PROBABILISTIC MODELS FOR FOCUSED WEB CRAWLING

by

Hongyu Liu

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
July 2007

© Copyright by Hongyu Liu, 2007
NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

AVIS:
L’auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l’Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L’auteur conserve la propriété du droit d’auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n’y aura aucun contenu manquant.
DALHOUSIE UNIVERSITY

To comply with the Canadian Privacy Act the National Library of Canada has requested that the following pages be removed from this copy of the thesis:

Preliminary Pages
Examiners Signature Page (pii)
Dalhousie Library Copyright Agreement (piii)

Appendices
Copyright Releases (if applicable)
Table of Contents

List of Tables ................................................. viii
List of Figures ................................................ xi
Abstract ....................................................... xvii
List of Abbreviations and Symbols Used .................. xviii
Acknowledgements ............................................ xix

Chapter 1  Introduction ........................................ 1
  1.1 Challenges of General Search Engines and Crawlers .... 2
  1.2 Focused Crawling ......................................... 3
  1.3 Literature Review ........................................ 4
  1.4 Our Approach ............................................. 10
  1.5 Applying Graphical Probabilistic Models ................. 12
    1.5.1 Directed Graph Models ............................... 12
    1.5.2 Undirected Graph Models ............................ 15
  1.6 Outline of the Thesis ................................... 15

Chapter 2  System Architecture Overview .................. 17
  2.1 Data Collection .......................................... 17
    2.1.1 User Data Collection ................................. 19
    2.1.2 Web Data Collection ................................ 21
  2.2 Pattern Learning ......................................... 28
    2.2.1 Pattern Learning with HMMs ........................ 28
    2.2.2 Pattern Learning with MEMMs and CRFs ............ 32
  2.3 Focused Crawling ....................................... 39
    2.3.1 Efficient Inference ................................ 40
    2.3.2 Calculation of the Priority ......................... 42
Chapter 3 Using HMM to Learn User Browsing Pattern for Focused Crawling

3.1 Concept Graph ........................................ 47
  3.1.1 LSI – Identification of Semantic Content .......... 47
  3.1.2 Clustering ........................................ 49
3.2 User Modeling ......................................... 51
  3.2.1 Parameter Estimation ............................... 51
  3.2.2 Efficient Inference ................................ 52
3.3 Focused Crawling ..................................... 54
  3.3.1 Calculation of the Priority ......................... 55
  3.3.2 Priority Queue Data Structure ..................... 55
  3.3.3 The Algorithm .................................. 55
3.4 Experiments .......................................... 57
  3.4.1 Algorithms for Comparison ......................... 57
  3.4.2 Training Data ................................... 58
  3.4.3 Results .......................................... 59
3.5 Summary ............................................. 67

Chapter 4 Focused Crawling with Maximum Entropy Markov Models

4.1 Maximum Entropy Markov Models (MEMMs) .............. 68
  4.1.1 Maximum Entropy Introduction ..................... 69
  4.1.2 Maximum Entropy Markov Model ................... 72
  4.1.3 Training MEMMs ................................ 74
  4.1.4 Smoothing ....................................... 75
4.2 Parameter Estimation .................................. 77
  4.2.1 Iterative Scaling Algorithms ....................... 77
  4.2.2 Limited-Memory Quasi-Newton Method (L-BFGS) .... 78
4.3 Focused Crawling ..................................... 78
List of Tables

Table 2.1 Summary of Features ................................. 35
Table 6.1 Computational Costs ................................. 125
Table 6.2 Pros (+) and Cons (−) of Three Graphical Models ................................. 127
List of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Typical Search Engine Structure</td>
<td>1</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Regular Crawling and Focused Crawling: Colored squares indicate the Web pages which are visited, while blank ones indicate avoided Web pages</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Context Graph Method for Focused Crawling</td>
<td>9</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>Our Approach: Model Focused Crawling as a Sequential Task, over an underlying chain of hidden states defined by hop distance from targets</td>
<td>11</td>
</tr>
<tr>
<td>Figure 1.5</td>
<td>Graphical model representation of simple HMM (A), MEMM (B) and linear-chain structured CRF (C). The nodes in the graph represent random variables either observed (white) or hidden (colored), and the arcs indicate conditional dependence between random variables</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>System Architecture: Data Collection, Pattern Learning, and Focused Crawling</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>User Data Collection. Double-bordered white nodes represent target pages. (a) Useful and Submit buttons. (b) User visited pages form the local Web graph</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Web Data Collection: it takes specified target pages as input, creates Web graph using Yahoo API, outputs extracted sequences of pages as training data. It consists of two components: Create Web graph and Extract page sequences</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>An example of selection of topic keywords, description and target pages from the Open Directory Project (ODP)</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Create Web graph layer-by-layer using Yahoo API inlink service. White nodes represent target pages, and grey nodes represent pages obtained through backlink service tracing with Yahoo API. Node 0 is the initial target page, and nodes 5 and 7 are target pages that are marked</td>
<td>25</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>A state transition diagram for a HMM with 4 hidden states. Nodes represent hidden states, and arrows represent allowable transitions, i.e., transitions with non-zero probability</td>
<td>29</td>
</tr>
</tbody>
</table>
Figure 2.7  Dependency structure of HMMs on modeling a sequences of Web pages, given state sequence \( s = \{ s_1, s_2, \ldots, s_n \} \) and observation sequence \( o = \{ c_1, c_2, \ldots, c_n \} \) representing the corresponding clusters to which the actual Web pages in the sequence \( \{ page_1, page_2, \ldots, page_n \} \) belong. \( t \) ranges over input positions. Arrow shows dependency (cause) ........................................ 30

Figure 2.8  The transition structure of a Hidden Markov Model for focused crawling with 4 hidden states \( T_0, T_1, T_2, T_3 \). Nodes represent hidden states. Arrows between different hidden states represent allowable transitions. Hidden states generate observations represented by cluster numbers ................................. 31

Figure 2.9  Dependency structure of MEMMs on modeling a sequence of Web pages. Directed graphical model, arrow shows dependency (cause). It defines separate conditional probabilities \( p(s_t|s_{t-1}, o_t) \) at each position \( t \) based on features such as title, keywords, and URL token. \( t \) ranges over input positions .................. 33

Figure 2.10  Graphical structure of CRFs on modeling a sequence of Web pages. This is an undirected graphical model. It defines a single conditional probability \( p(s|o) \) over the entire state sequence \( s \) given the observation sequence \( o \) based on features such as title, keywords, and URL token. \( t \) ranges over input positions ...... 34

Figure 2.11  Flow Chart of Focused Crawling with HMMs/MEMMs/CRFs Models ............................................. 40

Figure 3.1  System Architecture: User Data Collection, User Modelling via Pattern Learning, and Focused Crawling ................................. 47

Figure 3.2  User Modelling via Sequential Pattern Learning. \( C_i \) is the label of cluster \( i \), \( T_j \) is the estimated hidden state ................................. 50

Figure 3.3  Parameter Estimation of HMM ........................................... 51

Figure 3.4  Flow chart of focused crawling using HMM ............................. 54

Figure 3.5  Pseudocode of Crawling Algorithm using HMMs ....................... 56

Figure 3.6  Build context graph. Each circle represents one layer. Targets form layer 0, and layer \( i \) contains all the parents of the nodes in layer \( i - 1 \) ..................... 59

Figure 3.7  Topic \textit{Linux}: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarity of all downloaded pages .............................. 60

x
Figure 3.8  Topic Call for Papers: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarities to the set of target pages of all downloaded pages.

Figure 3.9  Topic Biking: (a) The number of relevant pages within the set of downloaded pages with threshold 0.6. (b) Average maximal similarity of all downloaded pages.

Figure 3.10  Topic Hockey: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarity of all downloaded pages.

Figure 3.11  The effect of different relevance threshold values on topic Linux: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.

Figure 3.12  The effect of different relevance threshold values on topic Call for Papers: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.

Figure 3.13  The effect of different relevance threshold values on topic Hockey: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.

Figure 3.14  The effect of different relevance threshold values on topic Biking: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) The number of relevant pages with threshold 0.5.

Figure 3.15  Comparison of HMM crawler using different training data on topic Linux: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data No.1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data No.3).
Figure 3.16 Comparison of HMM crawler using different training data on topic hockey: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data No.1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data No.3) ............................................................... 66

Figure 4.1 Dependency Graphical structure of MEMM for sequences, given state sequence $s = \{s_1, s_2, ..., s_n\}$ and input sequence $o = \{o_1, o_2, ..., o_n\}$, where $t$ ranges over input positions. Arrow shows dependency (cause) ................................................................. 72

Figure 4.2 Pseudocode of Crawling Algorithm with MEMMs ................. 82

Figure 4.3 Topic Fitnessyoga: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds .................................................. 84

Figure 4.4 Comparisons of different crawl methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Fitnessyoga. (b) Topic Linux ................................................................. 85

Figure 4.5 Topic Linux: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds .......... 86

Figure 4.6 Topic Callforpapers: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds .................................................. 87

Figure 4.7 Topic Balletdance: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds .................................................. 87

Figure 4.8 Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Callforpapers. (b) Topic Balletdance ................................................................. 88

Figure 4.9 Topic Hearthealthy: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds .................................................. 88

Figure 4.10 Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Internetlaw. (b) Topic Hearthealthy ................................................................. 89
Figure 4.11 Topic Skymaps: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

Figure 4.12 Topic Internetlaw: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds.

Figure 4.13 Topic Butterfly: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

Figure 4.14 Topic Linux: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.15 Topic Fitnessyoga: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.16 Topic Balletdance: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.17 Topic Hockey: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.18 Topic Biking: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.19 Topic Butterfly: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.20 Comparisons of different methods: BFS, memm marginal and memm_viterbi on the number of relevant pages. (a) Topic Internetlaw. (b) Topic Skymaps.

xiii
Figure 4.21 Comparisons of different methods: with Word feature only, simmeta-T feature only and the all features combination on the number of relevant pages within pages. (a) Topic Hearthealthy. (b) Topic Callforpapers ........................................... 96

Figure 5.1 An example of undirected graph: \((v_1, v_2, v_3)\) and \((v_3, v_4)\) are maximal cliques .................................................. 99

Figure 5.2 An example of CRFs .................................................. 101

Figure 5.3 Graphical structure of a linear-chain CRF for sequences, given state sequence \(s = s_1, s_2, ..., s_n\) and input sequence \(o = o_1, o_2, ..., o_n\), where \(t\) ranges over input positions. The dashed box over \(o\)'s denotes the sets of observation sequence variables. Although we have shown links only to observations at the same step, the state nodes can depend on observations at any time step .... 101

Figure 5.4 Topic Fitnessyoga: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds .... 110

Figure 5.5 Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. (a) Topic Linux. (b) Topic Fitnessyoga ........................................... 111

Figure 5.6 Topic Internetlaw: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds .... 111

Figure 5.7 Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. a) Topic Internetlaw.(a) Topic Hockey ........................................... 112

Figure 5.8 Topic Callforpapers: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds .... 112

Figure 5.9 Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. (a) Topic Balletdance. (b) Topic Callforpapers ........................................... 113

Figure 5.10 Topic Biking: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds .... 113
Figure 5.11 Topic Butterfly: Comparisons of different methods: BFS, CRF-
marginal and CRF-viterbi on the number of relevant pages in
the set of downloaded pages with different thresholds . . . . . 114

Figure 5.12 Topic Skymaps: Comparisons of different methods: BFS, CRF-
marginal and CRF-viterbi on the number of relevant pages in
the set of downloaded pages with different thresholds . . . . . 114

Figure 5.13 Topic Balletdance: Comparisons of different methods: BFS,
CRF-marginal and CRF-viterbi on the number of relevant pages
in the set of downloaded pages with different thresholds . . . . 115

Figure 5.14 Topic Internetlaw: Comparisons of different methods: with
Word feature only, sim-meta-T feature only and the all fea-
tures combination on the number of relevant pages within the
set of downloaded pages with different thresholds . . . . . . 116

Figure 5.15 Comparisons of different methods: with Word feature only, sim-
meta-T feature only and the all features combination on the
number of relevant pages within the set of downloaded pages.
(a) Topic Linux. (b) Topic Fitnessyoga . . . . . . . . . . . . . . . . . 116

Figure 5.16 Comparisons of different methods: with Word feature only, sim-
meta-T feature only and the all features combination on the
number of relevant pages within the set of downloaded pages.
(a) Topic Balletdance (b) Topic Hockey . . . . . . . . . . . . . . . . 117

Figure 5.17 Comparisons of different methods: with Word feature only, sim-
meta-T feature only and the all features combination on the
number of relevant pages within the set of downloaded pages.
(a) Topic Biking. (b) Topic Skymaps . . . . . . . . . . . . . . . . . 117

Figure 5.18 Comparisons of different methods: with Word feature only, sim-
meta-T feature only and the all features combination on the
number of relevant pages within the set of downloaded pages.
(a) Topic Hearthealthy. (b) Topic Callforpapers . . . . . . . . . . . 118

Figure 5.19 Topic Butterfly: Comparisons of different methods: with Word
feature only, sim-meta-T feature only and the all features com-
bination on the number of relevant pages within the set of down-
loaded pages with different thresholds . . . . . . . . . . . . . . . . 118

Figure 6.1 Topic Linux and Topic Butterfly: Comparisons of different
methods: BFS, MEMM-marginal and CRF-marginal on the
number of relevant pages in the set of downloaded pages . . . . 121
Figure 6.2 Topic Balletdance: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages in the set of downloaded pages. (b) Maximum Average Similarity ........................................ 121

Figure 6.3 Topic Biking and Topic Hockey: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal on the number of relevant pages in the set of downloaded pages .................. 122

Figure 6.4 Topic Fitnessyoga: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity ........................................ 122

Figure 6.5 Topic Hearthealthy and Topic Internetlaw: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal on the number of relevant pages within the set of downloaded pages with different thresholds ........................................ 123

Figure 6.6 Topic Callforpapers: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity ........................................ 123

Figure 6.7 Topic Skymaps: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity ........................................ 124

Figure 6.8 Average Precision Relative to BFS (APR) by CRFs, MEMMs, HMMs, CGS crawling methods. Precision relative to BFS is calculated by Precision using the method to be evaluated over Precision using BFS ........................................ 128

Figure 7.1 Behavior Modeling Problem as a Time-based Sequential Analysis Problem ........................................ 133
Abstract

A focused crawler is an efficient tool used to traverse the Web to gather documents on a specific topic. It can be used to build domain-specific Web search portals and online personalized search tools. Focused crawlers can only use information gleaned from previously crawled pages to estimate the relevance of a newly seen URL. Therefore, good performance depends on powerful modeling of context as well as current observations.

The goal of this research has been to design a robust method for the focused crawling problem, capable of collecting as many pages as possible that are relevant to the given topics. To address this challenge, we propose a new approach for focused crawling to capture sequential patterns along paths leading to targets based on probabilistic models. We model the process of crawling by a walk along an underlying chain of hidden states, defined by hop distance from target pages, from which the actual topics of the documents are observed. When a new document is seen, prediction amounts to estimate the distance of this document from a target.

Within this framework, we further investigate three probabilistic models for focused crawling. With Hidden Markov Models (HMMs), we focus on semantic content analysis with sequential pattern learned from the user's browsing behavior on specific topics. We extend our work to take advantage of richer representations of multiple features extracted from Web pages. With Maximum Entropy Markov Models (MEMMs), we exploit multiple overlapping features, such as anchor text, to represent useful context and form a chain of local classifier models. With Linear-chain Conditional Random Fields (CRFs), a form of undirected graphical models, we further focus on obtaining global optimal solutions along the sequences by taking advantage not only of text content, but also of linkage relations.

We conclude the thesis with an experimental validation and comparison of HMMs, MEMMs and CRFs for focused crawling. The experimental results of our model show significant performance improvement over Best-First crawling (BFC).
List of Abbreviations and Symbols Used

$A$ transition probability matrix
$APR$ Average Precision Relative to BFS
$B$ emission probability matrix
$D$ training data set
$E$ a set of edges in graph $G$
$E_i$ expected value
$G$ a graph
$H(p)$ conditional entropy
$L$ objective function
$N$ the number of sequence in training data
$T_i$ $i$ hops to reach a target
$V$ a set of nodes in graph $G$
$Z(o)$ global normalization factor on observation $o$
$\alpha$ forward probability of state $s$ at time $t$
$\beta$ backward probability of state $s$ at time $t$
$\gamma$ confidence threshold
$\lambda_i$ weight for the $i^{th}$ feature function
$\pi$ initial state distribution matrix
$\sigma$ Maximum Average Similarity
$\theta$ HMM parameters
$\bar{E}_i$ empirical value
$f_i$ the $i^{th}$ feature function
$k$ the number of hidden states
$m$ the number of clusters/feature functions
$o \in O$ observable input and input space
$s \in S$ state output and output space
$w_t$ web page seen at time $t$
$z(o_t)$ per-state normalization factor at time $t$
Acknowledgements

First of all, I would like to express my deepest and most sincere gratitude to my supervisors, Dr. Evangelos Milios and Dr. Jeannette Janssen, not only for their great supervision and support in research over the past six years, but also for all the guidance, assistance and encouragement in helping me through the difficult times. I am truly grateful to have them as my supervisors. Without them, this thesis would have not been accomplished.

I would also like to thank my committee members, Dr. Vlado Keselj and Dr. Nur Zincir-Heywood, for their invaluable comments, advice and help at all stages of this thesis.

I want to take the opportunity to give special thanks to the Wanzel family, especially Mrs. Lorna Wanzel and Grandma Mrs. Edith Price, who have become a second family to me in Canada.

I also want to thank my friends Xiaofen Zheng, Scott Sherrill-Mix, Bin He and Zhen Zhen, for their good friendship and support. In addition, I would like to thank all colleagues in the playground, group members in MoMiNIS and Lithel groups, making my study experience at Dalhousie University one of the most memorable experiences of my life.

Last, but certainly not least, I would like to thank my whole family. Throughout my Ph.D. studies, their endless love, support, and patience have been truly inspirational.
Chapter 1

Introduction

With the exponential growth of information on the World Wide Web, there is great demand for efficient and effective approaches to organize and retrieve the information available. Nowadays, more and more people are using search engines to search information and access documents from the Web. A search engine typically consists of four main components: Web Page Retrieval System, Web Page Indexing System (Indexer), Crawler, and a Web Page Repository (Fig. 1.1).

![Diagram of Search Engine Structure]

Figure 1.1: Typical Search Engine Structure

At the back end of the search engine, the Crawler is used to download Web pages from the Internet automatically, and send them to the Web Page Indexing System (Indexer). The Indexer extracts the keywords and links from each page, and stores the mappings from keywords to URLs (documents) in the Repository. In the front end of the search engine, when a user submits a query, the Web Page Retrieval
System receives keyword search terms from the user to find matches in the Web Page Repository. The search engine uses page-ranking algorithms to rank the pages, and returns the results to the user. Therefore, maintaining high coverage, high quality, and fresh data in the Repository is very important for the search engine.

1.1 Challenges of General Search Engines and Crawlers

With the exploding volume of the Web due to a large number of Web users with different search needs, the current search engines are not always sufficient to satisfy all needs. For example, search engines may find it difficult to refresh their indexes frequently enough to keep up with the pace of the changing Web. As well, a topic-oriented keyword search is more likely to yield thousands of relevant results based on only keywords specified in the query, which means that a user still needs to go through a selection and browsing process before finding the pages most relevant to him/her. On the other hand, popular topical pages cannot fit every user's interests equally well. Therefore, generic search engines will become less appealing to a user with specific information needs for which the proper search keywords are not well defined and difficult to come up with.

Therefore, the imbalance between the rapid growth of the Web and the limitations of network and storage resources pose many challenges for search engines and generic crawlers. First, the Web is huge. As we have seen, a page needs to be crawled and indexed in the repository before being retrieved by the user. Google, which is the biggest search engine in the world, indexes about 8.1 billion pages [34]. However, compared to the size of the Web, this is a very small percentage. A 2005 study [35] queried the Google, MSN, Yahoo!, and Ask Jeeves search engines and estimated the World Wide Web to contain over 11.5 billion pages in the publicly indexable Web as of January 2005. However, the estimates are for the "visible" web only, URLs that search engines can easily reach. The "invisible" or "deep" web refers to content locked behind databases or other systems that search engines haven't extracted. It was estimated that the deep Web contained 550 billion individual documents in the past [6]. Some estimate that search engines only reach 16% of the Web [74]. Second, the Web is dynamic. Pages and links are constantly being added, updated or deleted. For example, in the .com domain, half the pages are gone in 10 days. About 23%
of pages change daily [27]. Third, the Web is open. Anyone can post pages on any topics at any time.

Hence, to provide sufficient services to satisfy all needs, general search engines and crawlers are being evolved. There are many possible evolution paths in the market, such as personalization of search engines, for example, MyAsk, Google Personalized Search, My Yahoo Search; localization of search engines, for example, Google Local, Yahoo Local, Citysearch; topic-specific search engines and portals, for example, Kosmix, IMDB, Scirus, Citeeseer; and others, for example, Web 2.0, Blog. To achieve the goals of the topic-specific search tools, focused crawling is considered as a fundamental method to collect only pages relevant to specific topics. An application of extending digital libraries with a focused crawler can be found in [58].

1.2 Focused Crawling

The success of topic-specific search tools depends on the ability to locate topic-specific pages on the Web while using limited storage and network resources. This can be achieved if the Web is explored by means of a focused crawler. A focused crawler is a crawler that is designed to traverse a subset of the Web for gathering only documents on a specific topic, instead of searching the whole Web exhaustively.

![Diagram of Regular and Focused Crawling]

Figure 1.2: Regular Crawling and Focused Crawling: Colored squares indicate the Web pages which are visited, while blank ones indicate avoided Web pages.

In order to find pages of a particular type or on a particular topic, focused crawlers aim to identify links that are likely to lead to target documents, and avoid links to off-topic branches. As shown in Fig. 1.2, regular crawlers (or general-purpose crawlers) use a Breadth-First search strategy in order to download as many pages as possible,
while focused crawlers selectively choose links that quickly lead to target pages. As a result, with a small investment of network resources, a focused crawler is expected to collect high-quality domain-specific documents from the Web with respectable coverage at a rapid rate.

The challenge in designing a focused crawler is to predict which links lead to target pages, since the Web is noisy and multi-topic. Sometimes, relevant pages link to other relevant ones. However, it is very common that pages which are not on topic lead to topical pages indirectly. According to a study [7], most relevant pages are separated by irrelevant pages, the number of which ranges from at least 1, to a maximum of 12, commonly 5. The common approach to focused crawling is to use information gleaned from previously crawled pages to estimate the relevance of a newly seen URL. The effectiveness of the focused crawler depends on the accuracy of this estimation process.

