraints, the remaining thresholds are automatically determined by the Simplex-Downhill approach. Optimization of these parameters should be

conducted on a training dataset different from the test set in order to avoid overfitting.

Vector Compression. For compressing real-valued feature vectors, we consider Product Quantization as well. In contrast to Product Quantization based indexing based on LOPQ, however, we do not product quantize residual vectors, but rather the vectors themselves, as otherwise vectors belonging to different cells in the outer quantizer could not be compared efficiently. For compression, we split each feature vector in a set of $m = 8$ subquantizers and for each of these subquantizers build a codebook of $s = 256$ centroids. The distance between feature vectors can then easily be approximated as the sum of squared distances between the closest subquantizer centroids followed by a square root operation. As distances between cluster centroids can be stored in a lookup table of size $m * s * s$, distance computations reduce to m table look-ups and a single square root operation.

17.2 Experiments

17.2.1 Experimental Setup

Datasets. We evaluated the modified recognition pipeline on four datasets. The *Oxford5k* (O5k) building dataset [121] consists of 5063 images of common tourist landmarks in Oxford. The authors of the benchmark also provide a set of 55 queries including rectangular query regions and ground truth files listing, for each query, the images that contain at least parts of the query. Ground truth files are split into three categories: good, ok and junk. Good and ok files are considered for computing the Mean Average Precision (MAP) of the query. Junk images are neither scored as true hit nor as false hit and simply discarded for computing the MAP. We also included *Oxford105k* (O105k) in our evaluation which consists of the Oxford5k dataset in combination with about 100k distractor images [121] that do not contain images related to the query. The *Paris6k* (P6k) dataset [110], conceptually similar to the Oxford dataset, consists of 6412 images of common landmarks in Paris, and has the same structure as the Oxford dataset. As a third dataset we used the *INRIA Holidays* (Hol) dataset [126] which consists of 1491 images including 500 queries and their corresponding ground truth. In contrast to the Oxford and Paris dataset, Holidays contains more natural scenes and a lower number of result images for each query. Images of the Holidays dataset were scaled down to a maximum side length of 1024 before feature extraction.

Feature Extraction and Indexing. We used two different feature ex-

Table 17.1: Database Statistics

Dataset	Extractor	Features	⌀
O5k	BinBoost	10,640,081	2101.5
O105k	BinBoost	195,068,373	1855.4
O5k	SIFT (Hess.-Aff.)	13,516,675	2669.7
O105k	SIFT (Hess.-Aff.)	253,761,866	2413.7
P6k	SIFT (Hess.-Aff.)	16,073,531	2506.8
Hol	SIFT (Hess.-Aff.)	4,596,567	3082.9

traction techniques: a rotation-variant version of SIFT using affine invariant keypoints[1] made available by the authors of [125] and, as an instance of state-of-the-art binary descriptors, the BinBoost descriptor which is also publicly available [124]. We decided to include binary features in our evaluation as we see them as another mean of decreasing query complexity, however we will concentrate on SIFT features in our evaluation.

Concerning *Hessian-affine SIFT*, scale was separated from the affinity matrices according to [125], however for expanding matches we used the square root of this scale which roughly corresponds to the radius of the image patch used for SIFT extraction. The parameters of the feature extraction stage have been left at the default parameters. SIFT features are 128-dimensional real-valued vectors. These vectors were square-root weighted similar to RootSift[157], however without l_1 normalization. The weighted features were then indexed using LOPQ in combination with a multi-index [10]. We use a vocabulary of size $V = 2 * 1024$ for the inverted lists, and 8 subquantizers for vector quantization, each subquantizer with a vocabulary of size of 256 clusters. The corresponding source code has been kindly provided by the authors[2]. To compress the feature vectors for the expansion phase, we again used 8 subquantizers consisting of 256 clusters, reducing storage overhead of feature vectors to 6.25% of their uncompressed memory footprint. Codebooks for the Oxford and Holidays datasets were trained on Paris6k, and for Paris6k code books where trained on Oxford5k. During query processing, we applied a simple means of burst removal [156], scoring each query feature once even if it had more than one match.

BinBoost descriptors, i.e. 256-dimensional binary vectors, can be queried rather efficiently using exact indexing techniques optimized for binary feature vectors, e.g. [11]. We have used a publicly available implementation of their index during our experimental evaluation. We applied burst removal when querying these features as well.

[1] https://github.com/perdoch/hesaff/

[2] http://image.ntua.gr/iva/research/lopq/

Table 17.2: Parameters for Match Expansion

Extractor	Train	δ_{xy}	δ_s	δ_{d_v}	δ_α	δ_r	$\delta_{d_{xy}}$
SIFT	P6k	6	0.8	26.2	24.3	–	0.49
SIFT	O5k	6	0.8	26.9	18.9	–	0.56
BinBoost	P6k	4	0.8	73	21.1	26.0	0.46

An overview of the extracted features can be found in Table 17.1. Note that the number of query features was different to the number of database features on Oxford5k, Oxford105k and Paris6k due to the bounding boxes provided by the dataset authors, and for several queries the number of query features was less than 1000. The average number of features available over all queries was 1371.4 (BinBoost, $\sigma = 612.3$) and 1452.8 (SIFT Hessian-Affine, $\sigma = 950.2$) for queries on Oxford.

The code was written in C++ using OpenCV. Runtime experiments were conducted on an off-the-shelf Linux Machine with i7-3770@3.40GHz CPU and 32GB of main memory without parallelization. During our experimental evaluation we concentrate on analyzing the effectiveness of the approaches in terms of Mean Average Precision (MAP); we also provide numbers on the performance of the evaluated approaches concerning the runtime of the scoring, querying and ranking stages.

Parameters. The parameters for query processing were set as follows. First, range multiplier δ_{xy}, maximum scale change δ_s, k, and n were set by hand with computational efficiency in mind, as a lower number of features considered during expansion reduces the cost of this step. Given these manually set parameters, the remaining parameters of the expansion phase, i.e. feature distance threshold δ_{d_v}, angular threshold δ_α, gradient angle threshold δ_r and spatial distance ratio $\delta_{d_{xy}}$ were set to the outcome of a Nelder-Mead Downhill-Simplex optimization maximizing MAP; initialization was performed with reasonable seed values. Minimization was done on the Paris6k dataset (with LOPQ and quantization code books trained on Paris6k as well) for the Oxford5k, Oxford105k and Holidays datasets. For the Paris6k dataset, we optimized these parameters on the Oxford5k dataset. The parameters were selected for each of the descriptor types (SIFT and BinBoost) using ANMS ranking at $k = 100$, number of keypoints $n = 10$, recursively descending into every expanded match. The resulting parameters were reused for the remaining ranking approaches, different k, n and the non-recursive approach. An overview over the selected parameters is shown in Table 17.2.

We varied each of the optimized parameters by $\pm 10\%$ separately on Oxford 5k (ANMS ranking with match expansion) to get insights into their

Table 17.3: SIFT, Oxford5k, k=100

↓ Appr. → n	50	100	500	1000
RND	.616	.698	.810	.827
RESP	.557	.640	.787	.822
ANMS	.676	.727	.825	.836
RND+ME	.679	.749	.829	.838
ANMS+ME	.741	.780	.843	.844
RND+MER	.686	.752	.826	.832
ANMS+MER	.752	.786	.837	.838

Table 17.4: SIFT, Paris6k, k=100

↓ Appr. → n	50	100	500	1000
RND	.566	.652	.770	.786
RESP	.519	.594	.743	.775
ANMS	.578	.668	.783	.794
RND+ME	.629	.699	.781	.789
ANMS+ME	.648	.723	.793	.796

effect on MAP. The maximum deviation resulted from decreasing the feature distance threshold, which lead to a decrease in MAP of -0.012, indicating that while there is an impact of the optimized parameters on the performance of match expansion, there is still a range of relatively "good" parameters.

17.2.2 Experiments

We evaluated the algorithm's performance by varying k and n as these parameters affect the number of initial seed points that are expanded later. As a baseline for our experiments we implemented a scoring scheme based on [113] that considers the distances between features and the number of features in the image for score computation.

Keypoint Ranking. In our first experiment (see Table 17.3 and Table 17.6) we wanted to evaluate the performance difference in MAP when querying a low number of features (i.e. 50, 100, 500 and 1000 keypoints) with different keypoint ranking techniques, providing a baseline for further experiments. The simplest ranking (RND) takes random features from the extracted keypoints; we averaged this approach over 5 runs to get accurate results. Furthermore we evaluated a ranking based on keypoint responses (RESP), and a more sophisticated approach called Adaptive Non-Maximal

Table 17.5: SIFT, Holidays, k=10

↓ Appr. → n	50	100	500	1000
RND	.600	.662	.765	.792
RESP	.571	.630	.735	.770
ANMS	.642	.696	.779	.803
RND+ME	.646	.702	.764	.770
ANMS+ME	.699	.734	.780	.781

Table 17.6: BinBoost, Oxford5k, k=100

↓ Appr. → n	50	100	500	1000
RND	.390	.462	.586	.616
RESP	.389	.461	.600	.625
ANMS	.461	.508	.614	.620
RND+ME	.469	.529	.625	.638
ANMS+ME	.542	.588	.648	.644
RND+MER	.481	.539	.626	.634
ANMS+MER	.551	.591	.648	.640

Suppression [136] (ANMS) that aims at distributing keypoints relatively uniformly over the image. As expected, considering only few keypoints significantly reduces the MAP of all approaches. The MAP of the response-based ranking is worse or similar to the random baseline: for SIFT, the response decreases performance compared to the random approach, while for BinBoost (that is based on SURF Keypoints) results are sometimes slightly better than the random baseline. The ANMS ranking increases the MAP for all approaches. Note that the gain resulting from using ANMS is rather astonishing for the Oxford5k dataset; we can easily gain 0.03 (n=100) to 0.06 (n=50) points in MAP without significant computational overhead if the number of features queried is relatively low. Similar observations hold for Holidays (Table 17.5) but considering Paris6k (Table 17.4), the gain resulting from using ANMS instead of a random ranking is lower. Our results with BinBoost (Table 17.6) on Oxford5k indicate that ANMS without match expansion can increase performance by over 0.07 points in MAP (n=50), however its performance is generally lower than SIFT, even if SIFT vectors are quantized as in our case; the memory overhead (8 bytes) for quantized SIFT vectors is actually lower than for BinBoost (32 bytes) features.

