Faculty of Computer Science

Chair for Artificial Intelligence

CHEMNITZ UNIVERSITY
OF TECHNOLOGY

Diploma Thesis

Spatio – Temporal Data Mining with Bayesian Networks

Author: Thomas Liebig

Supervisors: Dipl. Winf. Christine Körner

Dr. Michael May

Prof. Dr. Werner Dilger

Date of Submission: 01.08.2007

Thomas Liebig

Spatio – Temporal Data Mining with Bayesian Networks

Diploma Thesis, Chemnitz University of Technology, 2007

Abstract

Traffic routes through a street network are no random walks but contain dependencies. Such dependencies exist for instance within streets or between neighbouring street-segments. They can be exploited by applications such as mobile communications, traffic management and location based services.

This work tackles the task to model correlations between the locations in space-time trajectories. Due to the nature of spatial data, we have to address the following challenges.First, it is hard to obtain a reasonably sized data-set of trajectories. Therefore, our first task is to generate a valid data-set which respects real-world traffic characteristics. For this, we use the A*-routing algorithm and extend it to produce routes that correspond to real-world traffic intensities. Second, the spatial application has serious demands on the complexity and performance of the applied methods and algorithms. We apply Bayesian Networks to achieve a compact representation of the conditional dependencies. We examine algorithms for structure learning of Bayesian Networks for large data-sets, show the inference and sampling processes, and finally apply these models in practice.

Acknowledgements

First and foremost, I would like to thank my thesis advisor Christine Körner and my supervisor Dr. Michael May. I wish to acknowledge their support and guidance throughout the project. By, numerous scientific discussions, and many constructive comments they have greatly improved this work. This includes in particular, proof-reading the innumerable drafts of this thesis.

I owe a debt of gratitude to my colleagues at the Fraunhofer Institute IAIS for patiently answering all my dumb questions concerning the usage of MapInfo and ORACLE. Each day since I have been here, widened my knowledge in the features of spatial data and spatial data mining in general

Therefore, I would like to acknowledge Prof. Dr. Werner Dilger not only for equipping me with the basic knowledge concerning artificial intelligence, but rather for raising my interest in intelligent data analysis and autonomous systems through numerous colourful creative sputtering talks. He was a very precious person and I am glad for knowing him.

Contents

List of Figures

Chapter 1

Introduction

A trajectory of a person (i.e. a person's movement through geographic space within a certain period of time) is not a random walk through the city, but shows spatial dependencies between the passed locations. For example, consider the daily path of a commuter. Starting at home, it mainly passes a motorway and ends at the place of work. During the trip it is more likely for the commuter to stay on the motorway than to leave it and enter one of the villages along the motorway. For this reason the probabilities of co-occurrence between locations on the motorway are higher than between a location on the motorway and within a village.

This example shows that the different locations of a region occur not independently within one trajectory, but correlate. These dependencies are highly valuable for various applications, for example mobile communications, traffic management and location based services. The goal of this diploma thesis is to model spatial dependencies between the various locations within trajectories. In addition, we provide a sampling method that allows the ad hoc generation of trajectories respecting a given application context.

Given a huge number of trajectories (for example in form of GPS-logs), the conditional dependencies between two locations can be determined by simply counting pairwise co-occurrences within the data. This results in a square matrix with one row and column for each location. However, in order to obtain dependencies between more than two locations the matrix has to be extended into higher dimensions. This means to consider not only pairs of locations, but tuples of various sizes.

This approach raises two problems. First, the matrix soon becomes huge and hard to handle in practice. It contains many empty entries as the co-occurrence decreases with the number of locations. Second, in order to represent the dependencies of all locations in the matrix the input data has to contain each location in at least one trajectory.

The approach in our thesis to tackle these two problems is to use Bayesian Networks for a compact representation of conditional dependencies and to generate artificial routes to obtain the necessary amount of data.

Bayesian Networks are graphical models which represent a probability distribution of random variables. Variables are modelled by nodes, and edges denote the conditional probability between the connected variables. Being a generative model, Bayesian Networks can also be used to draw samples according to the represented probability distribution.

In particular, our task involves the following steps (see figure 1.1):

- automatic generation of a set of routes,
- adjustment of the route set to satisfy real-world traffic intensities,
- structure learning of a Bayesian Network that models the correlations of locations within routes,
- sample generation from the Bayesian Network.

We tailor our work for the estimation of reach in poster-networks. Within the Swiss Poster Research project [IAI07b] Fraunhofer IAIS develops a general method to evaluate the contacts and reach of poster-networks in Switzerland. The reach of a poster-network states the percentage of people who notice at least one poster within a given network in a specified period of time. A challenge of modelling poster-reach in areas with sparse data lies in the identification of correlated poster locations. The result of this work can be used to meet this challenge.

In ongoing confidential work, Fraunhofer IAIS utilizes spatial sampling algorithms. This

thesis contributes to the sampling algorithm to include correlations between locations. In addition, the sampling algorithm can be used for various mobility connected applications. For example, a provider of location based services can infer the most likely future path given the movement history of a person.

The thesis has the following structure. In chapter 2 we represent the spatial data sources and the demands given by the geographic context of the application. Chapter 3 introduces Bayesian Networks and provides an overview of state of the art algorithms for structure learning, inference and sampling. In addition it discusses the applicability of the given algorithms for large datasets. Chapter 4 contains the practical unit of this thesis, where we generate routes, ensure there conformity with the traffic intensity, apply the structure learning algorithm and evaluate the sampling results. We conclude the thesis in chapter 5 with a summary and future work.

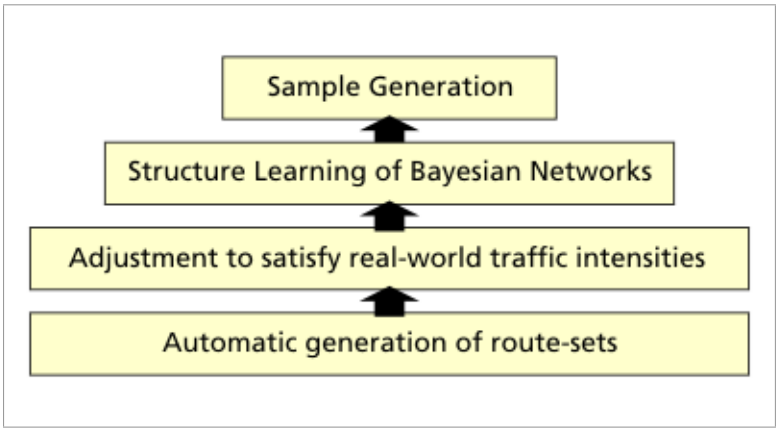


Figure 1.1: framework

Chapter 2

Data Sources and Requisitions

The following chapters utilize large spatio-temporal datasets that are hosted by databases. Usually, geographical information systems (GIS) are used to process this amount of data.

In this chapter we examine the common modalities of storing spatial data within a database in general (section 2.1), and describe the available data sources (street-network, trajectories and frequency map) in detail (section 2.2 to section 2.4).

Furthermore, as a result of the high quantity of data, we depict conditions attached to feasible solutions of our task (section 2.5).

2.1 Spatial Data Representation

Two distinct approaches are possible to store geometric information about spatial objects as those in figure 2.1 [Bar95] [BM98].

The first uses a tesselation grid (most often a regular rectangular grid, see figure 2.2) and stores rasterized information about the objects comparable with a digital image, and the second method describes the geometric boundary using vectors and compound vector-objects (figure 2.3).

Figure 2.1: spatial objects

Accumulating multiple objects in different layers, the rasterized data allows a much easier application of the spatial operators intersection and unification. But, to circumstantiate relative simple geometric objects (points, lines, polygons and their compounds), vectorized data needs less memory. It allows also an easy addition of further dimensions to the mostly two-dimensional table top projection of spatial objects. For raster models this would cause a transition from pixel- to voxel-space.
Vector models also provide easy mapping of data between various geographic coordinate systems with different range and precision.

In this thesis, the data sources exert the vector representation. Thereby, spatial objects are compounds of points, lines, areas and texts. This data structure, as well as advanced spatial features and methods, are implemented for (the object relational database management system) ORACLE by Oracle Spatial.

Therefore, one entity within a relation can store a spatial object. Additional data related to this object is saved within the same relation.

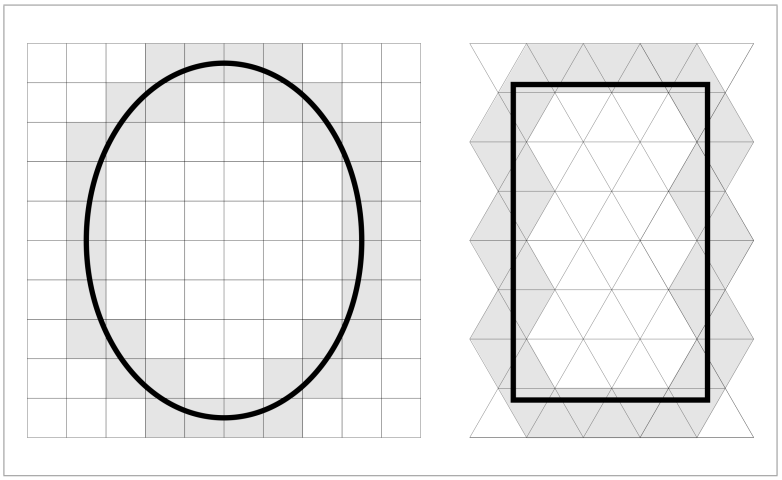


Figure 2.2: tesselated spatial objects

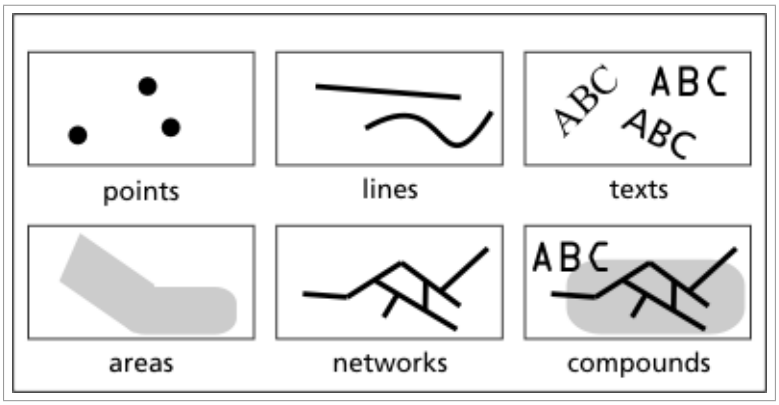


Figure 2.3: vectorized spatial objects [Bar95]

2.2 Street-Network

Dealing with streets in an arbitrary bounded spatial region, we are only interested in the streets location and connectivity relations rather than their appearance. This so called street-network is an preeminent data source for our task. It concedes the usage of algorithms and methods known by graph theory. Thereby, the real-world problem is transferred into the field of discrete mathematics and computer science.

In graph theory, the common method denoting a network structure $\mathcal{G}$ is to define a set of vertices $\mathcal{V}$ and a set of directed edges $\mathcal{E}$ between pairs of vertices ($\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$). Therewith, the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ provides a sufficient description of $\mathcal{G}$. Routing, search and other

network related algorithms based on this representation are well studied.

Mapping discrete spatial locations (i.e. streets or smaller street-segments) to the network vertices $\mathcal{V}$ and their connections to the set of edges $\mathcal{E}$, gives the structure of the street-network representation.

The NAVTAQ-atlas [dsG06], also used by many vehicle navigation systems, is a dataset of this type. Paths, streets and motorways are described by segments which are vectorized curves tagged with additional data. This includes for example the corresponding street-name, the type of street, topological relations of the segments or restrictions for several means of transport.

Hamburg (as one of the largest cities in Germany) consists of about 35'000 street-segments. Germany contains a total number of about 6 million segments.

2.3 Trajectories

Trajectories are paths of moving objects. As the introductory example already showed, a set of tracks contains correlations. Our goal is to model these correlations by a probabilistic model. For this reason, trajectories are our basic data source.

In physics, trajectories are described by functions $x(t)$ which map any time t to the current location x of an object.

In our case, the considered space is a finite set of discrete locations given by the vertices of the street-network $\mathcal{V}$. Each location x of a trajectory is an element of $\mathcal{V}$ ($\forall x : x \in \mathcal{V}$)

Furthermore, we only consider time discrete locations within a certain period $[t_{start}, t_{stop}]$.

In result, trajectories through the street-network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are defined by the following finite set of tuples:

$$\big\{(t, x(t)) | \forall t \in [t_{start}, t_{stop}], x(t) \in \mathcal{V}\big\}$$

The pairs of time and location are stored as an relation within the database.

2.4 Frequency Map

In order to ensure general applicability, the results from the probabilistic data analysis methods (which are presented in the next chapter) have to correspond with the real world. The only possible method to comply with this condition is a careful selection of representative data input.

We do not have sufficient real trajectories to guarantee this concordance. But, developed by previous Fraunhofer IAIS projects [IAI07a], we have a frequency map. This is a dataset which enriches the steet-network with real-world traffic statistics.

It denotes for any segment of the given NAVTEQ street-network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the traffic intensity in vehicles per hour. These values are the result of several counts and prudential reckonings which applies sustainable methods for knowledge discovery and data mining.

Mathematically, the frequency map $\mathcal{FM}$ can be formalised by a mapping from the set of locations $\mathcal{V}$ to the set of positive real numbers $\mathbb{R}^+$

$$\mathcal{FM} : \mathcal{V} \mapsto \mathbb{R}^+$$

Therewith, it is hosted by the database as an additional property of the street-segments.

2.5 Requirements

In the next chapters, we will discuss and improve algorithms to model and handle the inner-trajectory correlations in an arbitrary city. Therefore, the methods have to deal with small as well as large city sizes.

As we aim at a meaningful model for all street segments in the city, the given trajectory set

has to cover most of the contained street-segments at least once. Hence, we need a large set of trajectories with a cardinality that is at least as great as the number of street-segments.

It is not unlikely to obtain for a street-network of about 35'000 street-segments (in case of Hamburg) about 200'000 trajectories. The Bayesian Network structure learning algorithm and the sampling method have to cope with these problem dimensions.

Furthermore, we want to use the presented framework in practice. Thus, we are interested in methods with a reasonable runtime.

Chapter 3

Bayesian Networks

In this chapter we define Bayesian Networks (section 3.1), give an example (section 3.2) and explain the meaning of structure learning (section 3.3). Finally, we explain inference (section 3.4) and sampling (section 3.5) from a Bayesian Network. For all these steps, structure learning, inference and sampling, we give a few algorithms.

3.1 Definition

Before we define Bayesian Networks, we have to introduce the idea of conditional independence of two random variables:
Two random variables $\mathcal{A}$ and $\mathcal{B}$ are conditional independent with respect to a variable $\mathcal{C}$, if the property $p(\mathcal{A} \wedge \mathcal{B}|\mathcal{C}) = p(\mathcal{A}|\mathcal{C}) \cdot p(\mathcal{B}|\mathcal{C})$ holds [Pea88].
A Bayesian Network for a probability distribution $\mathcal{P}$ is defined by a tuple of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V})$ and a set Θ containing common probability tables (CPT). It is denoted by $(\mathcal{G}, \Theta)$ [Chi02].
The graph $\mathcal{G}$ has to be a directed acyclic graph (DAG) where the vertex set $\mathcal{V}$ contains all random variables. Descendants of one vertex x_i are all vertices that are reachable through a directed path, that is, following the direction of the edges. Nodes that are not descendants are called non-descendants $N(x_i)$ of x_i. For each vertex x_i, x_i and $N(x_i)$ have to be conditional independent given x_i's parents ($parents(x_i)$) [BH03].

We define Θ, the set of CPTs, such that for every vertex x_i there has to be exactly one CPT θ_i which denotes the probability of this variable depending on $parents(x_i)$ or, in case that it does not have any parents, by its marginal probability.

With a Bayesian Network representing a given distribution $\mathcal{P}$ we may compute the joint probability of the variables by the following product:

$$p(x_1, x_2, \ldots, x_n) = \prod_{\substack{i=1,\ldots,n \\ n=|\mathcal{V}|}} p(x_i|parents(x_i))$$

3.2 Watersprinkler Example

To show the advantages using a Baysian Network to represent the joint distribution of some random variables, we present a small network for the water sprinkler domain. In this domain, we observe four states: the weather is cloudy (C) it rains (R) a water sprinkler is on (S) and the grass is wet (W) the whole domain is visualized in figure 3.1.