1.3 Literature Review

A variety of methods for focused crawling have been developed. Although the term focused crawler appears to have been first coined by Chakrabarti et al. (1999) in [17], the concept of prioritizing unvisited URLs on the crawl frontier to accomplish specific searching goals was developed a few years earlier. Fish-Search [13] by De Bra et al. (1994) and Shark-Search [36] by Hersovici et al. (1998) were some of the earliest algorithms for crawling for pages with keywords specified in the query. In Fish-Search, the Web is crawled by a team of crawlers, which are viewed as a school of fish, based on whether the page is relevant or not. If a “fish” finds a relevant page based on keywords specified in the query, it continues looking by following more links from that page. If the page is not relevant, its child links receive a low preferential value. Specifically, each crawler takes as input a seed URL and a search query, and uses an arbitrary number width to select the number of children which are added to the frontier. For a relevant node, the first $a \times width$ children ($a = 1.5$) will be selected and added to the head of the frontier with the highest potential score 1, while for an irrelevant node, only the first width children are added to the frontier right after the last child of a relevant node and assigned with a lower potential score 0.5. The rest of the children are given the lowest potential score 0 and added to the tail (to
be handled only if time permits). Therefore, the limitations of this method are that there is very low differentiation of the priority of pages in the frontier, it does not discover “enough” relevant search directions, and the parameters to cut down the number of addressed children is arbitrary.

The underlying assumption of the Fish-Search is to capitalize on the intuition that relevant documents often have relevant neighbours. Shark-Search [36] follows the same assumption but with modifications in two different ways: a child inherits a discounted value of the score of its parent by multiplying that score by a decay factor, and this score is also combined with a value based on the anchor text that surrounds the link in the Web page. It uses a “fuzzy” relevance score, which is the similarity between a page and the query, to calculate the potential score.

Efficient Crawling through URL ordering [20] was proposed by J. Cho et al. (1998). URLs are ordered by an importance metric including:

- Similarity $IS(p, Q)$ – the similarity of a page $p$ to a query string $Q$
- Backlink $IB(p)$ – the count of pages which point to page $p$
- PageRank [14] $IR(p)$ – the PageRank of the page $p$
- Forward Link $IF(p)$ – the number of forward links on a page $p$
- Location $IL(p)$ – the domain of a page $p$ (.gov, .edu, etc.)

They found that if the similarity to a driving query is important, then pursuing URLs that have anchor text that is similar to the query, or have some of the query terms within the URL itself, or have a short link distance to a page that is known to be hot, improves the crawl performance. In general crawls, where high quality pages regardless of topic are sought, they found that PageRank $IR(p)$ is an excellent ordering metric when either pages with many backlinks or with high PageRank are sought. However, PageRank metric performs poorly when the task is to locate pages that are relevant to a particular topic or query [53]. This may be due to the fact that the PageRank score is calculated on a very small, non-random subset of the Web and also that the PageRank algorithm is too general for use in topic-driven tasks.

The focused crawler proposed by Chakrabarti et al. in [17] has three main components: a text classifier which determines the page relevance, a distiller which determines a measure of centrality of crawled pages to determine visit priorities (hub), and a crawler which adds URLs to a max-priority queue with their parent page's score,
and visits them in descending order. To train the text classifier which makes relevance judgments on a document to the pre-defined topic based on example pages provided by the user, it uses a canonical topic taxonomy. The user specifies a number of starting points, browses from these pages, and matches the pages to the best categories while browsing. This categorization is used to train the classifier. When the crawler crawls the Web for the specific topic $c^*$ and sees page $u$, then the visit priority of all the outlinks $v$ of page $u$ are determined by the initial probability $p(c^*|u)$ based on the classification and the hub-authority connectivity analysis of page $u$ governed by the classifier and distiller. The link analysis used in this method only considers links that give immediate benefit around currently-seen pages, which may miss relevant pages linked from irrelevant pages.

They have extended their work in [16] by adding a second classifier, called the apprentice, trained from the crawl database. The apprentice trains online continually as the crawl progresses. The first classifier is now used to obtain required training instances (feedback) for the second one, while the role of assigning visit priorities to unvisited URLs in the crawl frontier is now assigned to the apprentice. When the crawler crawls the Web for the specific topic $c^*$ and sees page $u$, then the visit priorities of each outlink $v$ of page $u$ (link $u \to v$) are no longer determined by the initial probability $p(c^*|u)$, but are calculated by an estimate of $p(c^*|v)$ based on the anchor text of $v$ gained by the Document Object Model or DOM tree of $u$ around defined HREF area for $v$ (i.e., $<a \ href=\ldots>$). The features used by the apprentice are derived from the HREF source page of a page $u$, represented by the Document Object Model or DOM tree (http://www.w3.org/DOM/). That is, the features associated with the link $u \to v$ in their experiments are the term tokens $<t, d>$ representing term $t$ at offset $d$ from $<a \ href=\ldots>$ which links to $v$ ($d \leq 5$) in the DOM tree of $u$. The probabilities $p(c^*|v)$ are estimated according to a Naïve Bayes (NB) classifier in the apprentice. Note that unlike their previous work in [17] discussed above, each outlink $v$ may have a different visit probability, but the estimation of $p(c^*|v)$ is only based on local structure features about $u$ in the vicinity of $v$ in the DOM tag tree, not larger contexts such as paths leading to $u$.

Aggarwal et al. have proposed an “intelligent crawling” framework described in [2]. The method involves looking for specific features in a page to rank the candidate links.
These features include content of in-linking pages, URL tokens, short-range locality information (e.g. parent does not satisfy predicate X but the children do) and sibling information (i.e. number of sibling pages matching the predicate so far). The visit priority value of the candidate link is calculated by a linear combination of these weighted features. The system introduced a concept of "intelligent crawling" where the system has the ability of self-learning, i.e. collecting statistical information during the crawl and adjusting the weight of these features to capture the dominant individual factor at that moment. It is also a generic framework that allows the user to specify the relevant criteria, that is, the user can specify an arbitrary predicate (e.g. keywords, document similarity), anything that can be implemented as a function which determines document relevance to the crawl based on URL, and page content. The "intelligent agents" literature has brought forth several systems such as Letizia [41] and WebWatcher [37]. Menczer and Belew proposed InfoSpiders [51], a collection of autonomous adaptive goal-driven crawler agents which search for pages relevant to the topic, using evolving query keyword vectors and Neural Networks with Q-learning to decide which links to follow. A perceptron represents a vector of real-valued weights. The neural net has a real-valued input for each keyword and a single output unit, and the agent estimates the visit priority of the links in the current page by considering the text surrounding those links. A complete framework to evaluate several crawling strategies is described in [53, 52, 75] including InfoSpiders. In these studies it was found that Best-First approach gives the best performance.

Lexical proximity and link context features have been used extensively in a variety of Web information retrieval and categorization tasks. Besides the work described in [2] above, different heuristics based on text and link context have been applied to guide topical crawlers in the literature. The work proposed by Johnson et al. [38] explores the space of potential strategies. Good strategies evolve based on the text and link structure of the referring pages. The strategies produce a rank function which is a weighted sum of several scoring functions about a page u, such as Hub score, Authority score, Link community scores (inlinks and outlinks), and SVM classification scores of the ancestor pages. A Genetic programming algorithm is used to evolve the weights from training data.

Recent papers [57, 59] systematically studied the use of different classification
algorithms to guide topical crawlers. In [57], the authors compare crawlers built upon three different classifier schemes based on the text content: Naïve Bayes, Neural Network, and SVM. The results show that the SVM classifier performs better than Naïve Bayes and is as good a choice as Neural Network. They continued their work in [59] to further examine link contexts using an SVM classifier. The link context they used consists of three options: simple full-page content, anchor text around a hyperlink on the parent page using arbitrary word size windows $T$, and anchor text derived from the HTML tag tree of the given page. They found that a crawler using a combination of anchor text and the entire parent page performs significantly better than a crawler that depends on just one of those cues.

In all the systems mentioned above, although different techniques and heuristics to guide focused crawling were used, the methods have in common that the underlying paradigm is baseline best-first search strategy, that is, to train a learner with only local features collected about relevant nodes alone from the immediate vicinity of a hyperlink $u \rightarrow v$ (i.e., the parent pages and sibling pages). This is based on the hypothesis that in the Web graph, Web pages on a given topic are more likely to link to those on the same topic (Linkage Locality), and if a Web page points to certain Web pages on a given topic, then it is more likely to point to other pages on the same topic (Sibling Locality). In other words, topics appear clustered in the Web graph [22, 50].

Two important advances have been made beyond the baseline best-first focused crawler including reinforcement learning in [67] and Context Graph algorithm in [24]. Both methods capture longer path information leading to targets rather than relevant nodes alone, as was the case with the baseline crawler. In [67], they modeled crawlers as autonomous agents to learn to choose optimal actions to achieve their goal. In detail, an action is following a particular hyperlink such as $u \rightarrow v$, and the reward for each action is the cumulative estimate $Q$-value of the number of relevant pages starting from $u$ that can be found as the result of following this hyperlink, which is calculated recursively by the immediate reward value from $u$ to $v$, plus the estimate $Q$-value of the immediate successor $v$, discounted by some factor $\gamma < \frac{1}{2}$ per hop. Therefore, the learning process is to train a Naïve Bayes text classifier, for which an instance consists of the features of a single hyperlink such as $u \rightarrow v$ with its
corresponding discretized $Q$-value pair. The features are words from the title and headers ($<$h1$>...<$h1$>$ etc.) of $u$, together with the anchor text around the link $u \rightarrow v$ in $u$. Given a hyperlink, the prediction is to calculate the probabilistic class membership to decide the visit priority.

![Context Graph Method for Focused Crawling](image)

Figure 1.3: Context Graph Method for Focused Crawling

The Context Graph method, proposed by Diligenti et al. [24], uses the text of page $u$ to estimate the link distance from $u$ to some target pages. Their method manually collects paths leading to relevant goals, following backlinks to a certain layer to build up a context graph for the goal page: goal page forms layer 0, and layer $i$ contains all the parents of the nodes in layer $i-1$ (Fig. 1.3). A Naïve Bayes classifier is trained for each layer based on the text of the pages in it. As a new document $u$ is found, it is classified into the layer corresponding to the estimated link distance from $u$ to the target page. Documents classified into layers closer to the target are crawled first, i.e., hyperlinks from the documents in layer 1 have the highest visit priority. The experiments showed that this approach maintained a higher level of relevance in the retrieved Web pages. However, two issues remain to be addressed. One is that the assumption that all pages in a certain layer from a target document belong to the same topic described by a set of terms does not always hold. This is due to the fact that many noisy or irrelevant links exist in Web pages, such as ad links, navigation panels,
or dynamically generated links. As pointed out in [15], the number of distracting
outlinks emerging from even fairly relevant pages has grown substantially since the
eyear days of Web authoring. Therefore, it is difficult to learn an effective classifier
from the pages in each layer based on only physical link layers. Second, there is no
discrimination among different links on a page. Since only a fraction of outlinks from
a page are worth following, offering additional guidance to the crawler based on local
features in the page to rule out some unimportant links can be helpful.

Their work inspired our work to capture longer path information leading to tar-
gets, but in a different way. We believe that in the Web graph, links do not point to
pages at random but reflect the authors’ idea of how to organize information seman-
tically or hierarchically. But in practice, the Web pages are constructed in different
ways, resulting in the number of links (hops) leading to unrelated topics varying
widely. Therefore, it may be difficult to classify all pages in given number of links
(hops) away from a target document to the same topic, as in the Context Graph
method [24]. However, the sequence information along the path leading to the tar-
gets can be further exploited to capture both content and linkage relations. This
provides the motivation to model focused crawling as a sequential task. We model
focused crawler as a ‘surfer or agent, moving from one state to another state based
on the actual pages as observations. The state is the set of defined hop distance from
targets and the observations are actual Web pages.

1.4 Our Approach

Unlike all the systems above, we model focused crawling as a sequential task and learn
the linkage patterns by using a combination of content analysis and link structure of
paths leading to targets. Typical examples of sequential patterns can be found in
topical hierarchies such as Open Directory Project (ODP) or Yahoo directory. They
are built by humans, and do not reflect the structure of the actual Web. In the real
Web, we conjecture that such sequential patterns exist, but in a more implicit way.
In sequential patterns, off-topic pages often lead to highly relevant pages. Due to
the small world nature of the Web [3], links may lead to unrelated topics within an
extremely short radius. At the same time, there exist long paths and large topical
subgraphs where topical coherence persists. This is often the result of webmasters
following some general rules to organize the pages of a Web site semantically or hierarchically. For example, university pages point to department pages, then to pages of faculty members; they rarely point to home gardening pages. Although different university Web sites may have different style and content presentations on the surface, the *Homepage → Department → People → Faculty → Research* is a very common underlying sequential pattern. In other words, off-topic, but semantically related, pages may often lead reliably to relevant pages. When looking for research publications on a specific topic, the crawler may have to traverse pages that are irrelevant to the topic before it reaches highly relevant ones. That is, there is significant amount of information in the sequential patterns of links in addition to content of individual pages. Good performance depends on powerful modeling of the context as well as current observations. Therefore, our hypothesis is that by learning such sequential linkage patterns hidden in the Web graph during crawling, a focused crawler may be able to follow links leading to targets more effectively.

![Web graph](image)

**Figure 1.4:** Our Approach: Model Focused Crawling as a Sequential Task, over an underlying chain of hidden states defined by hop distance from targets.

Our approach is unique in the following important ways. We model focused crawling as a sequential task, over an underlying chain of hidden states, defined by hop distance from targets, from which the actual documents are observed. As shown in Fig. 1.4, suppose page 4 is a target page, and the sequence $o = \text{page1} \rightarrow \text{page2} \rightarrow \text{page3} \rightarrow \text{page4}$ is an observed page sequence leading to the target in the graph. Then $s = T_3, T_2, T_1, T_0$ is the corresponding sequence of hidden states, indicating the number of hops each page in the sequence is away from the target. After training, our system may have learned patterns like "University pages are more likely to lead to Research papers than Sports pages". When a new document is seen, the prediction
is to estimate how many hops this document is away from a target based on the observations so far, and the crawler always follows the most promising link. The use of finite-state Markov models helps represent useful context including not only text content, but also linkage relations. For example, in the path \( p1 \rightarrow p2 \rightarrow p3 \rightarrow w \), the prediction of the link \( p3 \rightarrow w \) is based on the observable text content of \( w \) combined with features from its ancestor sequence \( p1, p2, \) and \( p3 \). Note that this is different from other systems in that we capture context and linkage structure relations along paths as sequential patterns to make predictions.

1.5 Applying Graphical Probabilistic Models

To capture such sequential patterns, we propose to apply probabilistic models for focused crawling. Graphical probabilistic models are a natural tool to deal with conditional probability distribution. The nodes in the graph represent random variables (either observed or hidden), and the arcs indicate conditional dependency between random variables. The graphical model not only gives a graphical representation of the joint probability distributions, but also provides inference methods for estimating the probability distribution. The graph can either be directed, also known as Bayesian Networks or Belief Networks (BNs), or undirected, also called Markov Random Fields (MRFs) or Markov networks.

![Graphical model representation of simple HMM (A), MEMM (B) and linear-chain structured CRF (C).](image)

Figure 1.5: Graphical model representation of simple HMM (A), MEMM (B) and linear-chain structured CRF (C). The nodes in the graph represent random variables either observed (white) or hidden (colored), and the arcs indicate conditional dependence between random variables.

1.5.1 Directed Graph Models

A directed graph is an acyclic graph \( G = \{V, E\} \) where \( V = \{v_1, v_2, \ldots, v_n\} \) is a set of nodes and \( E \) is a set of edges between nodes with directions. Each node \( v_i \) in
the graph represents a random variable and has a set of parent nodes $\text{parent}(v_i)$. The structure of the directed graph represents the conditional dependence between random variables. Namely, the joint probability over variables $V = \{v_1, v_2, ..., v_n\}$ can be calculated as the product of the conditional probability of each variable conditioned on its parents, that is,

$$p(v_1, v_2, ..., v_n) = \prod_{v_i \in V} p(v_i|\text{parent}(v_i)) \quad \text{(1.1)}$$

Hidden Markov Models (HMMs) are one of the most popular directed graphical models for sequential data and have been used very successfully in applications such as speech recognition [65], information extraction [72], image processing [42], and sequence analysis in bioinformatics [26]. Given an observable input sequence $o = o_1, o_2, ..., o_n$, the task is to infer the state output sequence (hidden) $s = s_1, s_2, ..., s_n$ for each position (See Fig. 1.5 (A)). In training an HMM, its model parameters are estimated in order to maximize the joint probability $p(s, o)$ given training state sequences $s = s_1, s_2, ..., s_n$ with observation sequences $o = o_1, o_2, ..., o_n$. Each observation sequence is considered to have been generated by a sequence of state transitions, therefore, it requires strict independence assumptions on the observations. The HMM formulation assumes the first-order Markov assumption, that is, the value of $s_{i+1}$ is independent of $s_{i-1}$ given the value of $s_i$. It also assumes the observation $o_i$ is independent of other states given the value of $s_i$. Therefore, the joint probability can be calculated from the conditional probabilities as follows:

$$p(s, o) = p(s_0) \prod_{i=1}^{n} p(s_i|s_{i-1})p(o_i|s_i) \quad \text{(1.2)}$$

To estimate the probability of the hidden states given the data, i.e., $p(s|o)$, $p(o|s)$ is calculated as surrogate to estimate the most likely hidden state labels based on Bayesian rules:

$$s^* = \arg \max_{s} p(s|o) = \arg \max_{s} \frac{p(s, o)}{p(o)} = \arg \max_{s} \frac{p(o|s)p(s)}{p(o)} \quad \text{(1.3)}$$
To apply HMMs for focused crawling, we focus on explicit semantic content analysis learned from the user’s browsing behavior on specific topics, as detailed in Chapter 3. Documents are grouped semantically by a clustering algorithm to build a concept graph, from which the learning and prediction are done to recognize sequences of topics leading to targets. Observations are the cluster numbers to which the observed pages belong, and the corresponding hidden states are based on the hop distance from the target.

Maximum Entropy Markov Models (MEMMs) [66, 47] are a variation on the traditional HMMs (See Fig. 1.5 (B)). MEMMs are discriminative models and attempt to calculate the probability of the hidden states directly given the data. Training an MEMM involves maximizing the conditional probability, \( p(s|o) \), of state sequences \( s = s_1, s_2, ..., s_n \) given observation sequences \( o = o_1, o_2, ..., o_n \), rather than the joint probability \( p(s,o) \). MEMMs consider the observation to be conditioned upon the state rather than generated by it, which allows us to use many, arbitrary, overlapping features of the observation. At every position of the observation sequence, MEMMs specify a set of distributions of possible states based on the observed feature values in the form of an exponential model. Each distribution function uses per-state normalization to define the conditional probability of possible next states given the current state and the next observation element. The conditional probability in MEMMs is defined as

\[
p(s|o) = \frac{1}{z_j} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(s_{j-1}, s_j, o_j) \right) \tag{1.4}
\]

where \( z_j \) is a normalizing factor over the \( j^{\text{th}} \) position in the sequence, the \( f_i \) are arbitrary features and \( \lambda_i \) is the weight of the feature \( f_i \). To apply this approach to the focused crawling problem, the hidden states are based on hop distance from the target and observations as a set of pre-defined feature values of observed pages, such as anchor text, URLs, and keywords extracted from the pages. Detailed descriptions are in Chapter 4.
1.5.2 Undirected Graph Models

An undirected graphical model is represented by $G = \{V, E\}$ where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes and $E$ is a set of undirected edges between nodes. A single node $v_i$ is independent of all the other nodes in the graph, given its neighbors. Linear-chain Conditional Random Fields (CRFs) are one type of widely used undirected graphical models for sequential data. It has been proven very effective in many applications including POS tagging, information extraction, document summarization and shallow parsing [39, 64, 61, 73, 28]. Given the graphical representation in Fig. 1.5 (C), with the potential function modeled as a form of exponential function of the features defined over the cliques, the conditional distribution $p(s|o)$ in CRFs is defined as

$$p(s|o) = \frac{1}{Z(o)} \exp \left( \sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_{ij} f_i(s_{j-1}, s_j, o) \right)$$  \hspace{1cm} (1.5)

where $Z(o)$ is a normalization on $o$:

$$Z(o) = \sum_{s' \in |S|^n} \exp \left( \sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_{ij} f_i(s_{j-1}, s', o) \right)$$  \hspace{1cm} (1.6)

Therefore, CRFs define a single distribution $p(s|o)$ over the entire state sequence $s = s_1, s_2, ..., s_n$ given the observation sequence $o = o_1, o_2, ..., o_n$, rather than defining per-state distributions over the next states, given the current state at each position, as in MEMMs. In Chapter 5, we apply CRFs to focused crawling by defining hidden states as $\{T_3, T_2, T_1, T_0\}$ and observations as a set of pre-defined feature values of observed pages, such as anchor text, URLs, and keywords extracted from the pages.

1.6 Outline of the Thesis

Our primary goal in this thesis is to develop effective algorithms for focused crawling problems. We aim to capture the link structure and content of documents leading to target pages by modeling the focused crawling problem as a sequential task. Within the framework we designed, we have applied three probabilistic graphical models to best capture the longer path information along the page sequence. Following this guideline, we organize the rest of the thesis as follows:
In Chapter 2, we present an overview of the system architecture, which is the general framework for all three models we applied in the thesis. We describe in detail the data collection for training and the structure of HMMs/MEMMs/CRFs for modeling a sequence of Web pages. We also define the features and feature functions used in both MEMMs and CRFs crawling. Finally we give the general flow chart of focused crawler implementation and the performance metrics used for evaluation.

In Chapter 3, we describe in detail, how to use the Hidden Markov Model to learn the user’s browsing patterns for focused Web crawling. We give the detailed algorithms for user modeling, pattern learning, and efficient inference specific to HMMs. Experimental results and discussions compared with different methods are also presented.

In Chapter 4, we focus on exploring multiple overlapping features using the Maximum Entropy Markov Models. We first give a brief overview of the Maximum Entropy Markov Models, and then describe in detail how to use them for focused crawling. This includes the algorithms for parameter estimation and efficient inference for both training and crawling. Last, we conduct experiments to investigate the impact of the multiple features and different inference algorithms on the performance.

In Chapter 5, we present how to use Linear-chain Conditional Random Fields for focused crawling. We describe in detail the learning and inference algorithms associated with the model to obtain global optimal solutions along the sequences. Experimental results and analysis are also presented.

In Chapter 6, we compare different models for the focused crawling problem. We also provide an estimation of computational costs and summarize the pros and cons of each method.

In Chapter 7, we discuss potential future work directions and conclude the thesis work.
Chapter 2

System Architecture Overview

The objective of the topic-specific search is to allow the user to perform the search according to his/her interest or specific topic. The objective of the focused crawler is to collect only pages relevant to specific topics. The problem description of the focused crawler in this thesis is the following:

Given a set of target pages, collect as many pages as possible that are relevant to the given topics.