Match expansion. Our second experiment aims at evaluating the gain in MAP that can be achieved for a low number of kNN queries when addi-

Table 17.7: SIFT, Oxford 105k, k=100

↓ Appr. → n	50	100	500	1000
ANMS	.489	.554	.710	.748
ANMS+ME	.584	.630	.753	.775

tional hypotheses are generated by match expansion (ME) and the same approach in its recursive version (MER). Affine-invariant SIFT (ANMS+ME, n=50) achieves about 90% of the random baseline (RND, n=1000) at 50 keypoints on Oxford5k, where the baseline only achieves 75%. At the same time the results at 1000 keypoints are similar for all approaches, showing that match expansion does not considerably affect MAP if a high number of keypoints is queried. This substantiates our statement made in the introduction: if a small number of features is queried, techniques that do not achieve significant performance gain for a high number of features can achieve considerable gain in performance. Results are similar for Holidays (88% for ANMS+ME@n = 50 vs. 76% for the random Baseline) while for Paris6k the gain of match expansion is lower (82% vs 72% for the random baseline). Further note that MAP for ANMS+ME decreases slower with decreasing n than without expansion (-0.003 (ANMS+ME) vs. -0.011 (ANMS) for $n : 1000 \rightarrow 500$ on Paris6k). Results for Oxford105k are shown in Table 17.7.

Using BinBoost (Table 17.6 and Table 17.8) the results are similar. The combination of ANMS ranking and match expansion at 100 keypoints performs similar to the random baseline at 500 keypoints, which is especially interesting as the exact indexing techniques used for BinBoost already lead to a relatively high runtime.

While there is some gain for recursively descending (MER) into matches, this additional step does not significantly improve the performance with both SIFT and BinBoost, while being computationally much more expensive. Therefore we will concentrate on ME in the following. ANMS+ME on Oxford using SIFT accepted about 16500 matches per query image ($n = 100$, $k = 100$), in contrast to the approximately 8800 tentative correspondences (less than $n * k$ due to burst removal) that have been generated using kNN matching alone (ANMS).

Note that we also evaluated the effect of a re-ranking stage, weak geometric consistency (WGC) [126] with 32 angular and 16 scale bins, on match expansion using BinBoost. Both pipelines, with and without match expansion were positively affected by WGC, gaining about 0.04 (ANMS and ANMS+ME) points in MAP at $n = 1000$, indicating that WGC is complementary to match expansion. We further observed that WGC did not have a large effect with a low number of features (0.013 with ANMS and 0.006 with ANMS+ME at $n = 100$) involved both with and without match expansion.

Table 17.8: BinBoost, Oxford 105k, k=100

↓ Appr. → n	50	100	500	1000
ANMS	.369	.412	.527	.558
ANMS+ME	.445	.477	.576	.590

Value of k. Not only the number of queried keypoints can be used to increase the number of seed hypotheses and therefore the matching quality, but also k. While increasing k comes at a lower query cost, it also produces hypotheses of lower quality. However, as it is well known for example in the context of kNN classification, reasonable values for k can increase the query performance. In this experiment we evaluate the effect of k if the number of features to be queried is fixed to a given number. For a large number of keypoints, a very high value of k adds a lot of false positives such that the MAP decreases [113]. On the other hand, if k is too small, only a small amount of correct hypotheses is found [113]. We reproduced this result with the ANMS ranker at 1000 keypoints (see Figure 17.2), as here the MAP at $k = 100$ is highest compared to $k = 10$ or $k = 1000$.

What happens when we decrease the number of keypoints? As shown in Figure 17.2, if a large number of keypoints is queried ($n = 1000$), then for all of the evaluated approaches a value of $k = 100$ performed better than $k = 1000$. So match expansion does not greatly affect the optimal value of k in this case. However, if only very few keypoints are used for query processing (e.g. $n = 10$), a large k performed better with match expansion. Without this additional step, performance decreased for large k (however at a larger k than at a higher number of keypoints queried), most likely because the additional noise introduced could not be out-weighted by the higher number of correct matches. This leads us to the following results: The best way to increase query performance, which is well known, is to increase the number of keypoints queried. In order to increase query performance however, it is possible to decrease the number of keypoints queried. In this case, some of the performance loss resulting from a lower number of keypoints can be compensated by a large k in combination with match expansion (and, at a lower degree, even without expanding matches).

Runtime. The cost of the evaluated *keypoint ranking* approaches is negligible for the random and response based ones, as these just have to sort the query features, and about 7ms (SIFT) and 5ms (BinBoost) for the ANMS ranker. For Hessian-affine SIFT on Oxford5k, scoring times (including ME) were about 6ms for processing all k results of a single kNN query ($k = 100$, $n = 100$), and therefore slightly lower than the runtimes of running a single

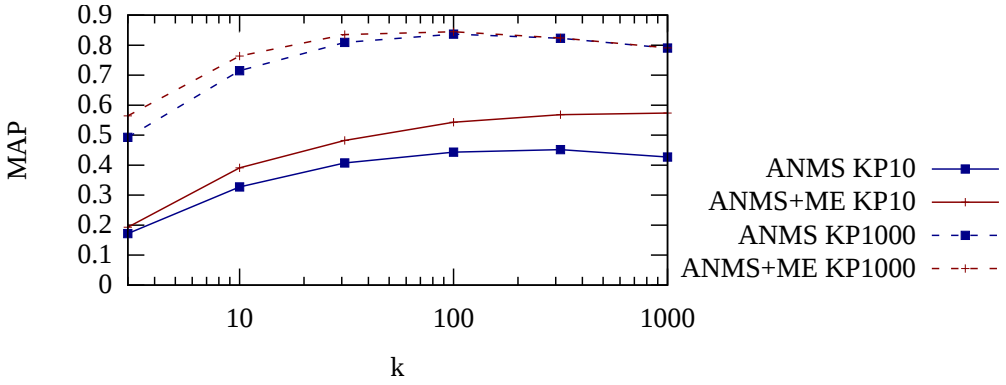


Figure 17.2: Performance for varying k (Hessian-affine SIFT). Straight lines show the performance for 10 keypoints, dashed lines for 1000 keypoints. Equivalent approaches have equivalent colors.

kNN query which took about 7ms, at the possible gain of adding additional matches and a rough geometric check. The feature quantization needed for match expansion took about 45ms for all features in a query image. For BinBoost features (including ME), the match expansion and scoring took less than 4ms for processing a single kNN result. Runtimes increase with k, as more correspondences have to be expanded. The overall runtime for Hessian-Affine SIFT at 100 keypoints (ANMS+ME) was about 1.34s ($k = 100$, $n = 100$), while for binary features it was higher (15s), as for this we used an exact, though state-of-the-art, indexing technique.

Setting runtimes in relation to MAP, it is possible to beat an RND ranker considering 100 keypoints with ANMS+ME considering 50 keypoints at a slightly lower runtime 0.69s vs 0.75s and a higher MAP (see Table 17.3, 0.741 vs. 0.698). For the holidays dataset runtimes of the random approach (RND) were about 0.9s ($k = 10$, $n = 100$) and for ANMS+ME it was only approximately 0.6s ($k = 10$, $n = 50$) at a higher MAP. The most time-consuming operation during match expansion is the search of spatially close features. Therefore we think that the runtimes of match expansion can be reduced significantly by optimizing this matching step, e.g. by ordering features in a kd-tree which can be realized without additional space overhead. This would also help on the Paris6k dataset where runtimes of the random approach (RND) were about 0.8s ($k = 100$, $n = 100$) and for ANMS+ME approximately 0.7s ($k = 100$, $n = 50$) at similar MAP. Runtimes have been measured using only a single core. As each keypoint is queried separately and match expansion is also achieved on a per-keypoint basis, query processing can be easily extended to a multi-core setting.

For Oxford105k the runtime for match expansion was similar to Oxford5k:

A single kNN query took slightly less than 8ms and expansion took about 7ms.

