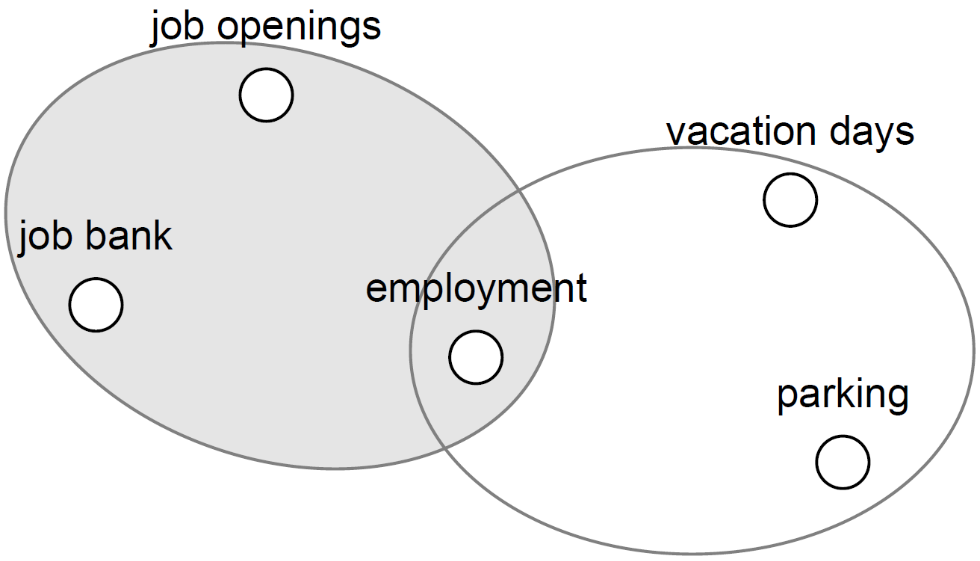


Query Concept Interaction over Time

Master of Science Thesis in the Programme Computer Science: Algorithms Languages and Logic

TOBIAS FÄRDIG,
FREDRIK JOHANSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, May 2012

Query Concept Interaction over Time

TOBIAS FÄRDIG,
FREDRIK JOHANSSON

Examiner: DEVDATT DUBHASHI

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Interaction graph for search queries

Department of Computer Science and Engineering
Göteborg, Sweden May 2012

Acknowledgments

We would like to express our sincerest gratitude towards our supervisor Vinay Jethava for immense help in the work with this thesis. We would also like to thank Findwise AB and their employees for their insight and help and for providing us with data. In particular, we thank Martin Johansson and Svetoslav Marinov for practical guidance in the work with our project.

Abstract

With the ever-increasing amount of data available through various kinds of search engines, the need for better ways of identifying what users are looking for increases as well. In recent years, there have been many attempts at improving search results by trying to identify the true nature of the users' intent. An aspect of this that is often overlooked is that the intent of users is dynamic. One example of this comes from health care. Queries such as *flu* or *fever* occur at a regular frequency under normal circumstances. However, a search for *bird flu* may indicate that an outbreak of the disease is at hand.

In this paper we present a novel method for discovering hidden, time-varying interaction patterns in search query relationships. The method revolves around a probabilistic graphical model, capable of inferring interactions between groups of queries. Given sequences of query expressions and a base graph, the model produces sequences of interaction strengths. We perform synthetic experiments confirming the effectiveness of our model in recovering latent interaction dynamics. Furthermore, we compare the performance of our model to existing methods for network dynamics. In an application to search query data, we use our model to perform keyword suggestion and evaluate the results.

The results of evaluating our model, shows that it has the potential to benefit a wide variety of applications within web-search including keyword suggestion and suggestion for related queries.

Contents

List of Figures

List of Tables

1 Introduction

WITH THE EVER-INCREASING AMOUNT OF DATA available through various kinds of search engines, understanding the intent of users becomes all the more important. In recent years, there have been many attempts at improving search results by trying to identify the true nature of the users' intent based on additional information. An aspect of this that is often overlooked is that the intent of users is dynamic. One example of this comes from health care. Queries such as *flu* or *fever* occur at a regular frequency under normal circumstances. However, a search for *bird flu* may indicate that an outbreak of the disease is at hand.

In many applications related to search, discovering user intent is of great importance. Examples of such applications include result ranking, suggestions for related queries and various kinds of statistics. When searchable content and the number of users grow large enough, the importance grows as well. To gain a more accurate knowledge of user intent, we propose an approach incorporating the dynamics of intent, resulting in a finer grained analysis.

In this thesis, we present a novel method for discovering hidden, changing interaction patterns in search query relationships. We argue that knowing when and how queries are related can help identifying user intent. In Figure 1 is a graph representation of three queries (nodes), *cough*, *flu* and *allergy*. The edges between them, and the weight of the edges represent relationships and strength of the relationships. The example depicted in the figure attempts to visualize the importance of incorporating time dynamics in the analysis of query interactions. In the summer, we might see that cough is more related to allergy than to flu. In the winter, we may see the opposite,

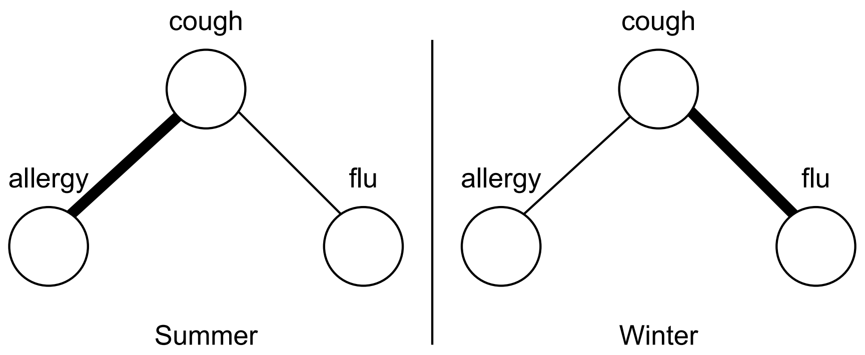


Figure 1: *Example of query interaction dynamics. Nodes represent queries and edges interactions between them. The thickness of an edge indicates the strength of the interaction.*

that cough and flu are more related. Using this knowledge, we could for instance rank results about flu higher when searching for cough in the winter, than in the summer.

In this thesis, we construct a probabilistic model for query interactions. We then use this model in an application to keyword suggestion. Keyword suggestion is a problem arising when entering new documents into a search engine. In such a setting, documents are often tagged with keywords to summarize them or boost their rank in the result list, cluster them, etc. Traditionally, documents are tagged with keywords either manually or by using keyword extraction software. Both approaches, however, neglect the search behavior of users and the queries that are actually used to find a document. Using a model for query interactions, we can make intent-aware, search-driven suggestions for keywords.

The methods presented in this thesis revolves around a probabilistic, Markovian graphical model, capable of inferring dynamic interactions in groups of queries. Rather than considering only pairwise interactions, it allows any number of queries to interact as one group. Given sequences of query expressions and a base graph, the model produces sequences of interaction strengths. We perform synthetic experiments confirming the effectiveness of our model in recovering latent interaction dynamics. Furthermore, we compare the performance of our model to existing methods for network dynamics.

1.1 Purpose and aims

With this thesis, we explore the possibility of applying probabilistic models to identify hidden interactions between search queries. The purpose of this is to open up new possibilities in search technology and to contribute to the research in probabilistic models.

The aims of the project behind this report are three-fold:

1. Apply and adapt an existing graphical model for network dynamics to the problem of identifying dynamic query interactions
2. Generalize and enhance the model mentioned in 1)
3. Apply the model to the problem of keyword suggestion

We aim to produce an implementation of the model mentioned above that can serve as a proof-of-concept for industry applications.

1.2 Scope

Within the realm of search technology, there has been an immense increase in research in recent years. This means that many models are available, attempting to solve problems closely related to the ones examined in this thesis. This thesis is not focused on such models, but rather attempts to apply a probabilistic model initially not intended for search applications. We therefore limit our result comparisons to other probabilistic models. In short, this thesis is more focused on the application and generalisation of an existing model to a new area, then to create a competitor for existing models.

In problems related to web search, typical datasets are often made up of search logs – records of search activity containing information on queries, time and what documents are accessed. Often, the documents themselves are not available, but only a link to them. Here, we only consider datasets where the documents *are* accessible as full texts.

In the tests we perform related to search queries, we only use datasets in Swedish, and only from one source. This is because there are very few publicly available datasets where full query logs and documents are accessible. However, there is no built in bias in our model, or the implementation we produce, towards any language.

Because of the fact that none of the datasets used are labeled, we can only compare the performance of our model with other unsupervised models.

1.3 Method

This thesis work was set up in a collaboration between Chalmers and Findwise AB, a company working with web search. The interest for the thesis work proposed is grounded in that it is closely related to research performed at Chalmers. From the point of view of Findwise, the interest lies in that the work is intended to make it easier for search engines to return better search results.

The model we are introducing is inspired by the NETGEM model [21], which makes it a good starting point for our studies. One of the main contributions of this thesis is to cast the problem of query correlations and keyword suggestion into a probabilistic framework. This allows systematic study of time-varying query relationships and allows us to discover hidden patterns in global user search behaviour. In order to handle the large amounts of query logs, we introduce a tractable probabilistic model inspired from recent work in NETGEM which identifies time-varying interactions. However, the NETGEM model is geared towards biological networks and is unsuited for query interaction partly because it considers only pairwise interactions. Furthermore, NETGEM is not suitable for datasets of the size that are handled in the applications we consider in this thesis. Last, the notion of gene functional categories does not apply to queries.

Our thesis proposes a hyper-graph based approach for modeling query relationships. We incorporate query intent models using the topic model LDA. We compare our approach to existing state-of-the-art methods and show that our approach performs significantly better by leveraging the multi-way interactions between queries.

The methods underlying our model and the NETGEM model are first studied to provide a full understanding of the model. We describe these general methods in the theory section of this thesis. Understanding these methods fully is necessary for us to be able to implement the model, as well as generalise it for our use.

Studies also need to be done in the query field to understand the problems that exist with previous attempts. This is to be able to motivate that our model is usable as well as propose use cases for in what way it is intended to be used.

The original NETGEM implementation is done in MATLAB, and the first step is to implement it in Java instead. This is to make the model more widely usable. Also, Findwise uses Java implementations for their search tools, therefore this model needs to be implemented using Java to be useful for the application it is intended. All parts of the model will be implemented by us, partly to be able to provide the model with almost no dependencies in the code, as well as giving a deeper understanding of the inner workings of the model. The only dependency used is an implementation of a topic model, described later in the thesis.

For increasing performance, our model is implemented with support for parallel estimation of variables. The specifics of this is described later in the thesis. Our model also differs from NETGEM as in it handles hypergraphs instead of simple, pairwise graphs. This change is done with the intention that the model will be able to handle interactions between multiple related queries, instead of just interactions between two.

The evaluation of *our model* is done using data provided by Findwise. This data consist of search logs from which we model the correlation changes between queries over time. To evaluate these results we need to find a measurement of the accuracy of the resulting correlations. This has proven to be a very difficult, since there is no correct solutions available. What we can do instead is to find queries that the model decides is correlated and say if this is reasonable or not. This, however, is not something that can be easily done for every example since some correlations can be unintuitive while still correct.

To get some measurable results we conduct a few other experiments. First we use synthetic data, generated to fit our model. We then use this data to compare with the original NETGEM model. We also use the search data from Findwise in a different application, namely keyword suggestion. The point of this application is that, given a new document, we want to suggest keywords for this document based on what users have searched for when looking for similar documents. For this application we can use one set of documents for the training of our model, and another set of documents for the testing procedure. The testing is done by checking what keywords are suggested by the model for a document, and checking to see if these keywords have been used to search for the document.

1.4 Report outline

The structure of this thesis is strongly related to the chronology of our project execution. In Section 2, we review the theoretical framework needed to understand the model we present in later sections. We cover concepts like graphical models, hidden Markov models and expectation-maximization.

In Section 3 we briefly survey research related to this thesis including probabilistic models and query applications.

After covering relevant research and theory, we present our approach to modelling query interactions in Section 4. The section covers the construction of our general graphical model and specifics regarding query applications.

In Section 5 we cover possible applications of our model and continue in Section 6 with a description of experiments conducted in the project behind this thesis. We cover both experiments on synthetic and real-world data.

The thesis ends with a discussion on results in Section 7 and conclusions in Section 8.

The very last part of this thesis is an appendix with some texts on theory detailed descriptions that did not fit into the main sections.

2 Theoretical framework

2.1 Graphical models

A graphical model is a way of representing random variables and the joint distribution of them as a graph [26]. Throughout this text we denote a graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$ by,

$$G = (\mathcal{V}, \mathcal{E}) \tag{1}$$

In the graph of a graphical model, edges represent variable dependencies and nodes random variables. Suppose that we have N random variables, some of which are dependent on each other. An example of a graphical model, and the joint distribution of the variables in the model, can be seen in Figure 2.