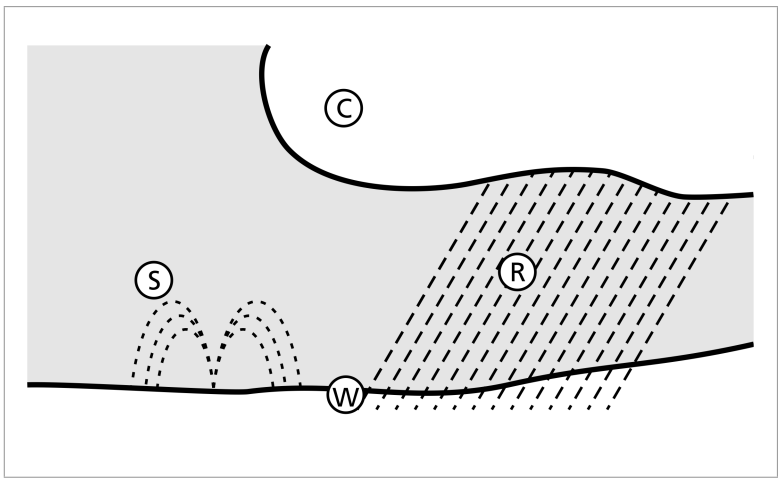


Figure 3.1: watersprinkler domain

The Bayesian Network which stores the conditional probabilities of the states is given by figure 3.2. In this example we only need to save $1 + 2 + 2 + 4 = 9$ probabilities instead of

$2^4 = 16$ without using a Bayesian Network.

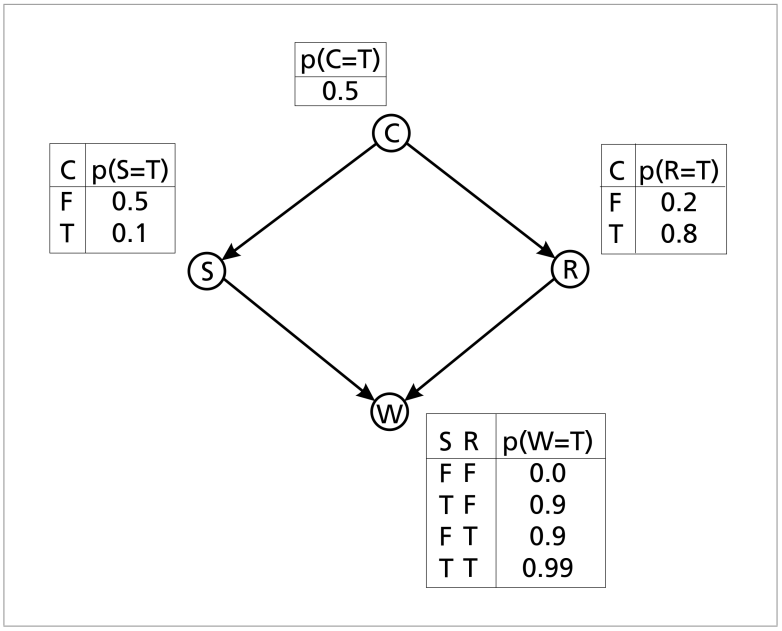


Figure 3.2: Bayesian Network for the watersprinkler domain

The joint distribution of the four variables $p(C, S, R, W)$ is given by:

$$p(C, S, R, W) = p(C) \cdot p(C|S) \cdot p(R|C, S) \cdot p(W|C, S, R)$$

When using the conditional independence relationship from the Bayesian Network this equation simplifies to:

$$p(C, S, R, W) = p(C) \cdot p(C|S) \cdot p(R|S) \cdot p(W|S, R)$$

Therefore, a Baysian Network is a compact representation of the common distribution of a set of random variables.
While we need 2^n values to store the joint probability table of n variables, we only need $O(n \cdot 2^k)$ values when using a Bayesian Network, where k is the maximum number of parent nodes for one vertex within the network [RN04].

3.3 Bayesian Structure Learning

In small domains, a Bayesian Network may be build by the help of a domain-expert who knows how the variables affect each other. He might model the vertex set and edge set manually or may at least support this process.
For our domain (routes through a network of street-segments) this seems to be hard or even impossible, therefore our main task is, to find a Bayesian Network (a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its parameters Θ) that represents the route matrix best. This problem is called structure learning problem. We already know the set of vertices $\mathcal{V}$ (As each vertex corresponds to exactly one street segment) of the Bayesian Network and just want to learn the edges and the corresponding CPTs.
So the Learning algorithm should search in the set $\mathcal{H}_n$ of all possible DAGs with a fixed number of vertices n (which equals the number of segments) for the BN that represents the data best.
The number $|\mathcal{H}_n|$ of possible network structures for n vertices increases super-exponential. Robinson calculated the number of possible DAGs for different numbers of vertices. For a number of 18 nodes there already exist about 1.5×10^{43} possible DAG structures [Rob77]. He also gave a recursive formula $f(n) = |\mathcal{H}_n|$ to compute the number of DAGs for n vertices that now is often quoted in the following form [CLDS99]:

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} \frac{n!}{(n-i)!i!} 2^{i(n-i)} f(n-i)$$

3.3.1 Scoring Bayesian Networks

To measure the quality of a Bayesian Network we have to define some scoring metrics. It should not only measure the ability to represent our original probability distribution but also penalize too complex structures (networks with high incoming-node-degree), as we want to be able to use the network later on.

3.3.1.1 Maximum Likelihood

The success of a model $\mathcal{M}$ to describe a distribution $\mathcal{P}$ is calculated by the maximum Loglikelihood estimate (MLL) [CLDS99] which is defined by the following equation [Stu06].

$$MLL(\mathcal{G}, \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{q(i,\mathcal{G})} \sum_{k=1}^{r(i)} d_{ijk} \cdot \ln \frac{d_{ijk}}{d_{ij}}$$

Where $\mathcal{G} \in \mathcal{H}_n$ is one DAG with n vertices, $\mathcal{D} \in DATA(n, d)$ and $0 \cdot \ln(0/*) \equiv 0$ is accepted. While d_{ij} is the number of all entries in database where the parent-configuration of node i equals to configuration $1 \leq j \leq q(i, G)$, d_{ijk} is the number of entries in database, where the parent-configuration of node i equals to configuration $1 \leq j \leq q(i, G))$ and the state of node i equals the state $1 \leq k \leq r(i)$. $q(i, \mathcal{G})$ is the number of parent configuration for a vertex x_i. As we deal with boolean variables, $q(i, \mathcal{G})$ equals $2^{|parents(x_i)|}$. $r(i)$ is the number of configurations for vertex x_i this is fixed to 2 when using boolean data.

Maximizing the MLL, the resulting Bayesian Network is not necessarily applicable, as this loglikelihood does not pay any attention to the possible complex structure. The resulting network might become too complex (the numbers of parent-nodes is too high) that it is hardly usable for inference, the Bayesian Infromation Criterion respects this kind of network complexity, and penalizes it.

3.3.1.2 Bayesian Information Criterion

A search that only maximizes the log-likelihood does not take the complexity of the Bayesian Network structure into account. Therefore, the resulting network would be hard to use for inference or sampling, because of the huge computational cost. So, another criterion for our search has to be the complexity and therefore the usability of the network. The Bayesian information Criterion (BIC) penalises complex structures by subtracting a measure for the complexity, the so called dimension of the network [Stu06]:

$$DIM(\mathcal{G}) = \sum_{i=1}^{n} q(i, \mathcal{G}) \cdot [r(i) - 1]$$

Here, $r(i)$ is again the number of configurations of a node i and $q(i, \mathcal{G})$ is the number of parent-configurations for node i.
The Bayesian Information Criterion (BIC) is given by this equation [Stu06]:

$$BIC(\mathcal{G}, \mathcal{D}) = MLL(\mathcal{G}, \mathcal{D}) - \frac{\ln d}{2} \cdot DIM(\mathcal{G})$$

BIC is not the only score for a Bayesian Network, there are also many others like AIC (Akaike Information Criterion) BDe (Bayesian Dirichlet equivalence [YSW+04]), BDeu (uniform Bayesian Dirichlet equivalence [SM05]), K2 and many others (see [YC02] [Chi96] for an overview of the most important scores).
But as BIC is claimed to be consistent [Nea04] and it is used by many applications (BNT [Mur], HUGIN [Hug], OpenBayes [Gai], Weka [oW]) we also focus on this score.

3.3.2 PDAGs and equivalence classes of Bayesian Networks

Two DAG's $\mathcal{G}$ and $\mathcal{G}'$ are called to be equivalent ($\mathcal{G} \sim \mathcal{G}'$), if for every Bayesian Network $\mathcal{B} = (\mathcal{G}, \Theta)$, there exists a Bayesian Network $\mathcal{B}' = (\mathcal{G}', \Theta')$ such that $\mathcal{B}$ and $\mathcal{B}'$ define the same probability distribution. [Chi02]. A quality criterion $g(BN : D)$ is called score equivalent if, for every $\mathcal{G}, \mathcal{G}' \in \mathcal{H}_n$ with $\mathcal{G} \sim \mathcal{G}'$ and each DATA(N,d) the equation $g(G_i : D) = g(G_j : D)$ holds. From a statistical point of view this requirement seems natural. Bayesian Information Criterion is score equivalent. [Stu06]

Given the equivalence of network structures, Chickering introduces a visual representation of the equivalence classes using partially directed acyclig graphs (PDAGs) [Chi02]. These are graphs that contain directed and undirected edges but no directed cycles. All edges belonging to a v-structure ($x \rightarrow z$, $y \rightarrow z$) [Pea88] within a DAG must also be directed in the corresponding PDAG. As there exist many possible PDAG structures for one equivalence class Chickering introduces the CPDAG (completed partially directed acyclic graphs) that

contain an undirected edge for each reversible edge in the equivalence class and a directed one for every other edge in a DAG. Two CPDAGs are equal if and only if they belong to the same equivalence class of DAG structures [Chi02]. An example of a CPDAG structure is given by figure 3.3.

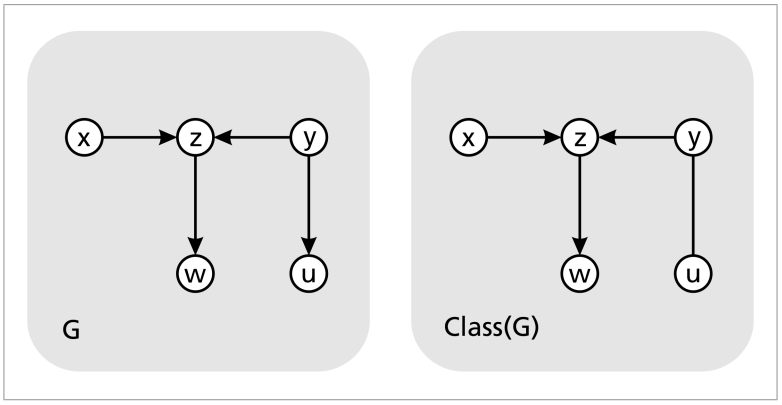


Figure 3.3: a DAG $\mathcal{G}$ and the completed PDAG for Class($\mathcal{G}$) [Chi02]

Therefore, the structure learning algorithm has to find the class of structures that maximize the score, but should not test multiple elements of one class for optimality.

3.3.3 Structure Learning Algorithms

We formalize the problem of learning the structure of a Bayesian Network $\mathcal{BN} = (\mathcal{G}, \Theta)$ for a known joint distribution $\mathcal{P}$ of n variables given by matrix $\mathcal{D}$, as the optimization problem, that is, to find the specific DAG $\mathcal{G}^*$ in the set of all possible DAGs with n vertices $\mathcal{H}_n$ that maximizes the score $g(\mathcal{G} : \mathcal{D}) \mapsto \mathbb{R}$.

$$\mathcal{G}^* = \arg\max_{\mathcal{G} \in \mathcal{H}_n} g(\mathcal{G} : \mathcal{D})$$

Learning a Bayesian Network $\mathcal{BN} = (\mathcal{G}, \Theta)$ is proven to be NP-hard for the score-metric BDe by Chickering [Chi96] and it is assumed also to be NP-hard for any other scoring function, although not yet proven.

Here, we give an overview of some structure learning algorithms: the Greedy Hill-Climbing Algorithm, the Sparse-Candidate Algorithm [FNP99] and the Screen-Based Network Search [GM04].

3.3.3.1 Greedy Hill-Climbing Algorithm

The first structure learning algorithm we present here, the Greedy Hill-Climbing Algorithm (algorithm 1), structures the set $\mathcal{H}_n$ of all possible DAGs with a neighbourhood relation by introducing a set of Operators $\{Op(\mathcal{G})|Op : \mathcal{H}_n \rightarrow \mathcal{H}_n\}$ that transform one element $\mathcal{G}$ of $\mathcal{H}_n$ into another: $Op(\mathcal{G})$.
Initially, the algorithm starts with an arbitrary DAG $\mathcal{G}^{(\tau=0)} \in \mathcal{H}_n$. Thus, applying all possible operators to this graph $\mathcal{G}^{(\tau)}$ generates the set of neighbour-structures $NB(\mathcal{G}^{(\tau)}) = \{\mathcal{G}'^{(\tau)}|\mathcal{G}'^{(\tau)} = Op(\mathcal{G}^{(\tau)}), \forall Op(\mathcal{G}^{(\tau)})\}$. The algorithm computes the score $g(\mathcal{G} : \mathcal{D})$ for all elements in this set and continues the search with $\mathcal{G}^{(\tau:=\tau+1)} := \arg\max_{\mathcal{G}\in NB(\mathcal{G}^{(\tau)})} g(\mathcal{G} : \mathcal{D})$. If the score did not improve ($g(\mathcal{G}^{(\tau+1)}) = g(\mathcal{G}^{(\tau)})$ the Hill-Climbing search stops and returns the current structure $\mathcal{G}^{(\tau)}$.
The operators the algorithm uses to transform the structures are: REVERSION, DELETION and INSERTION of edges within the DAG such that the DAG properties (to contain only acyclic, directed edges) are not violated.

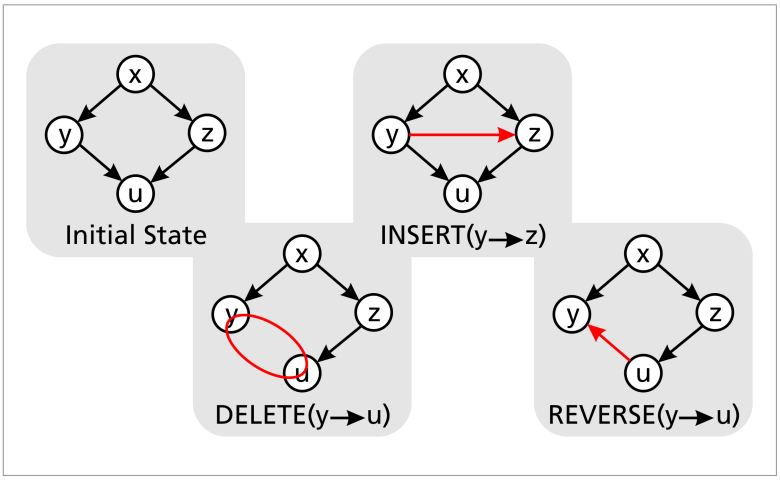


Figure 3.4: operators which the greedy Hill-Climbing Search uses

This operator set is complete, because each DAG $\mathcal{G}_1$ might be transferred into each other

DAG $\mathcal{G}_2$ by first removing all edges from $\mathcal{G}_1$ and then adding all edges contained by $\mathcal{G}_2$ into the network, and therefore it is proven that each DAG $\in \mathcal{H}_n$ can be transferred into each other.

Algorithm 1 Greedy Hill-Climbing Search

Input: $\mathcal{D}$, Data
$g(\cdot)$, a scoring metric for Bayesian Networks
Output: BN , Bayesian Network
INITIALISE($\tau \leftarrow 1$, $BN^{(\tau)} \leftarrow RANDOM$, $BN^{(0)} \leftarrow EMPTY$))
while $BN^{(\tau)} \neq BN^{(\tau-1)}$ **do**
$BN_{cand}^{(\tau+1)} \leftarrow BN^{(\tau)} \cup \{$all possible neighbours of $BN^{(\tau)}$ in $\mathcal{H}_n\}$
$BN^{(\tau+1)} \leftarrow \arg\max_{\forall BN \in BN_{cand}^{(\tau+1)}} g(BN)$
$\tau \leftarrow \tau + 1$
end while
return $BN^{(\tau)}$

Nevertheless, the algorithm only finds local maxima. To find a optimal structure it needs to be run very often with different initial networks.