Based on the framework we proposed, the system is designed based on a 3-tier architecture: Data Collection, Pattern Learning and Focused Crawling, as shown in Fig. 2.1. For simplicity, in our experiments the target pages are Web pages the user specified or extracted from DMOZ\(^1\). Given the target pages, the first tier Data Collection is to collect Web pages to build a Web graph and generate page sequences for training. The second tier Pattern Learning is to extract features from these sequences and learn model parameters. The third tier Focused Crawling layer has the pattern parameters as input, and crawls on the real Web to find relevant pages. When a new page is seen, the model predicts the distance leading to the targets, which determines a visit priority for crawling. Fig. 2.1 shows the general system structure. Details within each component may be different with different underlying models and algorithms. The following sections describe each of the components with specific variations when applying different probabilistic models.

2.1 Data Collection

In a supervised learning problem, we need to collect labeled page sequences to be used as training data. For each page sequence, the label or state sequence has to be

\(^1\)http://www.dmoz.org/
Figure 2.1: System Architecture: Data Collection, Pattern Learning, and Focused Crawling.

recorded as well. In our system, the hidden state of each page is defined as $T_i$, where $i$ represents the smallest number of hops required to reach a target from this page. Target nodes are always in state $T_0$. For instance, a page sequence is $p_1 \rightarrow p_2 \rightarrow p_3$ and its corresponding state sequence is $T_3 \rightarrow T_2 \rightarrow T_1$. All these labeled page sequences collected from data collection form the training data. We consider two options for collecting training data: User Data Collection and Web Data Collection. User Data Collection is designed to collect the sequences of pages visited by the user during his/her topic-specific browsing. The user starts browsing from a start page. Target pages will be marked by the user during browsing. User-visited pages define the local Web graph, these pages are clustered. The link structure among pages from different clusters is identified, and page sequences are extracted as training data from them. Details are described in the following Section 2.1.1. Web Data Collection, on the other hand, is designed to collect page sequences from the local Web graph constructed using Yahoo API inlink service\textsuperscript{2}. The target pages are selected from DMOZ (www.dmoz.com) and the graph is created backwards by finding backlinks pointing to them from the Web, up to 4 layers. Details are in Section 2.1.2 below.

We use User Data Collection to learn the user’s browsing patterns for focused crawling with HMMs, and Web Data Collection when applying MEMMs and CRFs for focused crawling. For a supervised learning problem, there are two main types of

\textsuperscript{2}http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html
models: generative models and discriminative models. HMMs are generative models which have more assumptions about the data. In contrast, discriminative models, such as MEMMs, have no independence assumption on observation and solve more direct problems with richer representation via kernels or feature selection; for example, overlapping features are trivial. There are empirical results showing that discriminative models have a lower asymptotic error as the training set size increases [54]. Therefore, when we extended our work to apply MEMMs and CRFs for the focused crawling problem, we chose to use Web Data Collection to collect sufficient data for training, since discriminative models require more data $O(d)$ to reach their asymptotic error, whereas generative models require $O(\log d)$, where $d$ is the number of parameters, as pointed out by A. Ng and M. Jordan in [54].

2.1.1 User Data Collection

In the User Data Collection stage, we collect the sequences of pages visited by the user during his/her topic-specific browsing. The user starts browsing with a given topic in mind and is intended to stay on the topic. If the user considers the current page relevant, he/she can click the *Useful* button which is added to each page (Fig. 2.2a), and the page will become an annotated target page. In order to represent the user browsing pattern, we construct a local Web graph (Fig. 2.2b). A Web graph is a directed graph whose nodes correspond to pages on the Web, and whose arcs correspond to links between these pages. To construct a Web graph, each browsed page is represented by a node, and all hyperlinks between pages are added as edges between the nodes (Fig. 2.2b). When a user requests a Web page by typing a URL in the location bar, choosing a URL from a bookmark, or following a link to reach a new Web page, a new node will be created and all the hyperlinks between it and existing nodes will be added into the Web graph as edges. For example, if the user follows the hyperlink in page 2 to access page 3 in Fig. 2.2, node 3 is created and edges from 2 $\rightarrow$ 3 are added. Each node is associated with a browsed HTML page with a unique URL, and an edge is established if a referring document has a hyperlink pointing to a referred document. Nodes corresponding to targets are marked as double-bordered white nodes in the Web graph, for instance, nodes 4 and 5. In the case of Web pages with frames, when a frame page is being requested all of its children pages will be
Figure 2.2: User Data Collection. Double-bordered white nodes represent target pages. (a) Useful and Submit buttons. (b) User visited pages form the local Web graph.

automatically downloaded by the browser. Therefore, instead of recording the frame page, all of its pages will be recorded in the Web graph.

Sometimes, the user may not make the best decision, following a longer path to reach a target page. In this case, we store other possible links between nodes. Continuing the example above, if the user follows a hyperlink in page 2 to access page 3, and if page 1 links to page 3, then a link from node 1 to node 3 will be added to the Web graph as well.

The local Web graph thus captures the link structure of the pages visited by the user. The way the user browses reflects the content and linkage structure leading to his/her targets, which is the information we try to capture, for example, path $1 \rightarrow 2 \rightarrow 4$ in Fig. 2.2(b). Sometimes, the user may follow links to unrelated pages and go ‘back’ from them, for example, using the “Back” button on the Web browser.
In this case, the page information, but not path information, is still stored in the graph. For example, the user follows the path $1 \rightarrow 2 \rightarrow 3$ in Fig. 2.2(b), then finds that page 3 is totally unrelated to the topic, so she goes 'back' to page 2. In this case, page 3 is still saved as part of the training data, but the move from page 3 to page 2 does not translate into a link between these pages. In other words, the Web graph records the actual hyperlinks between pages the user followed, rather than the user click streams. The reasoning is that the focused crawler will eventually crawl the real Web, so the local Web graph should reflect the actual Web linkage structure. If the user follows some hyperlinks that eventually leading to targets, these pages are considered as positive linkage examples. On the contrary, if the user follows hyperlinks that could not reach any targets, normally the user will go 'back', in this case, the pages will be treated as negative examples. However, the action of the user going back is not stored, because it is not part of the linkage structure between pages.

The reasons for considering only visited pages in defining a local Web graph is to reject noisy hyperlinks that may mislead the focused crawler, and to extract the dominant structure the user follows towards his/her topic. Furthermore, in addition to the sequences of pages which the user visited, recording links between nodes in the local Web graph reflects linkage structures in the real world, where focused crawling will take place. We argue that the behaviors of the user during topic-specific browsing reveal the user's real information needs.

After completing the browsing session, the user can then submit the entire session by clicking the Submit button on each browsed page (Fig. 2.2a). The entire local Web graph consisting of all user-visited pages, together with the user's annotated target information, will be saved as training data.

In our implementation, we use the Algorithmic Solutions Software LEDA package [1] to save the graph with a hash table for URLs and their corresponding document ID numbers.

2.1.2 Web Data Collection

The second option is to construct the local Web graph using ODP data (www.dmoz.com) and backlink (or inlink) services. We use Web Data Collection when applying both MEMMs and CRFs to focused crawling. It consists of two components: Local Web
graph creation and Page sequences extraction. It takes specified target pages as input, and builds the local Web graph from bottom to top using Yahoo inlink services\(^3\). Page sequences are extracted directly from the constructed local Web graph. Page sequences are generated randomly with sequence length between 2 to 10. Details are in the following sections. As an example on the right side in Fig. 2.3, the input is the node 0 representing the pre-specified target Web page on a particular topic, and the outputs are all page sequences extracted from collected local Web graph, such as page sequence \(7 \rightarrow 3 \rightarrow 1 \rightarrow 0\). Node 0 is the initial pre-specified target page, and node 5 and 7 are target pages that are marked after the whole graph is created.

![Diagram](image)

Figure 2.3: Web Data Collection: it takes specified target pages as input, creates Web graph using Yahoo API, outputs extracted sequences of pages as training data. It consists of two components: Create Web graph and Extract page sequences.

### Selection of Target Pages on Topics

The initial target pages are selected pages on a specified topic used in both the training and crawling procedures. They can be either manually created by the users for their

---

\(^3\)http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html
own target pages or user specified from existing pages. In the system we propose, for each topic we chose target pages from the Open Directory Project (ODP)\textsuperscript{4}. All the initial target pages we used in our experiments can be found in Appendix A.

Topic specification plays a very important role in evaluating focused crawlers, since we examine their performance on retrieving relevant pages on the topic. Ideally, relevant pages should be recognized by real users. However, involving real users for judging thousands of pages is impossible. Furthermore, topics can be obtained from different sources and can be defined at different levels of specificity: for example, the general topic “sports” or the more specific topic “Ice Hockey World Championships 2005”. Therefore, as seen in [75], we select topics from an existing hierarchical concepts index such as the Open Directory Project (ODP), and pick topics which are neither too general nor too specific. An example of such a topic is House Plants, to be found under the DMOZ topic hierarchy Home - Gardening - Plants - House Plants.

We collect three kinds of data about the topics: keywords, descriptions and target pages. Keywords are formed by concatenating the words appearing in the different levels along the topical hierarchy directory from the top. For example, as shown in Fig. 2.4, “Home, Gardening, Plants, House Plants” are extracted keywords for the topic. Descriptions are generated using the descriptive text and the anchor text in the page of the topic in the ODP. These hyperlinks and their descriptions are created by expert editors, which briefly summarize the contents of the pages linked and are independent of the page contents. Target pages are the external Web pages the hyperlinks point to. Such an example is illustrated in Fig. 2.4. We also further extract important words embedded in the target pages themselves, including words from title and headers (<title>...</title>, <h1>...</h1> etc.) and keywords and descriptions from meta data (<meta>...</meta>).

During the procedure of using the initial target pages to create the local Web graph, all the Web pages are examined using cosine similarity between them and the initial target pages with threshold 0.8 to find new possible targets in the training data. Therefore, the initial target pages and the target pages that are added later form the final target pages for training.

\textsuperscript{4}http://www.dmoz.org
Figure 2.4: An example of selection of topic keywords, description and target pages from the Open Directory Project (ODP).

Create Web Graph

In order to capture the page sequences leading to targets for training, first we construct a local Web graph to represent the content and linkage structure associated with the targets (Fig. 2.5).

To construct a local Web graph, each Web page is represented by a node, and all hyperlinks between pages are added as edges between the nodes. When a new Web page is found and added to the existing graph, a new node will be created and all the hyperlinks between it and existing nodes will be added into the Web graph as edges. For example, when the page 5 (in Fig. 2.5(c)) is added to the graph with existing nodes 0, 1, 2, 3, and 4, then node 5 is created and edges from 5 → 2 and 0 → 5 will be added.
Figure 2.5: Create Web graph layer by layer using Yahoo API inlink service. White nodes represent target pages, and grey nodes represent pages obtained through backlink service tracing with Yahoo API. Node 0 is the initial target page, and nodes 5 and 7 are target pages that are marked.

As shown in Fig. 2.5, the local Web graph is created layer-by-layer starting from one or more user-specified target pages, say node 0, which is obtained from ODP described in the section above. Note that we only use ODP to select initial target pages on the topic, rather than use the ODP hierarchy for training. ODP is a manually created hierarchical concept directory, which does not reflect the Web linkage structure in the real world. On the other hand, focused crawling is designed to retrieve Web pages from the actual Web, therefore we have to collect training data from the real Web to reflect the actual Web linkage structure and its dynamic nature. To do so, we make use of the inlink service from Yahoo Web APIs service\(^5\), which lists Web pages that have links to the specified Web page. Starting from the layer 0 with user-specified target page(s), the graph is created from bottom to top, up to layer 4.

There are no repeated nodes in the graph. When a new URL is seen, if it is not in the graph, both a new node and all corresponding edges (hyperlinks) are added into the graph; if it is already included in the graph, only all possible edges between it and existing nodes are added. For instance, when node 5 is added to the graph as shown in the third graph Fig. 2.5 (c), and the URL of node 0 is found as the inlink of node 5 by Yahoo inlink service, then only the edge from 0 → 5 will be added to the graph, due to the fact that node 0 is already in the graph. Since some Web pages

\(^5\)http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html
may have hundreds of backlinks pointing to it, while some may only have a few, we limit the total number of nodes in each layer to a maximum of 600 and average the number of backlinks for each node. For example, if the current layer $i$ contains 200 nodes, then the number of the backlinks for each node is limited to $600/200 = 3$, although some nodes may have hundreds of inlinks found by Yahoo API. As a result, the graph is created in a more balanced manner in terms of the number of nodes in each layer and the number of the backlinks for each node in order to extract page sequences effectively for training.

After the graph is created, all nodes are also classified to mark new targets based on the cosine similarity between them and the initial target pages, such as nodes 5 and 7 in the rightmost graph Fig. 2.5(e). If the similarity between a node and one of the initial target pages is equal to or greater than 0.8, it is marked as the target as well. In the figure shown, nodes corresponding to targets are marked as white nodes in the local Web graph, for instance nodes 0, 5 and 7, and grey nodes represent the others. The local Web graph thus captures sufficient content and link structure of the pages leading to targets, which is the information we try to capture. The motivation is that the focused crawler will eventually crawl the real Web, so the local Web graph should reflect the actual Web linkage structure.

The reasons for using backlinks from target pages in defining a local Web graph are to reject noisy hyperlinks that may mislead the focused crawler, and to extract the adequate structure directly or indirectly leading to the target topic. With backlink tracing, some noisy hyperlink branches such as advertisements can be efficiently eliminated from the graph and only useful Web pages leading to the targets are kept. We might create a local Web graph using Breadth-First search strategy in a top-to-bottom manner, however, it brings too much noisy data such as advertisement hyperlinks that never reach the targets. We may also continue to use User Data Collection described in the previous section, but User Data Collection reflects user's browsing behavior and is limited to learning the user's topic-specific browsing patterns, while the system we propose using MEMMs/CRFs is to train a model for focused crawling in a more general way. Sufficient and efficient training data reflecting the real Web is the important factor in taking full advantage of MEMMs/CRFs models. We believe that using backlinks to collect training data from the Web is a reasonable trade-off.
in terms of sufficiency and efficiency.

The complete training data includes page sequences and their corresponding layer sequences. For MEMMs and CRFs crawls, page sequences are referred to as observation sequences or observable input sequences, and layer sequences are referred to as hidden state sequences or state (label) sequences.

**Extract Page Sequences and State Sequences**

Given the created local Web graph, the next step is to extract page sequences from the graph.

The nature of the graph shows that one node may have multiple child URLs. For example, node 8 has nodes 4, 1, 5 as its children in the right side of Fig. 2.5. If we extract sequences following all the arcs in the graph, we may get many duplicated paths in the sequences, such as $9 \rightarrow 8 \rightarrow 4$ in the sequences $9 \rightarrow 8 \rightarrow 4 \rightarrow 1$ and $9 \rightarrow 8 \rightarrow 4 \rightarrow 2$. Therefore, to avoid this problem, we extract sequences in a totally random manner: every sequence to be extracted starts with a randomly-picked node, randomly selects one of its children for the next node, and repeats until a randomly-generated sequence length between 2 and 10 is reached.

The following rules are considered for the page sequence extraction:

- Only extract pages from higher layers to lower layers or from the same layer, without reverse the layer order in the sequences. For instance, in Fig. 2.5(e), $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is valid, and $9 \rightarrow 8 \rightarrow 4 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 0$ is not valid in this case, because $4 \rightarrow 7$ is the reverse order in the sequence. The advantage of this option is to try to collect all positive sequences as training data; in other words, these sequences are good examples for learning. Our goal is to predict how far away a page is from targets, the closer the better, so the position information a page in a sequence is an important role in training.

- Avoiding loops in the sequence. For example, $9 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is valid, and $9 \rightarrow 7 \rightarrow 4 \rightarrow 7 \rightarrow 4 \rightarrow 1$ is not valid.

The length of the sequences is randomly generated between 2 and 10, as a result, page sequences with length 2 are also included. For instance, $9 \rightarrow 8$, $4 \rightarrow 7$, $1 \rightarrow 0$. The training with short parent-child pairs is to focus on local features.
The hidden state for a page in our system is its lowest layer number. For example, given the rightmost graph of Fig. 2.5(e), sequence $9 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is a valid sequence and its corresponding state sequence is $T_1 \rightarrow T_0 \rightarrow T_1 \rightarrow T_1 \rightarrow T_0$. The rule is that a target node is always set to layer number 0, no matter which physical layer it is in the graph, and other nodes will be re-assigned a new layer number if the layer numbers of its adjacent nodes are changed. For example, the original state sequence is $T_4 \rightarrow T_3 \rightarrow T_2 \rightarrow T_2 \rightarrow T_1 \rightarrow T_0$ for the page sequence $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$, according to the graph of Fig. 2.5(d), since the original physical layer number of node 7 is layer 3. However, when nodes 7 and 5 are marked as target pages in Fig. 2.5(e), their layer number will be changed to 0; accordingly, the layer number of other nodes in the graph will also be changed. For example, the new layer number of node 8 will be changed from 3 to 1, since the new layer number of its direct child node 5 is now 0. That is, according to our definition of hidden state $T_i$ of a page, which represents the smallest number of hops to reach a target from this page, the hidden states of all nodes from node 1 to node 9 are \{$T_0, T_1, T_1, T_2, T_1, T_0, T_2, T_0, T_1, T_1$\} in Fig. 2.5(e). Therefore, the new state sequence for the page sequence $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$ is $T_1 \rightarrow T_0 \rightarrow T_2 \rightarrow T_1 \rightarrow T_1 \rightarrow T_0$.

2.2 Pattern Learning

The second stage of the system architecture is Pattern Learning. As shown in Fig. 2.1, the objective of this component is to take the training data (page sequences) as input, extract features and estimate the parameters for different underlying learning models (HMMs, MEMMs, CRFs).

2.2.1 Pattern Learning with HMMs

Hidden Markov Models (HMMs) [65], widely used in speech-recognition and information extraction, provide superior pattern recognition capabilities for sequential patterns. HMMs are useful when one can think of underlying unobservable events probabilistically generating surface events, that are observable. An HMM [65] is a stochastic finite automaton, where each state generates (emits) an observation. We use $s_t$ to denote the hidden state and $o_t$ to denote the observation at time $t$. HMMs assume $s_t$ is a discrete random variable. If there are $K$ possible states, then
\( s_t \in \{1, \ldots, K\} \). \( o_t \in \{1, \ldots, L\} \) if there are \( L \) possible symbols, or a feature-vector, \( o_t \in \mathbb{R}^L \). More specially, a HMM [65] is defined by a finite set of hidden states \( s = \{s_1, s_2, \ldots, s_k\} \) and a finite set of observations \( o = \{o_1, o_2, \ldots, o_m\} \), and is a generative model of how \( s_t \) generates or causes \( o_t \) and \( s_{t+1} \). The goal of inference is to invert this mapping, i.e., to infer \( s_{1:t} \) given \( o_{1:t} \). The parameters of the model are transition probability matrix \( A = [a_{ij}] \), where \( a_{ij} = p(s_t = j | s_{t-1} = i) \), emission probability matrix \( B = [b_{jk}] \), where \( b_{jk} = p(o_t = k | s_t = j) \), and the initial state distribution matrix \( \pi = [\pi_i] \), where \( \pi_i = p(s_0 = i) \).

The state transitions can be depicted graphically, as shown in Fig. 2.6. Nodes represent hidden states, and arrows represent allowable transitions, i.e., transitions with non-zero probability. The transition from state 1 to state 3 denotes \( p(s_t = 3 | s_{t-1} = 1) = A[1, 3] \).

![Figure 2.6: A state transition diagram for a HMM with 4 hidden states. Nodes represent hidden states, and arrows represent allowable transitions, i.e., transitions with non-zero probability.](image)

There are three assumptions made by HMMs:

- **Markov assumption:** the next state depends only on the current state.
- **Stationary assumption:** Each probability in the state transition matrix and in the emission matrix is time independent - that is, the matrices do not change in time as the system evolves.
- **Output independence assumption:** the current observation is independent of the previous observations given the current state.

Therefore, to model a sequence task with HMMs, first we have to specify a bounded number of set of state variables due to very strict independence assumption on the observations, that is, we need to enumerate all possible observation sequences. Second,
we need to model the joint probability $p(s, o)$ as shown in Equation 2.1.

$$p(s, o) = p(s_0) \prod_{t=1}^{n} p(s_t|s_{t-1}) p(o_t|s_t)$$  \hspace{1cm} (2.1)

**Structure of the HMM for Focused Crawling**

In the HMM used for focused crawling, the hidden states reflect the topology of the Web graph with respect to the target pages, while the observations reflect the content of the Web pages. The model aims to estimate the likelihood of topics leading to a target topic. However, it is difficult to enumerate all possible observation sequences, that is, to present all observation elements as isolated and independent elements from the others. If directly using Web pages or feature vectors as observations, it is hard to imagine how to trace a bounded number of Web pages as isolated observation elements, since there are numerous Web pages available and we always observe new pages.

![Diagram](image)

Figure 2.7: Dependency structure of HMMs on modeling a sequences of Web pages, given state sequence $s = \{s_1, s_2, ..., s_n\}$ and observation sequence $o = \{c_1, c_2, ..., c_n\}$ representing the corresponding clusters to which the actual Web pages in the sequence $\{page_1, page_2, ..., page_n\}$ belong. $t$ ranges over input positions. Arrow shows dependency (cause).

To do so, as shown in Fig. 2.7, the clusters to which Web pages belong are used to present observation elements to avoid the intractability nature of Web pages. Therefore, we model the focused crawling problem as a sequence analysis task where the hidden state sequence $s = s_1, s_2, ..., s_n$ probabilistically generates the observation sequence $o = c_1, c_2, ..., c_n$ representing the corresponding clusters to which the actual
Web pages in the sequence $page_1, page_2, ..., page_n$ belong (Fig. 2.7). Note that this graph represents the dependency structure of HMMs on modeling an observation sequence given a state sequence, which should not be confused with the state transition diagram introduced above.

The transition structure of the Hidden Markov Model for focused crawling is shown in Fig. 2.8. Note that this is not a fully connected graph, and arrows represent allowable transitions. For example, transition $T_3 \rightarrow T_1$ and $T_2 \rightarrow T_0$ are not allowed.

![Hidden states: T3, T2, T1, T0
Observations: cluster 1, 2, 3, 4, ...](image)

Figure 2.8: The transition structure of a Hidden Markov Model for focused crawling with 4 hidden states $T_0, T_1, T_2, T_3$. Nodes represent hidden states. Arrows between different hidden states represent allowable transitions. Hidden states generate observations represented by cluster numbers.

The key quantities associated with the model are the hidden states, observations, and the parameters (initial state probability distribution $\pi$, transition probability matrix $A$, and emission probability matrix $B$).

Let $k$ be the number of hidden states.