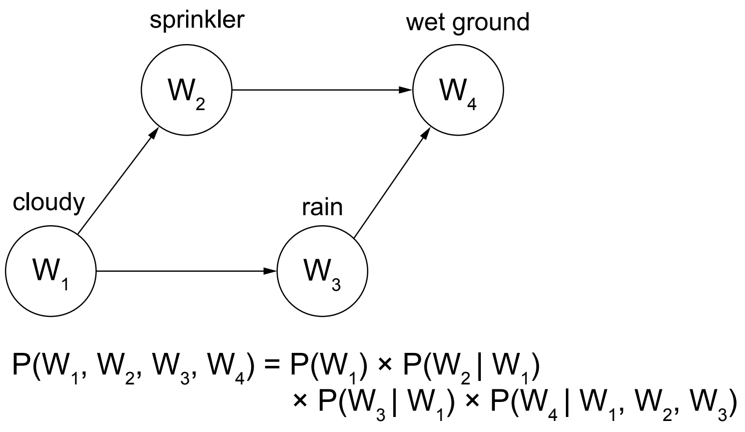

sprinkler
wet ground
cloudy
rain
W_1
W_2
W_3
W_4
$P(W_1, W_2, W_3, W_4) = P(W_1) \times P(W_2 \mid W_1)$
$\times P(W_3 \mid W_1) \times P(W_4 \mid W_1, W_2, W_3)$

Figure 2: *Directed graphical model of four variables. Arrows indicate dependencies betwen the variables. At the bottom is the joint distribution of the variables.*

There are two main kinds of graphical models, corresponding to two kinds of graphs, namely *directed* and *undirected* graphical models [26]. An edge $e = (v_i, v_j)$ in a directed graphical model specifies that the variable v_j is dependent on v_i. In an undirected graphical model, the same edge means that both variables depend on the other. The choice between directed and undirected graphical models depends on the application, as we will see in Section 2.2 about hidden Markov models.

In Figure 2, we have represented a classic example of 4 random variables. This example has an often repeated weather-related interpretation where all four variables are boolean. The variable W_4 is said to represent the variable *wet ground*, W_2 and W_3 the events *sprinkler on* and *rain* and W_1 represents *cloudy weather*. In other words, if we observe the ground being wet, this is assumed to depend on whether it has rained or if a sprinkler has been turned on, or both. Furthermore, the probability of rain depends on whether it is cloudy or not, and similarly the probability of using the sprinkler is also affected by the weather.

In many practical applications, we know the values of only a few of the variables in a graphical model. However, knowing the probabilities of the events means that we can draw conclusions based on the data at hand. In the weather example, if we observe the ground being wet and that the weather is cloudy, the likely cause of the wet ground is that it has rained. The process of drawing conclusions on the probability of unknown variable values from observing some variables in the model is called *inference* [7]. Inference and modelling are the two main uses of graphical models.

Inference algorithms for graphical models is a large topic and we intend only to cover a small fraction of it in this report. For a treatment of a sampling-based algorithm, see Section 2.5. In some cases, even the probabilities of the model are unknown and become parameters. Given some observations of the model, and suitable prior probabilities, such parameters can be estimated using a process known as *parameter estimation* [26]. Inference and parameter estimation are general terms within the realm of machine learning. In this report however, we only cover inference and parameter estimation directly related to graphical models. An example of algorithms for parameter estimation is Expectation Maximization (EM) [27], which we cover in Section 2.3.

Graphical models are widely used within the field of machine learning. Examples of applications include speech recognition [32], genetics [21], and as we will see later in this report, web search. A common type of graphical model is the hidden Markov model, described in the next section.

2.2 Hidden Markov models

In the previous section, we introduced the concept of graphical models. A common example of graphical models is the hidden Markov model. In this section we first briefly review the concept of Markov chains and Markov processes, before move onto the case of hidden Markov models.

A stochastic process is an ordered collection of variables that take on values from the same state space [35]. Markov processes are stochastic processes that evolve according to the Markov property, that transitions from one state to another depend only on the most recent state [31]. Formally, for a stochastic process $X = (X^t)_{t=1,...}$, we have

$$P(X^t = x^t \mid X^{t-1} = x^{t-1}, ..., X^1 = x^1) = P(X^t = x^t \mid X^{t-1} = x^{t-1}) \quad (2)$$

Markov processes that can take on only a finite number of states are often called Markov chains. In many applications when modelling data as a Markov chain, the state of the Markov chain isn't observable directly, but only through some function. This is sometimes referred to as the state being *partially observable* [26]. A Markov chain with a partially observable state is called a *hidden Markov model* or *HMM* [32].

We proceed to introduce some notation for HMM:s. We denote the state of the HMM by w^t taking on values from the set $W = \{W_i\}_{i=1}^N$ and $t = 1,2,...$ being the time. Observations at time t are denoted x^t and take on values in a discrete set, $x^t \in \{X_k\}_{k=1}^M$.

At the core of an HMM are two smaller models, governing the behavior of the state and observations of the model. One is called the *evolution model* and determines how transitions from one state to the next behave. The second is called the *observation model* and governs the observations of the HMM.

A HMM can be specified using two model parameters, N and M and three probability measures, A, O and π [32]. N is the number of states that the model can take on. M is the number of values, or symbols, that the observed variables can take.

$A = \{a_{ij}\}$ is a transition probability matrix, representing the evolution model, where element a_{ij} is the probability of moving from state i to state j where $i,j \in \{1,...,N\}$.

$$a_{ij} = P(w^{t+1} = W_j \mid w^t = W_i) \quad i,j \in \{1,...,N\} \quad (3)$$

$O = \{o_j(k)\}$ is the measure of observation probabilities, the observation model, where $o_j(k)$ is the probability of observing symbol X_k when the model is in state W_i.

$$o_i(k) = P(x^t = X_k \mid w^t = W_i) \quad i \in \{1,...,N\}, k \in \{1,...,M\} \quad (4)$$

$\pi = \{\pi_i\}$ is the initial state distribution, where π_i is the probability of the model being in state i at the beginning of the process.

$$\pi_i = P(w^1 = W_i) \quad (5)$$

Representing A and B as matrices, N and M can be determined by the sizes of A and B and we can therefore specify the entire HMM by the parameter set $\lambda = (A, O, \pi)$. For the parameters to be probabilities, we need,

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \text{for all } i \in \{1,...,N\} \tag{6}$$

$$\sum_{k=1}^{M} b_i(k) = 1 \quad \text{for all } i \in \{1,...,N\} \tag{7}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{8}$$

A hidden Markov model can be represented as a directed graphical model [26] where the state of the process is represented by a random variable, a node, and transitions are represented by dependencies, edges. Also, observed variables are represented as nodes with edges coming from the states. An graph representation of a typical HMM can be seen in Figure 3.

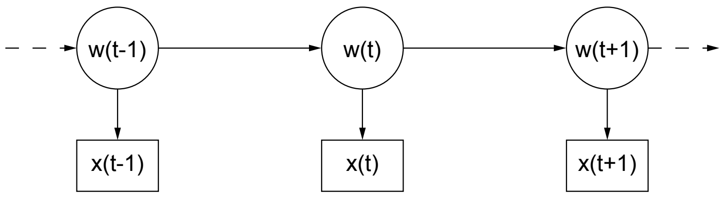


Figure 3: *Hidden Markov model. Circles represents the state of the model at a given time and boxes the observations.*

A common problem faced when modelling data with HMMs is to find the posterior marginal distribution for the state variables given a sequence of observations [32]. Finding the posterior is a problem of inference and can be done using the forward-backward algorithm, explained in Section 2.2.1. To perform inference, however, the parameter set λ must be specified. If the parameters are unknown, as in most practical cases, parameter estimation must be performed. This can be done using expectation-maximization which is covered in Section 2.3.

2.2.1 Forward-backward algorithm

When performing inference on an HMM, the popular choice is the forward-backward algorithm [32]. Based on a sequence of observations, the algorithm and its results can be used to estimate the likelihood of the model, the probability of the observations etc. Formally, the algorithm computes the posterior marginals of the hidden state variables given an observation sequence $x = (x^t)_{t=1}^T$. At the core of the forward-backward algorithm are three model parameters $\lambda = (A, O, \pi)$, specifying the HMM, and two variables as defined below.

The model parameters consists of the *transition probabilities*, $A = \{a_{ij}\}$, the *observation probabilities* $O = \{o_i(k)\}$ and the *initial state probabilities* $\pi = \pi_i$ as defined in Section 2.2. With the definition of the model parameters λ we can begin the description of the forward-backward algorithm. The key variables mentioned in the beginning of this section are the so-called *forward iterates*, $f^t(i)$ and *backward iterates*, $b^t(i)$.

The forward iterates, $f^t(i)$ represent the probability of observing the partial sequence $x^1, ..., x^t - 1$ and ending up in state $w^t = W_i$ given the model parameters λ. Formally, we have,

$$f^t(i) = P((x^1,...x^t), w^t = W_i \mid \lambda) \tag{9}$$

Using the definitions from earlier, we can compute the forward iterates using an inductive algorithm.

$$\begin{aligned} f^1(i) &= \pi_i o_i(x_1) \quad \text{for } i \in \{1,...,N\} && (10)\\ f^{t+1}(j) &= \left[\sum_{j=1}^{N} f^t(i) a_{ij}\right] o_j(x^{t+1}) && (11)\\ & \qquad \text{for } j \in \{1,...,N\}, \quad t = 1,...,T-1 && (12) \end{aligned}$$

In a similar way, we define the backward iterates $b^t(i)$ as the probability of observing the partial sequence $x^{t+1}, ..., x^T$ after being in state $w^t = W_i$ given the model parameter λ.

$$\begin{aligned} b^1(i) &= 1 \quad \text{for } i \in \{1,...,N\} && (13)\\ b^t(i) &= \left[\sum_{j=1}^{N} b^{t+1}(j) a_{ij}\right] o_j(x^{t+1}) && (14)\\ & \qquad \text{for } j \in \{1,...,N\}, \quad t = T-1, T-2, ..., 1 && (15) \end{aligned}$$

Now, with f and b, we can compute, for instance, the probability $\gamma^t(i)$ of being in a particular W_i state at a particular time t,

$$\begin{aligned} \gamma^t(i) &= P(w^t = W_i \mid O, \lambda) && (16)\\ &= \frac{f^t(i) b^t(i)}{\sum_{j=1}^{N} f^t(j) b^t(j)}. && (17) \end{aligned}$$

To compute the most likely state sequence, we can use,

$$w_{ML}^t = \underset{i=1,\ldots,N}{\arg\max} \left[\gamma^t(i)\right] \tag{18}$$

The calculations above can be performed only if we know the model parameters, λ. In the case where these are unknown, we can still use the forward-backword algorithms by estimating the parameters. One way of estimating the parameters is using the expectation-maximization (EM) algorithm, see Section 2.3. In the case of HMM:s, the parameter updates of the EM is identical to those of an algorithm called the Baum-Welch algorithm [32].

Here, we present the parameter updates according to Baum-Welch without proof. First we define the variable $\xi^t(i,j)$ for convenience.

$$\xi^t(i,j) = \frac{f^t(i)a_{ij}o_j(x^{t+1})b^{t+1}(j)}{\displaystyle\sum_{i=1}^{N}\sum_{j=1}^{N} f^t(i)a_{ij}o_j(x^{t+1})b^{t+1}(j)} \tag{19}$$

Now, we write the update equations,

$$\pi_i^* = \gamma^1(i) \tag{20}$$

$$a_{ij}^* = \frac{\displaystyle\sum_{t=1}^{T-1} \xi^t(i,j)}{\displaystyle\sum_{t=1}^{T-1} \gamma^t(i)} \tag{21}$$

$$b_j^*(k) = \frac{\displaystyle\sum_{\substack{t=1 \\ \text{s.t. } x^t=v_k}}^{T} \gamma^t(i)}{\displaystyle\sum_{t=1}^{T} \gamma^t(i)}. \tag{22}$$

This concludes our description of the forward-backward algorithm.

2.3 Expectation-maximization algorithm

The Expectation Maximization (EM) algorithm can be used to find maximum likelihood (ML) estimate of parameter values in problems with incomplete data [12]. The point of ML estimation of the parameters of a model is to find values of these parameters such that the observed data is most likely [29]. The algorithm involves two steps. The first step is the expectation step, where the missing or hidden data are estimated using the observed data. In the second step, the estimated data from the first step combined with the observed data is assumed to be the complete data. The maximization step then calculates the parameters that maximizes the likelihood function for this complete data. The two steps are typically called the E-step and M-step and can be defined as below, adapted from [12].

- E-step: Compute the log-likelihood of the model given the current estimate of parameters
- M-step: Choose parameters so as to maximize the log-likelihood

After this brief overview, we examine the details of the algorithm for better understanding.

Consider the complete data denoted by the random vector X and the model parameter θ. The log likelihood function for the complete data can be formed as

$$L(\theta) = \ln P(X \mid \theta). \tag{23}$$

Here, the maximum likelihood (ML) estimate of the data X is what we are interested in [14]. This means to find θ that maximizes $L(\theta)$, in other words, finding a value of the model parameter that maximizes the likelihood of the complete data.