Comparison to the State of the Art. While the primary goal of this research is not to increase the effectiveness of object recognition but rather to reduce the number of features queried, let us still compare the results from this paper to the state of the art in order to get insights into its performance. We will compare to [147], as the authors were using the same Hessian-Affine SIFT features as we do and a similar recognition pipeline involving Product Quantization. On the Oxford5k dataset the authors of [147] achieved 0.78 points in MAP using 8 subquantizers for indexing features using Product Quantization, i.e. a setting close to our scenario. This corresponds to the performance we could achieve when querying 100 keypoints. However, using the pipeline from this chapter requires a larger memory footprint; if we consider techniques with a memory footprint closer to ours, [147] was able to achieve 0.83 points in MAP by approximating features more accurately using 32 bytes per feature. On Paris6k, using 8 subquantizers, [147] achieved a MAP of 0.74, which is slightly better than the performance of ANMS+ME at 100 keypoints (at a higher memory footprint, performance of [147] was 0.76). Concerning Oxford105k, a larger number of features (about 300) is needed to achieve performance comparable to the state of the art (0.728 [147]). Finally note that the performance of our baseline is lower for Holidays than the state of the art performance of 0.80 (0.84 respectively) from [147]; this might be due to a different quantization training set or related to their similarity measure, which is however complementary to our approach and can be easily integrated into our pipeline.

Chapter 18

Retrieval of Binary Features in Image Databases: A Study

As we have already seen in Section 16.2, binary signatures are either derived by binarization of real-valued features or extracted directly from the image. They are usually queried using the Hamming Distance. Depending on the matching task and required accuracy, it is possible to use exact matching approaches (e.g. [11] and [158]), approximate solutions based on LSH [7], or approximate solutions based on quantization [119, 118]. During the last chapter we have queried binary features using exact approaches, more precisely Multi-Index Hashing [11], to maximize query accuracy. Unfortunately, given a large database, search times can increase up to tenths of a second. Given that a single image can contain hundreds of features, the matching times become several seconds, too slow for real-world applications. In the following chapter we will provide an overview over approximate query techniques and an evaluation of such approaches.

Many, if not all, of the approaches for querying binary features can be reduced to the idea of LSH. Such a mapping of indexing techniques to LSH has already been conducted in the case of real-valued features [123]; we will follow this path in this chapter, as it allows to compare different techniques in a general framework, increasing comparability. Additionally according to [118] the binary space is different to the real-valued space in its behaviour, making a specialized evaluation necessary.

LSH [7] and its derivatives have been successfully used for object recognition [114] based on binary features. The approach is furthermore often used as a baseline for other techniques [119, 118]. Note that, given that the extracted binary signatures are very short (e.g. 24 bits), they could be used as hash keys without processing, and queried directly without employing LSH. However currently, binary vectors have a length of several hundred bits (e.g.

ORB: 256 bits [114], FREAK: 512 bits [115]), and therefore an intermediate LSH-based hashing step has to be employed.

18.1 Querying Binary Features with LSH

Locality sensitive hashing has been initially proposed by Indyk and Motwani [7], proposing dedicated hash functions for Hamming space and theoretical approximation guarantees. LSH is based on a family of functions $\mathcal{H} = \{h : S \rightarrow U\}$ that map values from a space S (in our case the binary vector space $\mathbb{B}^d$) to binary strings from space U. To be useful for similarity search, $h(p)$ must be (r_1, r_2, p_1, p_2)-sensitive, i.e. points closer to a given reference point have a higher probability of being assigned the same hash as points further away. From this family $\mathcal{H}$, a family $\mathcal{G} = \{g : S \rightarrow U^k\}$ with $g_j(p) = (h_1(p), \ldots, h_k(p)), h_i \in \mathcal{H}$ is constructed, i.e. $\mathcal{G}$ corresponds to a concatenation of different hash functions of the same family. For index generation, t of these functions $g_j(p)$ are generated and for each of them a hash table is constructed from the features stored in the database. Based on the resulting hash tables, query processing for kNN and range queries can be achieved as shown in Figure 18.1. Given a query vector q, the hashes $g_j(q)$ are computed, then the corresponding entries from table j, are retrieved. Concatenating the resulting candidates of all tables leads to the candidate set. During refinement, the actual distance between the query and each candidate is computed and the query predicate is evaluated on this reduced set, speeding up the evaluation.

Initially, the following functions h_i (projecting a binary vector onto atomic random dimensions) and g_j (projecting on a random lower-dimensional space), were proposed for search in Hamming Space [7]:

$$\mathcal{H} = \{h : h_i(b) = b_i\}$$

$$\mathcal{G} = \{g : g(p) = (h_{i_1}, \ldots, h_{i_n}), \{i_1, \ldots, i_n\} \subset \{1, \ldots, d\}\}$$

For example, a function $h_2(b)$ would project a 3D binary vector to its second dimension, e.g. $h_2((1, 0, 1)) = 0$. A function $g(p)$ then concatenates different functions h, producing a subvector of the initial one. For a function $g(b) = (h_1(b), h_2(b))$ mapping to the first two dimensions, we would get a vector $g((1, 0, 1)) = (1, 0)$.

To be effective, LSH has to query a relatively high number t of different hash tables with different hash functions, incurring high storage overhead, as each of the tables has to store references to all of the indexed features, resulting in a memory consumption linear in t. To solve this issue, [32]

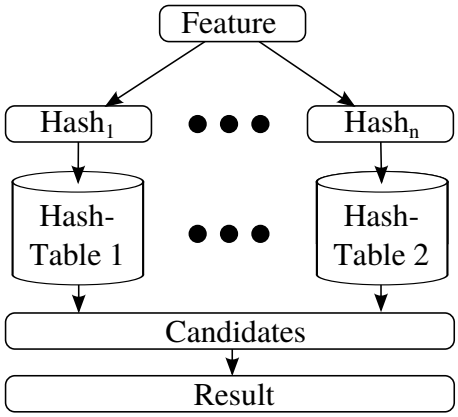


Figure 18.1: LSH-based indexing

proposed to use Multi-Probe LSH that visits different hash cells close to the query point in a single hash table, similar to the soft assignment policy in computer vision [110]. This procedure significantly reduces storage overhead and has been successfully used for querying binary features in the computer vision community [114]. Trzcinski et al. [118] proposed optimized hash functions for LSH-based search in Hamming space, aiming at improving the efficiency of this class of approaches.

Despite the success of LSH-based methods for querying binary features in Hamming space, techniques based on quantization (and therefore variants of k-means) have been investigated recently. The approaches are either plugged into the BoVW model [116, 117], or used for nearest neighbor search [119, 118]. Recall that quantization-based techniques first perform a k-means clustering of the features contained in the database. In a second step, each database vector is assigned to its closest cluster center. During query processing, the query vector is assigned to its closest cluster center, all database vectors assigned to the same cluster are returned.

For k-means the probability of two features assigned to the same cluster center being spatially close is high, while the probability of two features assigned to different cluster centers being spatially close is relatively low. Therefore, according to [123], quantization-based approaches can be seen as special LSH hash functions: the hash corresponds to the ID of the closest cluster center, i.e. features assigned to the same cluster center have the same hash. This results in the following mathematical definition:

$$\mathcal{H} = \{h : h_i(b) = \underset{c \in centers_i}{argmin}\, dist_H(v(c), b))\}$$

$$\mathcal{G} = \{g : g(p) = h_i(p)\}$$

Here, $centers_i$ denotes the cluster IDs of clustering i and $v(c)$ the vector corresponding to ID c, i.e. the c-th cluster mean. The hash function family $\mathcal{G}$ in this case is trivial, as each hashing function of $\mathcal{G}$ consist only of a single function h_i. On the other hand $h_i(p)$ becomes more complex as it consists of more than one bit. For standard k-means clustering, due to the assignment step, the hash computation is exponential in the hash length, therefore approximate solutions have been developed [122, 142, 121, 12, 159] in the real-valued domain. For quantizing binary vectors, solutions based on k-medoids, k-medians or k-means can be employed. Using k-means directly does not optimize according to l_1 distance, but rather according to the squared Euclidean distance, making this solution theoretically inapplicable to Hamming space. Most often, even if unknowingly, the k-medians [160] algorithm has been used which optimizes cluster centers according to the l_1 norm, directly corresponding to the Hamming distance on binary feature vectors. k-medians can be optimized for binary features, as the median computation degenerates to a majority voting in this case [161].

As BoVW is based on quantization, it can be abstracted to LSH as well [123], however without the costly refinement step that calculates exact distances on the candidate features retrieved from the hash table. Querying binary features in BoVW-based systems has most likely been initially proposed by Galvez-Lopez and Tardos [116, 117]. To achieve acceptable runtime performance, the authors employed the idea of hierarchical k-means (HkM)[122]. Hierarchical k-means (HkM) performs k-means clustering with a relatively low k'. It then assigns each point in the database to the corresponding cluster. For each of these sets of feature vectors, it then recursively clusters the set until a given height l is reached, resulting in a tree-like structure with k'^l leaf nodes. Due to the resulting tree structure, the computational complexity of assigning feature vectors to cluster centers decreases to $O(k'l)$, however the storage complexity of the approach remains in $O(k'^l)$ and some accuracy is lost. Utilizing the idea of HkM, Muja and Lowe [119] and Trzcinski et al. [118] simultaneously proposed to use a forest of random clustering trees on binary feature vectors. Mapping this to LSH, the leaves of a tree provide the hash functions, and each tree provides the hash function for a different hash table. Each of the trees is similar to a HkM tree, however cluster centers are not assigned by k-medians, but rather by randomly selecting cluster centers from the data: if multiple hash tables (i.e. trees) are employed, the corresponding hash functions should be *independent*. If a clustering would be performed, in the best case this clustering would represent a global optimum, generating exactly the same *non-independent* clustering for each function from $\mathcal{G}$. However, as the iterative k-medians based clustering approaches only find local minima instead of global ones, k-medians

clustering could be employed with different initial seeding, as it has been done in the case of real valued vectors [123]. We will evaluate both randomized and optimized clusterings in Section 18.2. In [119], better retrieval performance than LSH-based approaches is achieved with multi-probing. In contrast, [118] achieved slightly worse results than LSH, however without multi-probing. Hierarchical k-medians clustering has also been employed by [161], however with optimized cluster centers.