Chickering [Chi02] gives a modification of this algorithm that searches in the space of partially directed acyclig graphs (see section 3.3.2 for an introduction to PDAGs). This algorithm avoids visiting multiple equivalent structures.
A further speed up is possible, when using a decomposable score. That is a score fulfilling the following equation: [SM05]

$$g(\mathcal{G} : \mathcal{D}) = \sum_{i=1}^{n} NODESCORE(x_i | parents(x_i))$$

Therewith, the total score of a DAG can be expressed as a sum of scores for each vertex, where each i-th summand depends only on the parents of the i-th vertex x_i within the DAG structure. The Bayesian Information Criterion is a decomposable score that follows directly by its definition:

$$
\begin{aligned}
BIC(\mathcal{G},\mathcal{D}) &= MLL(\mathcal{G},\mathcal{D}) - \frac{\ln d}{2} \cdot DIM(\mathcal{G}) \\
&= \left[\sum_{i=1}^{n} \sum_{j=1}^{q(i,\mathcal{G})} \sum_{k=1}^{r(i)} d_{ijk} \cdot \ln \frac{d_{ijk}}{d_{ij}} \right] - \left[\frac{\ln d}{2} \cdot \sum_{i=1}^{n} q(i,\mathcal{G}) \cdot [r(i)-1] \right] \\
&= \sum_{i=1}^{n} \left[\sum_{j=1}^{q(i,\mathcal{G})} \sum_{k=1}^{r(i)} d_{ijk} \cdot \ln \frac{d_{ijk}}{d_{ij}} - \frac{\ln d}{2} \cdot q(i,\mathcal{G}) \cdot [r(i)-1] \right] \\
&= \sum_{i=1}^{n} NODESCORE(x_i | parents(x_i))
\end{aligned}
$$

The advantage using a decomposable score is the possibility to calculate the score of a network after applying one of the operators given above by just recomputing the local score of the vertices affected by the transformation. And so, we do not need to compute the score for the whole DAG in each step.

An open source software using this hill-climbing algorithm is for example the Open Bayes Python package [Gai]. We used this program with test-distributions of different size and may confirm that this algorithm is only applicable for relative small numbers of random-variables [FNP99]. But, as we want to learn the Bayesian Network with up to 35'000 variables (that is the total number of segments in Hamburg), we have to use another algorithm.

As the Greedy Hill-Climbing algorithm evaluates all possible changes to the current network structure in one step (by adding, reversing and deleting edges) and chooses the modifier that improves the score of the network maximally, it has to evaluate $O(n^2)$ possible changes [FNP99]. The following algorithms will reduce this cost by bounding the search-space $\mathcal{H}_n$.

3.3.3.2 Sparse Candidate Algorithm

The Sparse Candidate Algorithm [FNP99] (algorithm 2) describes a class of algorithms that bounds the set $\mathcal{H}_n$ containing all possible DAG structures for n variables as it restricts,

within a first step, for each vertex the set of parent-candidates. And then, within a second step, finds the optimal network due to this constraints. Both steps are done within a loop which stops when the score of the network stays equal and the candidate set for each node also does not change.

While Greedy Hill-Climbing searches in $\mathcal{H}_n$ the set of all DAG structures with n vetices, most of the considered candidates may be omitted in advance, with the following heuristic. If x_i and x_j are almost independent, we will not prefer x_i to be a parent-node of x_j or vice versa. Friedman assumes that this heuristic is reasonable for most domains. The Sparse Candidate algorithm will limit the number of possible parents for one vertex, which is $n-1$ using the greedy hill-climbing algorithm, by a $k \ll n-1$ this will restrict the search-space, and reduce the cost for structure learning.

To decide whether a vertex x_j becomes chosen as a parent-candidate for a node x_i the relevance of each x_j with $j \neq i$ is measured by the pairwise mutual information $I(x_i, x_j)$. The stronger the dependence between the two variables, the higher will be the mutual information. And in case of independence $I(x_i, x_j)$ will be zero. Friedman gives a lot of heuristics increasing the speed of the algorithm, but the basic algorithm containing its main idea is shown in Algorithm 2.

$$I(X,Y) = \sum_{x,y} freq_{\mathcal{D}}(x,y) \cdot log \frac{freq_{\mathcal{D}}(x,y)}{freq_{\mathcal{D}}(x) \cdot freq_{\mathcal{D}}(y)}$$

$freq_{\mathcal{D}}(a)$ denotes the frequency of a in the dataset $\mathcal{D}$

3.3.3.3 Screen Based Network Search Algorithm

This Algorithm combines the idea of bounding the search space, as shown by the Sparse Candidate algorithm [FNP99], with the concept of dealing with sparse datasets described by Chickering and Heckerman [CH99]. It uses the sparseness of the data by generating frequent item sets, which is a common way to use sparseness of a dataset [CH99] and computes the support for those frequent item sets.

Algorithm 2 The Sparse Candidate Algorithm

Input: $\mathcal{D}$, Data
k , maximal number of parent nodes
$g(\cdot)$, a scoring metric for Bayesian Networks
Output: BN , Bayesian Network

```
INITIALISE (τ ← 0, BN^(τ) ← an initial Bayesian Network)
repeat
  τ ← τ + 1
  for each x_i; i ∈ 1, ..., n do
    SELECT(C_i^(τ)) with |C_i^(τ)| ≤ k based on (D, BN^(τ−1))
  end for
  BUILD(H_n = (V, E)|E = {(x_j, x_i)|i, j, x_j ∈ C_i^(τ)})
  BN^(tau) = (G^(τ), Θ^(τ)) ← arg max_{G^(tau) ⊂ H^(τ)} g(BN^(τ) : D)
until convergence
return BN^(τ)
```

The basic Screen Based Network Search algorithm (SBNS) is shown in algorithm 3. In a first step, the algorithm enumerates all frequent sets with a support higher than a given threshold and a length less than a maximum length. For each of these frequent sets a local Bayesian Network is generated. The edges of all of the local Bayesian Networks are stored in a so-called edgedump ED that counts the occurrences of each edge within the network structures. After this first step, the edgedump becomes sorted by descending count value. And in this order each edge of the edgedump is inserted into an initially empty global Bayesian Network. To respect the DAG properties the edges might be reversed, this follows from the PDAG equivalence classes of DAG structures (see section 3.3.2).

The algorithm uses the sparseness of the given data to represent the co-occurrence of variables (and thus positive correlation) within the data. As in sparse data positive correlations are stronger than negative the strongest correlations are not omitted [GM04]. To model the pairwise negative correlations, mutual edges are added to the final Bayesian Network structure. Such edges can be detected cheaply using a technique from [Mei99]. Meilă showed that the mutual information $I(x_i, x_j)$ between two variables x_i and x_j may be calculated very efficiently when dealing with discrete binary data. If the two variables never co-occurred in the dataset the equation simplifies to [Mei99]:

$$
\begin{aligned}
I(x_i, x_j) \;=\; & \frac{1}{d}\big[d \log d + (d - freq_{\mathcal{D}}(x_i) - freq_{\mathcal{D}}(x_j)) \log(d - freq_{\mathcal{D}}(x_i) - freq_{\mathcal{D}}(x_j)) \\
& -(d - freq_{\mathcal{D}}(x_i)) \log(d - freq_{\mathcal{D}}(x_i)) - (d - freq_{\mathcal{D}}(x_j)) \log(d - freq_{\mathcal{D}}(x_j))\big]
\end{aligned}
$$

$freq_{\mathcal{D}}(a)$ denotes the frequency of a in the dataset $\mathcal{D}$
d denotes the total number of entries in the dataset $\mathcal{D}$

The number of significant entities is reduced further by considering only those variables in the dataset with a support higher than a given threshold. This step is again statistical justified, because lower support means less mutual information as shown by Meilă.
In a last step, some greedy Hill-Climbing can be done in the resulting network structure to achieve a higher score or a less complex structure.

Using this algorithm, we realized problems (described in the next chapter) due to the frequent set enumeration, which is very memory and time consuming in case of large maximum frequent sets. As presented in the work of Goldenberg [GM05] those problems do not occur with a maximal frequent set length less than 25. In section 4.3 we describe our solution to this problem.

Algorithm 3 Screen Based Network Search – SBNS

Input: $\mathcal{D}$, Data
K , maximal frequent set size
s , support threshold
$g(\cdot)$, a scoring metric for Bayesian Networks
Output: BN , Bayesian Network

```
INITIALISE(DS ← EMPTY, Ed ← EMPTY)
for k = 2 to K do
  OBTAIN COUNTS for all FREQUENT-SETS FS(D)
  for each FS_i ∈ {FS | with |FS| = k and COUNT(FS) ≥ s} do
    DAG* ← arg max g(DAG : FS_i)
    if DAG* contains a vertex with k − 1 parents then
      STORE DAG* in DS
    end if
  end for
end for
for each DAG ∈ DS do
  STORE all edges{source, dest, count++} in Ed
end for
ORDER Ed in decreasing order of edge counts
for each edge e ∈ Ed do
  if e does not form a cycle in BN and e improves g(BN : D) then
    ADD e to BN
  end if
end for
return BN
```

3.4 Inference Algorithms

The main task for probabilistic inference systems is to compute the conditional probability distribution for a set of variables with a given evidence. This is called inference. We present here some algorithms to answer queries for the probability of an event X given some evidence e, namely requests of the form $p(X|e)$.

Pearl describes that it is possible to model the query and also constraints as a sub-network into the Bayesian Network [Pea88] as shown in figure 3.5, where we model the query $q = p(wet|rainy = TRUE \vee skrinkler = TRUE)$ into the previously introduced water-sprinkler network (see section 3.2) by the help of the auxiliary vertices $q' = (rainy = TRUE \vee skrinkler = TRUE)$ and $q = (q' \wedge wet)$. The constraint of our query, in this

case ($rainy = TRUE \vee skrinkler = TRUE$), is then represented by adding the evidence $q' = TRUE$ to the network.

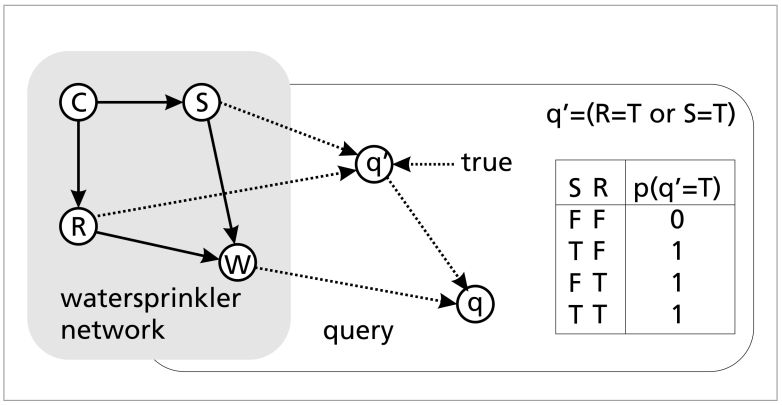


Figure 3.5: adding a query to a Bayesian Network

3.4.1 Enumeration Algorithm

From the Kolmogorov Axioms of probability follows that the probability of a proposition A equals to the sum of its atomic events e_i (as they are all disjunct by definition).

$$p(A) = \sum_{e_i \in e(A)} p(e_i)$$

This equation shows that it is also possible with a Bayesian Network to sum the probability of all atomic events of a proposition in order to get its probability $p(X|e)$.

$$p(X|e) = \alpha \cdot \sum_{y} p(X, e, y)$$

Where y enumerates all possible values of the unobserved variables Y, and α normalizes the probability afterwards. Reconsidering the watersprinkler domain from section 3.2 we now give the probability $p(W = w)$ for the state the grass is wet ($w = TRUE$) or not wet ($w = FALSE$) by the following sum of probabilites. Recalling, that each computation of $p(X, e, y)$ is performed by a product itself gives us the following formula.

$$
\begin{aligned}
p(W=w) &= \sum_c \sum_s \sum_r p(C=c, S=s, R=r, W=w) \\
&= \sum_c \sum_s \sum_r p(C=c)p(S=s|C=c)p(R=r|C=c)p(W=w|S=s, R=r)
\end{aligned}
$$

The complexity of such an computation for a network with n boolean variables is in $O(n \cdot 2^n)$ [RN04]. We reduce the cost when rewriting the term in the following form to $O(2^n)$

$$
p(W=w) = \sum_c p(C=c) \sum_s p(S=s|C=c) \sum_r p(R=r|C=c)p(W=w|S=s, R=r)
$$

This computation is generalized by the so called enumeration algorithm (algorithm 4), which also enumerates all atomic events of a proposition and sums their probabilities.

Algorithm 4 Enumeration Inference

Input: X , random variable
e , given evidence for a set of variables E
$\mathcal{BN}$, Bayesian Network for the variables $\{X\} \cup E \cup Y$
Output: $p(X|e)$
for each configuration x_i of X **do**
$e_i \leftarrow e \cup x_i$
$p(X = x_i|e) \leftarrow$ENUMERATE($VARS(\mathcal{BN})$,e_i)
end for
return NORMALIZE($p(X|e)$)

In the computation-tree (visualized by figure 3.6), we see that some partial sums are computed multiple times. The next algorithm (variable elimination) solves this problem.

Algorithm 5 ENUMERATE

Input: X , random variables
e , given evidence for a set of variables E
Output: a real number

```
if X = ∅ then
  return 1.0
end if
Y ← FIRST(X)
if Y is set to y by e then
  return p(y|parents(Y))· ENUMERATE(REMAINING(X),e)
else
  e_y ← e ∪ y
  return Σ_y p(y|parents(Y))· ENUMERATE(REMAINING(X),e_y)
end if
```

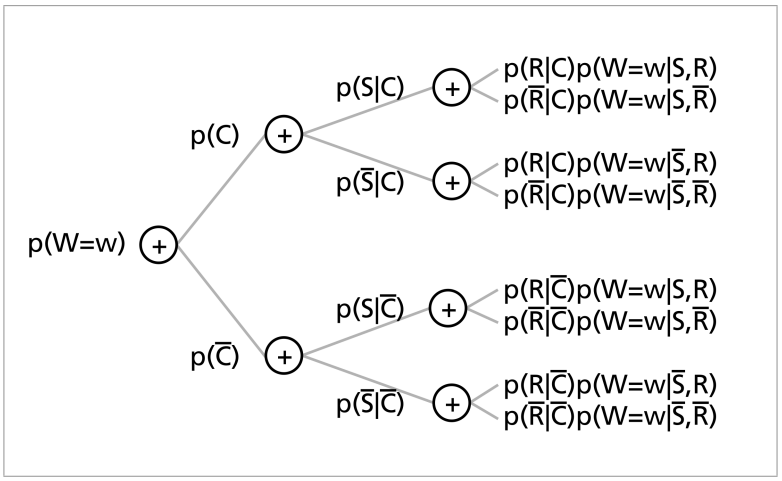


Figure 3.6: computation tree of the Enumeration Algorithm

3.4.2 Variable Elimination

While the enumeration algorithm computes partial probabilities many times, the variable elimination procedure keeps these values in memory (algorithm 6). Introducing some temporary variables T_i to keep partial results, the equation from the last section is factorized

to:

$$
\begin{aligned}
p(W=w) &= \sum_c p(C=c) \sum_s p(S=s|C=c) \sum_r p(R=r|C=c) p(W=w|S=s,R=r) \\
p(W=w) &= \sum_c p(C=c) \sum_s p(S=s|C=c) \underbrace{\sum_r p(R=r|C=c) p(W=w|S=s,R=r)}_{T_1(c,w,s)} \\
p(W=w) &= \sum_c p(C=c) \underbrace{\sum_s p(S=s|C=c) T_1(c,w,s)}_{T_2(c,w)} \\
p(W=w) &= \sum_c p(C=c) T_2(c,w)
\end{aligned}
$$

Temporary variables T_i are sums of the probability of some network variables over all possible configurations. In general, temporary variables are eliminated, when they are constantly 1. This occurs, if its network variables are unimportant for the query when they are no ancestors of evidence nor query variables.

Algorithm 6 VARIABLE ELIMINATION

Input: X , random variable
e , given evidence for a set of variables E
$\mathcal{BN}$, Bayesian Network for the variables $\{X\} \cup E \cup Y$
Output: $p(X|e)$

```
INITIALIZE(factors ← EMPTY)
for each var ∈ BN do
    factors ←MAKE-FACTOR(var,e)
    if var ∈ Y then
        factors ← SUM-OUT(var,factors)
    end if
end for
return NORMALIZE(PRODUCT(factors))
```

3.4.3 Complexity of Exact Inference

In general, exact inference in Bayesian Networks also contains boolean inference as a special case. It therefore has the same complexity as the Enumerate-Satisfiability-Problem

(#–SAT) of finding the number of distinct satisfying combinations for a boolean equation. As this problem is known to be $\mathcal{P}$#–complete [Val79], also exact inference is $\mathcal{P}$#–complete [BDP03]. But for singly connected networks (also called 'polytrees') the time and memory complexity of exact inference using the variable elimination algorithm is linear in the size of the network [RN04].
Due to the high complexity of exact inference, we give two algorithms that approximate the probability $p(X|e)$, in the next section. Namely, the Rejection algorithm (section 3.5.2) and a Monte-Carlo algorithm (section 3.5.3).