As stated in the introduction to this section, there are many situations where complete data is not available. For these cases the missing data has to be considered together with the observed data [29]. In this case, consider the observed data Y and the missing data Z, where the log likelihood function is now given by

$$L(\theta) = \ln P(Y{,}Z \mid \theta) \tag{24}$$

A straightforward approach for this would try every possible missing data Z, which in general is intractable. Also, the parameter θ is unknown. Clearly, another approach is needed here.

The EM algorithm suggests dividing this problem into two steps where estimating Z using an estimate of θ and then estimating θ using this estimate of Z [29].

The E-step:

$$Z^{(n)} = E_{Z|Y,\theta^{(n)}} \ln P(Y,Z \mid \theta^{(n)}) \tag{25}$$

The M-step:

$$\theta^{(n+1)} = E(Z \mid Y,\theta^{(t)}) \tag{26}$$

Iterating these steps an estimated solution can be found. It should also be noted that the expectation maximization algorithm never results in decreasing likelihood for each iteration, thus ensuring convergence [24].

The EM-procedure for an HMM is not covered in detail, but the results are stated in Section 2.2.1.

2.4 Generative models & mixture models

A generative model uses the assumption that the observed data is generated from some probability distribution. Using a simple example of a sequence of zeroes and ones, the generative model assumes that these are generated by some probability distribution. Consider the following sequence $S = 00101110110001010101$ (12 zeroes and 8 ones), this could have been created by the probability distribution described by

$$p(s_i = 0|S) = \frac{12}{20} = 0.6 \tag{27}$$

$$p(s_i = 1|S) = \frac{8}{20} = 0.4 \tag{28}$$

where s_i is value number i in the sequence.

By constructing a mixture model, generative models can be combined enabling more closely modelling of complex data. Consider the observable variable v. In the simple, generative model this would be the observed distribution, but in a mixture model this variable will depend on other models [7]. Let those models be the the probability distributions forming the set $H = \{h_1,...,h_m\}$. The probability density of the observed variable v is now given by

$$p(v) = \sum_{h=i}^{m} p(v \mid h_i)p(h) \tag{29}$$

The most common application for mixture model is to assume the data is divided into clusters. Consider an example where the data to be generated consists of points in 2D space. In this example h_i will describe the process of generating points in cluster i. Then v will describe the process of generating the observable points by choosing the points generated by all $h \in H$.

2.5 Inference in graphical models

As mentioned in Section 2.1 about graphical models, inferring values of the hidden variables of the model that maximizes the likelihood of the observed data must be done.

To compute the likelihood of the data, it can be seen that it must be possible to calculate the marginal distribution $p(x_A)$ over a subset A of the nodes N of the graphical model [36]. For a small example using a bivariate random variable $X = (x,y)$, finding the marginal distribution of a single node $i \in N$ includes summing over all possible configurations [36] of the form $\{x' \in X \mid i' = i\}$

Since there are only two possible values of i and $|X|$ nodes in the graph, the set of possible configurations consist of $2^{|X|-1}$ elements. This quickly becomes intractable to solve using a brute force approach, even for this small case with a bivariate random variable.

There are however a number of algorithms known that can be used to solve this approximately, such as Variational approximation [10], Laplace approximation [4] and Gibbs sampling [28].

In the following section Gibbs sampling is described, to introduce a sampling based algorithm for an approximate solution to the problem.

2.5.1 Gibbs sampling

Gibbs sampling is a Markov-chain Monte Carlo method [28]. The idea behind Markov chain Monte Carlo is that a Markov chain is created, where every state is an assignment of values to the variables which are being inferred [3]. This Markov chain can then be sampled via Gibbs sampling with the intent to find a state representing a probability distribution close to the target distribution [28].

Consider the small example with a bivariate variable $X = (x,y)$. Here we want to find the joint density $p(x,y)$. However, as mentioned earlier, in some cases this becomes intractable. The Gibbs sampler instead calculates a sequence of conditional probabilities, $p(x \mid y)$ and $p(y \mid x)$. The sampler then uses two steps (for the bivariate case)

$$
\begin{aligned}
x_i &\sim p(x \mid y = y_{i-1}) && (30)\\
y_i &\sim p(y \mid x = x_i) && (31)
\end{aligned}
$$

iteratively, starting with some initial values x_0 and y_0.

After a sufficient burn-in period of n runs of this algorithm, x_n and y_n will in this case make up an approximate joint probability distribution [28]. The burn-in period can be motivated by the fact that the contribution of the initial values x_0 and y_0 becomes small.

When working with distributions of more variables than two, the bivariate example can be extended in the obvious way. Consider the distribution θ, in this case the update equations can be generalised to

$$
\begin{aligned}
\theta_i^{(k)} \sim\ & p(\theta^{(k)} \mid \theta^{(1)} = \theta_i^{(1)}, ..., \theta^{(k-1)} = \theta_i^{(k-1)}, \theta^{(k+1)} = \theta_{i-1}^{(k+1)}, ...\\
& , \theta^{(n)} = \theta_{i-1}^{(n)}) \qquad (32)
\end{aligned}
$$

where the components are updated in order from $\theta_1^{(k)}$ to $\theta_n^{(k)}$ in each iteration.

2.6 Probabilistic topic models

A way of categorizing large collection of documents is usable in many applications, such as helping search engines find similar documents or returning documents of a particular topic. There are many available methods for performing such tasks. In this section probabilistic topic models are described.

Even with the focus on probabilistic topic models there are a number of algorithms, all using statistical methods to analyse and categorize texts. These algorithms do not require any preprocessing on the documents, using only the text itself to assign topics to the documents.

To describe more closely how these algorithms work, we look at one of the most popular [9, 34] topic models available, Latent Dirichlet Allocation.

2.7 Latent Dirichlet Allocation

The Latent Dirichlet Allocation model (LDA) introduced by Blei et al. in 2003 [11] is a generative probabilistic model for discrete data collections.

Table 1: *Latent Dirichlet Allocation, generative process*

```
1: Choose $N \sim \text{Poisson}(\xi)$
2: Choose $\theta \sim \text{Dir}(\alpha)$
3: for all $N$ words $w_n$ do
4:     Choose a topic $z_n \sim \text{Multinomial}(\theta)$
5:     Choose a word $w_n$ from the probability $p(w_n \mid z_n, \beta)$
```

More specifically, Blei uses the notion of documents as the data to describe the model. The point of the model is to represent each document i in a corpus as a topic distribution θ_i with

$$\sum_{j=1}^{m} \theta_{ij} = 1 \tag{33}$$

with the number of topics m for document. θ_{ij} represents the probability that document i has the topic j. Comparing these topic distributions can be used as a measure of similarity between documents. [11]

Consider a corpus $\mathcal{C}$ containing n documents $[\mathcal{D}_1,\mathcal{D}_2,...,\mathcal{D}_n]$. The generative model for each document $\mathcal{D}_i$ is assumed to be as shown in table 1 [11].

The randomness of the number of words N is not critical for the model's behaviour and a suitable value N can be selected depending on the application. The word probabilities, β decides with which frequency specific words is drawn, depending on the chosen topic. They are given by $\beta_{ij} = p(w_i = 1 \mid z_j = 1)$ and is a parameter in the model that needs to be estimated. This estimation procedure is described in Section A.2. α acts as a prior for selecting the topic distribution θ, and is to be supplied to the model. In this model the dimensionality of the distribution θ is assumed to be known and fixed for the entire corpus. This, in turn, means that the number of possible topics, Z is set before the start of the process.

The generative process is divided into three levels. The parameters ξ and α are corpus level parameters, set to the same value for the entire corpus $\mathcal{C}$. The parameters θ and N are sampled for each document $\mathcal{D}_i$, while the parameters z_n and w_n are sampled for each word. This means that a document d can be associated with more than one topic, specifically at most $\min(Z,N_d)$ topics. The inference of the hidden variables is done using Gibbs sampling, described in Section 2.5. Some details of the application to LDA can be found in Section A.1.

2.7.1 Implementation

The implementation of LDA used for *our model* is part of Mallet - A Machine Learning for Language Toolkit [25]. The Mallet implementation provides an easy to use interface for LDA, taking α and β as input parameters together with the input corpus. α and β are symmetric dirichlet parameters [25], which means that we assume no prior knowledge of the topic distribution over the words [11]. The Latent Dirichlet Allocation inference and parameter estimation is then performed by Mallet using Gibbs sampling for the inference. Returned from *Mallet* is a topic distribution for each of the documents from the input corpus.

Mallet uses a multi-threaded implementation of the topic model, which has a linear time complexity in the total number of words N and the number of topics K, NK [30].

3 Related research

In this paper we present a generative, graphical model to identify latent, dynamic interactions in a set of actors, for instance queries. Applications for such a model exist in a wide variety of settings, such as social [2, 17], technical [1] and biological [21, 33], and several related attempts of solving the problem have been presented earlier. While most such attempts are set in a specific application setting, almost all of them can be generalised to handle data independent of the context. Here, we present an overview of models related to the one presented in this paper, and highlight key differences.

The notion of finding interactions in a set of actors, e.g. queries, can be thought of as a clustering problem. Airoldi et al. [2] presented Mixed-Membership Stochastic Blockmodels (MMSB) for discovering latent clusters based on measurements of actor interactions. The original model was generalised by Song et al. [17] to Dynamic mixed-membership blockmodel for evolving networks, to consider evolving clusters. In both models however, only pairwise interactions were considered, representing interactions as a matrix. The matrix representation makes the model unsitable for handling interactions in groups of larger size than two since the matrix quickly becomes too large to handle. Furthermore, the model requires measurements of interactions as input, and that is partly what we want to infer.

Also related is Timeline [1], a model for discovering the birth and death of topics in streams of documents. While this methods are related in that they deal with dynamic relationships, they do not produce a measure of interaction strength at each time step.

In biology, one problem of interest is to discover how genes interact by inducing or repressing each other. This has motivated models such as KELLER [33]. KELLER produces a series of graphs, modelling gene interactions, based on sequences of gene expression measurements. While demanding little input, it has a problem with sparsity for low-expression input. This is mainly due to the fact that gene expression is assumed to be solely responsible for interactions.

A second model motivated by biology is NETGEM [21]. Given sequences of gene expressions and a base graph of interactions, NETGEM infers time sequences of interaction strength between genes. The inference is based on the assumption that interactions evolve according to, not only on gene expression, but on the functional category of the gene as well. By introducing the functional category, a type of class, NETGEM overcomes the problem of high sparsity found in KELLER. However, like KELLER, NETGEM considers only pairwise interactions, making it unsuitable for certain types of data. Of the related models we have examined, NETGEM is closest to the model presented in this thesis.

Identifying relationships among queries is of interest in many applications including query recommendation and query expansion. Attempts at creating methods for finding these relationships have been made using measures of query similarity and clickthrough URLs [5,6] and intent models [19].

Baeza-Yates et al. [5] approached the problem of query recommendation by relating queries through a similarity measure based on word occurrence in click-through URLs and query popularity. Another attempt [6], relates queries by analyzing a bipartite query-document graph created using query logs. Guo et al. [19] approached the problem using clickthrough documents to create an intent model related to topic models. The model does not take into account the time of the query.

In an application of the model presented in this thesis, we approach the problem of keyword suggestion for new documents in a search engine. Related to this problem is that of keyword extraction. Keyword extraction methods are typically used to assign new documents with a set of keywords when they are entered into a search engine. Such methods include approaches based on statistics, linguistics, machine learning and heuristics [22]. To the best of our knowledge, no existing approaches are search-driven, they do not consider search behavior of users. Basing our solution on query interaction allows us to suggest intent-aware keywords aimed at reflecting user search behavior.

4 Our approach

In this thesis, we present a model capable of inferring dynamic relationships between a set of actors given expression data for these actors at a number of time points an an initial interaction graph. While the model can be adapted to fit different settings, we will use terminology from an application to search query data throughout the paper.

4.1 Model overview

In many applications within the realm of search, it is of great importance to know the intent behind user search queries – what the user is looking for. Modelling how queries interact and how these interactions change over time enables us to give a good estimate of the users' intent. Applications of this knowledge should enable a better search experience for users in the future. We now proceed to describe our model for dynamic query interactions.

The model requires two sets of input data, consisting of a search log and a set of click-through documents. This is a log of queries performed by users, keeping the queries together with an associated document the user has clicked on after searching for the query. The query interactions are modelled using two steps. First, a graph of static interactions is calculated, using topic similarity between documents associated with queries. The second step involves inferring the dynamics of query interactions. To consider time-varying interactions, the query log needs to be divided in a small number of time periods.

The focus of this model is to find query interactions that changes smoothly and slowly over time, and therefore the periods should not be chosen too short. This also means we do not intend to capture extreme cases where a query is very popular for a very short time and otherwise not at all. The time frame should not either be divided into too long periods, as this would affect the significance of the time correlation aspect negatively. For instance, a query with a popularity heavily influenced by the changing of seasons will not be captured at all if the lengths of the time periods are 12 months.