Other quantization techniques that have been employed for real-valued image features could also be used for binary features, such as residual vector quantization [159], and product quantization [12]. In the context of residual vector quantization that involves subtraction operations, the subtraction operation would have to be replaced by an XOR operation to make the residuals remain in binary space.

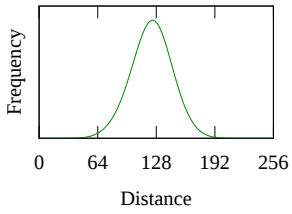
18.2 Experimental Evaluation

For our experiments we extracted ORB [114] features from the ALOI dataset[1] using OpenCV. We aimed at extracting binary features instead of binarizing real-valued ones as this can lead to performance advantages in real-world applications. Other binary features than ORB could be chosen as well; the decision depends on the application. While in Chapter 17 we concentrated on highly accurate binary features, in this section we chose ORB, as these features can be extracted extremely fast in only a few milliseconds. From the originally ∼30 million features we used a random sample of size 10^7 as database content. Distances in the dataset follow a normal distribution with a maximum frequency at a distance of 124 and covering nearly all possible distances, see Figure 18.2b. The different quantizations (k-medians, HkM and random HkM (RHkM)) were computed from a subset of about 7 million of the 30 million features containing, for each of the 1000 objects, 18 of the 72 images per object. We only considered the images varying in camera angle, but not the images varying e.g. in light color, as these are assumed to produce very similar features. Tailoring quantization to a specific database is reasonable as it makes quantization techniques more powerful. Our experiments are building upon [119, 118], who have initially proposed to use hierarchical k-means in Hamming space. We aim at reproducing and extending their experimental results by investigating the impact of system-inherent parameters on the performance of the different hashing techniques, namely the number of probes, the number of tables, the number of bits and the database size. As both of these publications did not plug the different solutions into the LSH

[1] http://aloi.science.uva.nl

Parameter	Value
#probes	1, **20**, 100, 1000, 10000
#tables	**1**, 2, 4, 8
#bits (LSH)	8, 10, 12, 14, **16**, 20
k (k-medians)	2^8, 2^{10}, 2^{12}, 2^{14}, $\mathbf{2^{16}}$
h (HkM)	**4**
k' (HkM)	2^2, 2^3, $\mathbf{2^4}$, 2^5, 2^6

(a) Experimental setup. Values in bold font denote default values.



(b) Distance Distribution (ORB, 10k Feature Vectors)

Figure 18.2: Parameter Settings and Distance Distribution.

scheme, we aim at making the different solutions comparable, e.g. in the number of bits of the underlying hash function, aiming at providing insights extending those from [119, 118]. In our experiments, we fix each of the free parameters to a given value and vary a single parameter to provide a wide image of the algorithms' performance. The default parameters are provided in Table 18.2a. As the parameter k is overloaded (for k-means and for the number of neighbors), we denote the number of neighbors retrieved in a kNN query as $|NN|$ during our evaluation.

To make the approaches comparable from an LSH-based point of view, all of the evaluated approaches are configured to generate hashes of the same length (16 bits by default). Equivalent to [123], who performed similar experiments for real-valued features, we set the number of hash tables to a single one in our default setting. Although this is not realistic, it gives insight into the performance of the actual hash function. As it does not provide insights into the *independence* of different hash functions which becomes relevant when using more than one hash table (the default with LSH), we will evaluate this behaviour in Figure 18.4. The number of neighbors $|NN|$ was set arbitrarily to $|NN| = 10$, our results on experiments for $|NN| = 2$ and $|NN| = 100$ show a similar behaviour. We will however mention variations in the results for different values of $|NN|$ in the corresponding sections. In our experiments we concentrate on analyzing the *Recall* for each set of parameters, for both range- and kNN queries; a short excursus also considers the BoVW paradigm. Where necessary we also consider different performance measures such as the number of distance calculations; we favoured this measure over runtime as it provides a platform- and implementation-independent measure of the computational complexity of a given approach. However note that other costs, such as restructuring the priority queue of

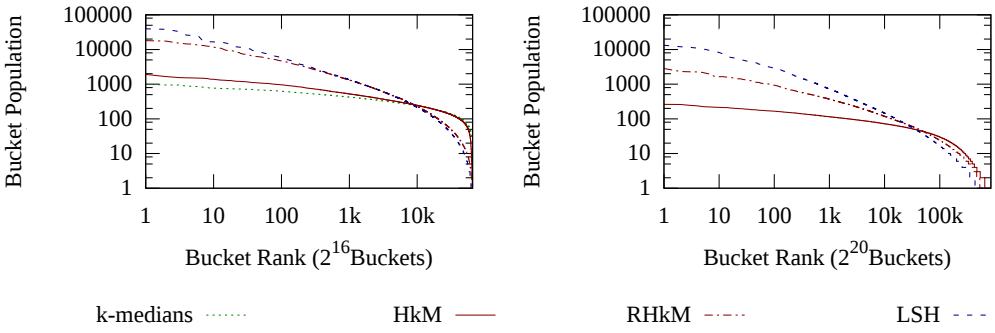


Figure 18.3: Populatation of buckets (log-log-space).

the quantization-based approaches, is not considered.

The optimized clusterings have been generated with a k-medoids approach. First, a random sample of 50% of the training features was selected and clustered with 5 iterations. Then, given the resulting centroids as initial cluster centers, the whole training set was clustered, stopping after an additional 10 iterations. The experimental evaluation has been conducted in the JAVA-coded framework ELKI[162] which has been specifically designed for the performance evaluation of data mining and query processing algorithms, providing variants of the k-means algorithm, including k-medians.

18.2.1 Nearest Neighbor Queries

Distribution of Hash-Code Frequencies. Following [123], let us first evaluate the population of hash buckets; Figure 18.3 visualizes the distribution for hashes of length 16 and 20 (for 20 bits, k-medians has not been evaluated due to its high complexity). If the population is equi-distributed, this leads to a good selectivity, as regardless of the location of a query in binary space, the number of candidates in the corresponding bucket is similar. The more uneven the distribution of features over different hashes, the more irrelevant features have to be refined if a highly populated bucket is found, leading to high runtimes; if a poorly populated bucket is accessed, the recall of the query becomes unnecessarily low. Similar to the case of real-valued features [123], the original LSH functions lead to an unbalanced distribution of hashes. On the other hand, both RHkM and HkM perform better. However, there is a significant difference between the random and the optimized version; RHkM performs very close to the original LSH, while the distribution of HkM is much more uniform, quite close to k-medians that performs best. The skewed distribution of the HkM-based approaches

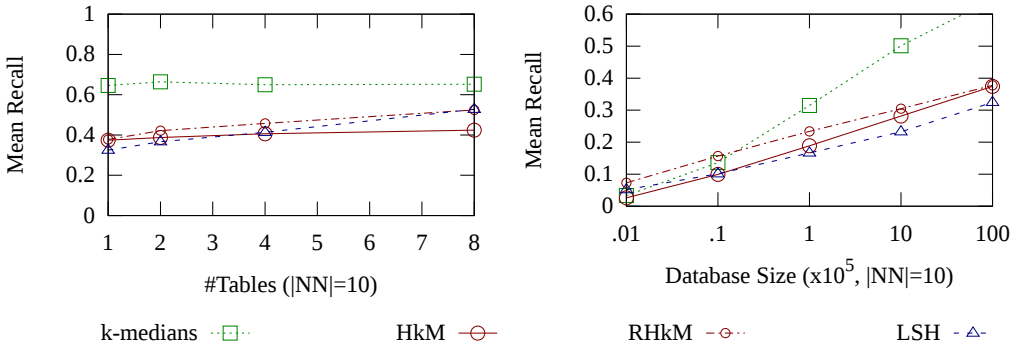


Figure 18.4: Varying # Tables (left) and Database Size (right)

is traded for a better computational performance: k-medians can only be seen as a theoretic solution, as it implies scanning all cluster centers during each query. The number of cluster centers scanned increases exponentially in the number of hash bits. Although these distance calculations are very fast in binary space, they can still incur a large overhead for higher bit lengths.

Number of Tables. In the following experiment we aim at investigating the effect of the number of tables on the different approaches (see Figure 18.4 left) in order to find out how well several hash tables complement each other. For this experiment, we kept the number of probes constant and split them between an increasing number of tables. For traditional hash functions, probes are split equally between tables. For quantization-based approaches, the cluster centers of all tables are ranked together, resulting in a not necessarily uniform split between tables. The experimental setup aims at showing if the performance gain is actually contributed to the higher number of tables rather than the increasing number of probes.