- **Hidden states:** $S = \{T_{k-1}, T_{k-2}, ..., T_1, T_0\}$
  - The focused crawler is assigned to be in state $T_i$ if the current page is $i$ hops away from a target. The state $T_{k-1}$ represent "$k - 1" or more hops to a target page.

- **Observations:** $O = \{c_1, c_2, c_3, c_4, ..., c_m\}$
  - Cluster number (1..m) of an observed Web page. The number of clusters $m$ is determined by the X-means algorithm [60].
• Set of HMM parameters $\theta = (\pi, A, B)$:

- Initial Probability Distribution
  
  $\pi = \{p(T_0), p(T_1), p(T_2), \ldots, p(T_{k-1})\}$.  
  
  These are the probabilities of being $0, 1, \ldots k - 1$ hops away from a target page at the start of the process. Due to lack of prior information, we use the uniform distribution $\pi = \{1/k, 1/k, \ldots, 1/k\}$.

- Matrix of Transition Probabilities: $A = [a_{ij}]_{k \times k}$, where $a_{ij}$ = probability of being in state $T_j$ at time $t+1$ given that the system is in state $T_i$ at time $t$.

- Matrix of Emission Probabilities: $B = [b_{ij}]_{k \times m}$, where $b_{ij}$ = probability that the current page belongs to cluster $j$ if the system is in state $T_i$.

Parameter estimation procedure for HMMs involves X-means clustering, concept graph construction, and learning the parameters $\theta = (\pi, A, B)$. Details are described in Chapter 3.

### 2.2.2 Pattern Learning with MEMMs and CRFs

MEMMs and CRFs allow the independence assumptions on the observations of HMMs to be relaxed. As introduced in Section 1.5, both MEMMs and CRFs are conditional probabilistic sequence models used to estimate conditional probability, $p(s|o)$, of state sequences $s = s_1, s_2, \ldots, s_n$ given observation sequences $o = o_1, o_2, \ldots, o_n$. Both of them use a linear combination of weighted feature functions $\sum_i \lambda_i f_i(s, o)$ to encode the information encapsulated in the training data, which allows non-independent, interacting, and arbitrary features of the observation sequence. Web pages are richly represented using extracted multiple features such as title, keywords, and URL token. This flexibility provides us the power to better model many real-world problems and gives better performance on a number of real-world sequence tasks. The parameters of MEMMs and CRFs are the feature weights $\lambda = \{\lambda_i\}$. However, they are different models: MEMMs are directed graphical models, while CRFs are undirected graphical
models (details in Chapter 4 and Chapter 5). The graphical dependency structures of MEMMs and CRFs for modeling the focused crawling problem in Fig. 2.9 and Fig. 2.10 show us the differences. MEMMs calculate the conditional probability $p(s|o)$ by multiplying a chain of local conditional probabilities $p(s_t|s_{t-1}, o_t)$ based on the multiple features, such as anchor text, at each position $t$, whereas, CRFs define a single conditional probability $p(s|o)$ over the entire state sequence $s$, given the observation sequence $o$.

As showed in Fig. 2.9 and Fig. 2.10, although the underlying models are different, the elements of the dependency structure and the features used in MEMMs and CRFs are the same, therefore, we describe them together in the following sections.

**Structure of MEMMs/CRFs for Focused Crawling**

Let $k$ be the number of hidden states. The key quantities associated with MEMMs/CRFs models are the hidden states, observations (features), and the parameters (feature weights vector $\lambda$).

- **Hidden states**: $S = \{T_{k-1}, T_{k-2}, \ldots, T_1, T_0\}$

  - The focused crawler is assigned to be in state $T_i$ if the current page is $i$
Figure 2.10: Graphical structure of CRFs on modeling a sequence of Web pages. This is an undirected graphical model. It defines a single conditional probability \( p(s|o) \) over the entire state sequence \( s \) given the observation sequence \( o \) based on features such as title, keywords, and URL token. \( t \) ranges over input positions.

- Observations: Collections of feature values of page sequences \( O = \{page_1, page_2, page_3, \ldots\} \)
  - Observable page sequences represented by a sequence of values for a set of predefined feature functions \( f = \{f_1, f_2, \ldots, f_m\} \). \( m \) is the number of feature functions.

- Set of parameters \( \lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\} \)
  - The parameter is a weight vector associated with each feature function \( f = \{f_1, f_2, \ldots, f_m\} \).

Parameter estimation in MEMMs/CRFs uses Limited Memory Quasi-Newton Method (L-BFGS) [55, 28] to iteratively estimate the model parameters \( \lambda \). Details are described in Section 4.1.3 for MEMMs and in Section 5.3 for CRFs.
Table 2.1: Summary of Features

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td>states transition features</td>
</tr>
<tr>
<td>Text</td>
<td>$\cos$ (text of current page, text of target pages)</td>
</tr>
<tr>
<td>Description</td>
<td>$\cos$ (description of current page, target pages)</td>
</tr>
<tr>
<td>Words</td>
<td>Words themselves (meta data, title, alt, head, important words)</td>
</tr>
<tr>
<td>URL Token</td>
<td>Contain at least one of target keywords</td>
</tr>
<tr>
<td>Pointing Anchor</td>
<td>Anchor surrounding text in parent page which points to the page</td>
</tr>
<tr>
<td>Child Anchor</td>
<td>Anchor surrounding text in the page</td>
</tr>
</tbody>
</table>

Features and Feature Functions

Each feature function $f(s, o)$ is defined as a factored representation. MEMMs and linear-chain CRFs make a first-order Markov independence assumption among states, that is, the current state depends only on the previous state and not on any earlier states. We use $s$ to represent the entire state sequence, $o$ for entire observation sequence, and $s_t$ and $o_t$ to indicate the state and observation at the position $t$, respectively. Since MEMMs define separate conditional probabilities $p(s_t|s_{t-1}, o_t)$ based on features at each position $t$, whereas CRFs define a single conditional probabilities $p(s|o)$ over the entire state sequence $s$ given the observation sequence $o$, to be distinct, we can rewrite $f(s, o)$ at position $t$ specifically into $f(s_{t-1}, s_t, o_t)$ in MEMMs, and $f(s_{t-1}, s_t, o, t)$ in CRFs. For simplicity, we use $f(s, o, t)$ in this section. Note that $f(s, o, t)$ in this work only depends on $s_{t-1}$ and $s_t$ and the content of pages observed at time $t$.

Formally, we define the feature functions to be factorized as:

$$f(s, o, t) = L(s_{t-1}, s_t, t) * O(o, t)$$

(2.2)

where $L(s_{t-1}, s_t, t)$ are transition feature functions, and $O(o, t)$ are observation feature functions. This allows us to combine information around the current position $t$. With defined feature functions, we construct a set of real-valued features to capture whether the observed Web pages have specific characteristics. In MEMMs and CRFs, each feature can be represented as either a binary or a real value. For binary values, a 1
indicates the presence of the feature, whereas a 0 means the absence of the feature. Real-valued features often represent cosine similarity values between texts.

First let us see an example. Suppose current page is the page at time \( t \), and parent page is the page at time \( t - 1 \). Let \( o = \text{page1}, \text{page2}, \text{page3} \) be a sequence of Web pages, and \( s = T_2, T_1, T_0 \) be the corresponding state sequence. To represent the information of \( \text{page3} \) being in state \( T_0 \) at current time \( t = 3 \) if the anchor text of the URL of \( \text{page3} \) in its parent page \( \text{page2} \) contains specified keywords “linux”, the feature functions \( f(s,o,t) \) can be written as:

\[
f(s,o,t = 3) = \begin{cases} 
1 & \text{if } L(s_{t-1}, s_t = T_0, t = 3) = 1 \text{ and } O(o, t = 3) = 1; \\
0 & \text{otherwise.}
\end{cases}
\]

For each \( i \) for which a transition \( T_i \rightarrow T_0 \) is possible,

\[
L(s_{t-1}, s_t = T_0, t = 3) = \begin{cases} 
1 & \text{if } t = 3 \text{ and } s_{t-1} = T_i \text{ and } s_t = T_0 \text{ exist;} \\
0 & \text{otherwise.}
\end{cases}
\]

\[
O(o, t = 3) = \begin{cases} 
1 & \text{if anchor text of } o_t = \text{page3 contains “linux”} \\
0 & \text{otherwise.}
\end{cases}
\]

In other words, \( L(s_{t-1}, s_t, t) \) captures the possible transition features of the states at time \( t - 1 \) and current states (in this case, \( T_1 \rightarrow T_0 \)), and \( O(o,t) \) expresses the some observation features at current time \( t \).

Table 2.1 summarizes all features we defined. We now describe each of them.

1. Edge Features: Transition feature functions \( L(s_{t-1}, s_t, t) \) can have two forms: \( L_1 \) and \( L_2 \). We use Edge feature \( L_1(s_{t-1}, s_t, t) \) to capture the possible transitions from states \( s_{t-1} \) to \( s_t \), and \( L_2(s_{t-1}, s_t, t) \) to capture the possible states at time \( t \).

Formally, for all \( i, j = 0, 1, \ldots, k-1 \) so that specified \( T_i, T_j \in S = \{T_{k-1}, T_{k-2}, \ldots, T_1, T_0\} \), we can have feature functions of the following form:
\[ L_1^{(i,j)}(s_{i-1}, s_t, t) = \begin{cases} 
1 & \text{if } s_{i-1} = T_i \text{ and } s_t = T_j \text{ is an allowed transition}; \\
0 & \text{otherwise}.
\end{cases} \]

\[ L_2^{(i)}(s_{i-1}, s_t, t) = \begin{cases} 
1 & \text{if } s_t = T_i \text{ exists}; \\
0 & \text{otherwise}.
\end{cases} \]

Some states and state transitions may not happen at some time positions, for example, it is impossible to have state transition \( T_2 \rightarrow T_0 \). If state transition \( T_3 \rightarrow T_2 \) is possible at time \( t \), then the feature value of \( L_1(T_3, T_2, t) \) is 1.

2. **Text Feature:** Maximal cosine similarity value between the content of a given candidate page and the set of targets. We define it as \( O_1(o, t) \):

\[ O_1(o, t) = \max_{d \in T} \cos(\text{text of current page } p \text{ at time } t, \text{ text of target page } d); \]

where \( T \) is the set of target pages, and \( \cos(\cdot, \cdot) \) is the standard cosine similarity function between two term vectors, i.e.,

\[ \cos(p, d) = \frac{\sum_{k \in p \cap d} w_{pk} \cdot w_{dk}}{\sqrt{\sum_{k \in p} w_{pk}^2 \sum_{k \in d} w_{dk}^2}} \]

where \( w_{dk} \) is the term frequency of term \( k \) in document \( d \).

3. **Description Feature:** Cosine similarity value between the page description of a given candidate page and the target description. Normally every page has meta data in addition to the text content of the page, including description and keywords, which briefly summarizes the whole page. We define feature function \( O_2(o, t) \) to capture the description feature:

\[ O_2(o, t) = \cos(\text{description of current page at time } t, \text{ the target description}); \]

4. **Word Feature:** Word feature \( O_w(o, t) \) identifies the important words (meta data,
title, head) appearing in the page text.

\[ O_w(o, t) = \begin{cases} 
1 & \text{if word } w \text{ appears in the current page at time } t; \\
0 & \text{otherwise.} 
\end{cases} \]

We may also use the counts of word \( w \) as the value of this feature, instead of the binary value.

5. **URL Token Feature**: The tokens in the URL of an observed page may contain valuable information about predicting whether or not a page is a target page or potentially leads to a target. For example, a URL containing “linux” is more likely to be a Web page about linux-related information, and a URL which contains the word “operating system” or “OS” indicates that with high probability, it may lead to a Linux page. There are two possible kinds of URLs related to the current observed page: one is the URL of the current page itself, and another one is the URL the current page is pointing to. We define two token feature functions \( O_3(o, t) \) and \( O_4(o, t) \) to identify if the keywords appear in the URLs.

\[ O_3(o, t) = \begin{cases} 
1 & \text{if any of URLs in the current page at time } t \\
& \text{contains at least one target keyword;} \\
0 & \text{otherwise.} 
\end{cases} \]

\[ O_4(o, t) = \begin{cases} 
1 & \text{if the URL of the current page at time } t \\
& \text{(contained in the parent page) contains at least} \\
& \text{one target keyword;} \\
0 & \text{otherwise.} 
\end{cases} \]

6. **Anchor Text Feature**: The anchor text around a link pointing to an observed page \( o \) is often closely related to the topic of the page. A human’s ability to discriminate between links mostly relies on the anchor text. We capture the important word \( w \) in the anchor text by defining two anchor features: \( O_{5,w}(o, t) \) is used to identify the anchor text in the parent page surrounding the link which points to the given page, and \( O_{6,w}(o, t) \) is to identify anchor text in the given
page.

\[ O_{5,w}(o,t) = \begin{cases} 
1 & \text{if word } w \text{ appears in the anchor text of the link in the} \\
& \text{the parent page linking to current page at time } t; \\
0 & \text{otherwise.} 
\end{cases} \]

\[ O_{6,w}(o,t) = \begin{cases} 
1 & \text{if word } w \text{ appears in the anchor text in the current} \\
& \text{page at time } t \text{ pointing to the page at time } t + 1; \\
0 & \text{otherwise.} 
\end{cases} \]

To extract and indicate each feature at each time step, we have used a feature vector \( F_t \) to include all the arguments for each feature function, which are needed for computing feature values at each time \( t \). For example, as shown in \( O_{5,w}(o,t) \), word \( w \) appearing in the anchor text in the previous page is used as a feature, then the feature vector \( F_t \) is assumed to include the identity of word \( w \).

Suppose we have \( m \) feature functions as the result of Equation 2.2, parameter estimation procedure for MEMMs and CRFs involves learning the parameters \( \lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\} \). Each \( \lambda_i \) is the weight associated with feature \( f_i \). More intuitively, each parameter \( \lambda_i \) can be considered to be a weight indicating the informativeness of feature \( f_i \). For each feature \( f_i \), this is an optimization problem to find the best \( \lambda_i \) so that the expected "count" of feature \( f_i \) using current value equals to the empirical "count" of feature \( f_i \) in the training data. Details are described in Chapter 4 and Chapter 5.

### 2.3 Focused Crawling

After the learning phase, the system is ready to start focused crawling. The third stage is the Focused Crawling layer which crawls on the real Web to find relevant pages based on learned parameters. An overview of the crawling algorithm is shown in Fig. 2.11. The crawler utilizes a queue, which is initialized with the starting URL of the crawl, and keeps all candidate URLs ordered by their visit priority value. We use a timeout of 10 seconds for Web downloads and filter out all pages except those with text/html content. The crawling respects the Robot Exclusion Protocol and distributes the load over remote Web servers. The crawler downloads the page
pointed to by the URL at the head of the queue and extracts all the outlinks and performs feature selections. The predicted state for each child URL is calculated based on the current observable features and corresponding weight parameters, and the visit priority values are assigned accordingly. In the flow chart, the rectangles for Analyze Page and Prediction are implemented differently for different underlying models (HMMs, MEMMs, CRFs).

Figure 2.11: Flow Chart of Focused Crawling with HMMs/MEMMs/CRFs Models

Since the queue is always sorted according to the visit priority value associated with each URL, we expect that URLs at the head of the queue will locate targets more rapidly.

We also set a relevance threshold $\gamma$ for determining whether a Web page is relevant to the user's interests or target topics. If its maximal cosine similarity to the target set is greater than $\gamma$, the URL is stored and presented to the user as a relevant page.

2.3.1 **Efficient Inference**

The goal of inference is to infer $s_t$ given $o_{1:t}$ based on the parameters of the underlying models. Given a downloaded Web page $w_t$ at current time $t$, the URLs associated with its outgoing links $w_{t+1}$ are inserted into the visit queue, sorted by the assigned priority value based on the probabilities of the predicted state of the children of $w_t$, $p(s_t|o_{1:t})$. 


So the task during the crawling phase is to associate a priority value with each URL that is being added to the queue. Efficient inference in HMMs/MEMMs/CRFs with specific parameters is performed for this task.

The estimated distribution of the state of \( w_t, p(s_t|o_{1..t}) \), also called belief state, is a probability distribution over all possible state values \( s_t = T_0, T_1, T_2, ..., T_{k-1} \), given observations \( o_{1..t} \). The distribution at time \( t \) is computed recursively based on the distribution at time \( t - 1 \), corresponding to the parent page \( w_{t-1} \) of \( w_t \). Parent page \( w_{t-1} \) of \( w_t \) is the Web page containing the URL of \( w_t \), which led to the insertion of \( w_t \) in the queue. The sequence \( 1..t \) refers to the visited pages along the shortest path from the seed page to the current page. The sequence does not necessarily reflect the temporal order in which pages have been visited.

The probability distribution \( p(s_t|o_{1..t}) \) of the state at the next step \( t \), given observations \( o_{1..t} \), can be calculated recursively from the result up to time \( t - 1 \) by conditioning on the previous state \( s_{t-1} \):

\[
p(s_t|o_{1..t}) = \sum_{s_{t-1}} p(s_t|s_{t-1}, o_t) p(s_{t-1}|o_{1..t-1})
\]  

(2.3)

where \( p(s_t|s_{t-1}, o_t) \) is the conditional probability of the transition from state \( s_{t-1} \) to state \( s_t \) on observation \( o_t \). The probability can be calculated efficiently by dynamic programming. We define forward vector \( \alpha_t \) with each visited page. Each forward value \( \alpha(s_t, t) \), is the probability that the system is in state \( s_t \) at time \( t \) given all observations made up to time \( t \). Hence the values \( \alpha(s_t, t) \) are the calculated values of \( p(s_t|o_{1..t}) \). HMMs, MEMMs and CRFs define forward value \( \alpha(s_t, t) \) differently depending on which model and which algorithm we use. The details of how to perform the computation are explained in Chapter 3, Chapter 4 and Chapter 5, respectively. In HMMs, \( p(s_t|s_{t-1}, o_t) \) is easily calculated by \( p(o_t|s_t)p(s_t|s_{t-1}) \) because of the first-order Markov assumption and the Markov property of evidence. That is, Equation 2.3 is calculated as

\[
p(s_t|o_{1..t}) = \sum_{s_{t-1}} p(o_t|s_t) p(s_t|s_{t-1}) p(s_{t-1}|o_{1..t-1})
\]  

(2.4)
In MEMMs, \( p(s_t | s_{t-1}, o_t) \) can be directly obtained by the definition

\[
p(s_t | s_{t-1}, o_t) = \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t))
\]

\[
z(o_t) = \sum_{s' \in S} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o_t))
\]

Therefore, Equation 2.3 is calculated in MEMMs by

\[
p(s_t | o_{1..t}) = \sum_{s_{t-1}} \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)) p(s_{t-1} | o_{1..t-1}) \tag{2.5}
\]

In CRFs, \( p(s_t | s_{t-1}, o_t) \) is calculated through the forward value \( \alpha(s_t, t) \),

\[
\alpha(s_t, t) = \sum_{s_{t-1}} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)) \alpha(s_{t-1}, t - 1) \tag{2.6}
\]

then \( p(s_t | o_{1..t}) \) is obtained by

\[
p(s_t | o_{1..t}) = \frac{\alpha(s_t, t)}{\sum_{s_i} \alpha(s_i, t)} \tag{2.7}
\]

where, \( s_i \) ranges all possible states \( s_i = T_0, T_1, T_2, ..., T_{k-1} \).

Note that with HMMs crawling, all children of page \( w_t \) receive the same priority value since there are no additional observation features from the next time step used for different child URLs in page \( w_t \); whereas in MEMMs/CRFs crawling, children of page \( w_t \) will have different visit priorities because additional features such as Anchor Text Feature and URL Token Feature of \( w_{t+1} \) extracted from page \( w_t \) are included in the feature vector \( F_t \) at time \( t \), as shown in Section 2.2.2.

### 2.3.2 Calculation of the Priority

The visit priority is determined based on the estimated state distribution \( p(s_t | o_{1..t}) \) for each URL, defining a vector key \( (p(T_0), p(T_1), p(T_2), ..., p(T_{k-1})) \), where \( k \) is the number of hidden states. The sorting keys are arranged according to the estimated probabilities of the \( k \) possible states in the order of \( (T_0, T_1, T_2, ..., T_{k-1}) \). URLs are sorted in a lexicographic manner based on the sequence of probabilities, beginning
with \( p(T_0) \).

If there are two or more items of approximately equal value in the first key, the key data items are ordered in decreasing order according to the second data item to break the tie. In our system, we use a threshold \( \varepsilon = 0.001 \) to define the equality of two key data items. i.e., for two state distributions \( X[p(T_0), p(T_1), p(T_2), ..., p(T_k)] \) and \( Y[p(T_0), p(T_1), p(T_2), ..., p(T_k)] \), if \( |X[p(T_0)] - Y[p(T_0)]| < \varepsilon \), the sorting process would use the second key data items pair \( X[p(T_1)] \) and \( Y[p(T_1)] \).

### 2.3.3 Priority Queue Data Structure

The priority queue contains the URLs of pages \( w_t \) to be visited sorted by the priority of the parent page \( w_{t-1} \) of page \( w_t \), defined as the page through which the URL of \( w_t \) was placed on the queue.

Each queue consists of the following three basic elements:

- the URL of page \( w_t \)
- the visit priority of \( w_t \)
- probabilities \( \alpha(T_j, t-1) \) that page \( w_{t-1} \) is in hidden state \( T_j \), for all \( j \), capturing \( p(s_{t-1}|o_{1..t-1}) \).

Other elements needed for inference may vary depending on different underlying models. In HMMs crawling, we add the cluster information \( C_{t-1} \) of the parent page \( w_{t-1} \) of page \( w_t \). With MEMMs/CRFs, we keep the anchor text feature and URL Token feature values of the parent page \( w_{t-1} \) of page \( w_t \). Detailed crawling algorithms with each model are described in the following chapters.

### 2.4 Evaluation Methods

It is important that the focused crawler returns as many relevant pages as possible while minimizing the portion of irrelevant ones. The standard information retrieval (IR) measures used to evaluate the performance of a crawler are **Precision** and **Recall**.

The **precision** is the percentage of the Web pages crawled that are relevant to the topic. It is ideal to recognize relevant pages by real users. However, involving
real users for judging thousands of pages is impossible. Application using cosine similarity to on-topic examples, queries, or profiles can be found in several topical crawling studies [36, 53, 76]. Since in general there are multiple target pages in our system either marked by the user or extracted from DMOZ, the relevance assessment of a page $p$ we are using is based on maximal cosine similarity to the set of target pages $T$ with a confidence threshold $\gamma$. That is, if $\max_{d \in T} \cos(p, d) \geq \gamma$ then $p$ is considered as relevant.