The changes of interaction is then based on how the number of searches for a specific query changes with time. Also the interaction between queries changes if their association with documents changes. Using two example queries *President of USA* and *Barack Obama*. The correlation between these two is assumed to go from weak to very strong with time (and weaker yet again in the future). The idea behind *our model* is to capture this by using the documents the queries are associated with. For this example the documents will start of being very different for the two queries. However, now they will instead probably be associated with the same documents, or at least documents that are very similar.

The focus on smooth time-varying interactions motivates a model based on Markov dynamics. Because we cannot observe interactions directly, we model the data as a hidden Markov model. We assume that interactions are affected by the topic of the queries involved. The notion of topic is brought into the evolution model of our HMM, using the probabilistic topic model LDA. The model is trained using expectation-maximization.

An overview of our model from an application perspective can be seen in Figure 4. The notation, terminology and concepts of the figure will be explained in detail in the following sections.

4.2 Hypergraph representation

For certain types of data, arising from interactions among actors, simple (pairwise) graphs are too crude models to properly represent relationships between the actors. One reason is that graphs can only represent pairwise relationships and not relationships in larger groups. This can lead to problems when trying to identify the nature of such groups and how their members interact.

In an application to query data, actors are queries, many of which are very similar because of the nature of search queries. Misspellings and different wordings cause similar phrases to be different queries, different nodes in a graph, but they should be strongly related. Because of this, the model we propose is based on hypergraphs [8] rather than graphs.

NETGEM [21], is an interaction model originally applied to gene interactions in cell evolution. The original model considers simple graphs and assumes that edges are independent of each other. This assumption can prevent the model from identifying relationships in groups of more than two members, since all interactions are assumed to be pairwise. We propose a generalisation to the approach of [21] by considering hypergraph representations of interactions. In doing so, we aim to indentify interactions in larger groups. Instead of edges, we assume hyperedges to be independent of each-other.

We proceed to introduce some notation for graphs and hypergraphs. In this thesis, we take hypergraph to be the norm and pairwise graphs to be a special case. We denote simple (pairwise) graphs $G = (\mathcal{V}, \mathcal{E}^P)$ and hypergraphs $H = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ denotes the set of nodes, $\mathcal{V} = \{v_1, ..., v_N\}$ where N is the total number of nodes. The edge sets are denoted $\mathcal{E}^P = \{e_1^P, ..., e_{M^P}^P\}$ and

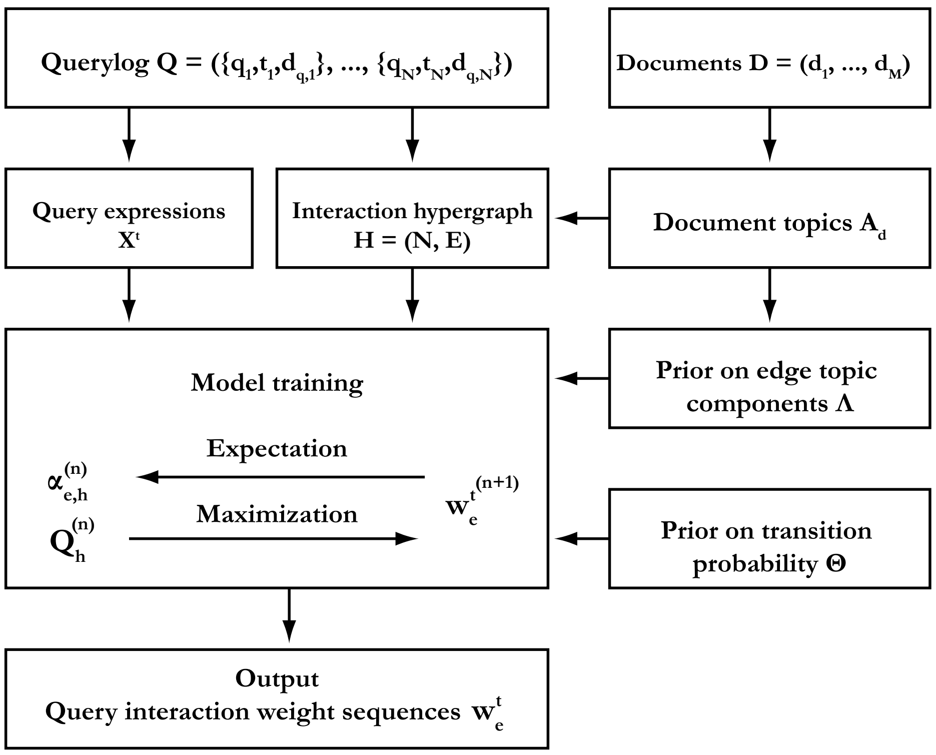

Querylog Q = ({q1,t1,dq,1}, ..., {qN,tN,dq,N})
Documents D = (d1, ..., dM)
Query expressions Xt
Interaction hypergraph H = (N, E)
Document topics Ad
Model training
Expectation
Maximization
α(n)e,h
Q(n)h
w(n+1)te
Prior on edge topic components Λ
Prior on transition probability Θ
Output
Query interaction weight sequences wte

Figure 4: *An overview of our model for query interaction from an application perspective. From the top, going down is the work flow of the model. At the top are the raw query inputs, the documents D and the query log, Q. Using the querylog data and the topic distribution of the documents, we construct the inputs to the model, X and H. The model also need prior distributions, Λ and Θ for the parameters of the model. Λ is created using query topic distributions. In the middle is the model training using expectation-maximization depicted. At the bottom is the output, the inferred weight sequences w_e^t*

$\mathcal{E} = \{e_1^H, ..., e_M^H\}$ respectively.

An pairwise edge is a pair of nodes $e^P = (v_i,v_j)$ and a hyperedge is a tuple (or a subset of $\mathcal{V}$), $e = (v_i,v_j,v_k,...)$ of arbitrary size $k_e \geq 2$. The size k_e of an hyperedge is also called *cardinality.* Note that a hypergraph $(\mathcal{V}, \mathcal{E})$ can be converted into a pairwise graph by adding an edge $e^P = (v_i,v_j)$ to $\mathcal{E}^P$ for every pair of nodes v_i,v_j in a hyperedge $e = (v_i,v_j,v_k,...)$.

We use the notation $n \in e$ to indicate a node n part of the hyperedge e. We will sometime use the notation $e = (i,j,..)$ to mean $e = (v_i,v_j,...)$ for convenience. In general, we consider *edge* to mean hyperedge, unless

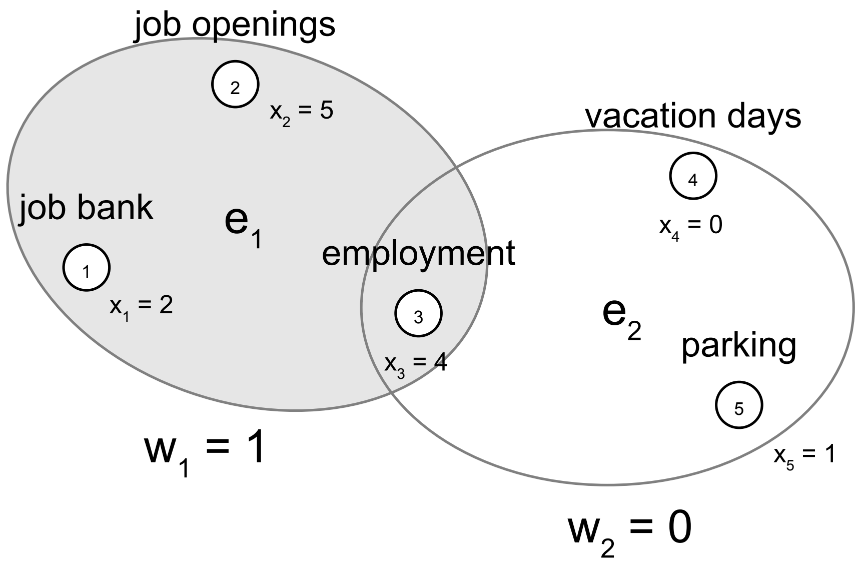


Figure 5: *Hypergraph representation of query interaction. Each node in the hypergraph represent a unique query. A hyperedge (circle) represents the interaction of the queries inside it. Each hyperedge is associated with a weight representing the strength of the interaction.*

otherwise specified. This means that $k_e \geq 2, \forall e \in \mathcal{E}$. In Figure 5 is a hypergraph representation of query interactions.

4.3 Data description

The input to our model consists of two components, node expression data, x and a hypergraph H, such as that of Figure 5. Node expressions are expected in the form of time sequences of actor expressions $x = ((x_j^t)_{t=1}^T)_{j=1}^N$ with j the index of the actor and t the time point of expression. We let $t \in \{1,...,T\}$ be an index representing the interval $[\tau(t), \tau(t+1)]$ where $\tau(t)$ is the starting time of the time period with index t. $\tau(0)$ is the earliest measurement point and $\tau(T)$ the latest.

In general, expressions are real numbers. In query data, actors are queries, uniquely identified by the words that make up the query and the expressions x_j^t represents the number of times a query j has been made at time t.

Our model also requires an initial specification of which interactions to consider. The hypergraph $H = \{\mathcal{V}, \mathcal{E}\}$ is a specification of which edges to infer interactions for. Each node in the graph represents a unique query. Edges represent query interactions and are denoted $e = (v_{e,1}, ..., v_{e,k_e})$ where k_e is the cardinality of edge e. Note that the nodes $\{v_{e,1}, ..., v_{e,k_e}\}$ are a subset of the node $\mathcal{V}$. This graph can be thought of as a static representation of relationships between the queries involved, i.e. which interactions that are expected to take place at all during the entire time period of the data.

Furthermore, our model requires a parameter to be set, namely the number of classes that a query or interaction can belong to. This can be thought of as the number of topics if the application is related to text data as in this case, or the number functional categories if actors are genes for example.

In the query application, data comes in the form of a query log $\mathcal{Q} = \{Q_1, ..., Q_M\}$ made up of tuples $Q_i = (q_i, \tau_i, d_i)$. We call such a tuple a *search query instance* or simply *instance*. Each such instance represents the event of a user searching for something and then clicking on a result. q_i are the query words for instance i, τ_i is the time stamp and d_i is a document identifier associated with the document the user clicked on after making the query. To transform the query logs into expression data, we simply count the occurrences of each query in each interval t,

$$x_j^t = \sum_{i=1}^{M} \delta_j(q_i, \tau_i) \tag{34}$$

$$\delta_j^t(q_i, \tau_i) = \begin{cases} 1 & q_i = \tilde{q}_j \text{ and } \tau_i \in [\tau(t), \tau(t+1)] \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

with $\tilde{Q} = \{\tilde{q}_j\}_{j=1}^{\tilde{M}}$ the set of unique queries.

Note that each set of query words q_i or document identifier d_i may occur arbitrarily many times in the log. This simply means that the same query has been made, or the same document has been clicked on at different times. These re-occurrences is what we count to form expression data. For instance, if the query *flu* has been made 13 times in February and our time intervals are the months of the year, we get $x_{flu}^2 = 13$.

4.4 Observation model

Here we begin the description of the components of our graphical model. In essence, our model is a type of hidden Markov model, a concept described in Section 2.2. The first component described here is the observation model.

We model the query expression data as observations in a hidden Markov model. This means that we assume that expression values occur with a probability governed by the state of the model. Specifically, our observation model models the probability of the expression levels $\vec{x}^t = \{x_v^t\}_{v\in\mathcal{V}}$, conditioned on the interaction strengths $\vec{w}^t = \{w_e^t\}_{e\in\mathcal{E}}$.

We assume that observations are formed according to a nearest-neighbour model, where edges are independent,

$$P(\vec{X}^t = \vec{x}^t \mid \vec{W}^t = \vec{w}^t) = \frac{1}{Z(w^t)} \exp\left(-\sum_{e\in\mathcal{E}} w_e^t \phi(e,t)\right) \tag{36}$$

$$= \frac{1}{Z(w^t)} \prod_{e\in\mathcal{E}} \exp\left(-w_e^t \phi(e,t)\right), \tag{37}$$

with $Z(w^t)$ the normalization factor. Here, $\phi(e)$ is a potential function. In this model we use,

$$\phi(e,t) = \prod_{v\in e} x_{i(v)}^t \tag{38}$$

with $i(v)$ the index of node v.

We note that the expression above is separable by edges and that this agrees with the treatment of the weight sequence of every edge as a separate Markov chain. Now we have our observation model described and we know move on to our evolution model describing how the weights change with time in *our model*.