3.5 Sampling Algorithms

Beeing a generative model, we want to use a Bayesian Network BN to draw samples from the joint distribution $\mathcal{P}$ of n variables. As this distribution is encoded by the tuple $(\mathcal{G}, \Theta)$ we draw samples respecting the conditional probabilities represented by the network structure. The process is called sampling. In this section, we present the sampling algorithms: Ancestral Sampling, Rejection Sampling and Gibbs Sampling.

3.5.1 Ancestral Sampling

Given no evidence for any variables in the Bayesian Network, Ancestral Sampling generates a sample in the order of the ancestral relation given by the DAG. First, the Bayesian Network is sorted topologically (see figure 3.7). Then, the sampling processes the vertices in this order and samples their state either from there marginal probability distribution (in case of a source node) or according to the corresponding common probability table (CPT) (for any other vertex but the sources). This is done for all vertices within the network. The result is a state of the network that is consistent with the joint distribution and therefore a sample of this distribution. The algorithm is given as Algorithm 7 [RN04][Bis06].

Algorithm 7 ANCESTRAL-SAMPLE

Input: BN , Bayesian Network with n vertices
Output: x , a sample with n elements
  TOP-SORT(BN)
  for $j = 1$ to n **do**
    $x_i \leftarrow$ randomly sample from $p(X_i|parents(X_i))$
  end for
  return x

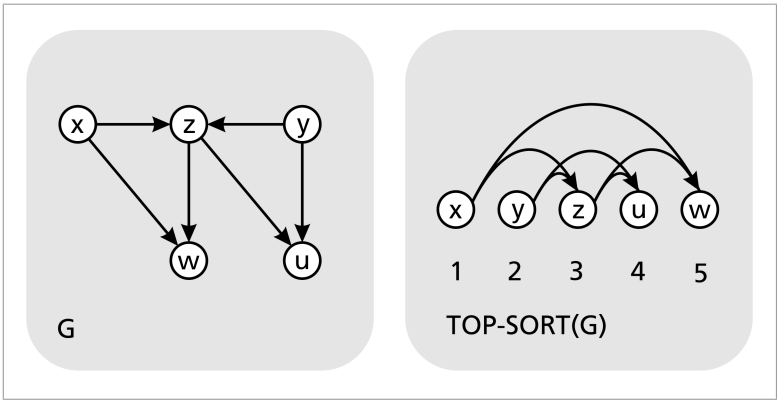


Figure 3.7: topologic ordering

3.5.2 Rejection Sampling

The basic idea of the Rejection Sampling algorithm is to use an easy to sample distribution for sampling from a more complex distribution. The algorithm draws samples from the easy distribution, and all samples that do not match the more complex distribution become rejected by the algorithm.

In the easiest case, it might be used to compute probabilities of the type $p(x|e)$. First the algorithm generates samples from the unconditioned distribution according to the distribution given by the Bayesian Network. And then it rejects all samples that do not match with the given evidence e. Afterwards, The probability for $p(X = x|e)$ will be estimated by counting the samples where $X = x$ in the set of all non rejected samples. The complete algorithm is given by Algorithm 8 [RN04].

Algorithm 8 REJECTION-SAMPLE

Input: X , the query
e , evidence
BN , Bayesian Network
N , Number of samples
Output: $p(X|e)$
for $j = 1$ to N **do**
$x \leftarrow$ ANCESTRAL-SAMPLE(BN)
if x is consistent with e **then**
$N[x] \leftarrow N[x] + 1$
end if
end for
return NORMALIZE($N[x]$)

3.5.3 Gibbs Sampling

Gibbs Sampling is a Markov Chain Monte Carlo (MCMC) algorithm. Considering the distribution $p(x) = p(x_1, x_2, \ldots, x_m)$ from which we wish to sample and an initial state for every variable, it chooses a x_i randomly or in a given order, in each step. For this x_i a value is drawn from the distribution $p(x_i|x_{\setminus i})$ where $x_{\setminus i}$ denotes $x_1, \ldots, x_m$ with x_i omitted. The complete algorithm is shown as Algorithm 9 [Bis06].

Algorithm 9 GIBBS-SAMPLE

Input: BN , Bayesian Network with n vertices
T , number of sampling-steps
Output: x , a sample with n elements
INITIALISE $x^{(1)} = (x_i : i = 1, \ldots, n)$
for $\tau = 1$ to T **do**
$x^{(\tau+1)} = x^{(\tau)}$
CHOOSE $(x_i; i \in 1, \ldots, n)$
SAMPLE $(x_i^{(\tau+1)} \sim p(x_i|\{x_1^{(\tau)}, x_2^{(\tau)}, \ldots, x_n^{(\tau)}\} \setminus x_i^{(\tau)}))$
end for
return $x^{(T)}$

The sampling of z_i from the joint probability distribution of all variables means in the case of a Bayesian Network to sample from z_i's Markov Blanket, as this blanket consists of the parent, children and co-parent vertices, it contains all nodes z_i may depend on (see figure 3.8).

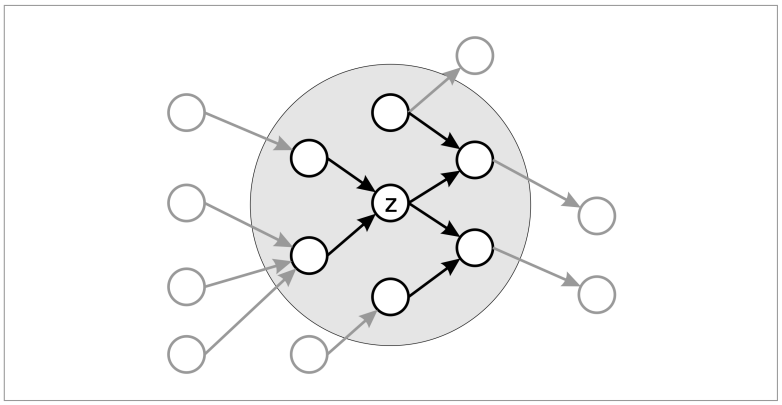


Figure 3.8: Markov Blanket of z

Chapter 4

From Trajectories to Correlations

In this chapter we describe the steps of the presented framework in detail, which are: First, the generation of artificial routes through the street-network and their adaption to real-world traffic intensities. Second, learning a Bayesian Network which represents the correlations within these trajectories. Finally, using this network as a generative model for drawing samples respecting the conditional probabilities.

In section 4.1, we present our routing algorithm that generates routes to learn the dataset with.

In section 4.2, we apply Bayesian Network structure learning algorithms to our spatio-temporal domain, as given in the previous chapter. Thereby, we examine and evaluate the applicability of these algorithms for the intermediate steps of our problem.

Afterwards (in section 4.3), we draw samples from the learned Bayesian Networks according to our framework. With these samples we evaluate the results of the performed structure learning algorithm.

Finally, we discuss the extension of the model in case of sparse real trajectories, which are provided as additional valuable data source (in section 4.4).

4.1 Data Generation

In this section we automatically create artificial routes. Thus, in a first approach, we utilize only the given NAVTEQ street-network $\mathcal{G}$ [dsG06]. In a second approach, we also use the frequency map $\mathcal{FM}$ (see section 2.4). The generated sets of routes are the data source for subsequent Bayesian Network structure learning.

The input for the previously described structure learning algorithms (section 3.3) is a joint probability distribution $\mathcal{P}$ given by a binary matrix. Therefore, the trajectories are described by a similar matrix $\mathcal{D}$ as following. In the dataset $\mathcal{D}$, routes are represented as binary vectors of size n that equals the total number of street-segments in the focussed region of interest. Claiming a fixed order of the street-segments, each element in the vector corresponds to a single street-segment. The vector $x_j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ is defined by the characteristic function of the j-th route $route_j$.

$$\mathcal{D} = (x_{ji})_{j=1,\ldots,d}^{i=1,\ldots,n}$$

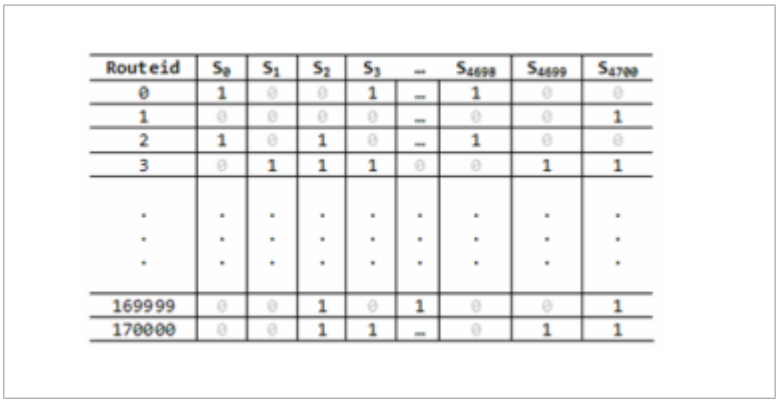
$$x_{ji} = \begin{cases} 1 & \text{, if } segment_i \in route_j \\ 0 & \text{, otherwise} \end{cases}$$

To get a distinct representation of a route, we have to claim each route to be acyclic. An example of such a dataset $\mathcal{D}$ shows figure 4.1.

The behaviour of the drivers and the necessary algorithms are given in the next sections.

4.1.1 Test Driver

As we generate the routes automatically, we need to define a consistent driver model. Therefore, we assume that in order to reach a goal-segment $segment_g$ from a start-segment $segment_s$ the test-driver chooses one acyclic path $path^*_{s \to g}(segment_s \to \ldots \to segment_g)$ through the street-network $\mathcal{G}$ that minimizes the required travel-time from $segment_s$ to $segment_g$.



Routeid	S_0	S_1	S_2	S_3	…	S_{4698}	S_{4699}	S_{4700}
0	1	0	0	1	…	1	0	0
1	0	0	0	0	…	0	0	1
2	1	0	1	0	…	1	0	0
3	0	1	1	1	0	0	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
169999	0	0	1	0	1	0	0	1
170000	0	0	1	1	…	0	1	1

Figure 4.1: binary path-matrix

$$path^*_{s\to g} \in \underset{\forall \text{ acyclic } path_{s\to g} \in \mathcal{G}}{\arg\min} time(path_{s\to g})$$

In comparison with real GPS-trajectories, this driver model is justified. But, it does not model inter-driver behaviour in times of high traffic. This is subject to game theory and psychology. We further assume by using this model that the driver creates a plan before starting a trip that he cannot change at any time.

The selection of start and goal segment is performed randomly using a uniform distribution of all segments within the given street-network.

4.1.2 The A*-Algorithm

To compute time-minimal paths from $segment_s$ to $segment_g$, we apply the A*-Algorithm (see algorithm 10 [RN04]). This algorithm evaluates each vertex, and therefore each segment, of a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a cost function $g(segment_i)$ that measures the time to reach this $segment_i$ from the start segment, and a cost estimate $h(segment_i)$ to reach the goal from the segment. The total cost estimate of a path from start to goal $f(segment_i)$ passing $segment_i$ then is given by the sum of those measurments:

$$f(segment_i) = g(segment_i) + h(segment_i)$$

In order to find the shortest path, it is reliable to start the search with those vertices where $f(segment_i)$ is minimal. If $h(segment_i)$ never overestimates the cost from $segment_i$ to the goal, the search algorithm finds the optimal path [RN04]. As the travel-time for one street-segment can be estimated by the segment-length divided by the speed-limit for this segment (which can be looked up according to the street-category stored in the databas), we define $h(segment_i)$ by the Euclidean distance between the centroids of $segment_i$ and $segment_g$ divided by the maximal speed-limit in the street-network.

$$\begin{aligned} t &= \frac{s}{v} \\ &\geq \frac{\|\overline{segment_i \rightarrow segment_j}\|}{\max v} \end{aligned}$$

To store the vertices with its current estimated distance, the algorithm is implemented using a priority queue Q that provides the functions: insertion of an entry with a certain key (ADD), remove the entry with the smallest key (REMOVEFIRST) and decrease the key of an entry (DECREASE-KEY). Very efficient implementations of such priority queues are possible with fibonacci heaps [CLRS01]. As we also log within each entry in this queue from which segment we gained the certain distance (key), we can reconstruct the full path (RECONSTRUCT-PATH) from source to target after reaching the target segment.

The A* algorithm, a so called best first search algorithm, finds an optimal solution that minimizes the required time from start to goal. Figure 4.2 shows an example of such a route.

Proof. When the algorithm halts, the cost of the path from $segment_s$ to $segment_g$ is given by

$$\begin{aligned} f(segment_g) &= g(segment_g) + \underbrace{h(segment_g)}_{0} \\ &= g(segment_g) \end{aligned}$$

For any remaining $segment_i$ in the priority queue Q the cost estimation is higher by

definition.

$$\begin{aligned} f(segment_i) &= g(segment_i) + h(segment_i) \, \forall segment_i \in Q \\ &> f(segment_g) \\ &> g(segment_g) \end{aligned}$$

As $h(segment_i)$ never overestimates the cost reaching the goal from $segment_i$, also $f(segment_i)$ never overestimates the cost of a path from $segment_s$ to $segment_g$ which passes $segment_i$. Thus, no possible solution is omitted and the search finds the shortest path. □

Algorithm 10 A*-Algorithm

Input: $segment_s$, source-segment
$segment_g$, goal-segment
$\mathcal{G}$, segment-network
$f(\cdot)$, estimated cost heuristic
Output: $path$, cost-minimal route from start to goal given as a list of segments

```
INITIALISE (Priority Queue Q = (id, key, parentid) ← EMPTY)
Q.ADD(segment_s, f(segment_s), NIL)
while Q ≠ ∅ do
  segment_test = Q.REMOVEFIRST()
  if segment_test = segment_g then
    path = Q.RECONSTRUCT-PATH(segment_test)
    return path
    halt
  end if
  for each segment_i ∈ SUCCESSOR(segment_test, G) do
    if (segment_i, *, *) ∉ Q then
      Q.ADD(segment_i, f(segment_i), segment_test)
    else
      Q.DECREASE-KEY(segment_i, f(segment_i), segment_test)
    end if
  end for
end while
return ERROR
```

4.1.3 Properties

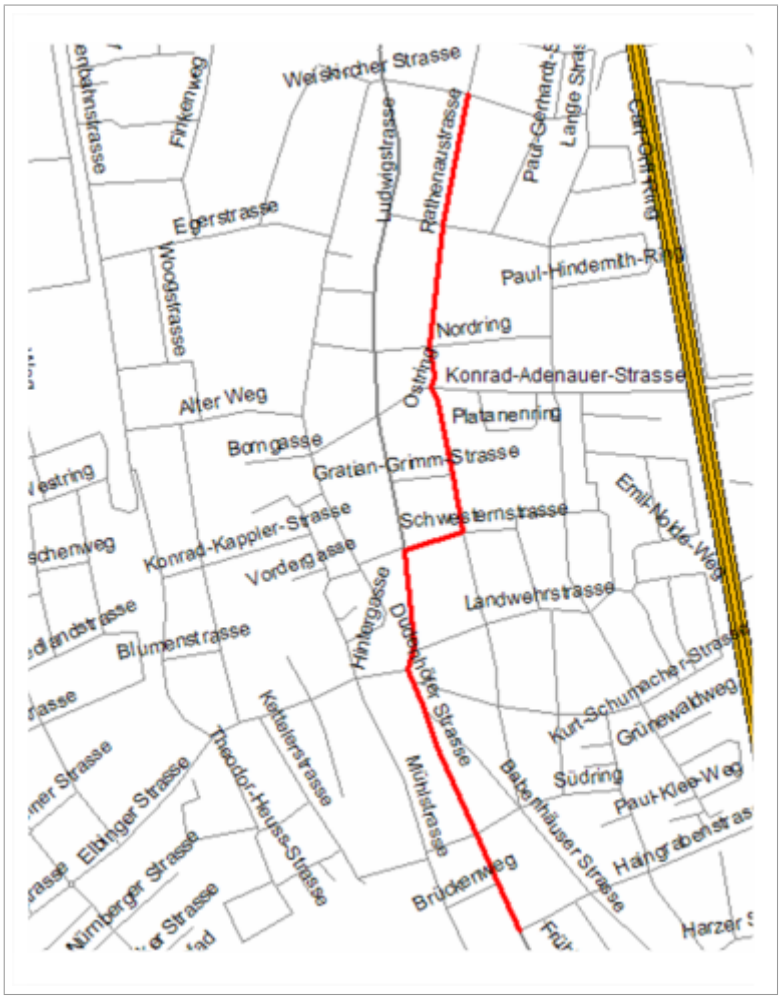


Figure 4.2: example of a shortest path

When generating many shortest paths with uniformly distributed start and goal segments, we can expect some locations to occur more frequent in the created dataset than others. To answer the question for there spatial distribution, we first give an easy example for a bounded region in $\mathbb{R}^2$ (figure 4.3). To have discrete locations, we work with a tessellation grid (in our case of the size 10 times 10). The shortest path between two points in $\mathbb{R}^2$ is (by definition) a straight line connecting them. When generating a large set of such routes (in this case 50'000), and counting the relative frequency of each location in the set of route afterwards, we see that locations in the centre of the bounded region have a higher probability to occur within a route than the locations at the boarder.