An increasing number of tables makes mainly sense for the traditional LSH functions, and less strongly for the randomized HkM approach. The performance gain of optimized quantization-based approaches diminishes with increasing number of tables, as the different quantizations are not independent. This observation is different to the real-valued case, where Pauleve et al. [123] stated that initializing k-means with different seeds can provide enough randomness to build independent hash tables. Note that, if the number of tables becomes sufficiently large, even the random HkM-based approach loses against the LSH-based hash functions. For $|NN| = 10$, the equilibrium was at eight tables; for $|NN| = 2$ it became lower (two tables), and for $|NN| = 100$ it became larger. Therefore, if memory is important, quantization-based approaches can be a useful solution. If recall is an issue and memory is not, LSH-based functions are the matter of choice.

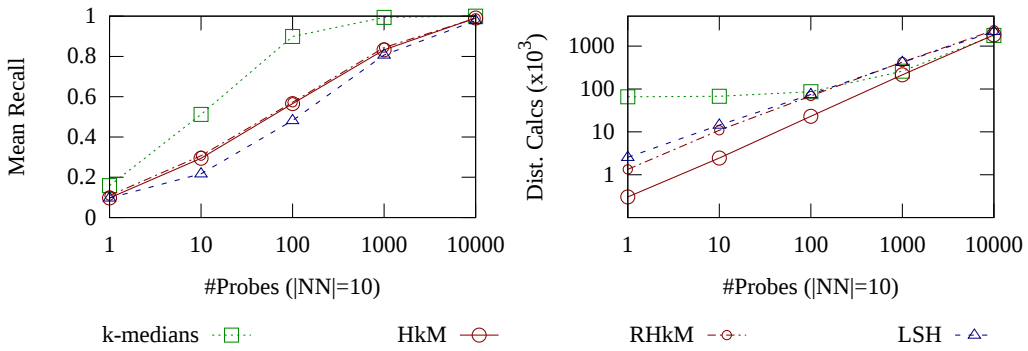


Figure 18.5: Varying # Probes, Recall (left) and Distance Calculations (right)

Database Size. As the number of objects indexed in a database affects the nearest neighbor distance of objects, varying the database size can also affect the performance of hash functions: If the number of bits in a LSH function is too high, the NN range of queries is severely restricted, such that in small databases the NNs will be found only with small probability. However if the database size increases, the NN-range of an object shrinks. Therefore, recall is positively affected by an increasing database size, the largest increase can be observed for the k-medians based quantization approaches (see Figure 18.4 right). As a result, given a large database size and a specified number of bits, it can make sense to use methods based on quantization.

Number of Probes. Besides varying the number of hash tables (which trades space for recall), recall can also be traded for computational complexity by increasing the number of probes, see Figure 18.5 (left). Our experiments indicate that for small $|NN|$ and with an increasing number of probes, the HkM, RHkM and LSH-based approaches become very similar, leading to no significant performance gain of quantization. For larger $|NN|$, LSH catches up only for larger number of probes. The k-medians based baseline shows a better performance than the rest, reaching recall rates of greater than 0.9 already at a hundred probes. By comparing the number of distance calculations (see Figure 18.5 (right)) we aim at providing implementation-independent insights into the computational efficiency of the different approaches. The experiment shows that HkM can significantly better reduce the number of distance calculations than LSH and k-medians, as this approach leads to a much more uniform distribution of values in different buckets than LSH (cp. Figure 18.3), and does not have the linear complexity of computing the closest cluster center as k-medians. This, however, does not hold as strictly for RHkM, indicating a similar runtime for RHkM and

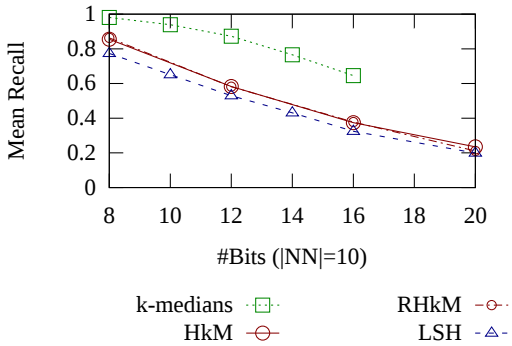


Figure 18.6: Varying # Bits

LSH. For additional runtime experiments, we refer to the original publications [119, 118].

Hash length. In the following experiment, Figure 18.6, we aim at investigating the impact of a varying hash length on the recall of the different approaches. For the sake of easy configurability it is important that all approaches behave similarly, i.e. the choice of the best algorithm does not depend on the number of bits used, which is the case in our setting. We would like to mention that with increasing $|NN|$ the difference between quantization-based approaches and LSH becomes larger: LSH loses recall relatively to the other approaches, indicating that it can make sense for large $|NN|$ (e.g. $|NN| = 100$) to use quantization. As in image recognition it is likely that $|NN|$ is chosen relatively large to generate more candidate matches and to be more robust to noise, HkM can achieve a better performance than the LSH hashing functions.

18.2.2 Range Queries and BoVW

Although this work concentrates on nearest neighbor queries, for the sake of completeness we also want to shed some light on range queries. Besides mean recall we evaluate the mean false positive rate. In Figure 18.7 we plot the recall given a specific radius and equivalently the false positive rate. Note that for small ranges results become more noisy as an object being a result of a range query with a small range is quite improbable (see Figure 18.2b). In our setting, the mean recall of the different approaches behaves very similar. Only k-medians manages to achieve a significantly higher recall, by increasing the probability of *wanted* hash collisions.

Although the false hit rate varies significantly for the different approaches, it does not have a large effect on range queries, as unwanted results are filtered

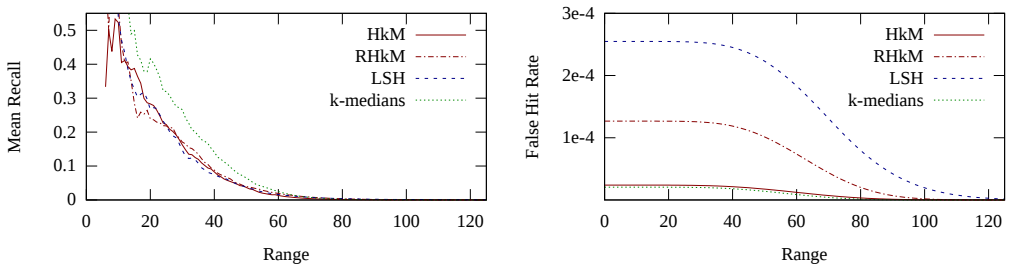


Figure 18.7: Recall and False Hit Rate for Range Queries

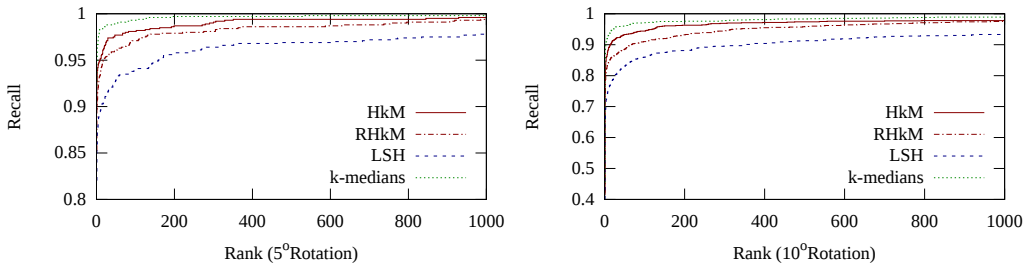


Figure 18.8: Applying the Hashing Functions to a BoVW-based Ranking

out during refinement. However, it provides useful insights into their performance when plugged into the BoVW framework: Considering that recall is similar for both HkM and LSH, would it make sense to use LSH functions for binary vectors in the BoVW-paradigm, directly treating keypoints falling into the same buckets as matches? Not necessarily, as not only recall matters, but also the false positive rate (see Figure 18.7 right). As the difference between BoVW and LSH is mainly that BoVW skips the pruning step, it is of major importance to avoid false positives: false positives put noise on the word vector by producing inaccurate word assignments. In Figure 18.8 we plugged the different hash functions into the BoVW pipeline (without geometric verification). We used the dataset from training the quantization approaches as database. As it contains all images in 20 degree angular steps, we queried with images having an angular distance of 5 and 10 degrees (Figure 18.8 left and right, respectively), visualizing the recall (of images, not features) with increasing rank. The different approaches have similar recall curves in Figure 18.7, but LSH performs significantly worse than the remaining approaches.

Chapter 19

Conclusions

This part of the thesis addressed nearest neighbor queries in image databases. As a first step, we evaluated an alternative pipeline for decreasing the runtimes of object recognition when kNN queries are used for the generation of tentative correspondences instead of Bags of Visual Words. While the reduction of query features can have negative effects on query performance, especially if the unmodified standard recognition pipeline is used, some simple modifications in the pipeline aiming at feature ranking and match expansion can already produce good results at only a fraction of kNN queries. Improvements in the match expansion stage should aim at increasing efficiency and effectiveness. Due to the simple structure of the pipeline used in this chapter, these improvements can be easily integrated.

As a second contribution we analyzed query processing on binary features. In this research we come to the following conclusions: given that a nearest neighbor search in Hamming space has to be performed, both quantization-based approaches and LSH provide good results, but LSH can not be beaten for larger number of tables. For a low number of tables, HkM and RHkM are the matter of choice. For BoVW-based image retrieval, similarly to the real-valued domain, the quantization-based approaches are the matter of choice: As the probability of false hits of quantization-based approaches is significantly lower, the necessity of refinement of these solutions decreases, making them applicable to the BoVW paradigm.