4.5 Evolution model

To complete the description of our HMM, we introduce the evolution model, the probability of moving to a given state from the current one. Given, the Markov assumption and the assumption that edges are independent, the transition probability of an edge depends only on the most recent state of the edge,

$$P(W_e^t = w_e^t \mid W^1, ..., W^{t-1}) = P(W_e^t = w_e^t \mid W^{t-1}). \tag{39}$$

We denote the probability of an edge $e \in \mathcal{E}$ moving from state w_l to w_k, both in $\mathcal{W}$,

$$Q_e(k,l) = P(W_e^t = w_l \mid W_e^{t-1} = w_k) \tag{40}$$

with

$$\sum_{l=1}^{K} Q_e(k,l) = 1 \quad \text{for all } k = 1,...,K. \tag{41}$$

We call Q_e the *edge transition probability.* The rows of Q_e can be thought of as drawn from a multinomial. We will use a Dirichlet distribution with parameter $\Theta = \{\theta_{i,h}\}$ as a prior for Q_e because it is the conjugate distribution to the multinomial [15].

$$\vec{Q_e}(i,:) \mid \Theta \sim Dir(Q_e(i,1), ..., Q_e(i, |\mathcal{H}|); \theta_{i,1}, ..., \theta_{i,|\mathcal{H}|}) \tag{42}$$

At this stage, we need to incorporate the notion of topics. We want transitions to be governed not just by edge weights, but by topic as well. To do this we formulate a mixture model.

4.6 Topic mixtures for evolution model

To allow edge weights, interactions strengths, to be governed by topics, we bring the notion of topics into the evolution model. We assume that there exists a set of topics $\mathcal{H} = \{h_1,...,h_{N_H}\}$. In other applications, this may be some other form of class. Also, we assume that for each $h \in \mathcal{H}$ there exist some state transition probability matrix Q_h, which we call *topic transition probability.* Furthermore, we assume that nodes and edges are associated with at least one topic and at most all of the topics.

In order to incorporate topics into the state evolution model, we construct a mixture model for edges and topics. In the mixture model, edge transition probabilitis Q_e depend on topic transition probabilities Q_h and mixture proportions $\alpha_{e,h}$.

We denote the set of mixtures proportions $\alpha = \{\{\alpha_{e,h}\}_{e\in\mathcal{E}}\}_{h\in\mathcal{H}}$. The $\alpha_{e,h}$ govern the influence of topic h in the evolution of edge e and the relationship between Q_e and Q_h can be written as follows,

$$Q_e(k,l) = \sum_{h\in\mathcal{H}} \alpha_{e,h} Q_h(k,l) \tag{43}$$

with

$$\sum_{h\in\mathcal{H}} \alpha_{e,h} = 1 \quad \text{for all } e \in \mathcal{E}. \tag{44}$$

We let Q_e be a matrix with elements $Q_e(k,l)$ where k denotes the row and l the column. We interpret an element $Q_e(k,l)$ as the probability of edge e moving from state w_k to state w_l. Note also that Q_h and $\alpha_{e,h}$ completely specifies Q_e. This means that we can now consider only Q_h and $\alpha_{e,h}$ as parameters to be learned, and we calculate Q_e only for inference.

α can be thought of as a matrix with element $\alpha_{e,h}$ at index (e,h). We will sometimes use the notation $\vec{\alpha}_e = \{\alpha_{e,h}\}_{h\in\mathcal{H}}$ which can be thought of as a row vector of α with dimension $|\mathcal{E}|$. Like with Q_e, we put a Dirichlet prior on α_e, with parameter $\Lambda = \{\lambda_{e,h}\}$,

$$\vec{\alpha}_e \mid \Lambda \sim Dir(\alpha_{e,1}, ..., \alpha_{e,|\mathcal{H}|}; \lambda_{e,1}, ..., \lambda_{e,|\mathcal{H}|}) \tag{45}$$

4.7 Generative model

With the definitions made earlier in this section, we can now construct an overview description of our model as a generative model. We assume that the data we use are observations generated from the probabilistic model described here.

As stated previously, we put a Dirichlet prior on the topic transition probability Q_h with hyperparameters $\Theta = \{\theta_e\}_{e\in\mathcal{E}}$ such that each row i of Q_h, $\vec{q}_h(i,\,:) \sim Dir(\vec{\theta}_h(i,\,:))$. Furthermore, we put a similar Dirichlet prior on the topic mixtures $\vec{\alpha}_e = \{\alpha_{e,h}\}_{h\in\mathcal{H}}$ with hyperparameters $\Lambda = \{\vec{\lambda}_e\}_{e\in\mathcal{E}}$ such that for each edge e, $\vec{\alpha}_e \sim Dir(\lambda_e)$.

To formalize the generative process we introduce the notion of active topic Y_e^t for edge e, defined by the equation,

$$P(W^{t+1} = w_m | W_e^t = w_l, Y_e^t = h) = Q_h(l,m). \tag{46}$$

The generative process based on the priors defined above is summarized in table 2. In Figure 6 is a plate notation representation of the resulting generative graphical model.

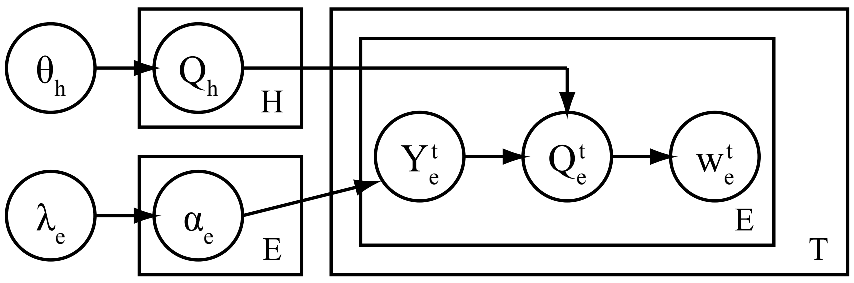


Figure 6: *The generative model in plate notation. A box with a letter X at the bottom right corner indicates a repetition of the contents of the box, X times.*

Table 2: *The generative process of our model*

```
Require: $\mathcal{H} =$ (set of topics)
Require: $G = (\mathcal{V}, \mathcal{E})$ (static interaction network)
Require: $\Lambda = \{\vec{\lambda}_e\}_{e \in \mathcal{E}}$ (prior on topic mixtures)
Require: $\Theta = \{\theta_h\}_{h \in \mathcal{H}}$ (prior on topic transition probability)
 1: for all $h$ in $\mathcal{H}$ do
 2:     Choose $Q_h|\theta_h$ s.t. each row $\vec{q}_h(i,:) \sim Dir(\vec{\theta}_h(i,:))$
 3: for all $e$ in $\mathcal{E}$ do
 4:     Choose $\vec{\alpha}_e|\lambda_e$ s.t. $\vec{\alpha}_e \sim Dir(\vec{\lambda}_e)$
 5: for $t = 1,...,T$ do
 6:     for all $e$ in $\mathcal{E}$ do
 7:         Choose $Y_e^t \sim P(Y_e^t = h) = \alpha_{e,h}$
 8:         Choose $Q_e^t|Y_e^t$ where $Q_e^t|\{Y_e^t = h\} = Q_h$
 9:         Choose $w_e^t \sim P_e(w_e^t|w_e^{t-1}, Q_e^{t-1})$
10:     Choose $\vec{X}^t|\vec{W}^t$
```

4.8 Graph construction in the query application

In the query application, the input hypergraph can be constructed using querylogs, click-through documents and a topic model. We assume that data is available in the form of a query log with records of query words, time of query and the document clicked on after using the query. Furthermore, we assume that a topic model, for instance LDA, has been used to associate each document d with a topic distribution $\vec{A}_d$.

From the query logs, each query q_i is associated with a set of documents, $\mathcal{D}_i = \{d_{i,1}, ..., d_{i,m}\}$, one document for each instance of the query. From a topic model, each document is associated with a topic distribution $\vec{A}_d = [A_{d,1}, ..., A_{d,H}]^\top$, such that $\sum_{h=1}^{H} A_{d,h} = 1$ for all d.

We calculate query-topic distributions $\vec{\kappa}_i = [\kappa_{i,1}, ..., \kappa_{i,H}]^\top$ by averaging over document-topic distributions for each query.

$$\kappa_{i,h} = \frac{1}{|\mathcal{D}_i|} \sum_{d \in \mathcal{D}_i} A_{d,h}. \tag{47}$$

We can construct a static interaction graph by comparing query topic distributions, κ. First we construct a pairwise graph, which is then converted to a hypergraph.

We denote the pairwise graph of query interactions, $\mathcal{G} = (\mathcal{V}, \mathcal{E}^P)$. We add a node v_i to $\mathcal{V}$ for each unique query q_i. Then we calculate the Kullback-

Leibler [23] divergence for each pair of queries (v_i,v_j) to get a measure of how related two queries are by topic.

$$D_{KL}(i,j) = D_{KL}(\vec{\kappa}_i||\vec{\kappa}_j) = \sum_h \kappa_{i,h} \log\left(\frac{\kappa_{i,h}}{\kappa_{j,h}}\right) \tag{48}$$

We add an edge $e^P = (v_i,v_j)$ to $\mathcal{E}^P$ if $D_{KL}(i,j) > \varepsilon_{KL}$ where ε_{KL} is a parameter of the tool.

The pairwise graph is then converted to a hypergraph $H = \{\mathcal{V}, \mathcal{E}\}$ using Bron-Kerbosch algorithm [13].

4.9 Parameter priors in the query application

In our model, as described in Section 4.6, each edge e is assumed to be associated with a topic mixture $\vec{\alpha}_e$ with elements $\alpha_{e,h}$ for topics $h \in \mathcal{H}$. $\alpha_{e,h}$ is a measure of how active the topic h is in the edge e.

In the generative model, the values of $\vec{\alpha}_e$ are assumed to come from a Dirichlet distribution with parameters $\vec{\lambda}_e$, i.e.

$$P(\vec{\alpha}_e) \sim \text{Dirichlet}(\alpha_{e,1}, ..., \alpha_{e,H}; \lambda_{e,1}, ..., \lambda_{e,|\mathcal{H}|}) \ . \tag{49}$$

We interpret the topic mixture hyperparameter, $\Lambda = \{\vec{\lambda}_e\}_{e\in\mathcal{E}}$ as a set of topic distributions, one for each edge e. When constructing a prior in the query application, e exploit the fact that we have calculated a static topic distribution for queries, as described in a previous section. As a prior $\vec{\lambda}_e$ for an edge $e = (v_1, v_2, ..., v_{k_e})$, we simply use the mean topic distribution over the queries in e,

$$\lambda_{e,h} = \frac{1}{k_e} \sum_{v\in e} \kappa_{i(v),h}, \quad e \in \mathcal{E}, \quad h \in \mathcal{H}. \tag{50}$$

where κ_i is the topic distribution for query i, k_e is the cardinality of edge e, and $i(v)$ is the query index of node v.

For transition probabilities we use a Laplacian prior,

$$\Theta(i,j) = \frac{e^{-\gamma|W_j|}}{\sum_{k=1}^{K} e^{-\gamma|W_k|}} \tag{51}$$

which controls the sparsity of the output interaction weight sequence by controlling the probability of reaching a certain weight state using the parameter γ. In other words, a high γ gives a high frequency of small weights (in terms of absolute value), such as 0, while a low γ gives more frequent larger weights.

4.10 Inference & parameter estimation

To estimate the parameters of our model we use a modified version of the the expectation-maximization (EM) procedure which itself uses the forward-backward algorithm. In essence, the algorithm computes the forward and backward iterates and then computes the maximum likelihood estimates of the model parameters. The algorithm is run separately on each edge of the graph, estimating Q_h and $\alpha_{e,h}$ in the process.

Here follows the definition of the forward and backward iterates as well as the observation probabilities used in the forward-backward algorithm.

4.10.1 Expectation-maximization

For the expectation step of the EM-algorithm, we first compute the forward backward iterates. In order to do this, we need to define our observation model and evolution model. Because of the assumption that edges are independent, we treat each edge as a separate HMM, and perform FB on each edge individually. Repeating the definition of the observation model from Section 4.4,

$$P(\vec{X}^t = \vec{x}^t \mid \vec{W}^t = \vec{w}^t) = \frac{1}{Z(w^t)} \prod_{e \in \mathcal{E}} \exp\left(-w_e^t \phi(e,t)\right), \tag{52}$$

$$\phi(e,t) = \prod_{v \in e} x_{i(v)}^t, \tag{53}$$

we can introduce the vague notion of *edge observations* x_e^t because of the fact that the equation above is separable by edges.

For an edge e, $o_e^t(l)$ is the probability of observing x_e at time t given that the edge is in state $W_e^t = w_l$. Formally, we have by (52),

$$\begin{aligned} o_e^t(l) &= P\left(X_e^t = x_e^t \mid W_e^t = w_l\right) & (54) \\ &= \frac{\exp\{-w_l \phi(e,t)\}}{\sum_{k=1}^{|w|} \exp\{-w_k \phi(e,t)\}}. & (55) \end{aligned}$$

In our model, we assume the observation probabilities to be fixed and do not reestimate them.

We denote our set of parameters $\Psi_e^{(n)} = \{Q_h^{(n)}, \alpha_{e,h}^{(n)}\}$. The superscript (n) indicates the nth estimate of Ψ. Q_h is a matrix of weight state transition probabilities with elements $Q_h(l,m)$ for topic h, and $\alpha_{e,h}$ are the mixture proportions for edge e as defined in Section 4.5. We assume a uniform initial state probability $\pi_k = 1/|w|$ for all k using the notation from Section 2.2.1.