If we have an arbitrary segment-network, we expect also the locations at the boundary to

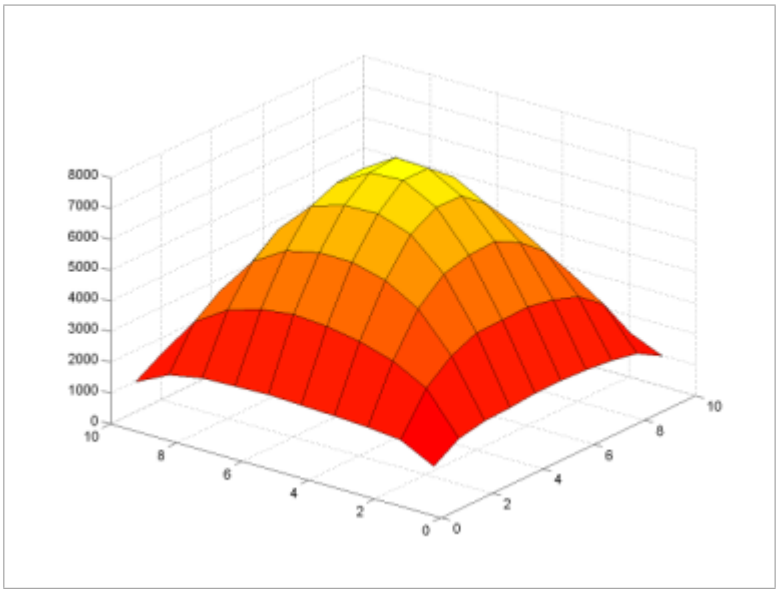


Figure 4.3: example in $\mathbb{R}^2$

be less frequent than central locations. The following figure (figure 4.4) shows the relative frequencies, when generating 50'000 random routes in the German city of 'Rodgau'.

Compared to the frequency map (figure 4.5), similarities in the centre are visible. The streets at the boarder and those that are mainly used by commuters (such as motorways) differ a lot in their frequency. For this reason the automatically generated route-set has to be generated such that it accomplishes the frequency map. The next section describes our method to cope with this problem.

4.1.4 Routing Algorithm

Using Bayesian Networks for real-world applications, we have to ensure that the represented probabilities are justified and correspond to real values.

For this reason, the automatically generated route-set (the one, we learn the networks with in a real-world task), has to be generated such that it accomplishes the frequency map $\mathcal{FM}$.

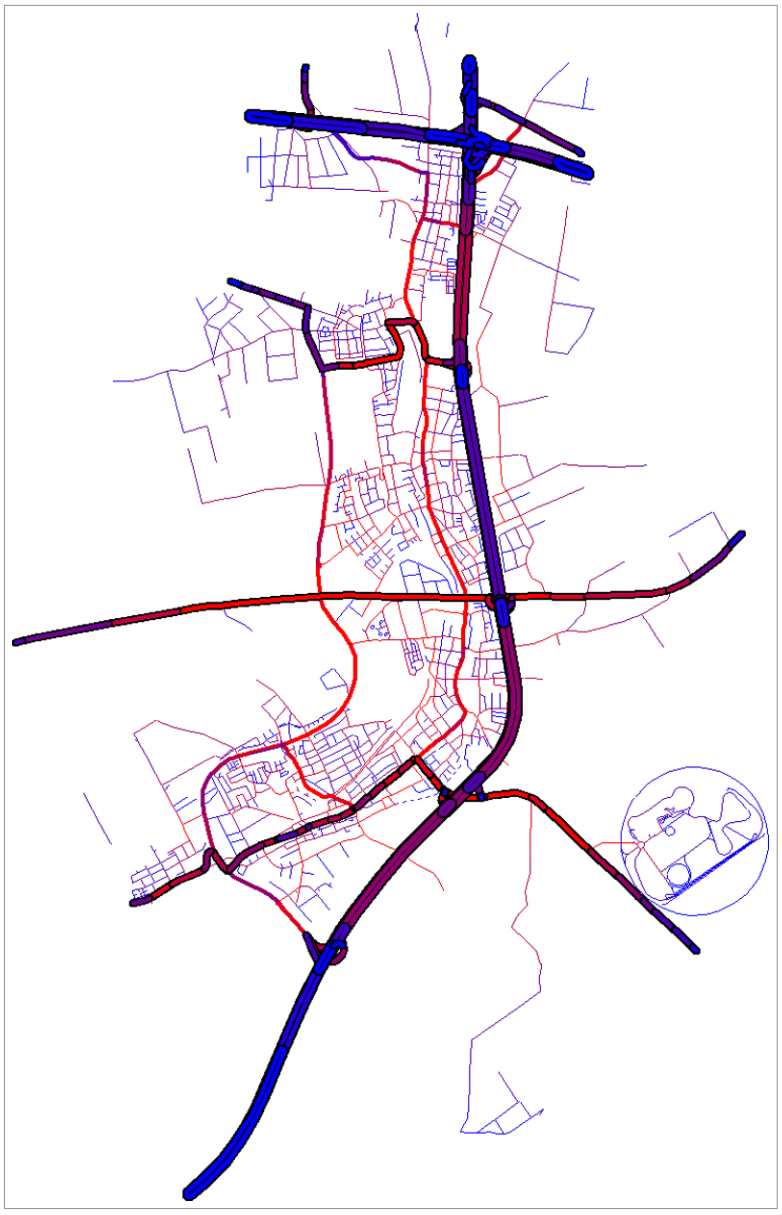


Figure 4.4: relative frequencies in the training-set

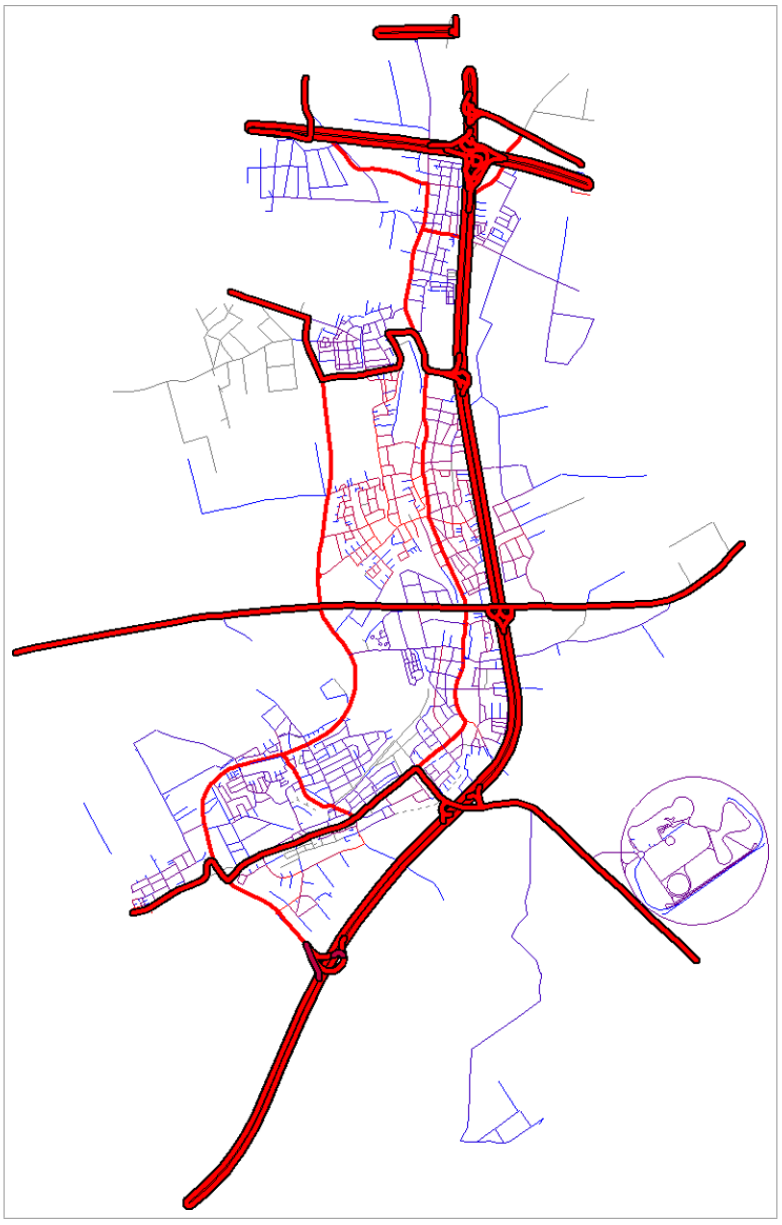


Figure 4.5: frequency map for Rodgau

Therefore, we developed a generative algorithm that creates a suitable set of acyclic routes (algorithm 11). It utilizes and extends the already described A*-algorithm.

Having a measure for the number of vehicles per segment within a certain period, by the frequency map, the algorithm generates random routes within a loop that stops when all segments are jammed. During the random route generation it removes already jammed street-segments from the segment-network and only considers the remaining segments as possible candidates for further time-minimizing routes.

This reflects the behaviour of traffic to choose alternative paths when the network-capacity is reached at certain edges and is similar to the Ford-Fulkerson algorithm that computes the maximum flow in a network from a source to a sink (compare [CLRS01]).

Algorithm 11 Route Generation

Input: $\mathcal{G}$, segment-network
$\mathcal{FM}$, frequency map
Output: $\mathcal{R}$, set of routes satisfying the frequency map

```
INITIALIZE(R ← EMPTY)
while G ≠ ∅ do
  SELECT(segment_s, segment_g ∈ G)
  R ← R ∪ A*(segment_s, segment_g, G, timeminimal)
  G ← G \ {segment_i | frequency(segment_i, R) = FM(segment_i))}
end while
return R
```

This algorithm returns a set $\mathcal{R}$ of routes satisfying the frequency map $\mathcal{FM}$.

Proof. Assume the case that $\mathcal{R}$ does not satisfy $\mathcal{FM}$. This means that it exists one segment $segment_i$, where frequencies differ in this two sets.

$$\exists\, segment_i \text{ s.t. } frequency(segment_i, mathcal R) \neq \mathcal{FM}(segment_i)$$

This raises the two cases:

$$\begin{aligned} frequency(segment_i, mathcal R) &< \mathcal{FM}(segment_i) \text{ and} \\ frequency(segment_i, mathcal R) &> \mathcal{FM}(segment_i)\ . \end{aligned}$$

As the generated trajectories are acyclic, the second case is in contradiction to the property that we delete the location in the considered network $\mathcal{G}$ in case of

$$frequency(segment_i, mathcal R) = \mathcal{FM}(segment_i) \ .$$

The first case is in contradiction to the stop criterion in the following way: if

$$frequency(segment_i, mathcal R) < \mathcal{FM}(segment_i)$$

the loop continues until

$$frequency(segment_i, mathcal R) = \mathcal{FM}(segment_i) \ .$$

Therefore the assumption '$\mathcal{R}$ does not satisfy $\mathcal{FM}$' leads in any case to a contradiction and the opposite property $\mathcal{R}$ satisfies $\mathcal{FM}$ must be true. □

As various source/goal distributions are possible, the algorithm is not the only generative method to create a route-set concurring with the frequency map.

4.2 Structure Learning

Now, that we have a dataset to learn the Bayesian Network from describing the routes, we apply two of the previously presented structure learning algorithms to obtain a Bayesian Network that represents the binary distribution given by the dataset best.

4.2.1 Greedy Hill Climbing

In section 3.3.3.1 we presented and discussed the Greedy Hill Climbing algorithm. Here, we evaluate its usability for the given structure learning task. Thus, we generate a set of 5'000

routes through the street-network of Rodgau which contains about 2'700 street-segments, and test the structure learning procedure by this dataset.

We already figured out that the Greeedy Hill Climbing algorithm has problems dealing with huge sets of random variables. But, because the computation of the score is the most time consuming part of the algorithm, also the length of the database is very important for the algorithm.

However, we hold on the previous constraints (see section 2.5) of our task. Particularly, we do not partition the focussed region into smaller units. It means in effect that we do not reduce the number of variables.

Anyhow, to perform a test of this structure learning algorithm we have to reduce the input data to a subset of our route-set. With more than 100 routes the time for computation endured already for longer than 48 hours (on an intel pentium 4 with 2 GHz and 2 GB RAM using OpenBayes [Gai], a free python package for working with Bayesian Networks).

Very unlikely, 100 routes are sufficient to represent the conditional dependencies of 2'700 street-segments. So, this algorithm is not suitable for our problem.

4.2.2 Screen Based Network Search

The advantage of the Screen Based Network Search (see section 3.3.3.3) is that it does not only restrict the search space but also benefits from the sparseness of the data. Our dataset is relatively sparse. In the previously described case of Rodgau, we have up to 120 out of 2'700 segments marked in one row.

Therefore, we evaluate in this section the usability of the algorithm for our domain. As in the previous section, our routes are a set of 5'000 randomly generated trajectories through the street-network of Rodgau.

First, we apply the algorithm with the recommended parameters [GM04] (frequent set length=4 and threshold=4). In this case, the cost for the frequent-set enumeration is very

high and we do not receive any resulting network structure, because it enumerates all possible frequent sets with the given parameters which are subsets of larger frequent sets.

Thus, we reduce the set of input routes to a number of 1'000. Furthermore, we apply different parameters (frequent set length=3 and threshold=6). With these settings, we obtain a Bayesian Network that hardly spans the city and contains only a few conditional dependencies.

Possible solution to this problem that enable us to use the SBNS algorithm in a spatio-temporal context to discover correlations of trajectories under the constraints given in section 2.5, are

- to bound the length of frequent sets that have a high support within the dataset, or
- to reduce the support of large frequent sets.

An approach for the first idea is to bound the length of the routes (and thus, of the possible frequent sets) in the input dataset by a certain distance. This is a simple way to obtain less marked entries per data-row in (D) and we achieve higher sparseness within the dataset. But, only spatially local dependencies can be represented by the network. And bottlenecks (which are regions of the street-network with a low cut, for example bridges) would complicate the structure learning. Also, the structure learning process would not be independent from the data generation step.

Hence, we use another method to bound the frequent set length. While bounding the length of the input-routes clears the ones in the input dataset in a certain order (the temporal order given by the trajectories, as described in section 2.3), we increase the sparseness by adding uniformly distributed zeros to the rows such that a maximal number of about 20 to 25 ones per row persist. Spatially, this heuristic adds uniformly distributed gaps to the routes.

We obtain the sparseness-value of about 25 ones per route from a comparison to Goldenbergs co-authorship network analysis [GM05], where the value density must be almost the same, and the algorithm works fine.

Utilizing such a method, the most significant co-occurrences (represented by frequent sets with a very high support) stay within the dataset and also long-distance relations can be found by the network search algorithm. For the learning process, we use the complete previously constituted dataset (consisting of 5000 routes), a frequent set support=4 and try different frequent set sizes.

As we are processing a sample of the given probability distribution, we recompute the common probability tables for the resulting network structure afterwards with the original distribution.

With this heuristic and these parameters a Bayesian Network for the routes through Rodgau can be computed respecting the requisitions in section 2.5. The resulting network spans the street-network with about 8'000 edges. In order to present a clearer visualization, we show in figure 4.6 an example of such a network that was learned with a reduced dataset (1'000 routes). The picture shows the meaning of strong dependencies. Thus, most conditional dependencies are along the main road.

The direction of the arrows do not have the semantic of travel-direction but the semantic of inference-direction. If we know something about the segment at the base of an arrow we can infer something about the segment at its other end.