Part V

Conclusions

In this thesis, we have addressed a variety of challenges from the wide research field of similarity search, concentrating on nearest neighbor queries. We addressed the development of efficient algorithms for complex query predicates, effective and efficient query processing for uncertain spatio-temporal data, and query processing on set-based objects in image databases.

RkNN Join Processing. In Part II of this work we addressed a complex query predicate derived from reverse nearest neighbor queries, namely the RkNN join, which has not been analyzed in depth until now. As a first step we first formalized the RkNN join, identifying its monochromatic and bichromatic versions and their self-join variants. While the bichromatic RkNN join can be transformed to a standard kNN join, such a transformation is not possible for the monochromatic case. To fill the research gap of an in-depth analysis of monochromatic RkNN joins, we developed solutions for this variant of the RkNN join, including a self-pruning and a mutual pruning algorithm. Our experimental evaluation indicates that, in a variety of scenarios, classic algorithms for performing single RkNN queries do not yield the required performance, and that our newly proposed algorithms often lead to better results. Our experiments however also indicated that in some scenarios, e. g. where the database is relatively static, other solutions based on traditional RkNN queries can achieve good performance. In the future we will explore further pruning strategies such as a combination of mutual pruning and self pruning in order to increase the number of pruned subtrees, decreasing the execution time of RkNN join algorithms and the number of page accesses. We would further like to extend our join algorithms to RkNN monitoring.

Similarity Queries on Uncertain Spatio-Temporal Data. The second challenge we tackled in Part III of this thesis addressed the development of efficient and effective techniques for query processing on uncertain spatio-temporal data. Starting from the traditional definition of kNN queries and a Markov-chain based approach for modeling uncertain trajectories, we developed efficient querying mechanisms for nearest neighbor queries on uncertain trajectories, considering temporal dependencies during query evaluation. To achieve this goal, we developed suitable query predicates that do not only return the objects closest to the query but also their probability of being a nearest neighbor, namely the P$\exists$NNQ, P$\forall$NNQ and the continuous PCNNQ query. In a theoretical evaluation we addressed the complexity of these query predicates; our sampling approach and an index-based filter-refinement approach aimed at increasing the practical efficiency of evaluating these query

predicates. We then extended these findings to reverse nearest neighbor queries, introducing the P∃RNNQ and the P∀RNNQ query predicates. During our experimental evaluation we have shown the efficiency and effectiveness of these approaches.

One drawback of the proposed techniques however is that they rely highly on the spatial correlation of uncertain objects between consecutive points in time. If to many states can be reached between tics, the efficiency of the proposed techniques will suffer, not only because the underlying state and transition matrices will become less sparse, but also because the selectivity of the proposed pruning techniques declines. Therefore, while the approaches work well for for a single mode of transport such as taxis, the introduction of an additional mode of transport such as airplanes will decrease the performance of these approaches. We aim at addressing this issue in the future.

kNN Queries in Image Databases. The third main part of this thesis, Part IV, addressed the problem of querying large datasets containing set-based objects, namely image databases. As we have seen, query processing in this area of research is drifting more and more towards accurate matching approaches based on nearest neighbor queries due to the invention of efficient and effective indexing techniques such as product quantization. These are however still relatively expensive from a computational point of view if hundreds or even thousands of keypoints are queried. In this context we evaluated a modified recognition pipeline optimized for kNN queries that combines a reduction of the number of matching queries to increase query efficiency with an additional match expansion step for increasing query effectiveness. As research on feature extraction is currently more and more shifting from the real-valued domain to the binary domain, we then evaluated efficient approximate indexing techniques for binary feature vectors based on the LSH scheme. Our experimental evaluation aimed at identifying advantages and disadvantages of the different approaches and providing suggestions for their use in real-world scenarios.

Bibliography

[1] I. EMC^2. (2014) The digital universe of opportunities. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf

[2] M. Deza and E. Deza, *Dictionary of Distances*. Elsevier Science, 2006.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*, 1996, pp. 226–231.

[5] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2006.

[6] M. M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. SIGMOD*, 2000, pp. 93–104.

[7] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. STOC*, 1998, pp. 604–613.

[8] G. R. Hjaltason and H. Samet, "Ranking in spatial databases," in *Proc. SSD*, 1995, pp. 83–95.

[9] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proc. SIGMOD*, 1995, pp. 71–79.

[10] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. CVPR*, 2014.

[11] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Proc. CVPR*, 2012, pp. 3108–3115.

[12] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE PAMI*, vol. 33, no. 1, pp. 117–128, 2011.

[13] J. Niedermayer, A. Züfle, T. Emrich, M. Renz, N. Mamoulis, L. Chen, and H.-P. Kriegel, "Probabilistic nearest neighbor queries on uncertain moving object trajectories," *PVLDB*, vol. 7, no. 3, pp. 205–216, 2013.

[14] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz, "Continuous probabilistic nearest-neighbor queries for uncertain trajectories," in *Proc. EDBT*, 2009, pp. 874–885.

[15] F. Korn and S. Muthukrishnan, "Influenced sets based on reverse nearest neighbor queries," in *Proc. SIGMOD*, 2000, pp. 201–212.

[16] W. Wu, F. Yang, C.-Y. Chan, and K. Tan, "FINCH: Evaluating reverse k-nearest-neighbor queries on location data," in *Proc. VLDB*, 2008, pp. 1056–1067.

[17] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. ICDE*, 2001, pp. 421–430.

[18] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proc. VLDB*, 2006, pp. 751–762.

[19] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[20] M. Busch. (2010) Twitter's new search architecture. [Online]. Available: https://blog.twitter.com/2010/twitters-new-search-architecture

[21] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. CVPR*, vol. 1, 2005, pp. 886–893.

[22] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE SMC*, vol. 3, no. 6, pp. 610–621, 1973.

[23] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proc. ICCV*, 2003, pp. 1470–1477.

[24] Idée Inc. (2015) How does tineye work? does it use image names? [Online]. Available: https://www.tineye.com/faq#how

[25] M. Müller, *Information retrieval for music and motion*. Springer, 2007, vol. 2.

[26] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Francisco: Morgan Kaufmann, 2006.

[27] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proc. SIGMOD*, 1984, pp. 47–57.

[28] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An efficient and robust access method for points and rectangles," in *Proc. SIGMOD*, 1990, pp. 322–331.

[29] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Boosting spatial pruning: On optimal pruning of mbrs," in *Proc. SIGMOD*, 2010, pp. 39–50.

[30] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, 1999, pp. 518–529.

[31] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. SOCG*. ACM, 2004, pp. 253–262.

[32] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proc. VLDB*, 2007, pp. 950–961.

[33] P. Kröger and M. Renz, "Multi-step query processing," in *Encyclopedia of Database Systems*, L. LIU and M. ÖZSU, Eds. Springer US, 2009, pp. 1858–1862. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9_227

[34] H.-P. Kriegel, P. Kröger, P. Kunath, and M. Renz, "Generalizing the optimality of multi-step k-nearest neighbor query processing," in *Proc. SSTD*, 2007, pp. 75–92.

[35] T. Seidl and H.-P. Kriegel, "Optimal multi-step k-nearest neighbor search," in *Proc. SIGMOD*, 1998, pp. 154–165.

[36] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas, "Fast nearest neighbor search in medical image databases," in *Proc. VLDB*, 1996, pp. 215–226.

[37] D. Gunopulos and G. Das, "Time series similarity measures (tutorial pm-2)," in *Tutorial notes of the sixth ACM SIGKDD*. ACM, 2000, pp. 243–307.

[38] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *Proc. SIGMOD*, 2002, pp. 334–345.

[39] S. Šaltenis, S. Jensen, S. Leutenegger, and M. Lopez, "Indexing the positions of continuously moving objects." in *Proc. SIGMOD*, 2000, pp. 331–342.

[40] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle, "Indexing uncertain spatio-temporal data," in *Proc. CIKM*, 2012, pp. 395–404.

[41] ——, "Querying uncertain spatio-temporal data," in *Proc. ICDE*, 2012, pp. 354–365.

[42] R. Cheng, T. Emrich, H. Kriegel, N. Mamoulis, M. Renz, G. Trajcevski, and A. Züfle, "Managing uncertainty in spatial and spatio-temporal data," in *Proc. ICDE*, 2014, pp. 1302–1305.

[43] P. Vagata and K. Wilfong. (2014) Scaling the facebook data warehouse to 300 pb. [Online]. Available: https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

[44] YouTube LLC. (2015) Statistics. [Online]. Available: https://www.youtube.com/yt/press/statistics.html

[45] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[46] T. Emrich, H.-P. Kriegel, P. Kröger, J. Niedermayer, M. Renz, and A. Züfle, "A mutual-pruning approach for rknn join processing," in *Proc. BTW*, 2013, pp. 21–35.

[47] T. Emrich, H.-P. Kriegel, P. Kröger, J. Niedermayer, M. Renz, and A. Züfle, "Reverse-k-nearest-neighbor join processing," in *Proc. SSTD*, 2013, pp. 277–294.

[48] ——, "On reverse-k-nearest-neighbor joins," *GeoInformatica*, pp. 1–32, 2014.