We use the notion of active topic Y_e^t for edge e by the equation,

$$P(W^{t+1} = w_m | W_e^t = w_l, Y_e^t = h) = Q_h(l,m) \tag{56}$$

as defined in Section 4.7.

We can now formulate the forwards probabilities, incorporating the topic of an edge, as,

$$\begin{aligned}
f_e^t(m,h) &= P(X_e^{1:t} = x_e^{1:t}, W_e^t = w_m \mid \Psi_e^{(n)}) && (57)\\
&= P(x_e^t \mid w_m) \sum_{l=1}^{M} \sum_{h'=1}^{H} P(Y_e^t = h \mid \alpha^{(n)}) && (58)\\
&\times P(w_m \mid W_e^{t-1} = w_l, Y_e^{t-1} = h') \times f_e^{t-1}(l,h') && (59)\\
&= o_e^t(m) \sum_{l=1}^{|w|} \sum_{h'=1}^{|\mathcal{H}|} f_e^{t-1}(l,h') \alpha_{e,h}^{(n)} Q_{h'}^{(n)}(l,m). && (60)
\end{aligned}$$

When going from (57) to (58), we make use of the Markov assumption.

We introduce the backwards probabilities as,

$$\begin{aligned}
b_e^t(m,h) &= P(X_e^{(t+1):T} = x_e^{(t+1):T} \mid W_e^t = w_m, Y_e^t = h, \Psi_e^{(n)}) && (61)\\
&= \sum_{l=1}^{M} \sum_{h'=1}^{H} P(X_e^{t+1} \mid W_e^{t+1} = w_l) P(Y_e^{t+1} = h' \mid \alpha^{(n)}) && (62)\\
&\times P(W_e^{t+1} = w_l \mid w_m, Y_e^t = h) \times b_e^{t+1}(l,h') && (63)\\
&= \sum_{l=1}^{M} \sum_{h'=1}^{H} o_e^{t+1}(l) b_e^{t+1}(l,h') \alpha_{h'}^{(n)} Q_h^{(n)}(m,l) && (64)
\end{aligned}$$

In the expectation step of the EM algorithm, we compute the log-likelihood, i.e $L(\Psi) = \ln P(x^{1:T}, \Omega^{1:T} \mid \Psi)$, with $\Omega^t = (W^t, Y^t)$ the state of the model including active topic. In the maximization step, we wish to find the parameters that maximizes $L(\Psi)$. With the procedure of Borman [12], we transform the problem using the likelihood term $\mathcal{L}(\Psi; \Psi^{(n)})$, resulting in the algorithm,

$$\text{E-step:} \qquad \mathcal{L}(\Psi; \Psi^{(n)}) = E_\Omega \left[\ln P \left(x^{1:T}, \Omega^{1:T} \mid \Psi(1:T) \right) \right] \tag{65}$$

$$\text{M-step:} \qquad \Psi^{(n+1)} = \arg\max_\Psi (\ln P(\Psi) + \mathcal{L}(\Psi; \Psi^{(n)}) \tag{66}$$

In order to perform the steps above, we calculate the expected number of transitions between each pair of states, given the observation sequence (and the current estimate of parameters). To compute the expectation, we first need to compute the probability of observing one single transition from one state (l,h) to another state (m,h') at a single time point, given the current estimate of parameters, and the observation sequence,

$$\xi_e^t(l,m,h,h') = P\left(\Omega_e^t = (w_l, h), \Omega_e^{t+1} = (w_{l'}, h') \mid x_e^{1:T}, \Psi^{(n)}\right) \tag{67}$$

From (54)–(64) and (67) we get,

$$\xi_e^t(l,m,h,h') \propto f_e^t(l,h) o_e^{t+1}(m) b_e^{t+1}(m,h') \alpha_{e,h'}^{(n)} Q_h^{(n)}(l,m) \tag{68}$$

Using ξ_e^t as defined in (68), we can redefine the likelihood term to maximize as,

$$\mathcal{L}(\Psi; \Psi^{(n)}) = \sum_{e \in E} \sum_{t=1}^{T-1} E_{\xi_e^t}\left[\ln Q_h(l,m) + \ln \alpha_{e,h'}\right] \tag{69}$$

and formulate the maximization problem as,

$$\Psi^{(n+1)} = \underset{\Psi=(Q_h,\alpha_{e,h})}{\arg\max}\ (\ln P(\Psi) + \mathcal{L}(\Psi; \Psi^{(n)})) \tag{70}$$

$$\sum_m Q_h(l,m) = 1,\ \forall h \tag{71}$$

$$\sum_h \alpha_{e,h} = 1,\ \forall e \tag{72}$$

The solution to the maximization problem gives us the update equations for the model parameters.

4.11 MAP estimate of weight sequence

The maximum-likelihood estimate of the edge weight sequences can be computed using the forward-backward iterates defined in the previous section. The ML estimate is in a broad sense the most likely *sequence*, emphasizing transitions rather than states at a particular time. If we want a good estimate of the weight at a certain time t, the MAP estimate of the weight sequence is a better choice. Here we define that sequence using the previous definition,

$$F_e^t(m) = \max_{l=1,\ldots,M} \left[Q_e(l,m) F_e^{t-1}(l) \right] \cdot o_e^t(m) \tag{73}$$

$$B_e^t(m) = \max_{l=1,\ldots,M} \left[Q_e(m,l) B_e^{t+1}(l) o_e^{t+1}(l) \right] \tag{74}$$

$$w_{MAP,e}^t = \arg\max_{m=1,\ldots M} \left(F_e^t(m) B_e^t(m) \right) \tag{75}$$

5 Applications

The uses of a time-varying measure of query interactions are many and varied. Examples include search result ranking, suggestions for related queries, keyword suggestion and statistical applications.

Regarding suggestions for related queries, the idea is that when a user searches for a query q, the system should suggest a list of queries q_1,...,q_n that are related to q based on how strongly they interact at the time of search. Such a system is similar to the application proposed by Guo et al. [19], but with the advantage of being aware not just of intent, but of time.

A second category of use is made up of statistical applications. By analyzing interaction sequences for query pairs, hidden patterns in user behaviour can be discovered. An example of such patterns are described in the experiments Section 6.3. Such an application would not likely need to run in an online fashion, but rather once for each dataset.

A third example is that of keyword suggestion, tested in this thesis and described below.

5.1 Keyword suggestions

In search engines, keywords are often used to summarize documents or boost their rank in the result list, cluster them, etc. Traditionally, documents are tagged with keywords either manually or by using keyword extraction software. Both approaches, however, neglect the search behavior of users and the terms that are actually used to find a document.

In Section 6.5, we explore using query interaction dynamics as part of a search-driven and intent-aware approach to keyword suggestion for new documents. We argue that in order for a keyword to be relevant, it should be used in practice to find a document. Our approach takes user intent into account by considering past searches reaching similar documents. The model we present is also sensitive to changes in intent by incorporating the dynamics of user search behavior.

6 Experiments

In order to verify that our model is capable of identifying the dynamic interactions of queries in practice, we have conducted a series of experiments. First we evaluate the performance of our model on synthetically generated data in Section 6.1.

In Section 6.2 we describe the setting in which we perform experiments on real-world data. This setting applies for the subsequent sections in which we perform evaluations first qualitatively and then quantitatively. The quantitative evaluation of our model is set in an application to keyword suggestions for documents in search engines.

In a final test in Section 6.6 we evaluate our model on the well-known Enron email dataset.

6.1 Evaluation on synthetic data

As a first experiment, we test the performance of our model, and the implementation of it, on synthetically generated data. The data is generated according to the generative process described in Section 4.7. We compare our performance to that of NETGEM, a related model handling only pairwise interactions.

We start off generating a random hypergraph, $H = (\mathcal{V}, \mathcal{E})$ using the procedure described by Ghosal [18]. A copy of the hypergraph is then converted to a pairwise (simple) graph $G = (\mathcal{V}, \mathcal{E}^P)$ by connecting all pair of nodes (n_j, n_k) of every hyperedge e with an edge $e^P = (j,k)$. This is to enable comparison with NETGEM. Next, any duplicate edges are removed.

We generate two types of hypergraphs in this procedure, one with uniform edge cardinality $k = 3$ and one with random cardinality $k \leq 4$. In the general case, the cardinality of hyperedges follow the distribution $P(k = 2) = 0.70$, $P(k = 3) = 0.25$, $P(k = 4) = 0.05$, $P(k = i) = 0, i \notin \{2,3,4\}$.

Then we generate random parameters, the transition matrices Q_h and the mixtures $\alpha_{e,h}$. These are now assumed to be known throughout the test and form the origin of the generative process.

We generate a weight sequence W_e^t and observations x_i^t for the hypergraph using the known parameters Q_h and $\alpha_{e,h}$. We construct one instance of our model using the hypergraph H and one using the pairwise graph G. We perform inference on both of the models using the observations x and the

Table 3: *Error in inferred weights on synthetic data. k is the cardinality of hyperedges. The error is the percentage of weights in the sequences inferred by the models that differed from the true sequence.*

Model	Error, $k = 3$	Error, $k \leq 4$
NETGEM	66%	37%
Our model	26%	30%

two graphs. This results in two weight sequences, $\tilde{W}_e^t$ and $\tilde{W}_e^{P,t}$, one inferred from each model, to compare to the original weight sequence, W^t.

Since the pairwise graph has more and different edges than the hypergraph used as the ground truth, we need to infer a weight sequence for the corresponding hypergraph using the pairwise weight sequence. We do this by using one of the simplest decision rules - majority vote. For each pairwise edge e_i making up hyperedge e_j, we perform majority vote. Furthermore, we compute Fleiss' Kappa [16] , κ_F as a measure of the agreement between edges in the vote.

As a measure of the error of the inferred weight sequence, we count the number of elements that differ from the truth in sequences W_e and $\tilde{W}_e$. For the experiment we use weights $W = \{-1{,}0{,}1\}, N = 20$ nodes, $T = 100$ time points, $H = 20$ topics, and $E = 50$ hyperedges. In table 3, we present the results of comparing performance for pair-wise graphs (NETGEM) and hypergraphs on synthetically generated data. The average Fleiss kappa was $\kappa_F = -0.01$ which indicates poor agreement in the majority vote [16].

From table 3, we can clearly see that for data that naturally lends itself to a hypergraph representation, our model outperforms the graph based method. For 3-hypergraphs the difference is larger in the general case because every edge has more than two nodes which means that the pairwise representation will always have to use majority vote.

6.2 Experimental setting

In our experiments, we have used query logs and a search index containing document texts from Västra Götalandsregionen, a county council in Sweden. Typical queries include "job vacancies", "smoking" and various disease names. The query logs contains click-through documents, or at least links to them, which makes the logs ideal for use with the topic model used in this project, see Section 2.6.

The logs comprise, in unfiltered form, around 12 000 000 query instances and 250 000 documents. The data was collected over nearly two years. In this setting we take a query instance to be a triple of 1) query words, 2) time and 3) click-through document URL. Together with the query logs is a search index that contains the text bodies of the documents that are clicked on. These texts are complete sentences with punctuation etc. Before doing any filtering, there are around 250 000 documents in the index.

For the experiments we kept only documents of length > 1000 characters and removed any document which hadn't been clicked on. This left around 20 000 documents. Furthermore, we kept only query instances for which click-through documents were in the filtered document set. This left around 1 000 000 query instances and 18 000 unique query words.

In the topic model we use, LDA, it is assumed that the ordering of words in a text isn't relevant. Instead we make the usual assumption [11] that a document is simply a bag of words. Because of this, we perform some preprocessing steps, removing any punctuation signs, removing stop words and making all letters lowercase.

Using the graph construction procedure described in Section 4.8 we can build an initial, static representation of query interactions, used as input to our model. An example of such a graph can be seen in Figure 7.