Figure 4.6: Bayesian Network structure after increasing the sparseness

We performed also experiments with other cities. For example we dealt with Hamburg which consists of about 35'000 segments to prove that our algorithm can still handle this

amount of variables (see section 2.5). And for Brandenburg an der Havel, a city with about 4'200 segments where we have many GPS-tracks we varied the parameters (cardinality of the routeset and the support threshold). In figure 4.8 the coverage of Brandenburg by the different route-sets is examined. And in figure 4.7 the required resulting running times in seconds are plotted.

Therefore, we showed that SBNS can handle the problem of learning a Bayesian Network with the constraint from section 2.5 when using our heuristic to increase the sparseness of the given data.

About the parameters we can state that increasing the support threshold increases the speed of the algorithm but the network achieves lower score values afterwards.

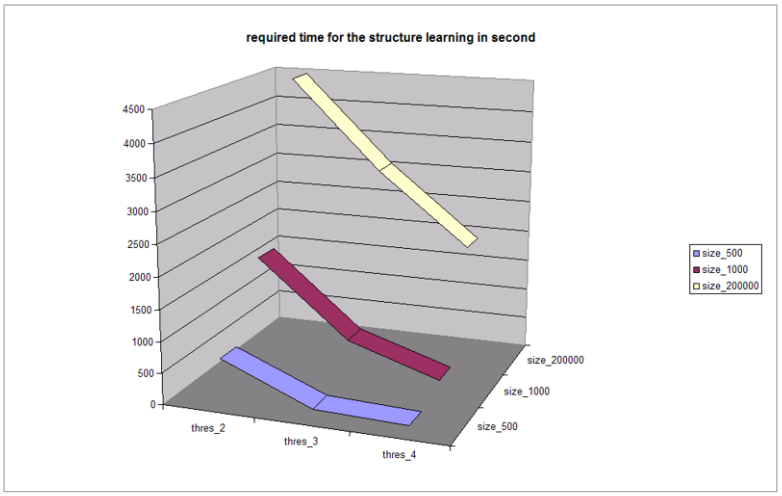


Figure 4.7: time requirement for different parameters

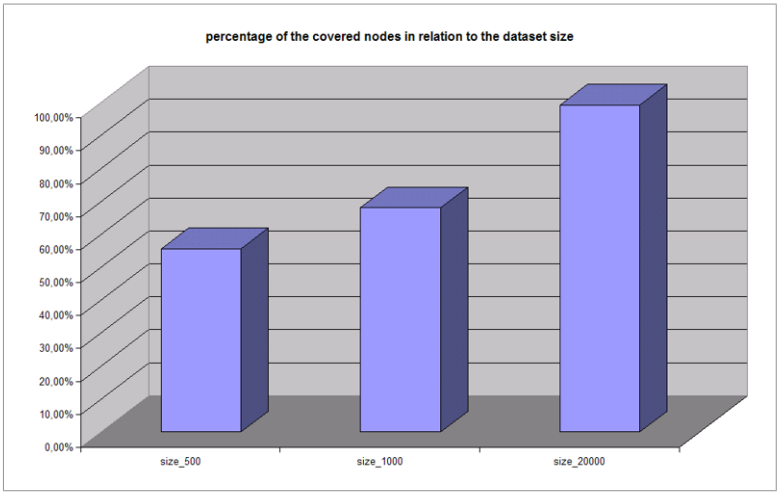


Figure 4.8: coverage of the street-network by different route-set sizes

4.3 Sampling and Evaluation

As the Bayesian Network will be used to draw samples from the distribution, we evaluate its quality also by drawing samples. We use the ancestral sampling algorithm (see section 3.5.1). Because the meaning of drawing spatial samples from a Bayesian Networks is not really intuitive, figure 4.9 gives an impression of this step and also visualizes the ancestral sampling procedure.

The samples will respect the conditional probability given by the Bayesian Network.

Proof. Ancestral sampling draws for each variable x_i a sample according to its probability $p(x_i|parents(x_i))$. Because the variables are previously topologically sorted and processed in this order, $parents(x_i)$ is always instantiated by a sample from the distribution $p(parents(x_i)|parents(parents(x_i)))$. Thus, a sample $(x_1, x_2, \ldots, x_n)$ is drawn from the following probability distribution :

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1,\ldots,n} p(x_i|parents(x_i))$$

The Bayesian Network describes the same distribution (see section 3.1). Hence, the samples respect the conditional probabilities given by the network structure. □

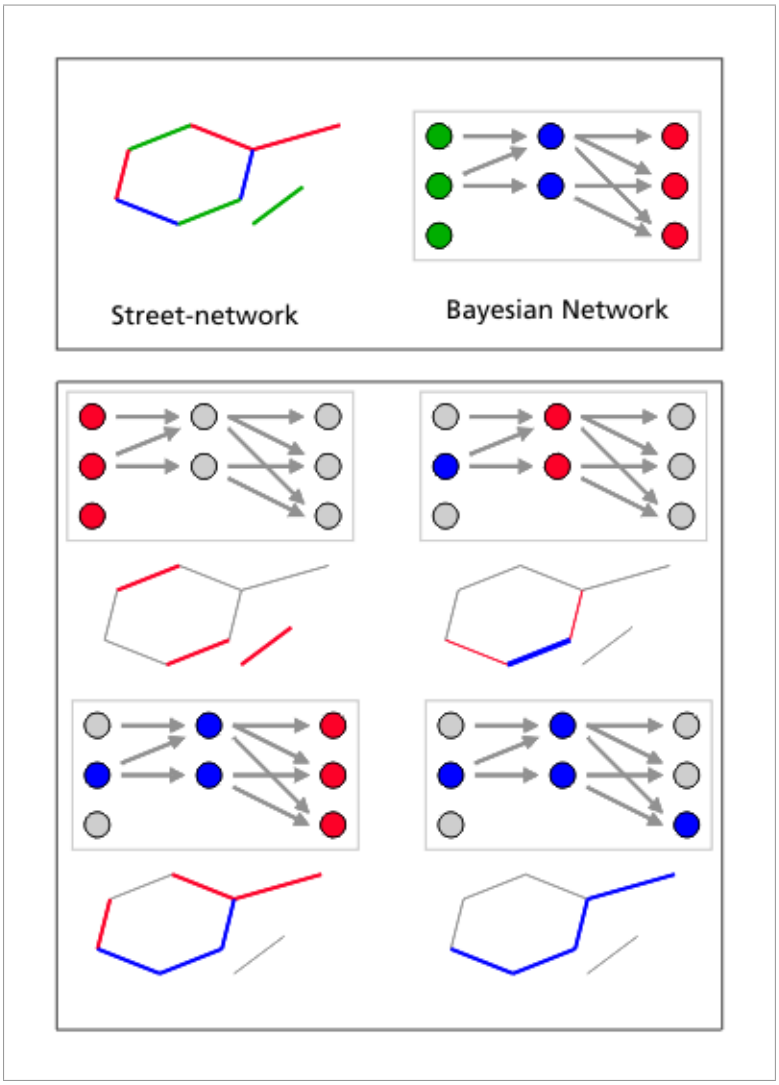


Figure 4.9: ancestral sampling from a Bayesian Network

Thus, when the Bayesian Network represents the distribution given by the original routes, the samples have the same properties as the original routes.

Therefore, we expect to receive connected acyclic paths from one segment to another with only small perturbances due to the probabilistic character of the algorithm (if the unlikely case occurs that a segment is never drawn, it will result in a gap in each sample drawn from the network). Furthermore, when sampling multiple times, we expect the relative frequency for each street segment to converge to the dataset.

This is investigated by the next figures, that show results of drawing 870 samples from the

network we have learned in the previous section using the SBNS with our heuristic. Figure 4.10 shows two samples and figure 4.11 visualizes the probability for each segment to be passed by a sample.

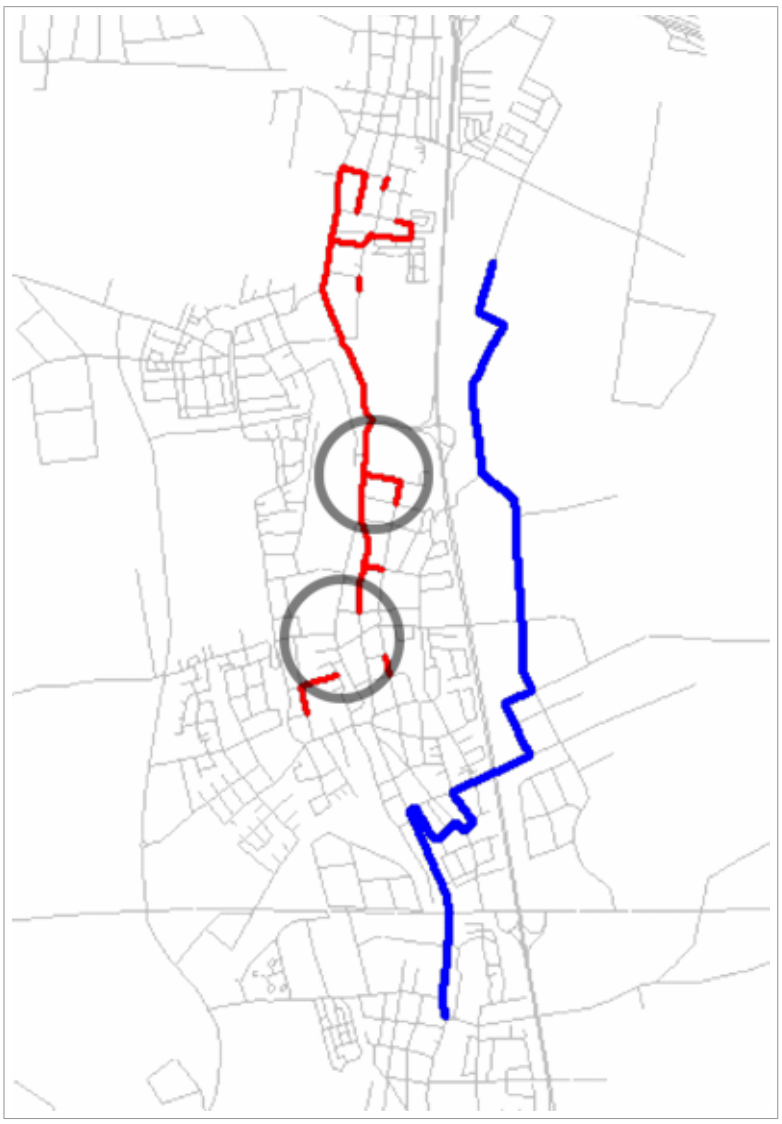


Figure 4.10: SBNS sample results after increasing the sparseness

The small perturbances visible in both figures result from a conceptual error in the sparseness increasing heuristic (described in the previous section). This heuristic prefers spatially local correlations, and the resulting input to the learning algorithm only contains most significant co-occurrences, as shown by the following figure 4.12.

Following the idea of Friedman [FNP99] that scoring functions of parent-candidates-sets

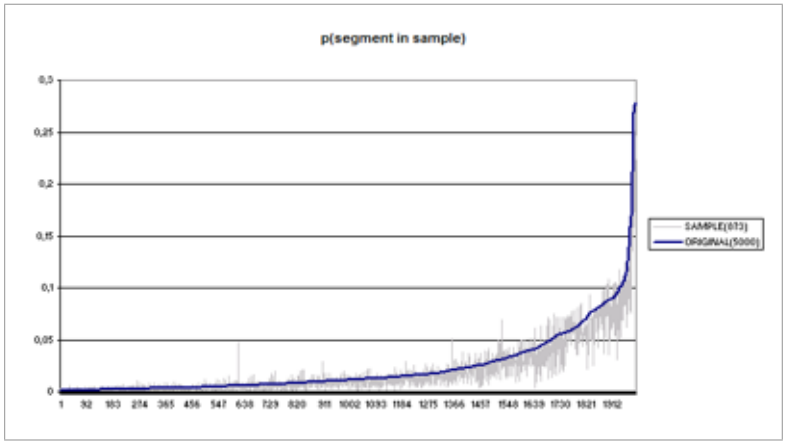


Figure 4.11: relative frequency per segment

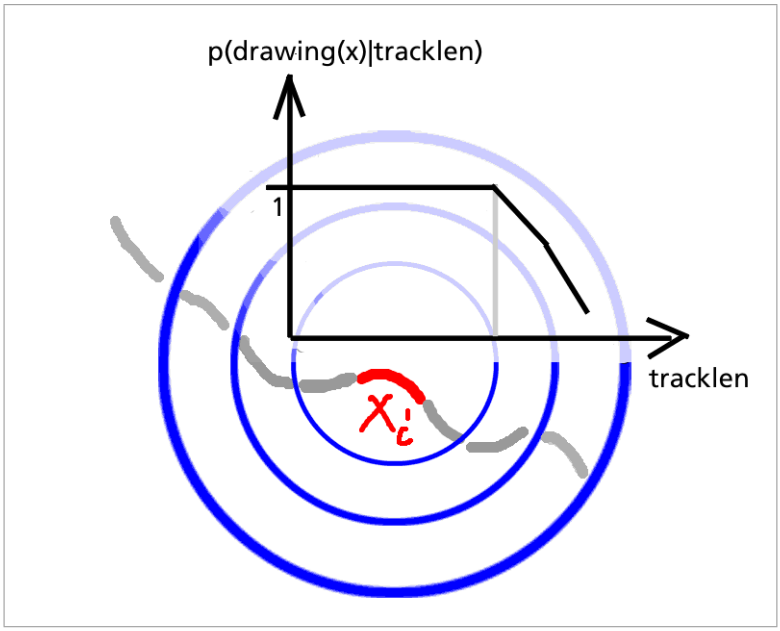


Figure 4.12: probability of co-occurrence decreases with increasing distance

which are bounded by the number of variables may not represent higher dimensional XOR relations, the bias for local dependencies in the dataset causes the network not to represent XOR relations of wide spread segments in general, when processing frequent sets. This means for the sampling that segment combinations might be sampled which could impossible occur within the same acyclic route (marked in figure 4.10).

That the correlation of the original data is not represented correctly and the network represents a slightly different probability distribution leads to the small errors of frequency at some segments within the network (see figure 4.11).

A solution to get rid of the bias is to reduce the number of ones per row within the dataset in a different way. Therefore, we suggest to represent each route by multiple rows. Enumerating all possible combinations of m segments within each route defines a set containing

$$\frac{length(route)!}{(length(route)-m)! \cdot m!}$$

elements, the new distribution can be sampled from. The idea, to represent a single route by an arbitrary selection of a fixed number of rows (for example exactly one, as above) containing only a reduced number of m segments of those many possible combinations, creates a bias for small routes.

But, using an algorithm that scales the support of each frequent set with a constant value removes the bias. This step is justified, as the most significant correlations (between less than m segments) are still in the dataset and are then learned and represented by the resulting Bayesian Network. The networks ability to describe spatial coherences will only be influenced by the value the support is scaled down to and the resulting maximal frequent set size as a boundary for the direct influence per segment.

A further speed up is possible when not enumerating all frequent sets but using only maximum frequent sets. They can be found by a method shown by Burdick [BCG01] or Lin and Kedem [LK02]. Then the edge-counter within the edgedump has to be increased in a different way then just counting the number of Bayesian Networks each edge was contained in. We suggest adding the lengths faculty of the corresponding maximal frequent set to the edgedump, as this is directly related (by the above relation) to the number of possible frequent sets within this maximal frequent set.

Applying both ideas, we get an improved Screen Based Network Search (algorithm 12), that first scales the sparseness to a certain level and then uses the exact knowledge of the sparseness by enumerating maximal frequent sets.

Algorithm 12 Modified Screen Based Network Search

Input: $\mathcal{D}$, Data
K , maximal frequent set size
s , support threshold
$g(\cdot)$, a scoring metric for Bayesian Networks
Output: $\mathcal{BN}$, Bayesian Network

```
INITIALISE(DS ← EMPTY, Ed ← EMPTY)
D' ← INCREASE-SPARSENESS(D,K)
for each MFS_i ∈ {MFS(D') | with SUPPORT(MFS) ≥ s} do
  DAG* ← arg max g(DAG : MFS_i)
  STORE DAG* in DS
end for
for each DAG ∈ DS do
  STORE all edges{source, dest, count+ = |DAG|!} in Ed
end for
ORDER Ed in decreasing order of edge counts
for each edge e ∈ Ed do
  if e does not form a cycle in BN and e improves g(BN : D) then
    ADD e to BN
  end if
end for
return BN
```

On the whole, the Screen Based Network Search can be applied in some street-networks of similar size as Rodgau, as presented in section 3.3.3.3 as the sparseness of the dataset differs for various street-network graphs. Using the heuristic given in section 4.2.2 enables the use of the SBNS-algorithm for any problem with the constraints presented in section 2.5 but also involves slight perturbances.

4.4 Extension

The previous section described the possibility to cope with the given spatio-temporal learning problem.