[49] T. Emrich, "Coping with distance and location dependencies in spatial, temporal and uncertain data," Ph.D. dissertation, Ludwig-Maximilians University Munich, 2013.

[50] A. Züfle, "Similarity search and mining in uncertain spatial and spatio-temporal databases," Ph.D. dissertation, Ludwig-Maximilians University Munich, 2013.

[51] T. Emrich, H.-P. Kriegel, N. Mamoulis, J. Niedermayer, M. Renz, and A. Züfle, "Reverse-nearest neighbor queries on uncertain moving object trajectories," in *Proc. DASFAA*, 2014, pp. 92–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05813-9_7

[52] T. Emrich, M. Franzke, H. Kriegel, J. Niedermayer, M. Renz, and A. Züfle, "An extendable framework for managing uncertain spatio-temporal data," in *Proc. SIGMOD*, 2014, pp. 1087–1090.

[53] J. Niedermayer and P. Kröger, "Retrieval of binary features in image databases: A study," in *Proc. SISAP*, 2014, pp. 151–163.

[54] ——, "Minimizing the number of keypoint matching queries for object retrieval," in *Proc. BMVC*, 2015 (to appear).

[55] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Züfle, "Inverse queries for multidimensional spaces," in *Proc. SSTD*, 2011, pp. 330–347.

[56] R. A. Jarvis and E. A. Patrick, "Clustering using a similarity measure based on shared near neighbors," *IEEE TC*, vol. C-22, no. 11, pp. 1025–1034, 1973.

[57] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proc. SIGMOD*, 1999, pp. 49–60.

[58] V. Hautamäki, I. Kärkkäinen, and P. Fränti, "Outlier detection using k-nearest neighbor graph," in *Proc. ICPR*, 2004, pp. 430–433.

[59] W. Jin, A. K. H. Tung, J. Han, and W. Wang, "Ranking outliers using symmetric neighborhood relationship," in *Proc. PAKDD*, 2006, pp. 577–593.

[60] C. Yang and K.-I. Lin, "An index structure for efficient reverse nearest neighbor queries," in *Proc. ICDE*, 2001, pp. 485–492.

[61] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz, "Efficient reverse k-nearest neighbor search in arbitrary metric spaces," in *Proc. SIGMOD*, 2006, pp. 515–526.

[62] I. Stanoi, D. Agrawal, and A. E. Abbadi, "Reverse nearest neighbor queries for dynamic databases," in *Proc. DMKD*, 2000, pp. 44–53.

[63] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun, "High dimensional reverse nearest neighbor queries," in *Proc. CIKM*, 2003, pp. 91–98.

[64] Y. Tao, D. Papadias, and X. Lian, "Reverse kNN search in arbitrary dimensionality," in *Proc. VLDB*, 2004, pp. 744–755.

[65] C. Böhm and F. Krebs, "The k-nearest neighbor join: Turbo charging the KDD process," *KAIS*, vol. 6, no. 6, pp. 728–749, 2004.

[66] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors queries in spatial databases," in *Proc. SSDBM*, 2004, pp. 297–306.

[67] C. Yu, R. Zhang, Y. Huang, and H. Xiong, "High-dimensional knn joins with incremental updates," *Geoinformatica*, vol. 14, no. 1, pp. 55–82, 2010.

[68] J. G. Venkateswaran, "Indexing techniques for metric databases with costly searches," Ph.D. dissertation, University of Florida, Gainesville, FL, USA, 2007, aAI3300799.

[69] Y. Tao, M. L. Yiu, and N. Mamoulis, "Reverse nearest neighbor search in metric spaces," *IEEE TKDE*, vol. 18, no. 9, pp. 1239–1252, 2006.

[70] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *ICDE*, 2011, pp. 577–588.

[71] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Reverse k-nearest neighbor search in dynamic and general metric databases," in *Proc. EDBT*, 2009, pp. 886–897.

[72] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler, "Reverse k-nearest neighbor search based on aggregate point access methods," in *Proc. SSDBM*, 2009, pp. 444–460.

[73] C. Xia, W. Hsu, and M. L. Lee, "ERkNN: efficient reverse k-nearest neighbors retrieval with local kNN-distance estimation," in *Proc. CIKM*, 2005, pp. 533–540.

[74] Y. Liu and Z.-X. Hao, "High-dimensional main-memory reverse k nearest neighbor query and join," *Jisuanji Gongcheng/ Computer Engineering*, vol. 37, no. 24, 2011.

[75] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient olap operations in spatial data warehouses," in *Proc. SSTD*, 2001, pp. 443–459.

[76] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler, "Incremental reverse nearest neighbor ranking," in *Proc. ICDE*, 2009, pp. 1560–1567.

[77] E. Achtert, A. Hettab, H.-P. Kriegel, E. Schubert, and A. Zimek, "Spatial outlier detection: Data, algorithms, visualizations," in *Proc. SSTD*, 2011, pp. 512–516.

[78] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: an efficient access method for similarity search in metric spaces," in *Proc. VLDB*, 1997, pp. 426–435.

[79] T. Emrich, F. Graf, H.-P. Kriegel, M. Schubert, and M. Thoma, "On the impact of flash ssds on spatial indexing," in *Proc. DaMoN*, 2010, pp. 3–8.

[80] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *Proc. SIGMOD*, 2008, pp. 715–728.

[81] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, and Y. Huang, "T-drive: Driving directions based on taxi trajectories," in *Proc. ACM GIS*, 2010, pp. 99–108.

[82] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma., "Understanding mobility based on gps data." in *Proc. Ubicomp*, 2008, pp. 312–312.

[83] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Algorithms for nearest neighbor search on moving object trajectories," *Geoinformatica*, vol. 11, no. 2, pp. 159–193, 2007.

[84] R. H. Güting, T. Behr, and J. Xu, "Efficient k-nearest neighbor search on moving object trajectories," *VLDB J.*, vol. 19, no. 5, pp. 687–714, 2010.

[85] G. S. Iwerks, H. Samet, and K. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *Proc. VLDB*, 2003, pp. 512–523.

[86] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *Proc. VLDB*, 2002, pp. 287–298.

[87] C. Xu, Y. Gu, L. Chen, J. Qiao, and G. Yu, "Interval reverse nearest neighbor queries on uncertain data with markov correlations," in *Proc. ICDE*, 2013, pp. 170–181.

[88] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and querying moving objects," in *Proc. ICDE*, 1997, pp. 422–432.

[89] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *Proc. SIGMOD*, 2004, pp. 611–622.

[90] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proc. ICDE*, 2005, pp. 631–642.

[91] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proc. ICDE*, 2005, pp. 643–654.

[92] H. Mokhtar and J. Su, "Universal trajectory queries for moving object databases," in *Proc. MDM*, 2004, pp. 133–144.

[93] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain, "Managing uncertainty in moving objects databases," *ACM TODS*, vol. 29, no. 3, pp. 463–507, 2004.

[94] G. Trajcevski, A. N. Choudhary, O. Wolfson, L. Ye, and G. Li, "Uncertain range queries for necklaces," in *Proc. MDM*, 2010, pp. 199–208.

[95] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE TKDE*, vol. 16, no. 9, pp. 1112–1127, 2004.

[96] S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, and M. Chau, "Putmode: prediction of uncertain trajectories in moving objects databases," *Appl. Intell.*, vol. 33, no. 3, pp. 370–386, 2010.

[97] G. Kollios, D. Gunopulos, and V. Tsotras, "Nearest neighbor queries in a mobile environment," in *Spatio-Temporal Database Management*. Springer, 1999, pp. 119–134.

[98] Y.-K. Huang, S.-J. Liao, and C. Lee, "Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction," in *Proc. SSDBM*, 2008, pp. 549–557.

[99] G. Li, Y. Li, L. Shu, and P. Fan, "Cknn query processing over moving objects with uncertain speeds in road networks," in *APWeb*, 2011, pp. 65–76.

[100] G. Trajcevski, R. Tamassia, I. F. Cruz, P. Scheuermann, D. Hartglass, and C. Zamierowski, "Ranking continuous nearest neighbors for uncertain trajectories," *VLDB J.*, vol. 20, no. 5, pp. 767–791, 2011.

[101] X. Lian and L. Chen, "Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data," *VLDB J.*, vol. 18, no. 3, pp. 787–808, 2009.

[102] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei, "Probabilistic reverse nearest neighbor queries on uncertain data," *IEEE TKDE*, vol. 22, no. 4, pp. 550–564, 2010.

[103] T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, S. Zankl, and A. Züfle, "Efficient probabilistic reverse nearest neighbor query processing on uncertain data," in *Proc. VLDB*, 2011, pp. 669–680.

[104] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. VLDB*, 1994, pp. 487–499.

[105] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas, "Mcdb: a monte carlo approach to managing uncertain data," in *Proc. SIGMOD*, 2008, pp. 687–700.

[106] L. R. Welch, "Hidden markov models and the baum-welch algorithm," *IEEE Information Theory Society Newsletter*, vol. 53, no. 4, pp. 1,10–13, 2003.

[107] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, pp. 13–30, 1963.

[108] L. Rabiner and B.-H. Juang, "An introduction to hidden markov models," *IEEE ASSP*, vol. 3, no. 1, pp. 4–16, 1986.

[109] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. KDD*, 2011, pp. 316–324.

[110] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Proc. CVPR*, 2008, pp. 1–8.