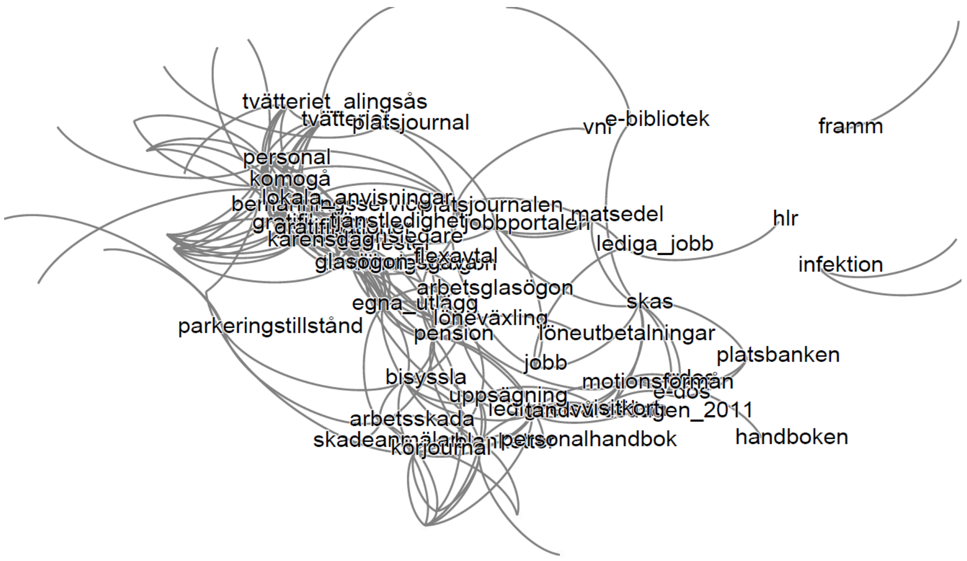


Figure 7: *Part of a static interaction graph used as input to our model. The words in the graph are queries that users have searched for and functions as nodes in the graph. Edges are between nodes of similar topic according to LDA. The edges with no nodes at the ends simply exist because this figure shows only part of a larger graph.*

6.3 Qualitative evaluation of interaction weight sequences

In this experiment we want to examine the results of our model in a qualitative fashion using the data described in the previous section. We trained our model on 75% of the data, sampled uniformly from the entire data set. The remaining 25% are considered the test set. Parameter estimation was performed, giving topic distribution $\alpha_{e,h}$ and class transition probability Q_h as results. These results were then used to infer query interaction dynamics on the test set.

With Q_h and $\alpha_{e,h}$ estimated in training, we have calculated the edge transition probability, Q_e for each edge of the test set according to (43). Using Q_e, we calculate the weight sequence MAP estimate for each edge of the test set, giving the most probable weight at each time step. Details on the MAP estimation procedure can be found in Section 4.11.

To briefly examine the quality of the weight sequences produced by the trained model, we have examined thetop 100 weight sequences having the highest variance. In Figure 8, we see an example weight sequence for the edge between queries "PM" and "ESBL". PM stands for promemoria in Swedish, and means an organizational directive in this setting. ESBL stands for Extended Spectrum Beta-Lactamase and is an enzyme breaking some types of down antibiotics. In general, we do not expect these queries to have a strong relationship since PM is something very general and ESBL something very specific. However, in january 2011 and august 2011 in the left figure, we note that the interaction is strong. Studying documents sent out by the organization, we can see that PMs on the subject of ESBL was sent out during the times of strong interaction.

We note that where both frequencies increase or both decrease, we see a high correlation. We also note that where the frequencies diverge, the output signifies a negative correlation. In general, we expect to see high correlation at a given time for queries that are of similar topic and of similar frequency at that time. The example with PM and ESBL indicates that our model is capable of capturing such events.

6.4 Qualitative comparison with KELLER

We compare our model against KELLER [33], a method for reverse-engineering dynamic interactions between genes based on gene expressions. In our setting, genes correspond to queries and gene expressions to query frequencies. KELLER does not consider the impact of functional classes (topics), so we expect the proposed model to outperform KELLER in cases where query

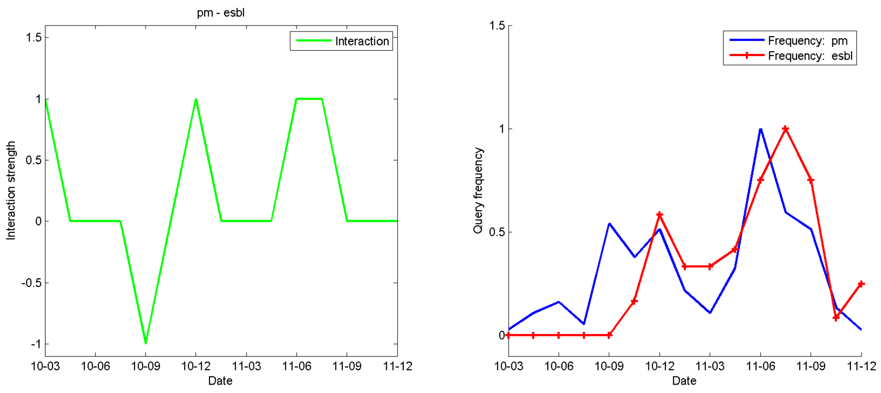


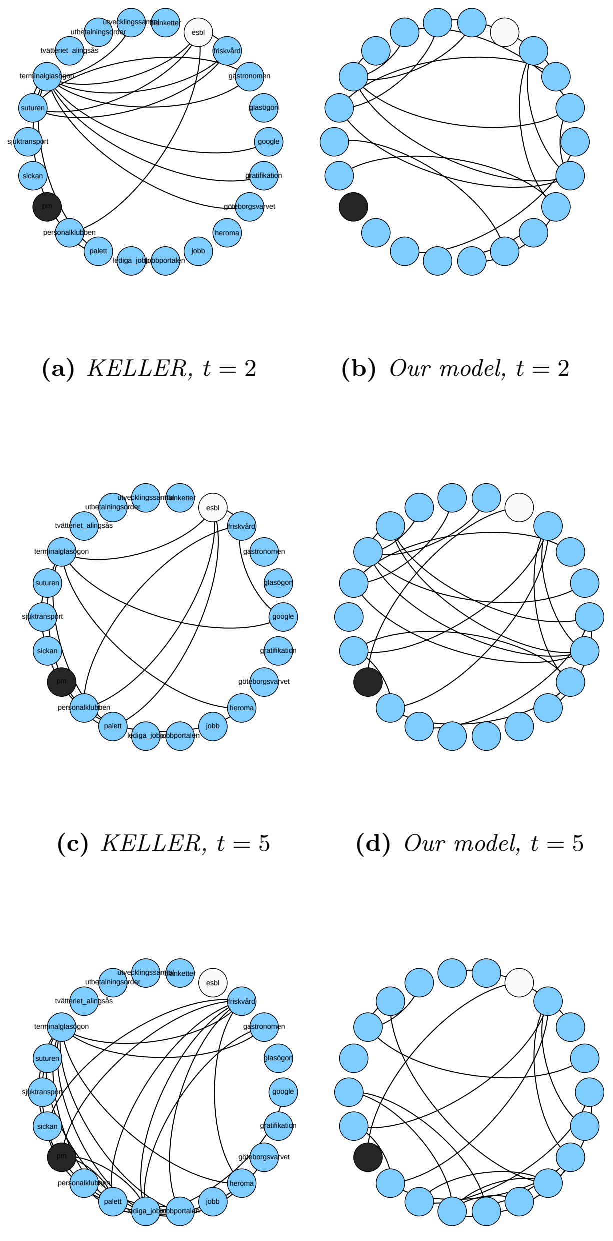
Figure 8: *Test results for query data. To the left is the interaction weight sequence for queries PM and ESBL. PM is an organizational directive and ESBL an enzyme breaking down some types of antibiotics. At times of high correlation, PMs about ESBL were found out to have been sent out in the organization. To the right are the query expressions for the two queries.*

interactions depend strongly on topic. Because KELLER is a model only considering pairwise interactions, we limit our own model to consider only edges of cardinality 2 in this experiment.

For the comparison we use a subset of our data consisting of 22 queries, the top 20 with highest variance in weight sequences, and the two queries *PM* and *ESBL*. The input to KELLER is made up solely of the instance counts for each query. From KELLER we get one graph per time period, in this experiment 15 in total. From our model we get weight sequences for each edge. We build a graph for each time period t by adding the edges e having interaction weight $w_e^t \neq 0$. This enables us to compare the evolving graphs directly.

In Figure 9 on page 40, the networks are presented at 3 time periods. We have selected three periods of interest in connection to the example with *PM* and *ESBL* from the previous section. At time 2, we can see that neither of the models have an edge between *PM* and *ESBL*, indicating that the two queries don't interact at all. For our model, this is expected from Figure 8. At the other two time points shown, our model captures the strong interaction between the two queries and KELLER does not.

Another aspect of the graphs in Figure 9 is that we can note very drastic changes in the graphs produced by KELLER. Almost all edges change over the course of the three time points, (note however that they are not in direct sequence). This goes against our assumptions that changes in interactions



(a) *KELLER, $t = 2$* **(b)** *Our model, $t = 2$*

(c) *KELLER, $t = 5$* **(d)** *Our model, $t = 5$*

(e) *KELLER, $t = 11$* **(f)** *Our model, $t = 11$*

Figure 9: *Graph evolution comparison between KELLER and our model. Nodes corresponds to queries and edges to interactions. Two of the nodes are marked,* esbl *(white) and* pm *(black). An edge (i,j) indicates that queries q_i and q_j are interacting.*

are smooth and slow.

KELLER has an unexpected problem with creating graphs where single nodes have a very high order, as can be seen at time 2 and 11. This is likely caused by these node having a very high expression compared to other nodes at that time. This is not the type of event that we want to capture, since we are more interested in the strength of interactions rather that of expressions. Furthermore, we note that the interactions our model identifies has better correspondence with topic than KELLER. This manifests in KELLER having edges between seemingly totally unrelated queries, as can be spotted when examining graphs with query labels at the nodes.

6.5 Evaluation of keyword suggestion

In this section we use the results of our model trained on the query data training set to suggest keywords for new documents. We will now formalize what we take to be a suggestion, and what an ideal suggestion is. A high-level view of the keyword suggestion process can be seen in Figure 10.

As stated in Section 5.1, we want to suggest keywords that are actually used as search queries. This would make the keywords relevant in terms of result ranking and query matching. The underlying idea for the suggestions we make is to make use of the query logs in the training set and suggest keywords that have been used as queries.

In this application we construct the training and testing sets by partitioning the data chronologically. Also, we make the training set contain 90% of the query instances. To do this, we sort the instances making up the total query log by time and take the first 90% of the to be the training sets and the remainder the test set.

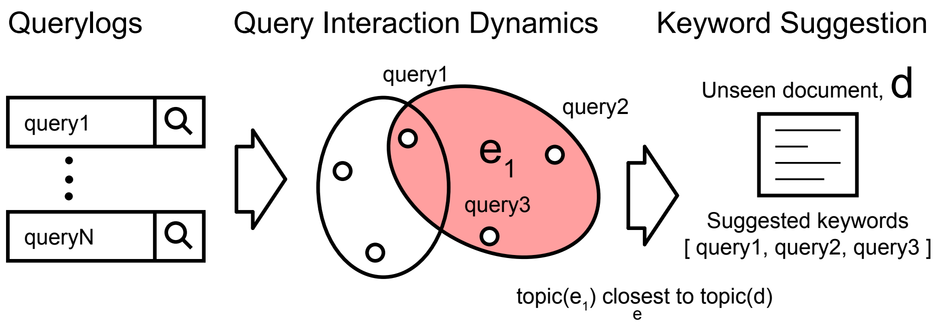


Figure 10: *Simplified view of the keyword suggestion process.*

Table 4: *Algorithm for keyword suggestion*

```
Require: $Q^{Tr}$ = training query logs with instances of time $\in [1,T-1]$
Require: $Q$ = testing query logs with instances of time $T$
Require: $D^{Tr}$ = training documents
Require: $D$ = testing documents such that $D \subseteq D^{Tr}$
 1: Construct graph $G = (\mathcal{V}, \mathcal{E})$ of training queries
 2: Train model on $G$, and $Q^{Tr}$
 3: for all $e$ in $\mathcal{E}$ do
 4:     Fetch mixture $\alpha_e$ from training results
 5:     Calculate MAP weight sequences $w_e$
 6: for all documents $d$ in $D$ do
 7:     Find $e \in \mathcal{E}$ with minimal $D_{KL}(\alpha_e||A_d)$ s.t. $|w_e^{T-1}| > 0$
 8:     Suggest keywords $K_d^T = \{q : q \in e\}$
```

Remember that in our model we partition data into intervals denoted by t ranging from 1,...,T in order model the data as a HMM. In this application, we use $T-1$ partitions for the training set and one for the test set.

We define a keyword suggestion for document d at time t as a set of queries $K_d^t = \{q_{K,1}, ..., q_{K,m}\}$. Please note that we only suggest keywords that have been used as queries in the training set, not any words occurring in documents. Here we differ from for example keyword extraction software [22].

Formally we define,

$$K_d^t = \{q : q \in e, e = \arg\min_{e \in \mathcal{E}} [D_{KL}(\alpha_e||A_d)] \text{ s.t. } |w_e^{t-1}| > 0\}, \tag{76}$$

where A_d is the topic distribution for document d and α_e is the topic distribution for edge e. $D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$ is the usual Kullback-Leibler divergence [23]. w_e^{t-1} is the weight of edge e at time $t-1$.

In words, the above definition means that the queries we suggest for keywords are those making up the edge with topic distribution closest to that of the document in question. The algorithm for suggesting keyword is described in pseudo-code in table 4.

To be able to evaluate our suggestions, we define for a document d a set $Q_d^t = \{q_{d,1}, ..., q_{d,n}\}$ where Q_d^t is the set of queries used to access d in the time interval t. We consider K_d^t a successful suggestion of keywords for d at time t if

$$Q_d^t \subseteq K_d^t \tag{77}$$

or in other words, if we at least suggest as keywords all queries that are used

Table 5: *Results for keyword suggestion on query dataset. A successful suggestion of keywords for a document is taken to be one where at least one of the keywords has been used as a query to reach document.*

Model	Avg. successful suggestions	Avg. recall
Our model	66%	0.62

to reach the document.