So far, the only data source that justifies the results is the frequency map. But, denoting only traffic intensities per segment, it does not define the distribution of sources and goals of the routes. Therefore, the smaller the considered region the more sustainable are the results (and vice versa).

When given sparse trajectories as an additional data source, they can be used to model the source/goal distribution as follows. We can partition the focussed region into multiple smaller zip-code regions. The real trajectories express the transition probabilities between those zip-code regions. Thus, we can learn a Bayesian Network which represents the probabilities of co-occurring zip-code areas within the real trajectories. Furthermore, we generate within each of these regions an artificial trajectory set that covers each location. Thereby, multiple Bayesian Networks can be learned for the zip-code areas representing the co-occurrences of the contained street-segments.

When drawing samples, we first draw zip-code areas and then continue drawing in the corresponding small Bayesian Networks.

Obviously, dependencies at the edge between two neighbouring areas are not modelled correctly. This can be avoided, when adding a random variable for each adjacent zip-code area to the small Bayesian Networks which represents the state that a trajectory passes this neighbour.

This bundle of Bayesian Networks then gives a compact representation of the co-occurrences within trajectories matching the frequency map and the source/goal distribution given by real trajectories.

Chapter 5

Conclusion and Future Work

In this chapter, we summarize the previous work and draw conclusions. Furthermore, we clarify the relation to applications and Fraunhofer IAIS projects. Finally, we give an outlook on open tasks and future work.

5.1 Summary

Within the previous chapters, we use Bayesian Networks to model the correlations contained by trajectories through a street-network.

Hence, we generate sets of routes which serve as input data for the learning process. Considering GPS-logs, these routes are reasonable claimed to minimize the required travel time for the trip. Thereby, we use the A*-algorithm to compute a set of trajectories that satisfies this condition.

To guarantee justified results without having enough real trajectories, we utilize the frequency map. We present a generative routing algorithm which ensures concordance between this additional data source and the resulting route set. Furthermore, we substantiate this by a proof.

Afterwards, we examine Bayesian Networks. This includes not only investigation and

comparison of state of the art algorithms that enable the use of graphical models for a huge number of variables, but also the development of heuristics to meet the requirements of the given spatio-temporal domain.

Evaluable information is extracted by drawing samples from the Bayesian Network. Therefore, we evaluate those samples and the intermediate steps.

As a part of this work, the presented algorithms were implemented.

Despite not yet evaluated, this thesis also explains a method to profit from sparse real trajectories in order to get more realistic results and to answer queries for conditional probabilities.

Literally, by the given knowledge that John Q. Public was within one track at the petrol station and at work, with our results, we can predict the probability of also passing the bakery.

5.2 Applicability

Modelling the conditional dependencies between locations, the hereby presented results are highly valuable for various applications (i.e. mobile communication, traffic management and location based services).

For example, consider the task to predict the reach of a network of advertisement posters. Reach states the percentage of people who notice at least one of the posters in a specified period of time. Within the Swiss Poster Research project Fraunhofer IAIS develops a general method to evaluate poster reaches.

The result of this work can be used to identify correlated poster locations.

Thus, we first generate a set of routes in the focussed region which tallies with the frequency map. Second, we learn the correlations between the tuples of the poster locations by Bayesian Networks. Finally, using inference algorithms, the Bayesian Network answers

queries for the conditional probability that a certain poster is passed by a given evidence of passing some other poster locations.

In case of real trajectories for the region we can even model realistic source/goal distributions of the routes by simply learning a Bayesian Network denoting the probabilities for zip-code area transitions of the given trajectories and multiple smaller Bayesian Networks for segments within the zip-code areas from automatically generated trajectories.

5.3 Prospect

We presented a routing algorithm, which generates routes according to the traffic intensities given by the frequency map. Taking this up, future work can investigate the possibilities of such generative routing algorithms to deal with constraints (i.e. a source/goal distribution) and to utilize additional data-sources. This may lead to a more realistic set of trajectories.

To enrich the prediction model by a real source/goal distribution, we described the possibility of learning different layers of Bayesian Networks: one for the zip-code area transitions (trained by real trajectories) and multiple smaller networks for the segment transitions within the zip-code areas (trained by generated routes). When dealing with this problem frequently, Hierarchical Bayesian Networks (HBN) [GF02] are a promising structure to describe the various conditional dependencies within one model. Hence, the spatial objects street-network, zip-code areas and the focussed region are part of the following taxonomy. The region contains the zip-code areas which contain the street-segments. Having taxonomic structured data, Hierarchical Bayesian Networks offer a method to build a probabilistic structure between arbitrary levels of this hierarchy. The complete model is a Bayesian Network and can be treated by all presented sampling and inference algorithms.

Furthermore, the processed problem to discover valuable knowledge in a set of trajectories can be understood to be a pattern recognition or sequence analysis problem.

The usual models for this class of problems are Hidden Markov Models (HMM) [RN04][Bis06], which extend Bayesian Networks due to the request of processing discrete time slices. When applied to our domain, these Hidden Markov Models store all transition probabilities be-

tween pairs of locations. Therefore, they are comparable to a square matrix which stores the pair-wise probabilities of co-occurrences. Hence, their drawback is also the extensive memory requirement and poor usability. A further disadvantage of this model is that it is only capable to describe data precisely which satisfies the Markov assumption [RN04]. Sequences holding this assumption satisfy the property that each contained value only depends on its previous one (They have the context length one.). In general, this Markov Assumption is unsatisfied by trajectories.

Nonetheless, the various researches concerning Hidden Markov Models raised new sophisticated models. Namely, the Variable Length Hidden Markov Model (VLHMM) [WZF+06]. It reduces the required memory by discarding weak dependencies and also enables variable lengths of context that are learned automatically. The application of this model will provide an online prediction model for locations contained by a trajectory.

Appendix A

Resulting Bayesian Networks

Bayesian Networks can generally be compared by their score and by visualization. As the route-sets vary in size, the scores of the networks are not comparable, and thus not denoted here. Therefore, this appendix visualises a selection of the resulting Bayesian Networks.

The meaningful information provided by the following images is the coverage of the street-network by the Bayesian Networks.

We show results for the work on Hamburg that prove that our approach can process large cities. Furthermore, we visualize the resulting Bayesian Networks for Rodgau, because they are mostly discussed in this work.

The utilized Parameters are denoted in the title of the images. For example, the following line: 'Rodgau - routes: 1'000 - maxfs: 20 - fss: 4 - sup: 4' describes:

The used route-set consists of 1'000 routes through Rodgau. The representing binary matrix has a maximal frequent set length (maxfs) of 20. The length of the considered frequent sets (fss) is set to 4. And the support (sup) is set to 4.