[111] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. CVPR*, 2013, pp. 2946–2953.

[112] A. Babenko and V. S. Lempitsky, "The inverted multi-index," in *Proc. CVPR*, 2012, pp. 3069–3076.

[113] H. Jégou, M. Douze, and C. Schmid, "Exploiting descriptor distances for precise image search," INRIA, Tech. Rep. 7656, 2011.

[114] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Proc. ICCV*, 2011, pp. 2564–2571.

[115] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Proc. CVPR*, 2012, pp. 510–517.

[116] D. Gálvez-López and J. D. Tardós, "Real-time loop detection with bags of binary words," in *Proc. IROS*, 2011, pp. 51–58.

[117] ——, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[118] T. Trzcinski, V. Lepetit, and P. Fua, "Thick boundaries in binary space and their influence on nearest-neighbor search," *Pattern Recognition Letters*, vol. 33, no. 16, pp. 2173–2180, 2012.

[119] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Proc. CRV*, 2012, pp. 404–410.

[120] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, "Total recall: Automatic query expansion with a generative feature model for object retrieval," in *Proc. ICCV*, 2007, pp. 1–8.

[121] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proc. CVPR*, 2007, pp. 1–8.

[122] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *Proc. CVPR*, 2006, pp. 2161–2168.

[123] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[124] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit, "Boosting binary keypoint descriptors," in *Proc. CVPR*, 2013, pp. 2874–2881.

[125] M. Perd'och, O. Chum, and J. Matas, "Efficient representation of local geometry for large scale object retrieval," in *Proc. CVPR*, 2009, pp. 9–16.

[126] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. ECCV*. Springer, 2008, pp. 304–317.

[127] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in hamming space with multi-index hashing," *CoRR*, vol. abs/1307.2982v3, 2014.

[128] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *Proc. CVPR*, 2008, pp. 1–8.

[129] A. Joly and O. Buisson, "Random maximum margin hashing," in *Proc. CVPR*, 2011, pp. 873–880.

[130] W. Zhou, Y. Lu, H. Li, and Q. Tian, "Scalar quantization for large scale image search," in *Proc. ACM MM*, 2012, pp. 169–178.

[131] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. CVPR*, 2012, pp. 2957–2964.

[132] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proc. CVPR*, 2013, pp. 2938–2945.

[133] W. Hartmann, M. Havlena, and K. Schindler, "Predicting matchability," in *Proc. CVPR*, 2014.

[134] K. Hajebi and H. Zhang, "Stopping rules for bag-of-words image search and its application in appearance-based localization," *arXiv preprint arXiv:1312.7414*, 2013.

[135] S. Lee, K. Kim, J.-Y. Kim, M. Kim, and H.-J. Yoo, "Familiarity based unified visual attention model for fast and robust object recognition," *Pattern Recognition*, vol. 43, no. 3, pp. 1116–1128, 2010.

[136] M. Brown, R. Szeliski, and S. Winder, "Multi-image matching using multi-scale oriented patches," in *Proc. CVPR*, vol. 1, 2005, pp. 510–517.

[137] T. Sattler, B. Leibe, and L. Kobbelt, "Improving image-based localization by active correspondence search," 2012, pp. 752–765.

[138] ——, "Fast image-based localization using direct 2d-to-3d matching," in *Proc. ICCV*, 2011, pp. 667–674.

[139] G. Tolias, Y. Kalantidis, Y. Avrithis, and S. Kollias, "Towards large-scale geometry indexing by feature selection," *Computer Vision and Image Understanding*, vol. 120, pp. 31–45, 2014.

[140] Y. Li, N. Snavely, and D. P. Huttenlocher, "Location recognition using prioritized feature matching," in *Proc. ECCV*, 2010, pp. 791–804.

[141] P. Turcot and D. G. Lowe, "Better matching with fewer features: The selection of useful features in large database recognition problems," in *Computer Vision Workshops, 2009*, 2009, pp. 2109–2116.

[142] G. Schindler, M. Brown, and R. Szeliski, "City-scale location recognition," in *Proc. CVPR*, 2007, pp. 1–7.

[143] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proc. CVPR*, 2008, pp. 1–8.

[144] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." in *VISAPP (1)*, 2009, pp. 331–340.

[145] M. Patella and P. Ciaccia, "Approximate similarity search: A multi-faceted problem," *J. Discrete Algorithms*, vol. 7, no. 1, pp. 36–48, 2009.

[146] L.-f. Ai, J.-q. Yu, Y.-f. He, and T. Guan, "High-dimensional indexing technologies for large scale content-based image retrieval: a review," *Journal of Zhejiang University SCIENCE C*, vol. 14, no. 7, pp. 505–520, 2013.

[147] D. Qin, C. Wengert, and L. J. V. Gool, "Query adaptive similarity for large scale object retrieval," in *Proc. CVPR*, 2013, pp. 1610–1617.

[148] C. Schmid and R. Mohr, "Combining greyvalue invariants with local constraints for object recognition," in *Proc. CVPR*, 1996, pp. 872–877.

[149] I.-K. Jung and S. Lacroix, "A robust interest points matching algorithm," in *Proc. ICCV*, vol. 2, 2001, pp. 538–543.

[150] F. Schaffalitzky and A. Zisserman, "Multi-view matching for unordered image sets, or "how do i organize my holiday snaps?"," in *Proc. ECCV*. Springer, 2002, pp. 414–431.

[151] V. Ferrari, T. Tuytelaars, and L. Van Gool, "Simultaneous object recognition and segmentation by image exploration," in *Proc. ECCV*. Springer, 2004, pp. 40–54.

[152] X. Guo and X. Cao, "Good match exploration using triangle constraint," *Pattern Recognition Letters*, vol. 33, no. 7, pp. 872–881, 2012.

[153] C. Cui and K. N. Ngan, "Global propagation of affine invariant features for robust matching," *IEEE TIP*, vol. 22, no. 7, pp. 2876–2888, 2013.

[154] Z. Wu, Q. Ke, M. Isard, and J. Sun, "Bundling features for large scale partial-duplicate web image search," in *Proc. CVPR*, 2009, pp. 25–32.

[155] O. Chum, M. Perdoch, and J. Matas, "Geometric min-hashing: Finding a (thick) needle in a haystack," in *Proc. CVPR*, 2009, pp. 17–24.

[156] H. Jégou, M. Douze, and C. Schmid, "On the burstiness of visual elements," in *Proc. CVPR*, 2009, pp. 1169–1176.

[157] R. Arandjelovic and A. Zisserman, "Three things everyone should know to improve object retrieval," in *Proc. CVPR*, 2012, pp. 2911–2918.

[158] X. Zhang, J. Qin, W. Wang, Y. Sun, and J. Lu, "Hmsearch: an efficient hamming distance query processing algorithm," in *Proc. SSDBM*, 2013, pp. 1–12.

[159] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.

[160] P. S. Bradley, O. L. Mangasarian, and W. N. Street, "Clustering via concave minimization," in *NIPS*, 1996, pp. 368–374.

[161] Q. Luo, S. Zhang, T. Huang, W. Gao, and Q. Tian, "Scalable mobile search with binary phrase," in *Proc. ICIMCS*, 2013, pp. 66–70.

[162] E. Achtert, H.-P. Kriegel, E. Schubert, and A. Zimek, "Interactive data mining with 3d-parallel-coordinate-trees," in *Proc. SIGMOD*, 2013, pp. 1009–1012.

Acknowledgements

This work would not have been possible without the scientific and personal support of my colleagues, friends and family. I would like to use this opportunity to express my unlimited gratitude to all of you who have been there for me during the last years.

First and foremost I would like to thank my thesis supervisor, PD Peer Kröger, for his great support during my whole time as a research assistant. I would especially like to thank him for giving me the opportunity and freedom to work on the topics I always wanted to work on, and for providing me the funding to do so – including one of the best industry projects that I could ever imagine. Peer Kröger also deserves my gratitude for his scientific input on the publications we worked on.

Second I am indebted to Prof. Michael Gertz who agreed to serve as a second reviewer of my thesis, which is always tied to a huge amount of work.

As a leader of the Database Systems Group I was working in, Prof. Hans-Peter Kriegel deserves special thanks for providing the positive and team-oriented working atmosphere that made the work in his group so enjoyable.

My sincere thanks also go to my colleagues who always supported me in my scientific research, be it as coauthors of the papers we published, or simply as partners for scientific discussions, providing valuable inputs for this thesis. During my work at this group I learned to appreciate the good working atmosphere of our group that helped me considerably during the completion of my thesis. I would especially like to thank Tobias Emrich, Matthias Renz and Andreas Züfle who coauthored a variety of papers included in this thesis, always having interesting ideas for new research projects. I would also like to thank Erich Schubert for his support with the Elki framework, Markus Mauder and PD Matthias Schubert for the valuable discussions and all of my colleagues for providing such a great working environment.

Many thanks also go to Susanne Grienberger who was always helping me to cope with the bureaucratic matters of university life, and to Franz Krojer who always provided us with a system infrastructure greatly supporting our work. I would especially like to thank him for always having an eye on the

hardware market, making it possible to test new hardware.

Finally I would like to thank my parents, my brother and my future wife Bianca for supporting me during the whole time of my doctoral research studies, in good times and in bad, and helping me not to give up when suffering a setback. Without their help this thesis would not have been possible. The same holds for my friends who have always supported me in my undertaking.