This definition suggests that we can evaluate suggestions in terms of recall r defined by,

$$r(d) = \frac{TP}{TP + FN} \tag{78}$$

where TP, true positives, in this case is *the number of queries used to reach document d AND suggested as keywords.* FN are the false negatives, *the number of queries used to reach document d and NOT suggested as keywords.*

In addition to the recall calculation, we also simply count the number of documents that are given successful suggestions according to (77) and calculate a percentage. This is done for every document in the test set. Both the recall and the percentage of successful suggestions averaged over 10 tests are presented in table 5.

We see that for 66% of documents, our algorithm is successful in suggesting keywords that are used as queries to reach the document. One should also note that this indicates predictive capabilities since suggestions are made for unseen data.

6.6 Evaluation on Enron email networks

In the last experiment, we apply our model to a social application, rather than to one of search queries. We use the publicly available email data set from the Enron corporation to compare our model with the original NETGEM model as well as the KELLER model. The Enron dataset is made up of data on employees' sent and received e-mail. Here we use data on mail that is sent within the Enron corporation between 2000-11-07 and 2002-06-12.

A pairwise, static graph, G is created, representing the email networks, by inserting one node per employee and an edge $e = (i,j)$ if at least one email has been sent from employee i to employee j during the entire measurement period. Employees who haven't sent any emails are neglected, leaving a total of $N = 135$ nodes and employees. Out of these we select the 30 most

Table 6: *Enron test comparison. The figures indicate precision and recall for weight sequences inferred using the models in the table compared with the true sequences of email activity.*

	Precision	Recall
KELLER	0.24	0.14
Our model	0.55	0.51

active people in the network. The graph is then converted to a hypergraph H using Bron-Kerbosch algorithm [13]. H is taken as the base graph, part of the input to our model.

We partition the data into 12 different time periods, each representing a month, $\{T_1,T_2, ...,T_{12}\}$ based on the date where the mail was sent. The node expression x_i^t for person i at time t is taken as the total number of emails sent in the period $[T_t, T_{t+1}]$. For convenience, we also define $S_{i,j}^t$ to be the number of emails sent from person i to j in time period $[T_t, T_{t+1}]$.

Since we know how many emails have been sent during each time period, we can construct a *true* weight sequence to compare to the output of our model. We set the weight of an edge e to 1 at time t if at least one mail has been sent from one of the employees in the edge to another in the edge, or formally

$$W_e^t = \begin{cases} 1, & \text{if } \exists(v_i,v_j) : v_i,v_j \in e, i \neq j, S_{i,j}^t > 0 \\ 0, & \text{otherwise} \end{cases} \tag{79}$$

We compare our model to an existing model for network interaction dynamics, namely KELLER. KELLER is a pairwise model, so as in the synthetic experiment, we run it on the pairwise version of H and perform majority vote on the resulting weight sequence to get weights for the hypergraph. We compare the resulting weight sequence to the true sequence using precision and recall calculated by counting where the weight sequences agree and where not. Recall has been defined earlier, and precision is defined as,

$$p = \frac{TP}{TP + FP} \tag{80}$$

where TP denotes number of true positives, and FP number of false positives.

The results of the experiment are presented in table 6. We note that our model is better than KELLER in terms of both precision and recall.

7 Discussion

From the qualitative evaluation of weight sequences in Section 6.3, we gain confidence that our model is capable of recovering hidden interactions between search queries. We successfully identified a real-world event in the test set by looking only at the interaction sequence produced by a model trained on the training dataset. This result shows promise for the use of our model in a practical application.

In the dataset we have used for our tests, we have no notion of *ground truth* since there is no labeled data, no true weight sequences. This is problematic when evaluating accuracy in a quantitative manner. The problem arises from the data itself, because it is very hard, even manually, to identify a true weight sequence for query interactions. In static models, it is easier, because often a manual labeler has an intuitive feel of what queries should be related. Also, one can look at which queries are used to find the same thing. In our case, manual labeling is harder, because of the assumption that interactions are dynamic. There is little intuition for when two queries are more related and when they are less so.

In the comparison with KELLER in Section 6.4, we found that our model produced more intuitive results than KELLER. While this is a vague notion, we can still note specifics like at least finding clear events of interest (PM-ESBL) and the drastic changes in the KELLER graphs. We believe that much of the difference between the two models comes from the fact that KELLER does not consider the impact of topics, while our model does. We expect this to be more important for certain queries than for others, and especially for cases where query expression is low, but the activity of a certain topic is high. We have yet to identify such a case, and this is work for the future.

In another application of our model, for instance to social interactions, manual labeling can be easier. For instance, in email networks, we have a measure of the number of emails sent between people at each time. This is why we tested our model on the Enron email network dataset, presented in Section 6.6. For this data we got good precision and recall, which increases our belief in that our model performs well for data that can be modelled as a hypergraph.

Because of the lack of direct comparison with a labeled set, we chose both to apply the results of our model to a practical application – keyword suggestion and to evaluate the model on synthetically generated data. In the synthetic evaluation we found that our hypergraph based model outperforms an existing graph based model for network interaction dynamics on data that is best represented as a hypergraph, see table 3. The difference was especially large for 3-uniform hypergraphs.

The query data naturally lends itself to a hypergraph representation for several reasons. First, different users typically use similar queries to find the same thing. An example of this are queries, found in the health care dataset used in this project and translated to English, such as “free jobs”, “job portal”, “job bank” etc. Typically, all such queries are related to each other in a grouped fashion, and not pairwise. This motivates the use of hypergraphs. Second, spelling and wording may differ for queries, as well as word order. In the hypergraph representation, all such variations are modelled as one group.

To get an indication of the performance of our model in a practical application, we applied the results of our model to keyword suggestion, see Section 6.5. We found that the proposed algorithm gave good results in terms of recall, and suggested keywords that were used to find documents in practice.

We have done very little discussion of time complexity this far. Both training and inference can be performed in linear time, which indicates that we can use the model for reasonably large scale dataset. One should note that *linear* is in terms of edges and nodes, and not number of query instances. Also, training complexity depends linearly on the number of iterations. In tests, we have found that around 10 iterations have sufficed, but we have no theoretical results backing this up. In all, the time complexity has not been analyzed to any further extent yet, and more tests are needed to confirm the usability of our model.

8 Conclusions & future work

In this thesis we have constructed a probabilistic graphical model for dynamic search query interaction. We have found through tests on synthetic as well as real-world data that our model is capable of recovering hidden interaction dynamics in search query networks.

We have found that our model can be used in an application to document keyword suggestions, and that the suggested keywords are relevant in that they are actually used as search terms.

In synthetic tests, we have found that the hypergraph based model of this thesis outperforms existing graph-based models. In a small test, we also found our model to be capable of identifying dynamic social interactions using the Enron email dataset.

Future work should include theoretical analysis of time complexity to determine the practical usability of our model. A more rigorous set of tests of the quality of query interaction weight sequences is also needed to confirm the accuracy of the model. Another area to explore is that of efficient training algorithms. Instead of the EM-procedure, we could construct a parameter-estimation algorithm based on sampling.

References

[1] A. Ahmed and E. Xing. Timeline: A dynamic hierarchical dirichlet process model for recovering birth/death and evolution of topics in text stream. *Uncertainty in Artificial Intelligence*, 2010.

[2] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res.*, 9:1981–2014, June 2008.

[3] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1):5–43, January 2003.

[4] Adriano Azevedo-Filho and Ross D. Shachter. Laplace's method approximations for probabilistic inference in belief networks with continuous variables. In *UAI*, pages 28–36. Morgan Kaufmann, 1994.

[5] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the 2004 international conference on Current Trends in Database Technology*, EDBT'04, pages 588–596, Berlin, Heidelberg, 2004. Springer-Verlag.

[6] Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 76–85, New York, NY, USA, 2007. ACM.

[7] D. Barber. *Bayesian Reasoning and Machine Learning.* Cambridge University Press, 02-2012 edition, 2012. In press.

[8] C. Berge. *Hypergraphs: Combinatorics of Finite Sets.* North-Holland, 1989.

[9] David M. Blei. Introduction to probabilistic topic models. *Communications of the ACM*, 2011.

[10] David M. Blei and Michael I. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1:121–144, 2005.

[11] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[12] Sean Borman. The expectation maximization algorithm: A short tutorial. unpublished paper. http://www.seanborman.com/publications, 2004.

[13] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[14] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[15] Daniel Fink. A compendium of conjugate priors, 1997.

[16] J.L. Fleiss et al. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.

[17] Wenjie Fu, Le Song, and Eric P. Xing. Dynamic mixed membership blockmodel for evolving networks. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 329–336, New York, NY, USA, 2009. ACM.

[18] Gourab Ghoshal, Vinko Zlatic, Guido Caldarelli, and M. E. J. Newman. Random hypergraphs and their applications, 2009. cite arxiv:0903.0419Comment: 11 pages, 7 figures.

[19] Jiafeng Guo, Xueqi Cheng, Gu Xu, and Xiaofei Zhu. Intent-aware query similarity. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 259–268, New York, NY, USA, 2011. ACM.

[20] Gregor Heinrich. Parameter estimation for text analysis. Technical report, 2004.

[21] Vinay Jethava, Chiranjib Bhattacharyya, Devdatt Dubhashi, and Goutham N. Vemuri. Netgem: Network embedded temporal generative model for gene expression data. *BMC Bioinformatics*, 12(327), 2011.

[22] Jasmeen Kaur and Vishal Gupta. Effective approaches for extraction of keywords. *Journal of Computer Science*, 7(6):144–148, 2010.

[23] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[24] G.r. Sell L.e. Baum. Growth functions for transformations on manifolds. *Pac. J. Math*, 27(2):211–227, 1968.

[25] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit, 2002.

[26] Kevin P. Murphy. An introduction to graphical models. http://www.cs.ubc.ca/~murphyk/Papers/intro_gm.pdf, 2001.

[27] Radford Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[28] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto, September 1993. 144pp.

[29] Radford M. Neal and Geoffrey E. Hinton. *A view of the EM algorithm that justifies incremental, sparse, and other variants*, pages 355–368. MIT Press, Cambridge, MA, USA, 1999.

[30] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed Algorithms for Topic Models. *Journal of Machine Learning Research*, 10:1801–1828, 2009.

[31] J. R. Norris. *Markov chains.* Cambridge Univ. Press, Cambridge, reprinted edition, 1998.

[32] Lawrence R. Rabiner. Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[33] Le Song, Mladen Kolar, and Eric P. Xing. KELLER: estimating time-varying interactions between genes. *Bioinformatics*, 25(12):i128–i136, June 2009.

[34] Mark Steyvers and Tom Griffiths. *Probabilistic Topic Models.* Lawrence Erlbaum Associates, 2007.

[35] Henk C. Tijms. *A first course in stochastic models.* Wiley, New York, 2003.

[36] M.J. Wainwright and M.I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[37] Han Xiao and Thomas Stibor. Efficient collapsed gibbs sampling for latent dirichlet allocation. In *Asian Conference on Machine Learning (ACML)*, volume 13 of *JMLR W&CP*, Japan, 2010. (AR: 31

A Topic Modeling

A.1 Inference via Gibbs Sampling

To use LDA we have to do inference of the posterior distribution of hidden variables given a document,

$$p\left(\theta, \mathbf{z} \mid \mathbf{w}, \alpha, \beta\right) = \frac{p\left(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta\right)}{p(\mathbf{w} \mid \alpha, \beta)} \tag{81}$$

For this application it can be shown that this is intractable to solve exactly. If we, for a moment, omit the hyper parameters from eqn. (81) [20]

$$p(\vec{z} \mid \vec{w}) = \frac{p(\vec{z}, \vec{w})}{p(\vec{w})} = \frac{\prod_{i=1}^{W} p(z_i, w_i)}{\prod_{i=1}^{W} \sum_{k=1}^{K} p(z_i = k, w_i = k)} \tag{82}$$

we can see the main problem which occurs in the denominator where there exists a sum over K^W terms. Using Gibbs sampling this can be done approximately using a collapsed Gibbs sampling algorithm for LDA [37]. Gibbs sampling for the application for LDA means using the fixed collections of words, i.e documents, to guide where the samples of possible topics should be searched for [9]. The derivation of the collapsed Gibbs sampler for LDA is not given here but a description can be found written by G. Heindrich [20].

A.2 Parameter Estimation

For the LDA model, the parameters that need to be estimated is α and β. These parameters should be chosen to maximize the log likelihood of the data, i.e the words.

$$L(\alpha, \beta) = \sum_{d=1}^{M} logp(\vec{(w)}_d \mid \alpha, \beta) \tag{83}$$

This problem is also intractable due to the number of possible combinations of α and β. However, the collapsed Gibbs sampler for LDA described in [20], also solves this step of estimating the parameters in an approximate way.