The recall of the standard evaluation measure is the ratio of the relevant pages found by the crawler to all relevant pages on the entire Web. It is not possible to get the exact total number of relevant pages on the Web, so recall cannot be directly measured. It is reasonable to use an estimate of recall to compare different retrieval algorithms based on a document corpus, such as Target Recall described in [56, 75]. They treat the recall of the target set, i.e., target recall (R), as an estimate of true recall. That is,

$$
R = \frac{|C \cap T|}{|T|}
$$

(2.8)

where $C$ is the set of the pages crawled, $T$ is the set of targets, and $|T|$ is the number of targets.

However, the goal of the focused crawler is to find as many relevant pages as possible to a given topic, rather than to collect the given target pages only. It is hard to expect target recall to give us a good estimate of the actual recall. Therefore, we do not compare the crawler performance based on the estimated recall in general. Instead, we use the Maximum Average Similarity as the second evaluation metric.

The ability of the crawler to remain focused on the topical Web pages during crawling can also be measured by the average relevance of the downloaded documents [53, 52]. Average Similarity is the accumulated similarity over the number of the crawled pages. The ideal crawler will always remain focused on the topic and have the maximum possible similarity all the time. This relevance measure also has been used in [53, 52, 75, 24]. In our system, since there are multiple user-marked or pre-specified target pages, if the query document is close to any one of the targets, it is considered as matching the user's interests or the topic. The query document is compared to all the target documents, and the highest similarity value is used to
calculate Average Similarity, which we called the *Maximum Average Similarity* $\sigma$.

$$\sigma = \max_{d \in T} \frac{\sum_{p \in S} \cos(p, d)}{|S|}$$

(2.9)

where $T$ is the set of target pages, $S$ is the set of pages crawled, $|S|$ is the number of targets, and

$$\cos(p, d) = \frac{\sum_{k \in p \cap d} w_{pk} \cdot w_{dk}}{\sqrt{\sum_{k \in p} w_{pk}^2 \sum_{k \in d} w_{dk}^2}}$$

is the standard cosine similarity function, and $w_{dk}$ is the weight of reduced vector term $k$ in document $d$. 
Chapter 3

Using HMM to Learn User Browsing Pattern for Focused Crawling

This chapter describes how to use Hidden Markov Model to learn a user’s browsing patterns for focused Web crawling. We focus on learning from user browsing behavior and to emulate these to find more relevant pages for the user. This work was published in [44, 43].

Consider the way a user searches for information on the Web. Usually the surfer has a specific topic in mind before starting a search. Then he/she will search for relevant pages by entering topic keywords into a search engine like Google or Yahoo. Keyword search typically results in thousands of hits. He/She will then select some results returned by the search engine and start browsing there. While browsing, he/she selectively follows links from page to page to identify what he/she really wants. If he/she reaches a page he/she is interested in, he/she may take time to read, print or save it. The process continues by choosing another link, or going back to the list of search engine results to follow other returned results, or typing new keywords to start a refined topic search. In short, the user refines keyword-based search by combining searches with further browsing to fulfill her topic-specific information need. What the user really seeks goes beyond lexical keyword search.

User surfing behavior is likely to follow an intuitive understanding of semantic topic hierarchies. The decision by the user to follow or ignore a specific link is based on a variety of clues on the page and humans are very good at making such decisions. The above observations suggest that the context of the hyperlinks the user follows reveals the user’s information need. If we can detect such sequential patterns hidden in the surfer’s topic-specific browsing, we may be able to build an effective focused crawler.

We use a light modification to Fig. 2.1 to focus on the user modeling. The system consists of three components: User Data Collection, User Modeling via Pattern
Learning, and Focused Crawling, as shown in Fig. 3.1. User Data Collection is already described in Section 2.1.1 in Chapter 2. The following sections mainly describe how to create a concept graph from the user data collection and the other two components.

![System Architecture Diagram]

Figure 3.1: System Architecture: User Data Collection, User Modelling via Pattern Learning, and Focused Crawling.

### 3.1 Concept Graph

To capture the semantic relations from the user's browsing session, we first categorize the Web pages visited by the user into different types. Given the local Web graph from the User Data Collection stage (in Section 2.1.1), we first use Latent Semantic Indexing (LSI) [11, 71] to identify semantic content, apply a clustering algorithm, then use the cluster information and the user Web graph to build a concept graph. Finally a HMM is built to learn sequential patterns leading to targets from the concept graph.

#### 3.1.1 LSI – Identification of Semantic Content

A topic the user browses on can be characterized by the keywords contained in Web pages on the topic. Different words may be used to describe the same concepts in different pages, so we bring context into consideration by representing documents using LSI. The documents in the Web graph are processed to extract the semantic
relationships among words in the collection. Latent Semantic Indexing (LSI) [11, 71] captures the semantic relationships among different words on the basis of co-occurrences in the document collection. It uses the singular value decomposition (SVD) [32, 29] to reduce the dimensions of the term-document space. Its effectiveness has been demonstrated empirically in many information retrieval applications leading to increased average retrieval precision [23, 8, 31, 12]. The main drawback of LSI is that the computation cost for large document collections is high. However, in our system, the user visited pages form a small collection with limited topic categories, thus LSI performance limitations are less of an issue.

In detail, we implement document representation using the following steps:

1. Form the dictionary of the collection by extracting all meaningful content words;
   We perform stemming, convert all characters to lower case, and truncate words
   with more than 20 characters;

2. For each document and term from the dictionary, calculate its normalized TF-IDF
   (Term Frequency, Inverse Document Frequency) score;

3. Construct the term-document matrix using the TF-IDF scores as entries;

4. Apply the SVD algorithm to obtain a low dimensional LSI space representation
   for the document collection.

The normalization in the weighting Step 2 is a scaling step designed to keep
large documents with many keywords from overwhelming smaller documents. The
normalized term weight for term $i$ in document $k$ is

$$W_{ik} = \frac{t_{f_{ik}}(\log_2 \frac{N}{df_i} + 1)}{\sqrt{\sum_{k=1}^{T}(t_{f_{ik}})^2(\log_2 \frac{N}{df_i} + 1)^2}}$$  \hspace{1cm} (3.1)

where,

$t_{f_{ik}}$ = the frequency of term $i$ in document $k$

$N$ = the number of documents in the collection

$df_i$ = the number of documents in which term $i$ appears

$T$ = the number of terms in the collection
In step 4, singular value decomposition breaks the term-document matrix $A$ down into the product of three matrices,

$$A = U \Sigma V^T,$$

where the columns of matrices $U$ and $V$ are the left and right singular vectors, respectively, and $\Sigma$ is a diagonal matrix of the singular values of $A$ sorted in decreasing order. The closest rank-$k$ approximation to the original matrix $A$ is constructed from the $k$-largest singular values. The number of reduced dimensions $k$ and the associated singular values and matrices $U_k$ and $V_k$ are determined by means of an iterative process using Lanczos algorithm [9, 10] with tolerance of the approximation $10^{-6}$. The implementation provided in [11] is used. Within the reduced space, semantically related terms and documents presumably lie near each other so that clusters of terms and documents are more readily identified.

### 3.1.2 Clustering

After obtaining the low-dimensional LSI representation, pages are grouped together to form semantically coherent clusters and a concept graph is created. In order to do this, a clustering algorithm is applied.

An important question is how to find the right number of clusters in the data. Therefore, we apply the X-means algorithm [60], which extends k-means with efficient estimation of the number of clusters within a given interval. We first normalize the reduced LSI document vectors, then we cluster all documents, except the pages marked as target by the user, using the X-means algorithm, where the number of clusters is in the interval $[3, 8]$. We felt that the interval $[3, 8]$ is a good compromise between flexibility in the number of clusters and constraining that number to a range that is neither trivially small (under 3) or excessively fragmented (over 8). All user marked target pages form a separate target cluster (cluster 0). They are not part of the clustering process. By clustering documents we try to capture the semantic relation between different Web pages (clusters) leading to the target pages (cluster 0).
We might use topic categorization instead of clustering based on pre-existing categories available online in the Open Directory Project (ODP)\(^1\) or Yahoo\(^2\). However, compared to user visited pages produced by topic-specific browsing, the categories in the ODP are too general. Furthermore, the ODP hierarchy does not reflect real-life Web linkage structure, and standard categorization in this way lacks page context information. Therefore, applying clustering within the context of user’s browsing is more reasonable than using standard categorization. In fact, our goal is to be able to capture the concept associative relation between different types of documents, i.e., document type A is more likely to lead to targets than document type B, rather than what exact categories they belong to.

After clustering, the associative relationships between groups are captured into a concept graph $G = (V, E)$, where each node is assigned to $C_i$, the label of the cluster it belongs to, as shown in Fig. 3.2(b). In the concept graph, we do not merge different nodes with the same cluster number, since doing that would cause linkage information between nodes to become lost.

![Web graph](a) ![Concept graph](b) ![Browsing Pattern](c)

Figure 3.2: User Modelling via Sequential Pattern Learning. $C_i$ is the label of cluster $i$, $T_j$ is the estimated hidden state.

\(^1\)http://dmoz.org
\(^2\)http://www.yahoo.com
3.2 User Modeling

We model focused crawling in the following way: We assume that there is an underlying Markov chain of hidden states, from which the actual topics of the documents are generated. Each state is defined by the distance, in number of hops, of the current page to a target page. In other words, hidden states represent the closeness of a page to a target, and the actual document is the observation that is probabilistically generated. Given a sequence of observations, we predict the next state the page will be in based on the observations so far. During the crawling, those pages with lower state subscript such as $T_1$ will be assigned higher visit priorities than those with higher state subscript such as $T_2$, and those pages with the same state are then sorted by the prediction scores, as shown in Fig. 3.2(c). The pattern learning procedure involves the estimation of the parameters of the underlying probabilistic model (HMM), that is, $\theta = \{\pi, A, B\}$. Details of the defined structure of HMMs for focused crawling and the parameters have been described in Section 2.2.1.

3.2.1 Parameter Estimation

Once the state-transition structure is defined, the model parameters are the state transition probability matrix ($A = [a_{ij}]$) and the emission probability matrix ($B = [b_{ij}]$). We use annotated training data, i.e. the concept graph with the identified target pages (see Section 3.1). We “label” all nodes in the concept graph as $T_0$, $T_1$, $T_2$, ..., $T_{k−1}$ in a Breadth-First search out of the set of target pages ($T_0$) as shown in Fig. 3.3. Probabilities are estimated by maximum likelihood as ratios of counts, as

![Concept graph](image)

Figure 3.3: Parameter Estimation of HMM
follows:

\[ a_{ij} = \frac{|L_{ij}|}{\sum_{r=0}^{k-1} |L_{rj}|} \]  

(3.3)

where, \( k \) is the number of hidden states, \( i, j = 0..k - 1 \), \( L_{ij} = \{ v \in T_i, w \in T_j : (v, w) \in E \} \).

\[ b_{ij} = \frac{|N_{ij}|}{\sum_{r=1}^{x} |N_{rj}|} \]  

(3.4)

where, \( x \) is the number of the clusters, \( i = 0..k - 1 \), \( j = 0..x \), \( N_{ij} = \{ C_i : C_i \in T_j \} \).

### 3.2.2 Efficient Inference

The task during the crawling phase is to associate a priority value with each URL that is being added to the queue. Inference using the HMM model is performed for this task. Given a downloaded web page \( w_t \), the URLs associated with its outgoing links are inserted into the queue and they all receive the same priority value, computed on the basis of the prediction of the state of the children of \( w_t \).

As we introduced before in Section 2.3.1, the estimated distribution of the state of \( w_t \), \( P(s_t|o_{1..t}) \), is a probability distribution over all possible state values \( T_0, T_1, \ldots, T_{k-1} \), given observations \( o_{1..t} \) and the distribution at step \( t - 1 \), corresponding to the parent page of \( w_t \), \( w_{t-1} \). This estimation problem is known as filtering. Parent page \( w_{t-1} \) of \( w_t \) is the web page containing the URL of \( w_t \), which was inserted earlier into the queue. The sequence \( 1..t \) refers to the visited pages along the shortest path from the seed page to the current page. The sequence does not necessarily reflect the temporal order in which pages have been visited.

In HMMs, the calculation of \( P(s_t|o_{1..t}) \) can be viewed as being composed of two steps: first the current state distribution is forwarded from previous time \( t - 1 \) to \( t \), then it is updated using the new observation \( o_t \).

\[
p(s_t|o_{1..t}) = p(s_t|o_{1..t-1}, o_t) \quad \text{(dividing the observation)}
\]

(3.5)

\[
= p(o_t|s_t, o_{1..t-1})p(s_t|o_{1..t-1}) \quad \text{(Bayes' rule)}
\]

(3.6)

\[
= p(o_t|s_t, o_{1..t}) \sum_{s_{t-1}} p(s_{t}|s_{t-1}, o_t)p(s_{t-1}|o_{1..t-1}) \quad \text{(Markov property)}
\]

(3.7)

\[
= p(o_t|s_t) \sum_{s_{t-1}} p(s_{t}|s_{t-1})p(s_{t-1}|o_{1..t-1}) \quad \text{(Markov property)}
\]

(3.8)
p(o_t|s_t, o_{1..t}) is replaced by p(o_t|s_t) because of the Markov assumption on o_t. Now we have the desired recursive formulation. The first term, p(o_t|s_t) is obtained directly from the emission probability matrix (B = [b_{ij}]_{k \times m}) (See Section 2.2.1 for the definition). Within the summation, the first factor is simply obtained from the state transition probability matrix (A = [a_{ij}]_{k \times k}). Hence, we can view the p(s_{t-1}|o_{1..t-1}) as a message that is propagated forward along the sequence from t - 1 to t, modified by the current transition and emission distributions. If we define the forward value α(s_j, t) as the probability that the system is in state s_j at time t given all observations up to time t, then according to the definition in Section 2.2.1, p(s_t|o_{1..t}) is easily calculated using dynamic programming. Given the values α(s_i, t - 1), i = 1..k of the parent page at time t - 1, we can calculate the values α(s_j, t) using the following recursion, derived from Equation 3.8:

\[ \alpha(s_j, t) = b_{jj'} \sum_{s_t} a_{ij} \alpha(s_i, t - 1) \]  

(3.9)

where, i, j = 1..k, j' = 1..m, s_i, s_j ∈ \{T_0, T_1, ..., T_{k-1}\}.

Based on p(s_t|o_{1..t}), a prediction step can be further carried out to estimate the distribution of the state of the children t + 1, which have not been seen yet, and therefore there is no observation o_{t+1} associated with them.

The probability distribution p(s_{t+1}|o_{1..t}) of the state at the next step t + 1 given observations O_{1..t} can be calculated from p(s_t|o_{1..t}) using the following equations:

\[ p(s_{t+1}|o_{1..t}) = p(s_{t+1}|s_t) \cdot p(s_t|o_{1..t}) \]  

(3.10)

where p(s_{t+1}|s_t) are the state transition probabilities.

From α(s_j, t) in Equation 3.9, we can calculate the one-step prediction for the future state s_{t+1} using Equation 3.10. Let α'(s_j, t) be the probability that the system will be in state s_j at the next time step, given the observations thus far. We have

\[ \alpha'(s_j, t) = \sum_{s_t} a_{ij} \alpha(s_i, t) \]  

(3.11)

The prediction for the next step involves only the transition probabilities because there is no additional evidence. All the α'(s_j, t) values are used to calculate the visit
priority. All the $\alpha(s_j, t)$ values are stored for each item of the priority queue for the prediction in next time step.

### 3.3 Focused Crawling

After the learning phase, the system is ready to start focused crawling. An overview of the crawling algorithm using HMMs is shown in Fig. 3.4. The crawler utilizes a queue, which is initialized with the starting URL of the crawl, and keeps all candidate URLs ordered by their visit priority value. We use a timeout of 10 seconds for Web downloads and filter out all but pages with text/html content. The crawler downloads the page pointed to by the URL at the head of the queue, calculates its reduced dimensionality (LSI) representation, and extracts all the outlinks. Then all child pages are downloaded and classified using the $K$-Nearest Neighbor algorithm into one of the clusters. The prediction of future state is calculated for each parent-child pair based on the current observations and corresponding HMM parameters $\theta = \{\pi, A, B\}$, and the visit priority values are assigned accordingly. The crawling algorithm with efficient inference using HMMs is described in detail in Section 3.3.3.

![Flow chart of focused crawling using HMM](image)

We also set a relevance threshold $\gamma$ for determining if a Web page is relevant to
the user’s interests. If its maximal cosine similarity to the target set is greater than \( \gamma \), the URL is stored and presented to the user as a relevant page.

3.3.1 Calculation of the Priority

The visit priority is determined based on the estimated future state distribution \( P(S_{t+1}|O_{1..t}) \), defining a vector key \((P(T_0), P(T_1), P(T_2), ..., P(T_{k-1}))\), where \( k \) is the number of hidden states.

If there are two or more items of approximately equal value in the first key, the key data items are ordered in decreasing order according to the second data item to break the tie. In our system, we use a threshold \( \varepsilon = 0.001 \) to define the equality of two key data items, i.e., for two state distributions \( X[P(T_0), P(T_1), P(T_2), ..., P(T_k)] \) and \( Y[P(T_0), P(T_1), P(T_2), ..., P(T_k)] \), if \( |X[P(T_0)] - Y[P(T_0)]| < \varepsilon \), the sorting process would use the second key data items pair \( X[P(T_1)] \) and \( Y[P(T_1)] \).

3.3.2 Priority Queue Data Structure

The priority queue contains the URLs of pages \( w_t \) to be visited sorted by the priority of the parent page \( w_{t-1} \) of page \( w_t \), defined as the page through which the URL of \( w_t \) was placed on the queue.

Each queue element consists of:

- the URL of page \( w_t \)
- cluster \( C_{t-1} \) of the parent page \( w_{t-1} \) of page \( w_t \)
- the visit priority of \( w_t \)
- probabilities \( \alpha(s_{j, t-1}) \) that page \( w_{t-1} \) is in hidden state \( s_j \), for all \( j \), capturing \( P(S_{t-1}|O_{1..t-1}) \). For simplicity, we use \( \alpha(j, t-1) \) instead of \( \alpha(s_{j, t-1}) \) in the algorithm below.

3.3.3 The Algorithm

The basic crawling algorithm is summarized in Fig. 3.5. The crawler utilizes a queue, which initialized with the starting seed URLs, repeatedly downloads the page pointed to by the URL at the head of the queue until the termination condition is satisfied.
Algorithm Focused_Crawler(HMM, $\gamma$, $k$)

$urlQueue := \{\text{Seed URLs}\};$

WHILE ( not(termination) ) DO

$w_t := \text{dequeue head of } urlQueue;$

extract from $w_t$ its URL, parent cluster $C_{t-1}$, and 
$\alpha(j, t - 1)$ for all possible states $j$;

Download contents of $w_t$;

Parse and classify page $w_t$ to cluster $C_t$;

IF $cos(w_t, TargetSet) > \gamma$, THEN store $w_t$ as relevant;

Calculate $\alpha(i, t)$ and priority for $w_t$'s children

IF $w_t$ is start URL,

THEN $\alpha(i, t) = \pi(i) \ b_{iC_t};$

ELSE $\alpha(i, t) = \sum_{j}(\alpha(j, t) \ a_{ji}) \ b_{iC_t};$

calculate $\alpha'(i, t);$ 

calculate prediction $[P(T_0), P(T_1), P(T_2), \ldots, P(T_k)];$

calculate the visit priority;

FOR EACH outlink $w_{t+1}$ of $w_t$ with $url$ (has the same priority)

$urlQueueEntry := (\text{priority, url, } C_t, \alpha(i, t));$

Enqueue $(urlQueueEntry); // \text{ entries sorted by priority}$

Figure 3.5: Pseudocode of Crawling Algorithm using HMMs

The termination condition can be specified differently. In the experiments of this thesis, the crawler terminates when the number of the downloaded pages reaches 20000. During the crawling, each Web page $w_t$ is associated with information about its parent in the form of cluster number $C_{t-1}$ and partial probabilities $\alpha(j, t - 1)$. If page $w_t$ is a start URL, its $\alpha(., .)$ will be calculated using the initial matrix, otherwise, it will be calculated based on information of its parent and current observations.

We now analyze the computational complexity of the crawling algorithm. For training, HMM training requires $O(NM)$ time, where $N$ is the size of training set, and $M$ is the number of state transitions which is not greater than $k^2$ ($k$ is the number of hidden states). The complexity of HMM crawling algorithm is $O(L + Q \log Q + M)$, where $L$ is the average number of outlinks per page, $Q$ is the size of URL queue, and $M$ is the number of state transitions. We use a hash table to implement the operations of inserting and removing URLs from the queue and checking if a newly extracted URL is already in the queue, yielding $O(L + Q \log Q)$ complexity. The HMM inference
takes $O(M)$ operations, since we must do a matrix-vector multiply for each page. Therefore, for HMM crawling, it takes $O(L + Q \log Q + M)$ in the general case.

### 3.4 Experiments

To collect training data, we selected topics from the Open Directory Project (ODP)$^3$ categories, which are neither too broad nor too narrow, such as Linux. Users are graduate students from computer science for each topic and they were asked to search for Web pages they are interested in related to the given topic. We also compare our focused crawler with a Best-First Search (BFS) crawler and a Context-Graph (CGS) crawler.

#### 3.4.1 Algorithms for Comparison

We now briefly describe the focused crawling algorithms against which we compare our focused crawler.

1. Best-First Search crawler (BFS):
   This crawler assigns priorities to all children of the current page using standard lexical cosine similarity between the set of target pages and the content of the current page. Since we have multiple targets, the visit priority order of a URL here is based on maximal cosine similarity, and the priority queue is sorted by this value.

2. Context-Graph crawler (CGS):
   We implemented the focused crawler which is based on the Context Graph method [24], in which Naïve Bayes classifiers are built for each layer, and a category "other". We use the likelihood with a threshold of 0.5 to assign weakly matching pages to "other". When a new page is seen, it will be classified to the winning layer $j$ it belongs to and its visit priority order is sorted by the link distance to layer 0, that is, the closer the layer $j$ is to layer 0, the higher its visit priority. Pages in the same layer are sorted by the likelihood score. For fair comparison of the learning performance, during the focused crawling, we didn’t

---

$^3$http://dmoz.org/
use the extra back-crawling feature which obtains and includes all immediate parents of every relevant page into the queue when it is discovered.

3.4.2 Training Data

We now describe the details of collecting training data for the three focused crawlers being compared.

1. **Building a Web graph using user visited pages for HMM crawling:**
   
   Data is collected from a Data Collection session as described in Section 2.1.1. Targets are all pages the user marked as relevant during browsing.

   To collect training data, the user is given a topic such as *Hockey* and is asked to search for Web pages he/she is interested in related to this topic. The user is asked to start browsing from 1 or 2 levels above the given topic in the ODP or from results returned by Google using keywords on topics 1 or 2 levels above the desired topic in ODP. Note that ODP was just used to determine starting pages, while user browsing was performed on the actual Web. For example, for the topic *Hockey*, he/she may start from the *Sports* level URLs in the ODP, or type *Sports* keywords in Google and start from some of the returned URLs. As a result, we will be able to capture for training hierarchical pages sequences. We found that not sufficiently many of such sequences were captured when the user started browsing from pages too close to the target pages (e.g., if he/she started browsing from pages on *Hockey* in our example).