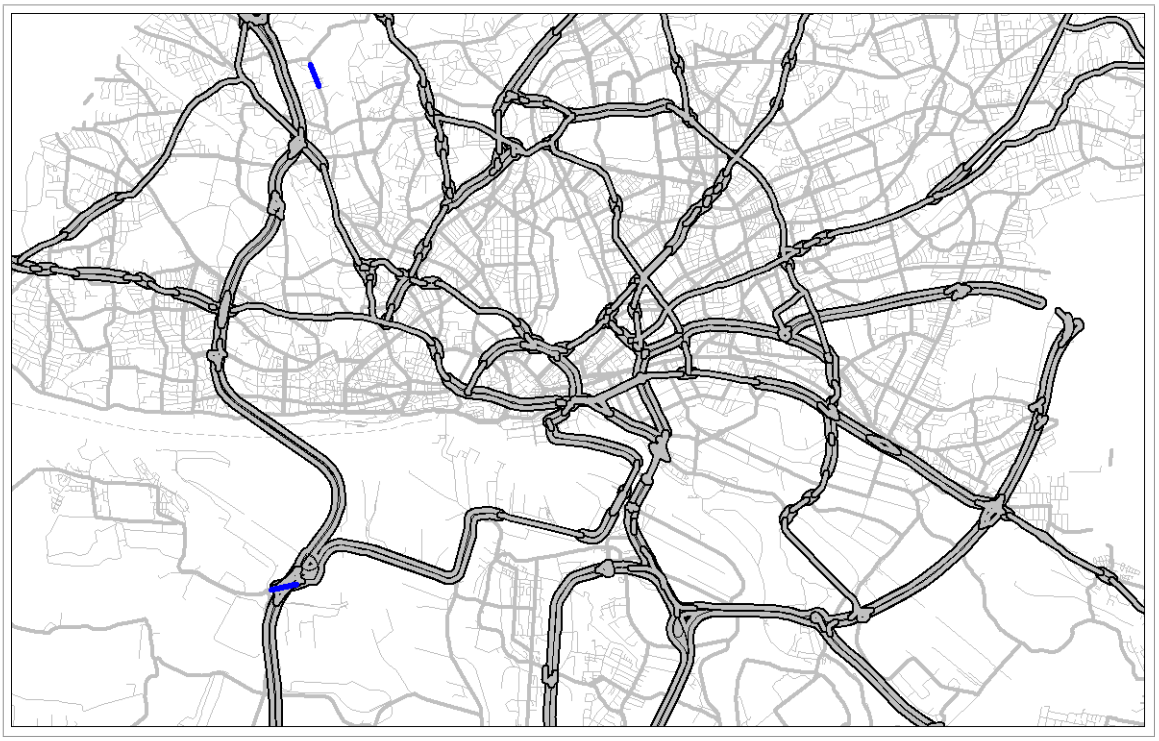


Figure A.1: Hamburg - routes: 600 - maxfs: 20 - fss: 4 - sup: 4

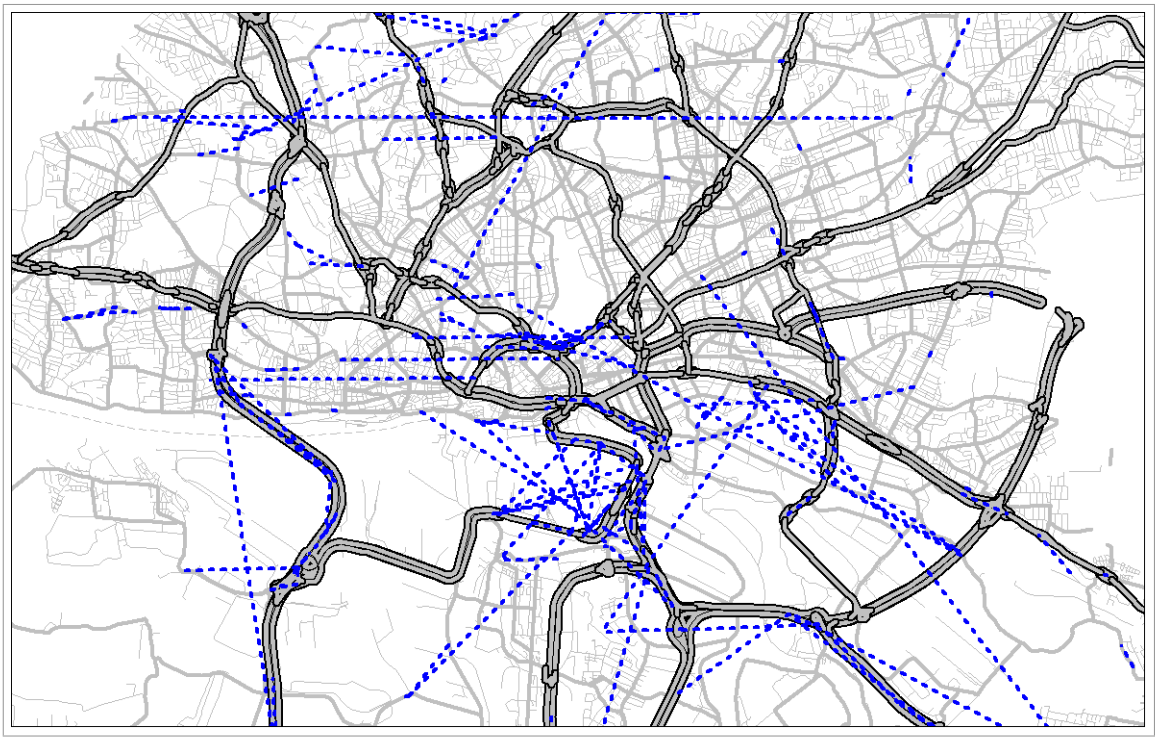


Figure A.2: Hamburg - routes: 600 - maxfs: 20 - fss: 4 - sup: 3

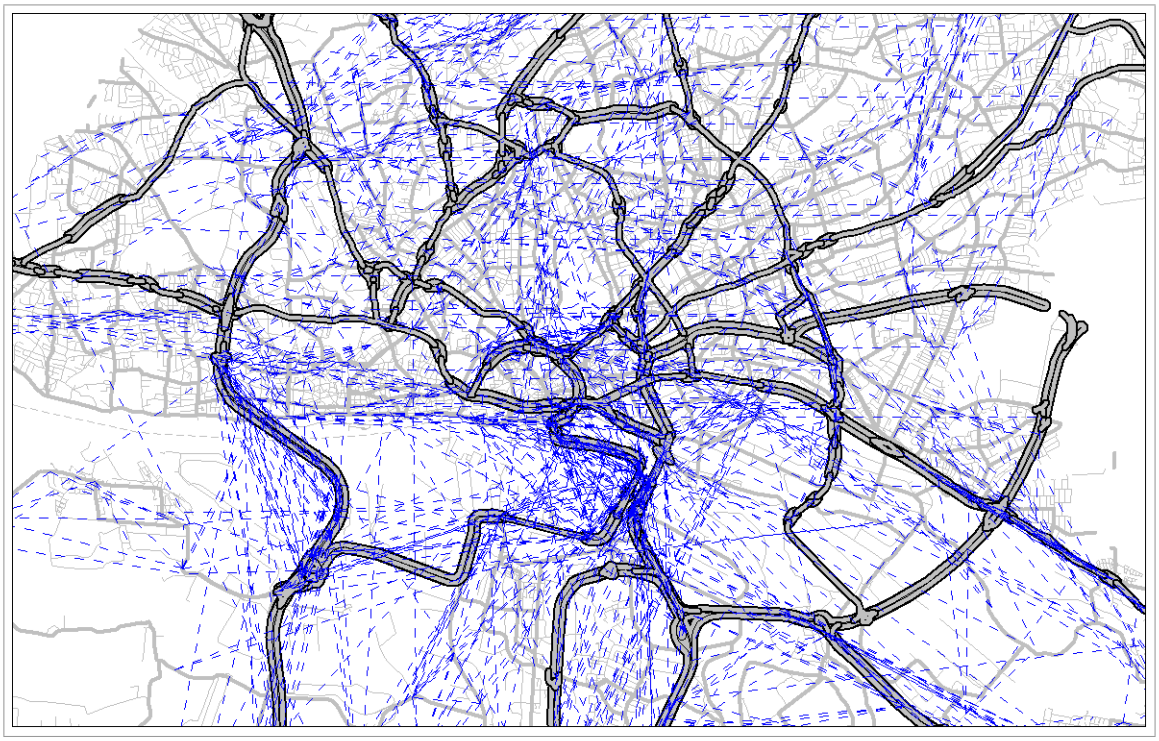


Figure A.3: Hamburg - routes: 600 - maxfs: 20 - fss: 4 - sup: 2

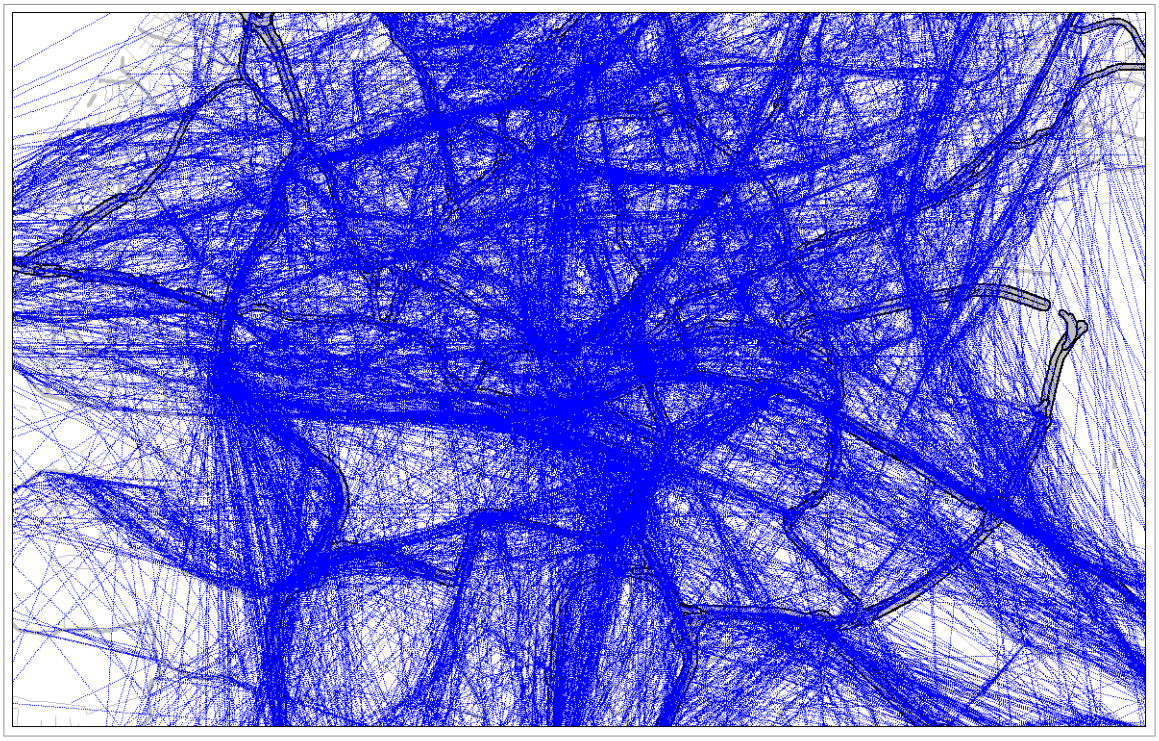


Figure A.4: Hamburg - routes: 50'000 - maxfs: 20 - fss: 4 - sup: 4

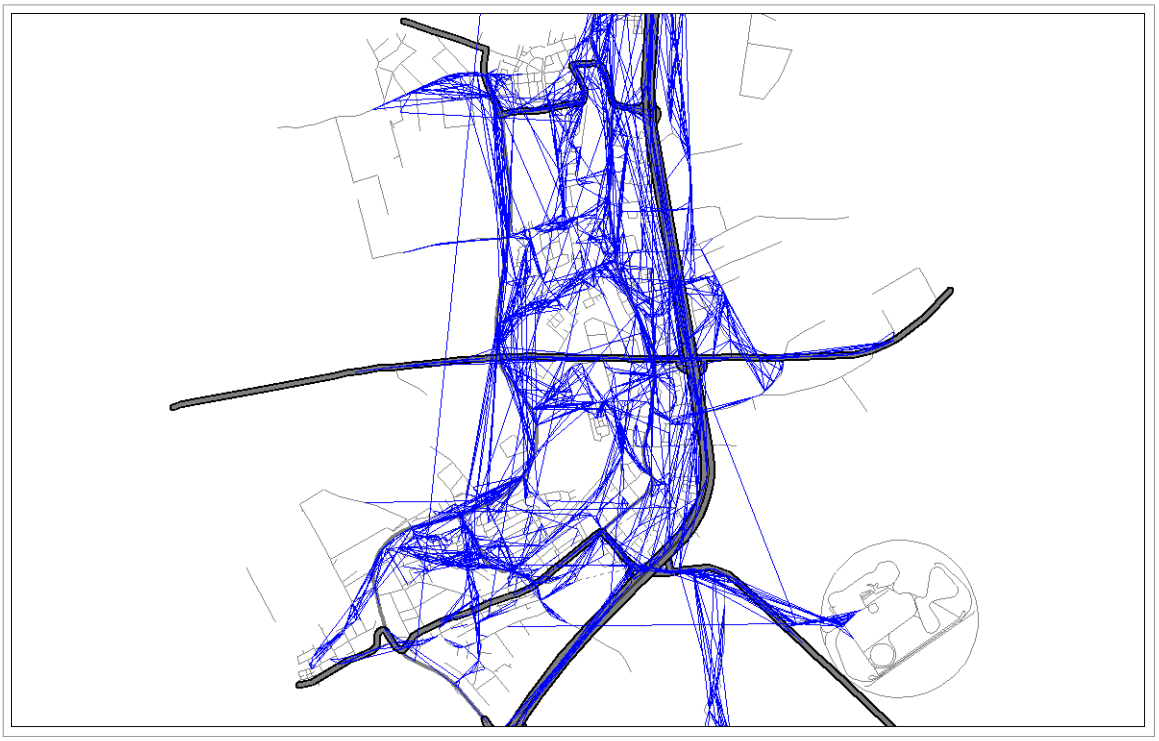


Figure A.5: Rodgau - routes: 1'000 - maxfs: 20 - fss: 4 - sup: 4

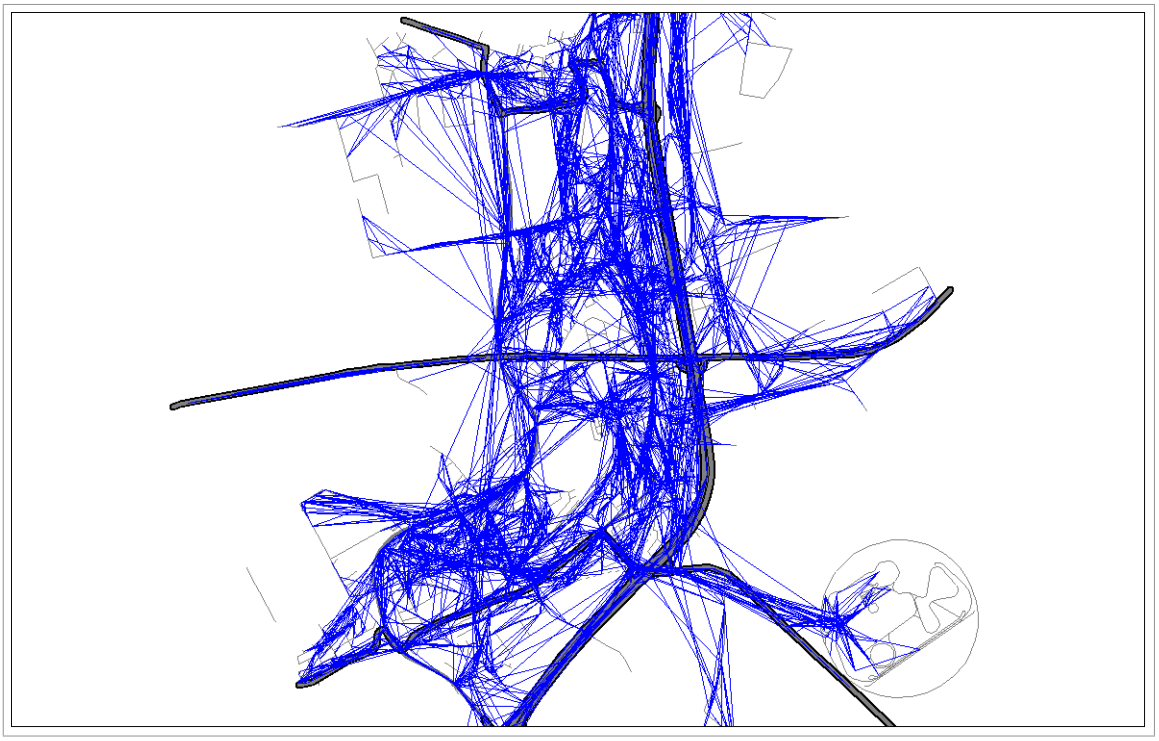


Figure A.6: Rodgau - routes: 5'000 - maxfs: 20 - fss: 4 - sup: 4

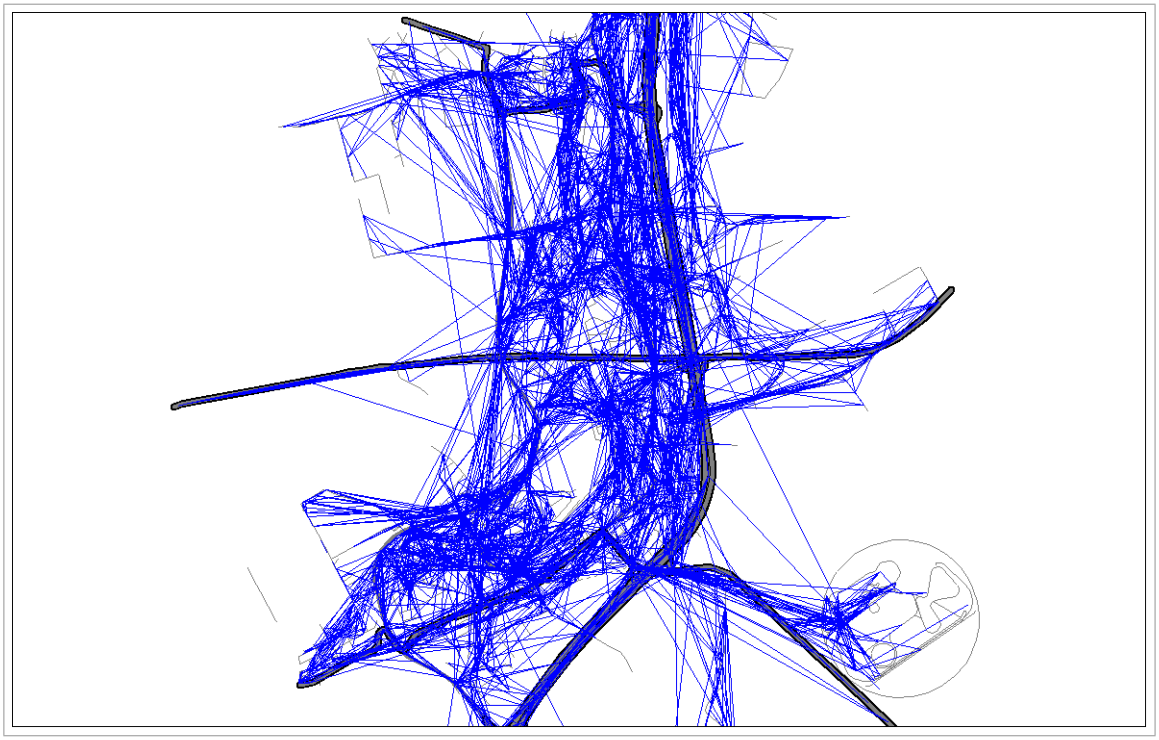


Figure A.7: Rodgau - routes: 5'000 - maxfs: 20 - fss: 4 - sup: 3

Bibliography

[Bar95] N. Bartelme. Geoinformatik. Springer-Verlag Berlin Heidelberg, 1995.

[BCG01] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A Maximal Frequent Itemset Algorithm for Transactional Databases. In Proceedings of the 17th International Conference on Data Engineering (ICDE'01), pages 443–452. IEEE Computer Society, 2001.

[BDP03] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and Complexity Results for #sat and Bayesian Inference. In Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03), pages 340–351. IEEE Computer Society, 2003.

[BH03] M. R. Berthold and D. J. Hand. Intelligent Data Analysis. Springer, 2003.

[Bis96] C. M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, 1996.

[Bis06] C. M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, 2006.

[BM98] P. A. Burrough and R. A. McDonnell. Principles of Geographical Information Systems. Oxford University Press, 1998.

[Bun91] W. Buntine. Theory Refinement on Bayesian Networks. In Proceedings of the 7th Annual Conference on Uncertainty in Artificial Intelligence (UAI'91), pages 52–60. Morgan Kaufmann, 1991.

[Bur94] R. E. Burkard. Methoden der ganzzahligen Optimierung. Springer, 1994.

[CH99] D. M. Chickering and D. Heckerman. Fast Learning from Sparse Data. In Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI'99), pages 109–115. Morgan Kaufmann, 1999.

[Chi96] D. M. Chickering. Learning Bayesian Networks is NP-Complete. In D. Fisher and H. J. Lenz, editors, Learning from Data: Artificial Intelligence and Statistics V, pages 121–130. Springer-Verlag, 1996.

[Chi02] D. M. Chickering. Learning Equivalence Classes of Bayesian-Network Structures. J. Mach. Learn. Res., 2:445–498, 2002.

[Chr00] G. Christakos. Modern Spatiotemporal Geostatistics. Oxford University Press, 2000.

[CLDS99] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. Probabilistic Networks and Expert Systems. Springer-Verlag New York, Inc., 1999.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. MIT Press, 2001.

[Cre93] N. A. C. Cressie. Statistics for Spatial Data. John Wiley & Sons Ltd, 1993.

[dsG06] Digital data service GmbH. Navteq atlas. `http://www.dds.ptv.de`, 2006.

[FNP99] N. Friedman, I. Nachman, and D. Peér. Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI'99), pages 206–215. Morgan Kaufmann, 1999.

[Gai] K. Gaitanis. Open Bayes for Python. `http://www.openbayes.org`.

[GF02] E. Gyftodimos and P. Flach. Hierarchical Bayesian Networks: A Probabilistic Reasoning Model for Structured Domains. In Proceedings of the ICML-2002

Workshop on Development of Representations, pages 23–30. The University of New South Wales, July 2002.

[GM04] A. Goldenberg and A. W. Moore. Tractable Learning of Large Bayes Net Structures from Sparse Data. In Proceedings of the twenty-first international conference on Machine learning (ICML'04), pages 44–52. ACM Press, 2004.

[GM05] A. Goldenberg and A. W. Moore. Bayes net graphs to understand co-authorship networks? In Proceedings of the 3rd international workshop on Link discovery (LinkKDD'05), pages 1–8. ACM Press, 2005.

[HGC94] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. In KDD Workshop, pages 85–96. Morgan Kaufmann, 1994.

[Hug] Hugin Expert A/S. Hugin. `http://www.hugin.com`.

[IAI07a] Fraunhofer IAIS. Faw - Fachverband Außenwerbung e.V. Project. `http://www.iais.fraunhofer.de/2235.html?&L=2`, 2007.

[IAI07b] Fraunhofer IAIS. Spr+ - Swiss Poster Research Plus Project. `http://www.iais.fraunhofer.de/2231.html?&L=2`, 2007.

[LK02] D.-I Lin and Z. M. Kedem. Pincer-Search: An Efficient Algorithm for Discovering the Maximum Frequent Set. IEEE Trans. Knowl. Data Eng., 14(3):553–566, 2002.

[Mei99] M. Meilă. An Accelerated Chow and Liu Algorithm: Fitting Tree Distributions to High-Dimensional Sparse Data. In Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99), pages 249–257. Morgan Kaufmann Publishers Inc., 1999.

[MH01] H. J. Miller and J. Han. Geographic Data Mining and Knowledge Discovery. Taylor & Francis, Inc., 2001.

[Mur] K. Murphy. Bayesian Network Toolbox for Matlab. `http://bnt.sourceforge.net`.

[Mur98] K. Murphy. A Brief Introduction to Graphical Models and Bayesian Networks. `http://www.ai.mit.edu/~murphyk/Bayes/bnintro.html`, 1998.

[Nea04] R.E. Neapolitan. Learning Bayesian Networks. Pearson Prentice Hall, 2004.

[oW] The University of Waikato. Weka. `http://www.cs.waikato.ac.nz/ml/weka`.

[Pea88] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.

[RN04] S. J. Russell and P. Norvig. Künstliche Intelligenz. Ein moderner Ansatz. Pearson Studium, 2004.

[Rob77] R. W. Robinson. Counting Unlabeled Acyclic Digraphs. In Proceedings of the 5th Australian Conference of Combinatorial Mathmatics, pages 28–43. Springer, 1977.

[RSV01] P. Rigaux, M. Scholl, and Agnès Voisard. Spatial Databases with Application to GIS. Morgan Kaufmann, 2001.

[Ski98] S. S. Skiena. The Algorithm Design Manual. Springer-Verlag New York, Inc., 1998.

[SM05] A. Singh and A. W. Moore. Finding Optimal Bayesian Networks by Dynamic Programming. Technical report, Carnegie Mellon University, June 2005.

[Stu06] M. Studený. An Algebraic Approach to Structural Learning Bayesian Networks. In Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'06), pages 2284–2291. ESIA-Université de Savoie, 2006.

[Val79] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. SIAM Journal on Computing (SICOMP), 8(3):410–421, 1979.

[WZF+06] Yi Wang, Lizhu Zhou, Jianhua Feng, Jianyong Wang, and Zhi-Qiang Liu. Mining Complex Time-Series Data by Learning Markovian Models. In Proceedings of the Sixth International Conference on Data Mining (ICDM'06), pages 1136–1140. IEEE Computer Society, 2006.

[YC02] Shulin Yang and Kuo-Chu Chang. Comparison of Score Metrics for Bayesian Network Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 32(3):419–428, 2002.

[YSW+04] J. Yu, V. A. Smith, P. P. Wang, A. J. Hartemink, and E. D. Jarvis. Advances to Bayesian Network Inference for Generating Causal Networks from Observational Biological Data. Bioinformatics, 20(18):3594–3603, 2004.