   During browsing, the user clicks the *Useful* button to mark relevant Web pages, and clicks the *Submit* button to save the training data anytime.

2. **Building a Context Graph using backlinks for Context-Graph crawling:**

   Data is collected using the Google [34] backlink service, starting from target pages and proceeding up to 4 layers with a maximum of 300 pages in a single layer. Layer $i$ includes all nodes reaching targets by $i$ hops. As shown in Fig. 3.6, each circle indicates one layer, target pages form layer 0, and layer $i$ contains all the parents of the nodes in layer $i - 1$. We do not check for links between nodes
in the same layer and links between nodes in different layers except neighboring layers, as shown in Fig. 3.6. This data was used for the Context-Graph crawler (CGS).

![Diagram](image)

Figure 3.6: Build context graph. Each circle represents one layer. Targets form layer 0, and layer \( i \) contains all the parents of the nodes in layer \( i - 1 \).

3. **Building a Web graph using all nodes from the Context Graph for HMM crawling:**

   Combination of the above two. We use all nodes collected in the context graph to build a Web graph which includes all links between the nodes, not just the ones going to the next layer.

   For HMM crawling, we performed two sets of experiments using two kinds of training data (No. 1 and 3 above) for comparison. The motivation is to observe the effect on the performance of using data obtained from topic-specific browsing against using the complete Web graph.

3.4.3 **Results**

The number of relevant pages plotted against the number of pages downloaded for the topic *Linux* with relevance threshold 0.7 is shown in Fig. 3.7 (a), and the average similarity over all downloaded pages on the same topic is shown in Fig. 3.7 (b). In the following figures, in addition to the crawling methods we mentioned above (HMM,
CGS, and BFS), we also include noLSI method which is HMM crawling without LSI dimensionality reduction. It is used to demonstrate the impact of LSI on the performance and will be explained later.

![Graphs showing the number of relevant pages and average maximal similarity](image)

**Figure 3.7**: Topic *Linux*: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarity of all downloaded pages.

The system collected a total of 200 user browsed Web pages including 30 marked target pages. For the CGS crawler, we used the same target pages as layer 0 and constructed the context graph of depth 4. The results show that our system performs significantly better than Context-Graph crawling and Best-First crawling.

In this case, Best-First crawling performs very poorly and when we examine closely Fig. 3.7(b), at the very earlier stage, Best-First crawling pursued the links that appear the most promising (with higher similarity score) at the expense of large longer term loss, whereas our focused crawler explored suboptimal links that eventually lead to larger longer term gains. A typical example during the crawls is that starting from [http://www.comptechdoc.org](http://www.comptechdoc.org), our system follows links quickly leading to [http://www.comptechdoc.org/os/linux/manual](http://www.comptechdoc.org/os/linux/manual) and crawls a lot of links during the same domains, whereas Best-First crawler picks the branch leading to [http://www.iceteks.com/forums/](http://www.iceteks.com/forums/) and crawls many links under the branches that are far away from relevant pages. The URL visit order in Context-Graph crawling is based on the classification score on each layer, so the followed links include [http://www.comptechdoc.org/os/linux/](http://www.comptechdoc.org/os/linux/), [http://www.comptechdoc.org/os/windows/](http://www.comptechdoc.org/os/windows/), [http://www.comptechdoc.org/independent/](http://www.comptechdoc.org/independent/).
A different topic, Call for Papers is shown in Fig. 3.8(a) and Fig. 3.8(b). Again our system still showed very good performance compared to the other two. But in this case the Best-First crawler performs much better than Context-Graph crawler overall. The system collected a total of 153 user browsed Web pages and 4 marked target pages with the threshold $\gamma = 0.7$. The basic principle behind the Best-First crawler is linkage locality of the Web, that is the assumption that Web pages on a given topic are more likely to link to those on the same topic. The Context-Graph crawler captures physical layers leading to targets, however, if common hierarchies do not exist for each layer due to mixing links (cross links between topics) in the Web graph, it performs poorly, which reflects our arguments before. In contrast, our model takes both content and linkage structure into account during the learning procedure in order to learn semantic relation to targets, and is shown here to work well on both cases.

Other topics, Biking and Hockey, are shown in Fig. 3.9 and Fig. 3.10. For the topic Biking, the system collected a total of 697 user browsed Web pages and only 1 marked target pages. Since the target page information is very specific, consequently all methods return much fewer relevant pages and less values on maximum average similarity, as shown in Fig. 3.9, so we lower the threshold by 0.1 for comparison. In this case, HMM crawler still performs the best for the number of relevant pages (Fig. 3.9a). The Context-Graph crawler initially beats HMM crawler on the maximum
average similarity, but the number of relevant pages received is dramatically lower, indicating that at the earlier stage of crawl, the Context-Graph crawler gets high quality pages, but much fewer of them (Fig. 3.9b). In the long run, HMM crawler returns more relevant pages and shows better performance.

On the different topic Hockey the Best-First crawler shows very good performance compared to HMM crawler (Fig. 3.10). Both of them returned a large number of relevant pages with high maximum average similarities. The system collected a total of 201 user visited pages and 30 marked targets. As we mentioned before, Best-First crawler always pursues the links that have the highest cosine similarity, so if the
Figure 3.11: The effect of different relevance threshold values on topic Linux: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.

crawler enters a large topical subgraph where topical coherence persists, it normally performs very well. The drawback in some cases is that the crawler then tends to get stuck within the subgraph or one Web site, while other crawlers more easily leave the subgraph, resulting in a small proportion of relevant pages immediately, but reaching to more diverse target pages in the long run.

It is also worth mentioning that in the original work on the Context Graph method [24], it was demonstrated that the linkage locality of the Web does boost the performance when including the back-crawling feature: when a page is discovered as a relevant page, its immediate parents are obtained from the search engine and added to the crawling queue automatically. We omitted this feature in order to obtain a fair comparison, since the other two methods do not have access to the feature. Apparently our system could be further enriched by combining such features.

Next we examine the impact of LSI on the performance. As we mentioned before, LSI is useful in situations where traditional lexical information retrieval approaches fail. By reducing the dimensionality of the term-document space, the underlying semantic relationships between documents are revealed, and much of the noise is eliminated. Web pages contains more noisy data than regular text documents, since they may contains irrelevant information such as advertisements. We proposed to use LSI to minimize the word-based retrieval problems. To compare the performance without using LSI, we use only the regular term-document matrix for X-means clustering as
Figure 3.12: The effect of different relevance threshold values on topic Call for Papers: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.

Figure 3.13: The effect of different relevance threshold values on topic Hockey: (a) The number of relevant pages within the set of downloaded pages with threshold 0.8. (b) The number of relevant pages within the set of downloaded pages with threshold 0.6.
Figure 3.14: The effect of different relevance threshold values on topic *Biking*: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) The number of relevant pages with threshold 0.5.

described in Section 3.1.2. The results, as shown in Figs. 3.7 - 3.14, demonstrated that LSI does significantly boost the performance in almost all cases, except for the topic *Biking* with threshold 0.7 as shown in Fig. 3.14(a).

Since cosine similarity with a relevance threshold can be sensitive to the value of the threshold, we further investigate the effect of different threshold values. As shown in Fig. 3.7(a), Fig. 3.11, Fig. 3.8(a), and Fig. 3.12, our system shows robust results on both topic *Linux* and topic *Call for Papers* respect to change of the threshold. In fact, the better performance of HMM crawler is achieved, the higher threshold is used. For the topic *Hockey*, we saw that in Fig. 3.10, Best-first crawler performs almost matched HMM crawler for threshold 0.7. We observe similar behavior with threshold 0.6 and 0.8: for threshold 0.6, the behaviors of HMM and Best-First crawlers are almost identical, while for threshold 0.8, Best-First method is even better than HMM crawler (Fig. 3.13). On topic *Biking*, the HMM crawler showed stable behaviors on threshold 0.5 (Fig. 3.14b) and 0.6 (Fig. 3.9a), and extreme sensitivity to the choice of threshold of 0.7 (Fig. 3.14a). For threshold 0.7, the HMM crawler performs poorly and is outperformed by noLSI crawler. We observe that maximum average similarity is very low (see Fig. 3.9b) and the number of relevant pages is small for all crawlers, indicating that none of the crawlers performs well, because there is only one target page whose content is also too specific. Therefore, in this case, a higher threshold amplifies the sensitivity.
Figure 3.15: Comparison of HMM crawler using different training data on topic Linux: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data No.1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data No.3).

Figure 3.16: Comparison of HMM crawler using different training data on topic hockey: (a) The number of relevant pages within the set of downloaded pages with threshold 0.7. (b) Average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data No.1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data No.3).
Next we observe the effect of the training data on the learning performance. We compared the performance on two types of training data described in Section 3.4.2 for HMM-learning: HMM using user data (HMM), and HMM using the context graph with all links added (HMM_alldata) as described in Section 3.4.2 (3), shown in Fig. 3.15 and Fig. 3.16. The results show that using user visited pages as training data performs best. As we expected, building the Web graph from the user's browsing pattern not only extracts the dominant link structure leading to target pages, but also incorporates the judgment of the user about the nature of the Web pages. We found that constructing the Web graph using all nodes can bring too many noisy links into the learning process, resulting in bad performance.

3.5 Summary

This chapter described the use of Hidden Markov Models (HMMs) to learn from the user's browsing behavior for focused crawling. Following the system framework described in Chapter 2, we presented detailed algorithms for training and crawling. The motivation is that the context of the hyperlinks the user follows reveals the user's information need. In particular, we first collect the Web pages visited during a user browsing session. These pages are clustered, and the link structure among pages from different clusters is then used to learn page sequences that are likely to lead to target pages. We compared the performance with Context-Graph crawling and Best-First crawling. Our experiments demonstrated that HMM crawler trained from User Data performs better than Context-Graph crawler and Best-First crawler.
Chapter 4

Focused Crawling with Maximum Entropy Markov Models

In the HMM crawl we proposed in the previous chapter, all information about a visited page is captured in its cluster number. Keywords are only used implicitly in the construction of clusters and classification of pages. Features associated with anchor text, URL token, title or metadata are not accounted for. For instance, a document relevant to a specific topic frequently contains explicitly a set of topic-specific keywords, showing that the lexical keywords are significant factors and it is beneficial to use them directly. It may, therefore, be useful if a composite criterion were formed, that directly included keywords and other important features to capture sequential patterns leading to targets. The hope is that with a richer representation of multiple features, relevant paths will be effectively identified.

In this chapter, we extend our work on prediction of the links leading to relevant pages by using Maximum Entropy Markov Models (MEMMs). This will allow us to explore multiple overlapping features for training and crawling. All features and feature functions have been described in Section 2.2.2.

4.1 Maximum Entropy Markov Models (MEMMs)

MEMMs [66, 47] are a variation on the traditional Hidden Markov Models (HMMs). However, unlike generative HMMs, MEMMs maximize the conditional probability of the state sequence given the observation sequence, by expressing the conditional probability as a weighted sum of a set of features, and training a model to estimate the weights of these features that are consistent with the training data. These features can interact. Examples of such features for Web pages would be word identity, keywords in anchor surrounding text, keywords contained in the title/meta data of current page.

Intuitively, the principle of maximum entropy is simple: model all that is known and assume nothing about that which is unknown. In other words, given a collection
of facts, choose a model consistent with all the facts, but otherwise as uniform as possible. Accordingly, the model seeks to maximize the entropy of the posterior conditional distribution subject to the constraint that the expected values of certain feature functions as predicted by the model should comply with their corresponding empirical frequencies observed in a training set. Namely, we would like to choose an optimal model so as to minimize the difference between the predicted values and the observed values. The parametric form of the maximum entropy probability function is explained as follows.

4.1.1 Maximum Entropy Introduction

Maximum Entropy (ME or MaxEnt for short) model is a framework to estimate a probability distribution from data under given constraints. MaxEnt is more widely known as multinomial logistic regression. Maximum Entropy Modeling has been successfully applied to Computer Vision, Natural Language Processing and many other fields [5, 68]. Entropy is a measure of uniformity of a probability distribution, or its underlying uncertainty. Formally, consider a stochastic model of a random variable $x$, which produces an output value $y$. The model distribution $p(y|x)$ is the conditional probability of $y$ given a context $x$, and the conditional entropy of $y$ given $x$, $H(p)$, is given by definition:

$$H(p) = -\sum_x \bar{p}(x) \sum_y p(y|x) \log p(y|x)$$

$$= -\sum_{x,y} \bar{p}(x)p(y|x) \log p(y|x)$$

where $\bar{p}(x)$ is obtained by its empirical distribution of $x$ in the training data, and $p(y|x)$ is the model distribution to be estimated from the training data.

$H(p)$ will be maximized when the distribution $p(y|x)$ is as uniform as possible. In order to build a model that accurately encodes important information contained in the training data, we view the important information as constraints and represent them in the form of $m$ feature functions $f_i(x, y), i = 1..m$.

To ensure that the model agrees with the information encapsulated in the training data, the model distribution $p(y|x)$ is constrained so that the expectation of each
feature $f_i(x, y)$ with respect to the training data, given by

$$
\hat{E}_i = \sum_{x,y} \hat{p}(x, y)f_i(x, y)
$$

(4.3)

is equal to the expected value of that feature with respect to the model distribution $p(y|x)$:

$$
E_i = \sum_{x,y} \hat{p}(x)p(y|x)f_i(x, y)
$$

(4.4)

where $\hat{p}(x, y)$ is the empirical distribution of $x, y$ in the training data. The goal of maximum entropy is to seek the distribution $p(y|x)$ that satisfies all such constraints on feature values, while otherwise remaining as uniform as possible. In other words, to select the conditional probability $p^*(y|x)$ with maximum entropy $H(p)$ that satisfies the feature constraints $\forall i, \hat{E}_i = E_i$:

$$
p^*(y|x) = \arg \max_p H(p)
$$

(4.5)

This is a problem of constrained optimization on objective function $H(p)$ (Equation 4.2), given $m$ feature functions $f_i(x, y), (i = 1..m)$ as constraints. The solution is to apply the method of Lagrange multipliers to $H(p)$ from the theory of constrained optimization, and to consider the Lagrangian objective function $L$ with parameters $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ corresponding to weights for each feature function and $\gamma$ corresponding to the constraint that $p$ is a conditional probability distribution, that is, for all $x, \sum_y p(y|x) = 1$. Substituting Equation 4.2, 4.3 and 4.4 with the constraints that is $\hat{E}_i = E_i$, $L$ is written as

$$
L = - \sum_{x,y} \hat{p}(x)p(y|x) \log p(y|x)
$$

$$
+ \sum_i \lambda_i \left( \sum_{x,y} \hat{p}(x)p(y|x)f_i(x, y) - \hat{p}(x, y)f_i(x, y) \right)
$$

(4.6)

$$
+ \gamma \left( \sum_y p(y|x) - 1 \right)
$$

where $\hat{p}(x, y)$ is the empirical distribution of $x, y$ in the training data. This now gives us an equivalent unconstrained optimization problem, which we can solve by taking
partial derivatives of $L$ with respect to $p(y|x)$ from Equation 4.6:

$$\frac{\partial L}{\partial p(y|x)} = -p(x) (1 + \log p(y|x)) + \sum_i \lambda_i p(x) f_i(x, y) + \gamma$$

$L$ is maximized when setting the partial derivatives of $L$ to zero,

$$\frac{\partial L}{\partial p(y|x)} = -p(x) (1 + \log p(y|x)) + \sum_i \lambda_i p(x) f_i(x, y) + \gamma = 0$$

then we obtain $p(y|x)$:

$$p(y|x) = \frac{\exp \left( \sum_i \lambda_i f_i(x, y) \right)}{z(x)}$$

(4.7)

$$z(x) = \exp \left( 1 - \frac{\gamma}{p(x)} \right)$$

Recall that $p$ is a conditional probability distribution, that is, for all $x$, $\sum_y p(y|x) = 1$, we have

$$\sum_y p(y|x) = \sum_y \frac{\exp \left( \sum_i \lambda_i f_i(x, y) \right)}{z(x)} = 1$$

so we obtain $z(x)$:

$$z(x) = \exp \left( \sum_i \lambda_i f_i(x, y) \right)$$

(4.8)

Therefore, combining Equation 4.7 and Equation 4.8, we use the final format for $p(y|x)$ written as:

$$p(y|x) = \frac{\exp \left( \sum_i \lambda_i f_i(x, y) \right)}{z(x)}$$

(4.9)

$$z(x) = \sum_y \exp \left( \sum_i \lambda_i f_i(x, y) \right)$$

$z(x)$ is called the normalizing factor.

In other words, Maximum Entropy is based on an exponential conditional probabilistic model that has the form of a linear combination $\sum_i \lambda_i f_i(x, y)$. The weights $\lambda$ are the parameters of the model. Given the training data and the model form,
learning is to find the best set of feature weights $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ to make the conditional probability $p(y|x)$ of the observed $y$ values in the training data to be the highest, given the observations $x$, that is,

$$\lambda^* = \text{arg max}_\lambda \prod_j p(y^j|x^j)$$

where superscript $j$ represents training instances in the training set.

### 4.1.2 Maximum Entropy Markov Model

Maximum Entropy Markov Model or MEMM is an augmentation of the basic MaxEnt Model so that it can be applied to calculate the conditional probability for each element in a sequence. MEMMs were introduced by Ratnaparkhi [66] and McCallum et al [47]. MEMMs are probabilistic sequence models that define conditional probabilities of state sequences given observation sequences.

![Dependency Graphical structure of MEMM](image)

Figure 4.1: Dependency Graphical structure of MEMM for sequences, given state sequence $s = \{s_1, s_2, ..., s_n\}$ and input sequence $o = \{o_1, o_2, ..., o_n\}$, where $t$ ranges over input positions. Arrow shows dependency (cause).

Formally, let $s$ and $o$ be random variables ranging over observation sequences and their corresponding state (label) sequences respectively. We use $s = s_1, s_2, ..., s_n$ and $o = o_1, o_2, ..., o_n$ for the generic state sequence and observation sequence respectively, where $s$ and $o$ has the same length $n$. As shown in Fig. 4.1, state $s_t$ depends on observations $o_t$ and previous state $s_{t-1}$. Therefore, MEMMs are discriminative models that define the conditional probability of a state sequence $s$ given an observation sequence $o$, $p(s|o)$. 
Let $n$ be the length of the input sequence, $m$ be the number of features. Then $p(s|o)$ is written as

$$p(s|o) = p(s_1, s_2, ..., s_n|o_1, o_2, ..., o_n) \quad (4.10)$$

MEMMs make a first-order Markov independence assumption among states, that is, the current state depends only on the previous state and not on any earlier states, so $p(s|o)$ can be rewritten as:

$$p(s|o) = \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t) \quad (4.11)$$

where $t$ ranges over input positions $1..n$. Applying the maximum entropy principle as in Equation 4.9 by replacing $x$ with $o_t$ and $y$ with $s_t$ in each position $t$, we can rewrite $p(s_t|s_{t-1}, o_t)$ as the following:

$$p(s_t|s_{t-1}, o_t) = \frac{1}{z(o_t)} \exp\left(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)\right) \quad (4.12)$$

$$z(o_t) = \sum_{s' \in S} \exp\left(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o_t)\right) \quad (4.13)$$

where, $s_t$ is the state at time $t$, $o_t$ is the observation at time $t$, $S$ indicates a set of possible states corresponding to finite state machines (FSMs). Combining Equation 4.11 and Equation 4.12, we get

$$p(s|o) = \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t)$$

$$= \prod_{t=1}^{n} \frac{1}{z(o_t)} \exp\left(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)\right)$$

$$= \prod_{t=1}^{n} \exp\left(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)\right) \prod_{t=1}^{n} z(o_t)$$

$$= \frac{\exp\left(\sum_{i=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)\right)}{\prod_{t=1}^{n} z(o_t)} \quad (4.14)$$

$z(o_t)$ is calculated using Equation 4.13. As we have seen from Equation 4.14, MEMM defines the conditional probability of state sequence $s$ given observation sequence $o$ as the product of separate conditional probabilities $p(s_t|s_{t-1}, o_t)$ at each position $t$. 
ranging over all positions. Therefore, $z(o_t)$ is called the per-state normalizing factor. In other words, MEMM uses local normalization factor $z(o_t)$ to form a chain of local models, which implies each position $t$ contains a “next-state classifier” that makes the distribution sum to 1 across all next state $s_t$ at current position $t$.

The use of feature functions allows arbitrary, non-independent features in the observation sequence $o$. It relaxes the strong observation independence assumption in generative models HMMs (see Section 2.2.2). All feature functions we defined for MEMM crawling are described in Section 2.2.2.

4.1.3 Training MEMMs

MEMMs have parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ which are the weights for each feature function $f_1, f_2, ..., f_m$. Training means estimating these parameters from the training data. Given the training data $D$ consisting of $N$ state-observation sequences (described in Section 2.1.2), $D = \{S, O\} = \{(s^j, o^j)\}_{j=1}^N$ (we'll use superscripts $j$ to represent training instances), where each $o^j = \{o^j_1, o^j_2, ..., o^j_n\}$ is a sequence of observations with length $n$, and each $s^j = \{s^j_1, s^j_2, ..., s^j_n\}$ is the corresponding sequence of states. The task of training MEMMs is to choose values of parameters $\{\lambda_i\}$ which maximize the log-likelihood, $L'_\lambda = \log p(S|O)$, of the training data. We use $(s^j, o^j)$ to present the $j^{th}$ state-observation sequence from the training data set, $s^j_t, o^j_t$ to indicate the state and observation at position $t$ of the $j^{th}$ state-observation sequence respectively, then based on Equation 4.14, we have

$$L'_\lambda = \log p(S|O) = \log \prod_{j=1}^{N} p(s^j|o^j)$$

$$= \log \prod_{j=1}^{N} \prod_{t=1}^{n} \frac{\exp(\sum_{i=1}^{m} \lambda_i f_i(s^j_{t-1}, s^j_t, o^j_t))}{\prod_{t=1}^{n} \sum_{s' \in S} \exp(\sum_i \lambda_i f_i(s^j_{t-1}, s', o^j_t))}$$

$$= \sum_{j=1}^{N} \left( \sum_{t=1}^{n} \lambda_i f_i(s^j_{t-1}, s^j_t, o^j_t) \right) - \sum_{j=1}^{N} \log \prod_{t=1}^{n} \sum_{s' \in S} \exp\left( \sum_i \lambda_i f_i(s^j_{t-1}, s', o^j_t) \right)$$

$$= N(\lambda) - M(\lambda) \quad (4.15)$$
To perform the optimization of $L'_\lambda$ with respect to $\lambda$, we differentiate the log-likelihood $L'_\lambda$ to seek the zero of the gradient. For simplicity, we use $N(\lambda), M(\lambda)$ to denote the first sum and the second sum. Differentiating the first component $N(\lambda)$ with respect to parameter $\lambda_i$, gives:

$$
\frac{\partial N(\lambda)}{\partial \lambda_i} = \frac{\partial }{\partial \lambda_i} \sum_{j=1}^{N} \sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}^j, s_t^j, o_t^j) = \tilde{E}_i
$$

(4.16)

$\tilde{E}_i$ gives the empirical "count" for feature $f_i$ of the training data, equal to the sum of feature $f_i$ values for over position $t$ in all $N$ sequences. Similarly, differentiating the second component $M(\lambda)$ with respect to parameter $\lambda_i$, gives:

$$
\frac{\partial M(\lambda)}{\partial \lambda_i} = \frac{\partial }{\partial \lambda_i} \sum_{j=1}^{N} \log \prod_{t=1}^{n} \sum_{s' \in S} \exp \sum_{i} \lambda_i f_i(s_{t-1}^j, s', o_t^j) = \frac{\partial }{\partial \lambda_i} \sum_{j=1}^{N} \sum_{t=1}^{n} \log \sum_{s' \in S} \exp \sum_{i} \lambda_i f_i(s_{t-1}^j, s', o_t^j)
$$

$$
= \sum_{j=1}^{N} \sum_{t=1}^{n} \sum_{s \in S} \exp \sum_{i} \lambda_i f_i(s_{t-1}^j, s', o_t^j) \frac{\partial }{\partial \lambda_i} \sum_{s' \in S} \exp \sum_{i} \lambda_i f_i(s_{t-1}^j, s', o_t^j)
$$

$$
= \sum_{j=1}^{N} \sum_{t=1}^{n} \sum_{s \in S} p(s|s_{t-1}^j, o_t^j) f_i(s_{t-1}^j, s', o_t^j) = E_i
$$

(4.17)

$E_i$ gives us the expected "count" for feature $f_i$ using current parameters with respect to the model distribution $p$. So the derivative of $L'_\lambda$ is $\tilde{E}_i - E_i$. Therefore, the optimum parameters $\lambda$ are the ones for which the expected value of each feature equals its empirical value, that is, the derivative of $L'_\lambda$ is set to zero.

Before we discuss how to optimize this to estimate $\lambda$ by the iterative approaches, we need to solve the optimization problems, that is, feature weights can be very large (infinite), and iterative methods can take a long time to get to those infinities.

### 4.1.4 Smoothing

The goal of smoothing is to penalize large weights, since MaxEnt will learn very high weights which overfit the training data. There are a number of smoothing methods
for maximum entropy models [33, 25, 18, 19]. To avoid overfitting and sparsity, we use smoothing to penalize the likelihood with a Gaussian weight prior [18, 19] assuming that weights are distributed according to a Gaussian distribution with mean $\mu$ and variance $\sigma^2$.

$$p(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp \left[ -\frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} \right]$$

One way to understand the smoothing is that if we had a prior expectation that the weight values would not be very large, we could then balance the evidence suggesting large or infinite weights against the prior. Since the evidence would never totally defeat the prior, the weight values would be smoothed and kept small or finite. We can do this explicitly by changing the optimization objective function $\log p(S|O)$ in Equation 4.15 to maximum posterior likelihood $\log p(S, \lambda|O)$:

$$\log p(S, \lambda|O) = \log p(S|O) + \log p(\lambda)$$  \hspace{1cm} (4.18)

*Posterior  Evidence  Prior*

After substituting Equation 4.15 into Equation 4.18, we get $L_\lambda$

$$L_\lambda = N(\lambda) - M(\lambda) + \log p(\lambda)$$

$$= N(\lambda) - M(\lambda) + \sum_{i=1}^{m} \log p(\lambda_i)$$

$$= N(\lambda) - M(\lambda) + \sum_{i=1}^{m} \log \frac{1}{\sigma_i \sqrt{2\pi}} \exp \left[ -\frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} \right]$$

$$= N(\lambda) - M(\lambda) - \sum_{i=1}^{m} \frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2} + K$$

as the new objective function. Thus combining with Equation 4.16 and Equation 4.17, its derivative with respect to $\lambda_i$ is:

$$\frac{\partial L_\lambda}{\partial \lambda_i} = \sum_{j=1}^{N} \sum_{t=1}^{n} f_i(s_{t-1}^{j, i}, s_{t}^{j, i}, o_{t}^{j, i}) - \sum_{j=1}^{N} \sum_{t=1}^{n} \sum_{s' \in S} p(s'|s_{t-1}^{j}, o_{t}^{j}) f_i(s_{t-1}^{j}, s', o_{t}^{j}) - \sum_{i=1}^{m} \frac{\lambda_i}{\sigma^2}$$

$$= \tilde{E}_i - E_i - \sum_{i=1}^{m} \frac{\lambda_i}{\sigma^2}$$  \hspace{1cm} (4.19)
\( \tilde{E}_i, E_i \) are defined in Equations 4.16 and 4.17, representing empirical value and expectation value with respect to the current \( \lambda_i \) of each feature function \( f_i \) in the training data, respectively. The equation 4.19 is the first-derivative with respect to \( \lambda_i \) of the new objective function \( L_\lambda \) to be optimized. Therefore, the optimum parameter \( \lambda_i \) is obtained by setting equation 4.19 to zero, as discussed below.

4.2 Parameter Estimation

Now it is the time to discuss how to optimize \( L_\lambda \) given its derivative in Equation 4.19. The task is to estimate the parameters \( \lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\} \) to maximize \( L_\lambda \) when the gradient is zero. We now describe two iterative training methods used in the literature to compute the parameter \( \lambda \) on maximum entropy models: Iterative Scaling algorithms [21, 62, 47], Limited-Memory Quasi-Newton method L-BFGS [55, 28]. Methods can differ significantly in the number of iterations required to reach convergence.

4.2.1 Iterative Scaling Algorithms

Iterative Scaling methods such as GIS (Generalized Iterative Scaling) [21] and IIS (Improved Iterative Scaling) [4, 62, 63] are simple and guaranteed to converge. They are similar in form and computational cost to the expectation-maximization(EM) algorithm. IIS usually converges more quickly than GIS. The basic idea of iterative scaling algorithms is to find iteratively the parameter set \( \lambda \) that maximizes the Log-likelihood \( L_\lambda \). For each feature weight \( \lambda_i \), it starts with arbitrary initial value, say 1, and updates the weights as \( \lambda_i = \lambda_i + \delta_i \) for appropriately chosen \( \delta_i \) until convergence.

We now briefly describe the steps for each feature weight \( \lambda_i \):

1. Calculate the "count" of each feature \( i \) in the training data, \( \tilde{E}_i \).

2. Start with arbitrary initial \( \lambda_i = 1 \).

3. Repeat until converge: use current \( \lambda_i \) to calculate the expected value of feature \( i: E_i \)

4. Update \( \lambda_i = \lambda_i + \frac{1}{C} \log \frac{\tilde{E}_i}{\tilde{E}_i} \), where \( C \) is the total feature count.

See [21, 4, 63, 62, 47] for the more detailed algorithm.
4.2.2 Limited-Memory Quasi-Newton Method (L-BFGS)

Pure Newton methods are iterative approximation algorithms which construct the tangent quadratic objective function and use second order (curvature) information, or the gradient of the gradient, to find the search direction to shift the current estimate of the parameters. This involves the Hessian matrix inverse computation on each iteration, which is slow and impractical. The Quasi-Newton method avoids the exact Hessian inverse computation by building a simpler approximation from previous gradients and weight updates. However, for large scale problems with hundreds of thousands of parameters, the approximate Hessian is too large to store.

Limited-Memory Quasi-Newton method (L-BFGS) [55, 28] uses the last few gradient values and weight updates to implicitly approximate the Hessian matrix. There is no theoretical guidance on how much information should be kept. Earlier work [28] showed that 3 to 10 pairs works well. It can offer a substantial savings in storage requirements, while still giving favorable convergence properties. Earlier work [46, 28] also showed L-BFGS performs well and converges more quickly than Iterative Scaling algorithms and Conjugate Gradient methods. The later is a first-order method searching along a linear combination of the gradient and the previous search direction.

In our work, we use L-BFGS method for parameter approximation. It can simply be seen as a black-box procedure, only requiring the first-derivative of the function to be optimized, in our case, Equation 4.19.

4.3 Focused Crawling

We now put all parts specific to MEMMs crawling together following the general flow chart of focused crawling described in Fig. 2.11. The crawler downloads the page pointed to by the URL at the head of the queue, extracts all the outlinks, and creates a feature vector $F_t$ for each parent-child pair to include all the arguments to each feature function, which are needed for computing feature values (Section 2.2.2). Then the prediction of future state for each URL is calculated based on the feature functions and corresponding MEMM parameters $\lambda$, and the visit priority values are assigned accordingly (Section 4.3.1).

Like in HMMs crawling, which uses a relevance threshold $\gamma$, in MEMM crawling,
we apply the same method to determine if a Web page is relevant to targets or not. If so, the URL is stored and returned as a relevant page.

4.3.1 Efficient Inference

We now discuss two kinds of inference we are going to use in Focused Crawling stage. When the crawler sees a new page, the task of the inference is to estimate the probability that the page is in a given state \( s \) based on the values of all observed pages already visited before. We are using two major approaches to compute the probabilities in our experiments: marginal probability and the Viterbi algorithm, and they can be performed efficiently using dynamic programming.

Marginal Mode

The marginal probability of states at each position \( t \) in the sequence is defined as the probability of states given the observation sequence up to position \( t \). Specifically, the forward probability, \( \alpha(s,t) \) is defined as the probability of being in state \( s \) at position \( t \) given the observation sequence up to position \( t \). We set \( \alpha(s,1) \) equal to the probability of starting with state \( s \) and then the recursive steps are:

\[
\alpha(s,t) = \sum_{s'} \alpha(s', t-1) \ p(s'|s', o_t) \tag{4.20}
\]

The calculation of \( p(s'|s', o_t) \) is straightforward in MEMMs. As we discussed above, MEMM directly defines separate conditional probabilities \( p(s'|s', o_t) \) at each position \( t \), that is,

\[
p(s'|s', o_t) = \frac{1}{z(o_t)} \exp\left( \sum_{i=1}^{m} \lambda_i f_i(s', s, o_t) \right)
\]

\[
z(o_t) = \sum_{s' \in S} \exp \sum_{i=1}^{m} \lambda_i f_i(s', s'', o_t)
\]

substituting it into Equation 4.20, the forward probability values \( \alpha(s,t) \) is calculated easily.

In the notations for our system, hidden states are defined as \( T_j, j = 0..k-1 \), therefore we associate values \( \alpha(T_j, t) \) with each visited page. Forward value \( \alpha(T_j, t) \)
is the probability that the system is in state $T_j$ at time $t$, based on all observations made thus far. Given the values $\alpha(T_j, t - 1)$, $j = 0..k - 1$, of the parent page, we can calculate the values $\alpha(T_j, t)$ using the following recursion, derived from Equation 4.20:

$$\alpha(T_j, t) = \sum_{j' = 0}^{k-1} p(T_j|T_{j'}, o_t) \alpha(T_{j'}, t - 1)$$

(4.21)

Hence the values $\alpha(T_j, t)$ in our focused crawling system are calculated as:

$$\alpha(T_j, t) = \sum_{j' = 0}^{k-1} \alpha(T_{j'}, t - 1) \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o_t))$$

(4.22)

$$z(o_t) = \sum_{j' = 0}^{k-1} \exp(\sum_{i=1}^{m} \lambda_i f_i(T_{j'}, T_{j'}, o_t))$$

(4.23)

All the $\alpha(T_j, t)$ values for all $j = 0..k - 1$ are stored in the priority queue, which are used to determine the visit priority as described in Section 2.3.2, and to perform inference for the next time step.

The Viterbi Algorithm

In Viterbi decoding, the goal is to compute the most likely hidden state sequence given the data:

$$s^* = \arg \max_s p(s|o)$$

(4.24)

By Bellman’s principle of optimality, the most likely path to reach state $s_t$ consists of the most likely path to some state at time $t - 1$ followed by a transition to $s_t$. Hence the Viterbi algorithm can be derived accordingly. $\delta(s, t)$ is defined as the best score (the highest probability) over all possible configurations of the state sequence ending at the position $t$ in state $s$ given the observation up to position $t$. That is

$$\delta(s, t) = \max_{s'} \delta(s', t - 1) p(s'|s', o_t)$$

(4.25)

This is the same as the forward values (Equation 4.20), except we replace sum with max.

Similarly, back to our system detailed notations with hidden states defined as $T_j$, \ldots
\[ \delta(T_j, t) = \max_{j^* = 0}^{k-1} \delta(T_{j^*}, t - 1) \frac{1}{z(o_t)} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j^*}, o_t) \right) \quad (4.26) \]

\[ z(o_t) = \sum_{j^* = 0}^{k-1} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(T_{j^*}, T_{j^*}, o_t) \right) \quad (4.27) \]

4.3.2 Priority Queue Data Structure

To perform the dynamic programming for efficient inference during crawling, since MEMM crawling uses some features from parent page, we need to keep them in the queue. When a new page is seen, all features including features extracted from current page and those from parent page are collected together for prediction. Therefore, anchor text feature and URL token feature in the previous page should be kept in addition to the three basic elements. In summary, the following are four elements of the priority queue in MEMMs crawling:

- the URL of page \( w_t \)

- anchor text feature and URL Token feature vector \( P_t \) extracted from the parent page \( w_{t-1} \) of page \( w_t \)

- the visit priority of \( w_t \)

- probabilities \( \alpha(T_j, t-1) \) that page \( w_{t-1} \) is in hidden state \( T_j \), for all \( j \), capturing \( p(s_{t-1}|o_{1:t-1}) \).

4.3.3 The Algorithm

The flow chart of focused crawling with MEMMs is in Fig. 2.11 in Chapter 2. The pseudocode of crawling algorithm with MEMMs is in Fig. 4.2. The classifier\((d)\) is used for relevance judgement of document \( d \), and also can be used for computing Text Feature value \( O_1 \) and Description Feature value \( O_2 \) described in Section 2.2.2. In our experiments, we use the maximal cosine similarity to the set of target pages \( T \) with a confidence threshold \( \gamma \). That is, if \( \max_{t \in T} \cos(d, t_k) \geq \gamma \) then \( d \) is considered as relevant. Forward value \( \alpha(T_j, t) \) is Equation 4.22 for marginal mode and Equation
Algorithm Focused_Crawler(MEMM, classifier(d), k)
urlQueue := {Seed URLs};
WHILE( not(termination) ) DO
    \( w_t := \text{dequeue head of urlQueue}; \)
    extract from \( w_t \) its URL, possible anchor text
    and URL Token feature vector of parent page \( P_{t-1} \),
    and \( \alpha(T_j, t - 1) \) for all possible states \( j \);
    Download contents of \( w_t \);
    Parse and preprocess content, feature extraction vector \( F_t \);
    IF classifier(\( w_t \))=1, THEN store \( w_t \) as relevant;
    FOR EACH outlink \( w_{t+1} \) of \( w_t \) with url
        Calculate \( \alpha(T_j, t) \) for all \( j \) and priority for \( w_t \)'s children:
        \[ \alpha(T_j, t) = \sum_i \alpha(T_i, t - 1)p(T_j|T_i, a_t) \]
        \[ \alpha(T_j, t) = \max_i \alpha(T_i, t - 1)p(T_j|T_i, a_t) \]
        Assign the visit priority based on \( \alpha \);
        Update \( P_t \);
        urlQueueEntry := (priority, url, \( P_t \), \( \alpha \));
    Enqueue (urlQueueEntry); \// entries sorted by priority

Figure 4.2: Pseudocode of Crawling Algorithm with MEMMs

4.26 for Viterbi algorithm. Each feature vector \( F_t \) is different for each URL, therefore, the children of page \( w_t \) have different visit priorities.

We now analyze the computational complexity of the crawling algorithm with MEMM. For training, MEMM training requires \( O(I \times N \times F^2) \) time, where \( I \) is the number of iteration, \( N \) is the size of training set, and \( F \) is the number of features. The complexity of MEMM crawling algorithm is \( O(L + Q \log Q + F^2) \), where \( L \) is the average number of outlinks per page, \( Q \) is the size of URL queue, and \( F \) is the number of features. We use a hash table to implement the operations of inserting and removing URLs from the queue and checking if a newly extracted URL is already in the queue, yielding \( O(L + Q \log Q) \) complexity. The MEMM inference takes \( O(F^2) \) operations, since we must do a matrix-vector multiply for each page. Therefore, for MEMM crawling, it takes \( O(L + Q \log Q + F^2) \) in the general case.
4.4 Experiments

In this section, we conduct experiments to test our MEMM-based focused crawling approach empirically. The topics are chosen from the Open Directory Project (ODP)\textsuperscript{1} categories, and the target pages are chosen based on the listed URLs under each category. The 10 topics are:

/Computers/Software/Operating_Systems/Linux/
/Home/Gardening/Gardens/Wildlife/Butterfly/
/Arts/Performing_Arts/Dance/Ballet/
/Sports/Cycling/Mountain_Biking/Downhill/
/Business/Management/Management_Information_Systems/Call_For_Papers/
/Society/Religion_and_Spirituality/Yoga/
/Health/Nutrition/Disease_Prevention/Heart_Disease/
/Sports/Hockey/Ice_Hockey/Training/
/Science/Astronomy/Amateur/Sky_Maps_and_Atlases/
/Society/Law/Legal_Information/Computer_and_Technology_Law/Internet/

Detailed information about each topic such as target pages, start URLs and the number of training sequences can be found in Appendix A.

4.4.1 Evaluation

We continue to use two methods to evaluate the results. The first one is the number of pages crawled that are relevant. The second one is Maximum Average Similarity $\sigma$ as discussed in 2.4.

A baseline method Best-First Search (BFS) crawler is used for comparison. BFS crawler assigns priorities to all children of the current page using standard lexical cosine similarity between the set of target pages and the content of the current page. Since we have multiple targets, the visit priority order of a URL here is based on maximal cosine similarity, and the priority queue is sorted by this value.

\textsuperscript{1}http://dmoz.org/
4.4.2 Results

We have conducted two experiments. One is to compare MEMM-based method with different inference algorithms against BFS crawl. Another one is to test the impact of the multiple features on the performance.

Comparison with Different Inference Algorithms: Viterbi Algorithm and Marginal Mode

This section shows the results of MEMM-based crawler using marginal mode and the Viterbi algorithm for the inference, as discussed in Section 4.3.1. We use BFS, memm-marginal, memm-viterbi in the figures to denote Best-First Search crawler (BFS crawl), MEMM crawl with marginal mode inference (MEMM-marginal crawl), and MEMM crawl with Viterbi algorithm (MEMM-Viterbi crawl), respectively. We choose the threshold between 0.5-0.8 for general comparisons. Too high or too low threshold may result in too few or too many relevant pages based on the target pages and the start URLs, which does not provide sufficient information for comparison.

![Graphs showing comparisons of different methods](image)

(a) Threshold = 0.6  
(b) Threshold = 0.7

Figure 4.3: Topic Fitnessyoga: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds.

First we compare our MEMM-based methods with all the features against BFS crawl. We find that our MEMM-based crawl significantly outperforms BFS crawl on 8 out of 10 topics. Fig. 4.3 to Fig. 4.13 show some of the results.
Figure 4.4: Comparisons of different crawl methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Fitnessyoga. (b) Topic Linux.

Fig. 4.3 shows the performance of three different crawls on the topic Fitnessyoga. All the crawl methods: BFS crawl, MEMM-marginal crawl, and MEMM-Viterbi crawl give very good results on this topic, however, two MEMM-based crawls still work better than BFS crawl on the number of relevant pages returned, which also is confirmed on the Maximum Average Similarity metric, as shown in Fig. 4.4(a).

Fig. 4.5 shows the results of topic Linux. Both MEMM-marginal crawl and MEMM-Viterbi crawl also outperform BFS crawl. MEMM with marginal mode shows significant improvement over BFS crawl, while MEMM with Viterbi algorithm shows slight improvement on the number of the relevant pages. However, MEMM-Viterbi crawl gives a very close performance to MEMM-marginal crawl on the Maximum Average Similarity, which significantly outperforms BFS crawl as shown in Fig. 4.4(b). This shows that two MEMM-based methods stay on the topic, whereas BFS method crawls away from the topic and gives very poor performance. When we examine the crawl histories of these crawls, we found that BFS crawl chooses to follow links http://opensource.linux-mirror.org/, http://www.ddj.com/linux/, and sticks in searching for URLs in the http://www.ddj.com/ and http://os3.fsfmirror.com/, in which topics are mixed with other software development and operating systems. MEMM-marginal crawl makes a better judgement to follow the links to reach http://www.linux-m68k.org/ and http://howtos.linux.com/howtos/,
Figure 4.5: Topic *Linux*: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

where many relevant pages are found. MEMM-Viterbi crawl follows the links in http://linux-directory.com/, http://tlp.org/FAQ/Linux-FAQ/, and http://howtos.linux.com/howtos/. A similar situation happened in topic *Butterfly* shown in Fig. 4.13, in which BFS crawl also performs very poorly. This reflects the major problem of the Best-First search crawl criteria, which is pursuing short-term gains from the local pages at the expenses of less-obvious crawl paths that ultimately yield larger sets of valuable pages in a longer run.

The crawl performance of topic *Callforpapers* and the topic *Balletdance* are shown in Fig. 4.6 and Fig. 4.7, which also give constant better performance over BFS crawl on both the number of relevant pages and on the Maximum Average Similarity (Fig. 4.8). Compared to the topic *Balletdance*, the number of relevant pages returned for topic *Callforpapers* is much smaller. This probably is due to the fact that there are very fewer relevant pages on the topic *Callforpapers* based on the target pages selected, while apparently topic *Balletdance* is a much popular topic therefore there are tons of relevant pages existing on the Web.

BFS crawl shows better performance only on the topic *Hearthealthy* and the topic *Skynaps*, as shown in Fig. 4.9 and Fig. 4.11. Although MEMM-marginal crawl and MEMM-Viterbi crawl perform better than BFS crawl on topic *Hearthealthy* in the
Figure 4.6: Topic Callforpapers: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds.

Figure 4.7: Topic Balletdance: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages with different thresholds.
Figure 4.8: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Callforpapers. (b) Topic Balletdance.

Figure 4.9: Topic Hearthealthy: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages within the set of downloaded pages with different thresholds.
Figure 4.10: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the Maximum Average Similarity. (a) Topic Internetlaw. (b) Topic Hearthealthy.

Figure 4.11: Topic Skymaps: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.
Figure 4.12: Topic Internetlaw: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages within the set of downloaded pages with different thresholds.

Later stage of the crawl (Fig. 4.9), the performance on the Maximum Average Similarity in Fig. 4.10(b) shows that BFS crawl gives overall better performance than MEMM-marginal crawl and MEMM-Viterbi crawl. As we mentioned before, BFS crawl works very well when it enters large topical subgraphs where topical coherence persists, since BFS crawl always follows the links with highest similarity to the targets. Another possible factor is that topic Hearthealthy and the topic Skymaps are more domain-specific topics, in which keywords specific to these topics in the page content play an important role for the URL expansions.

Compared to the results only based on MEMM-marginal and MEMM-viterbi crawls, we found that the marginal mode outperforms the Viterbi algorithm on 7 topics out of 10. Topic Butterfly (Fig. 4.13) and Topic Hearthealthy (Fig. 4.9) showed us a very close or mixed performance when using MEMM-marginal crawl and MEMM-Viterbi crawl. With Topic Butterfly, both MEMM-marginal crawl and MEMM-Viterbi crawl outperform BFS crawl significantly, but MEMM-marginal crawl performs better than MEMM-Viterbi crawl with threshold 0.7, while performs worse with threshold 0.8. With Topic Hearthealthy, MEMM-marginal crawl and MEMM-Viterbi crawl have a very close performance against each other, on the number of the relevant pages (Fig. 4.9) and Maximum Average Similarity (Fig. 4.10(b)).
Figure 4.13: Topic *Butterfly*: Comparisons of different methods: BFS, MEMM-marginal and MEMM-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

As we can see in Fig. 4.6, MEMM-marginal crawl performs worse than MEMM-Viterbi crawl at the beginning of the crawl time, but performs better in the later stage of the crawl, which is also presented in Fig. 4.8(a) on the *Maximum Average Similarity* measure. Similar performance can also be found in topic *Internetlaw* (Fig. 4.12). As we have described in previous sections, MEMM-Viterbi crawl uses the exactly same underlying MEMM model with the same features as MEMM-marginal crawl, except we replace sum with max to calculate the state probability distribution in the inference during the crawling stage (Section 4.3.1). Recall that MEMMs is a chain of a "next-state classifier" that all the probability mass that arrives at a state must be distributed among the possible successor states, and at each position $t$, the marginal probability is to find the most likely state at current position given the observation sequence up to $t$, while the Viterbi algorithm is to find the most likely path to reach state $s_t$ based on the principle that the most likely path to reach state $s_t$ consists of the most likely path to *some* state at time $t-1$ followed by a transition to $s_t$. In the focused crawling problem, finding the distribution for each individual hidden state at a particular instant is more important than finding the best "string" of hidden states of each Web page along the sequence, since there may be many very unlikely paths that led to large marginal probability. To be more specific, let us see an example. If we have
“aaa” 30% probability
“abb” 20% probability
“bab” 25% probability
“bbb” 25% probability

In this example, “aaa” is the most likely sequence, ‘a’ is the most likely first character, ‘a’ is the most likely second character, ‘b’ is the most likely third character, but the string “aab” has 0 probability.

Therefore, as we expected, MEMM-marginal crawl shows better performance than MEMM-Viterbi crawl in most of the cases in our experiments.

Comparison with Different Features

In this section, we test the impact of the features on the performance. The last section shows that MEMM-marginal crawl is better than MEMM-Viterbi crawl, so in this experiment, we choose to compare MEMM-marginal crawl with different features: with all features, with Word feature only, and with the remaining features: Text feature, Description feature, URL token feature and Anchor text feature (See all features in Section 2.2.2). We denote them as MEMM-marginal, MEMM-marginal-word, and MEMM-marginal-sim-meta-T in the figures, respectively.

As figures Fig. 4.14 to Fig. 4.21 show, MEMM-marginal crawl with all features combination performs constantly better than with Word feature only and with sim-meta-T features on almost all topics, except topic Hearthealthy (Fig. 4.21(a)), in which MEMM-marginal-word shows much better performance than MEMM-marginal crawl with all features, and topic Callforpapers (Fig. 4.21(b)), in which MEMM-sim-meta-T shows better performance over MEMM-marginal with all features. This confirms our approach that with multiple features representation, even if only some of them are present, relevant path can be effectively identified.

Individual features also play important roles on the performance. As we can see from the figures, MEMM-marginal with Word features performs better than MEMM-marginal with sim-meta-T on topics Balletdance (Fig. 4.16), Fitnessyoga (Fig. 4.15), Hearthealthy (Fig. 4.21(a)), and Linux (Fig. 4.14), whereas Memm-marginal with sim-meta-T features performs better than MEMM-marginal with Word feature on topics
Figure 4.14: Topic Linux: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.15: Topic Fitnessyoga: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.
Figure 4.16: Topic Balletdance: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.17: Topic Hockey: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.
Figure 4.18: Topic **Biking**: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 4.19: Topic **Butterfly**: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.
Figure 4.20: Comparisons of different methods: BFS, memm\_marginal and memm\_viterbi on the number of relevant pages. (a) Topic *Internetlaw*. (b) Topic *Skymaps*.

Figure 4.21: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within pages. (a) Topic *Hearthealthy*. (b) Topic *Callforpapers*. 
Skymaps (Fig. 4.20(b)), Callforpapers (Fig. 4.21(b)), and Internetlaw (Fig. 4.20(a)).

4.5 Summary

This chapter described the use of Maximum Entropy Markov Models (MEMMs) for the focused crawling problem. As a discriminative model, MEMMs have the flexibility to include multiple overlapping features. Following the system framework described in Chapter 2, we presented detailed algorithms for training and crawling for the focused crawling. Experimental results showed that MEMM-based crawl performs better than BFS crawl, and the performance using MEMM crawl with marginal mode is better than using Viterbi algorithm. We also investigated the impact of the features on the performance, and found that the crawler using the combination of all features performs constantly better than the crawler that depends on just one or some of them.
Chapter 5

Focused Crawling with Conditional Random Fields

This chapter describes focused crawling with Conditional Random Fields (CRFs). CRFs are a form of undirected graphical models. It specifies a single log-linear distribution \( p(s|o) \) over the entire state sequence \( s = s_1, s_2, ..., s_n \) given the observation sequence \( o = o_1, o_2, ..., o_n \), rather than defining per-state distributions over the next states given the current state at each position, as in MEMMs.

5.1 Undirected Graphical Models

As we discussed in previous chapters, HMMs and MEMMs are directed models. In directed models, the conditional independence relationships between the nodes can be clearly identified by the topology of the graph. For example, the conditional probability \( p(s|o) \) in MEMMs is defined as

\[
p(s|o) = \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t)
\]  

(5.1)

and the joint probability \( p(s, o) \) in HMMs is defined as

\[
p(s, o) = \prod_{t=1}^{n} p(s_t|s_{t-1})p(o_t|s_t)
\]  

(5.2)

Formally, the probability distributions in a directed graphical model \( G' = \{V', E'\} \) where \( V' = \{v'_1, v'_2, ..., v'_n\} \), can be written into a product of conditional probabilities between the nodes and their parents in the graph \( G' \). That is, a directed model is a family of distributions that factorizes as:

\[
p(v'_1, v'_2, ..., v'_n) = \prod_{v' \in V'} p(v'|parent(v'))
\]  

(5.3)

However it is different in an undirected graphical model. A Markov Random
Figure 5.1: An example of undirected graph: \((v_1, v_2, v_3)\) and \((v_3, v_4)\) are maximal cliques.

Field [30], also known as Markov networks or undirected graphical model, is a graph \(G = \{V, E\}\) where \(V = \{v_1, v_2, ..., v_n\}\) is a set of nodes and \(E\) is a set of undirected edges between nodes (for example see Fig. 5.1). The nodes \(V\) represent a set of continuous or discrete random variables. The edges \(E\) represent the probabilistic dependencies between these variables. The undirected nature means that it is difficult to ensure that the conditional probability of any node given its neighbors is consistent with the conditional probabilities of the other nodes in the graph. That is, we cannot ensure that the conditional probabilities assigned to the nodes yield a single joint distribution over all random variables in the graph. Therefore, the joint distribution of an undirected model cannot be factorized directly as a product of conditional probabilities.

To model the probability distributions over these random variables, we need to explore conditional independence relations between the random variables within the graph. The solution is to define the joint probability \(p(v_1, v_2, ..., v_n)\) as the product of a set of potential functions \(\Psi\) such that all independent random variables in the model do not appear in the same potential function, that is, each potential function only operates on the clique. A clique of a graph is its maximal complete subgraph that cannot be extended to include additional nodes and simultaneously remain fully connected (for example see Fig. 5.1). Therefore the joint probability of the variables represented in the undirected graph \(p(v_1, v_2, ..., v_n)\) can be defined as the product of the potential functions over all the maximal cliques in \(G\), that is,

\[
p(v_1, v_2, ..., v_n) = \frac{1}{Z} \prod_{c \in C} \Psi_c(v^c)
\]  \hspace{1cm} (5.4)
where $C$ is the set of all cliques in $G$, $v^c$ are the nodes $v$ in the clique $c$, $\Psi_c(v^c)$ is a potential function over the nodes of clique $c$, $Z$ is a normalization factor to satisfy the axioms of probability, which is defined as:

$$Z = \sum_v \prod_{c \in C} \Psi_c(v^c)$$  \hfill (5.5)

For example, the model in Figure 5.1 represents

$$p(v_1, v_2, v_3, v_4) = \frac{1}{Z} \Psi(v_1, v_2, v_3) \Psi(v_3, v_4)$$  \hfill (5.6)

Potential functions $\Psi$ can be defined differently. For example, if we assume that each potential function has the exponential family form of $m$ weighted features \{\textit{f}_i\}, then

$$\Psi_c(v^c) = \exp \left( \sum_{i=1}^{m} \lambda_i f_i(v^c) \right)$$  \hfill (5.7)

5.2 Conditional Random Fields (CRFs)

A conditional random field (CRF) [39, 77] is a form of undirected graphical model or Markov Random Field. General CRFs are defined as follows in [39]: let $G = \{ V, E \}$ be a undirected graph such that $s = \{ s_v \}_{v \in V}$, so that $s$ is indexed by the vertices of $G$. Then $(s, o)$ is a conditional random field in case, when conditioned on $o$, the random variables $s_v$ obey the Markov property with respect to the graph: $p(s_v | o, s_w, w \neq v) = p(s_v | o, s_w, w \sim v)$, where $w \sim v$ means that $w$ and $v$ are neighbors in $G$. In other words, a single node $s_v$ is independent of all the other nodes in the graph given its neighbors, i.e., $p(s_v | o, \text{all other } s) = p(s_v | o, \text{neighbors}(s_v))$; this is known as the local Markov property. For example, suppose $s = \{ s_1, s_2, s_3, s_4, s_5 \}$ as shown in Fig. 5.2, if $p(s_3 | o, s_1, s_2, s_4, s_5) = p(s_3 | o, s_2, s_4)$, then $o$ with $s$ is a conditional random field.

Thus, a CRF is a random field globally conditioned on the $o$. Think of $o$ as observations and $s$ as states (labels). As a typical special case, when modeling sequences, $G$ is a simple chain structure, so that $s = \{ s_1, s_2, ..., s_n \}$, $o = \{ o_1, o_2, ..., o_n \}$ for $t = 1, n$, and $s$ is the state sequence of the observation sequence $o$. If each potential function takes the exponential form (Equation 5.7), let $\lambda = \{ \lambda_1, \lambda_2, ..., \lambda_m \}$ be a weight vector,
then CRFs define the joint probability:

$$p(s, o) = \frac{1}{Z} \prod_{c \in C, c \neq o} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(s_c, o) \right)$$ (5.8)

where $Z$ is a constant, $C$ is the set of all cliques in $G$ and $f_i$ is feature function with weight $\lambda_i$. Thus, the conditional distribution $p(s|o)$ is

$$p(s|o) = \frac{p(s, o)}{\sum_{s'} p(s', o)} = \frac{\prod_{c \in C, c \neq o} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(s_c, o) \right)}{\sum_{s'} \prod_{c \in C, c \neq o} \exp \left( \sum_{i=1}^{m} \lambda_i f_i(s'_c, o) \right)}$$ (5.9)

Figure 5.3: Graphical structure of a linear-chain CRF for sequences, given state sequence $s = s_1, s_2, ... , s_n$ and input sequence $o = o_1, o_2, ... , o_n$, where $t$ ranges over input positions. The dashed box over $o$'s denotes the sets of observation sequence variables. Although we have shown links only to observations at the same step, the state nodes can depend on observations at any time step.

When modeling sequences, the simplest and most common linear-chain structure (Fig. 5.3) is used, one where all the nodes in the graph form a linear chain. Such
models are extensively used in POS tagging, information extraction, document summarization and shallow parsing [39, 64, 61, 73, 28]. In these models, the set of cliques \( C \) is just the set of all cliques of size 1 (the nodes) and the set of all cliques of size 2 (the edges), although CRFs can be used more generally. When we model the focused crawling problem with hyperlinked Web pages, the state (label) of a page is assumed to be dependent upon the page itself and the states of the pages that link into it or out of it, which corresponds to an undirected graphical model in the shape of a linear chain. Thus the conditional probability \( p(s|o) \) for a linear-chain CRF can be rewritten as the following, derived from Equation 5.9:

\[
p(s|o) = \frac{1}{Z(o)} \exp \left( \sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o) \right) \tag{5.10}
\]

where \( Z(o) \) is a normalization on \( o \):

\[
Z(o) = \sum_{s' \in |S|^n} \exp \left( \sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o) \right) \tag{5.11}
\]

The space of \( s' \) is now the space of state sequences, thus the number of all possible state sequences is exponential in the length of the input sequence \( n, |S|^n \).

Note the difference compared with MEMMs: the capitalized \( Z(o) \) represents a global normalization factor on the whole sequence \( o \), while \( z(o_t) \) in MEMMs represents a local normalization factor at position \( t \) of the sequence \( o \).

As mentioned above, in the case of linear-chain CRFs, each feature function will operate only on pairs of state variables \( s_{t-1} \) and \( s_t \) for each state transition, therefore, we can simply define \( f_i(s_{t-1}, s_t, o) \) as \( f_i(s, o, t) \), and rewrite the equation above as

\[
p(s|o) = \frac{1}{Z(o)} \exp \left( \sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s, o, t) \right) \tag{5.12}
\]

\[
Z(o) = \sum_{s' \in |S|^n} \exp \left( \sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s', o, t) \right) \tag{5.13}
\]
5.3 Parameter Estimation in CRFs

Assume we have training data $D$ with $N$ sequence pairs $\{(s^j, o^j)\}_{j=1}^N$, the training is to find $\lambda$ to maximize the conditional log-likelihood, $L_\lambda$, of all sequences in the training data set. We use $(s^j, o^j)$ to present the $j^{th}$ page sequence pair from the training data set, then

$$L_\lambda = \sum_{j=1}^N \log p(s^j | o^j)$$

$$= \sum_{j=1}^N \log \left[ \frac{1}{Z(o^j)} \exp(\sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s^j, o^j, t)) \right]$$

$$= \sum_{j=1}^N \left[ \log \frac{1}{Z(o^j)} + \sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s^j, o^j, t) \right]$$

$$= \sum_{j=1}^N \left[ \sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s^j, o^j, t) - \log Z(o^j) \right]$$

(5.14)

where

$$Z(o^j) = \sum_{s' \in \mathcal{S}^n} \exp \left( \sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s', o^j, t) \right)$$

(5.15)

To perform this optimization, differentiate the log-likelihood with respect to parameter $\lambda$ to seek the zero of the gradient

$$\frac{\partial L_\lambda}{\partial \lambda} = \sum_{j=1}^N \left[ \sum_{t=1}^n \sum_{i=1}^m f_i(s^j, o^j, t) - \sum_{s' \in \mathcal{S}^n} p(s' | o^j) \sum_{t=1}^n \sum_{i=1}^m f_i(s', o^j, t) \right]$$

(5.16)

$$\frac{\partial L_\lambda}{\partial \lambda_i} = \sum_{j=1}^N \sum_{t=1}^n f_i(s^j, o^j, t) -$$

$$\frac{\sum_{j=1}^N \sum_{s' \in \mathcal{S}^n} (\exp(\sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s', o^j, t)) \sum_{t=1}^n f_i(s', o^j, t))}{\sum_{j=1}^N \sum_{s' \in \mathcal{S}^n} \exp(\sum_{t=1}^n \sum_{i=1}^m \lambda_i f_i(s', o^j, t))}$$

$$= \sum_{j=1}^N \left[ \sum_{t=1}^n f_i(s^j, o^j, t) - \sum_{s' \in \mathcal{S}^n} p(s' | o^j) \sum_{t=1}^n f_i(s', o^j, t) \right]$$

(5.17)
Note, it looks similar as that in MEMMs, but they are different. The first part inside the parenthesis of the right-hand side of Equation 5.17 is the total sum of the values of feature function $i$ for all positions given sequences $o^j$ and $s^j$, and it can be calculated easily from the training data set. The second part is to compute the sum of the expectation value of each feature function with respect to the current CRF parameter for every observation sequence $o^j$ in the training data. There are $|S|^n$ possible state sequences, so the computation is prohibitively expensive and must be summed over using dynamic programming.

Since the conditional distribution obeys the Markov property, the second part inside the parenthesis of the right-hand side of Equation 5.17 can be written as

\[
\sum_{s' \in |S|^n} p(s'|o^j) \sum_{t=1}^{n} f_i(s', o^j, t) = \sum_{t=1}^{n} \sum_{v, w} p(s_{t-1} = v, s_t = w|o^j) f_i(v, w, o^j, t)
\]  

(5.18)

Thus it eliminates the need to sum over $|S|^n$ possible state sequences, and breaks the calculation into pairs.

The second part of Equation 5.17 can be done efficiently using a dynamic programming method. To do so, first we augment the graph with two additional state nodes, $s_0$ and $s_{n+1}$, with corresponding observations start and stop respectively, to the top (left) and bottom (right) level of the Web graph. That is, $s = s_0, s_1, s_2, ..., s_n, s_{n+1}$ is the state sequence, $o = start, o_1, o_2, ..., o_n, stop$ is the corresponding observation sequence. For a given $o$, we define a set of $n+1$ transition matrices for position $t$, ($t = 1..n+1$), and each matrix $M_t$ is a $n + 1$ by $n + 1$ matrix with elements of the form

\[
M_t[s', s] = \exp(\sum_{i=1}^{m} \lambda_i f_i(s', s, o, t))
\]

Thus, the probability of $s$ given observation sequence $o$ can be written as the product of the appropriate elements of the $n + 1$ transition matrices as following:

\[
p(s|o) = \frac{1}{\mathcal{Z}(o)} \prod_{t=1}^{n+1} M_t[s_{t-1}, s_t]
\]  

(5.19)
We define forward vector $\alpha_t = \alpha(s, t)$ as the unnormalized probability of being in state $s$ ending at position $t$ and backward vector $\beta_t = \beta(s, t)$ as the unnormalized probability of being in state $s$ at position $t$ given the observation sequence after position $t$ respectively for each node in the graph as

$$
\alpha_t = \begin{cases} 
\alpha_{t-1} M_t & 0 < t \leq n + 1 \\
1 & t = 0.
\end{cases}
$$

$$
\beta_t = \begin{cases} 
M_{t+1} \beta_{t+1} & 0 \leq t \leq n \\
1 & t = n + 1.
\end{cases}
$$

Thus, we get:

$$
Z(o) = 1 \cdot \alpha_{n+1} = \sum_s \alpha(s, n + 1)
$$

We also define a matrix for features as $f_t(s', s) = f_t(s', s, o, t)$ and the second part of Equation 5.17 is calculated as

$$
\sum_{t=1}^{n} \sum_{v, w} p(s_{t-1} = v, s_t = w|o) f_t(v, w, o, t) = \sum_{t=1}^{n} \frac{\alpha_t(f_t \ast M_t)\beta_t}{Z(o)}
$$

We use smoothing to penalize the likelihood over parameters, thus the derivative in Equation 5.17 is changed into

$$
\frac{\partial L_\lambda}{\partial \lambda_j} = \sum_{k=1}^{N} \left[ \sum_{t=1}^{n} f_j(s^k, o^k, t) - \sum_{s' \in |S|^n} p(s'|o^k) \sum_{t=1}^{n} f_j(s', o^k, t) \right] - \sum_{i=1}^{m} \frac{\lambda_i}{\sigma^2}
$$

Therefore the parameter estimation is to find the best model parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ using L-BFGS method [55, 28] (Section 4.2.2) with the first-derivative of the objective function, Equation 5.21.

### 5.4 Efficient Inference

Following the discussion in Section 4.3.1 about efficient inference in MEMMs, the marginal mode and the Viterbi algorithm for CRFs can be derived accordingly.
5.4.1 Marginal Mode

Back to our system notations, the hidden states $s$ are defined as $T_j \in \{T_0, T_1, \ldots, T_{k-1}\}$. The forward probability values $\alpha(T_j, t)$ denotes the probability of being in state $T_j$ at time $t$ given the observed page sequence up to time $t$. Thus, $\alpha(T_j, t)$ in CRFs is calculated as:

$$
\alpha(T_j, t) = \sum_{j' = 0}^{k-1} \alpha(T_{j'}, t - 1) \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o)) \tag{5.22}
$$

Unlike in MEMMs, forward values are not calculated as $p(s_t|o_{1..t})$, since CRFs use global normalization $Z(o)$ to determine the conditional probability. $\alpha(T_j, t)$ is unnormalized probability. Therefore, the probability of state $s_t = T_j$ given the observation sequence $o$ so far, $p(T_j|o_{1..t})$ is

$$
p(T_j|o_{1..t}) = \frac{\alpha(T_j, t)}{Z(o)} = \frac{\alpha(T_j, t)}{\sum_{j=0}^{k-1} \alpha(T_j, t)} \tag{5.23}
$$

The global normalization factor $Z(o)$ is defined when the recursion terminates at time step $t$, that is, $Z(o) = \sum_{j=0}^{k-1} \alpha(T_j, t)$.

5.4.2 The Viterbi Algorithm

Recall that the Viterbi algorithm is to find the best choice of hidden state assignments for a sequence given the model parameters. In CRFs, $\delta(T_j, t)$ maintains the unnormalized probability of the best labeling ending at time $t$ with the state $T_j$.

$$
\delta(T_j, t) = \max_{j' = 0}^{k-1} \delta(T_{j'}, t - 1) \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o)) \tag{5.24}
$$

Similarly, the normalized probability of the best labeling is given by $\frac{\delta(T_j, t)}{Z(o)}$.

5.5 Focused Crawling with CRFs

The system structure and algorithm for focused crawling with CRFs are the same as that in MEMMs, only with CRFs parameters and inference forward values. The flow chart of focused crawling with CRFs is in Fig. 2.11 in Chapter 2.
We now analyze the computational complexity of the crawling algorithm with CRF. For training, CRF training requires $O(I \times N \times T \times F^2)$ time, where $I$ is the number of iteration, $N$ is the size of training set, $T$ is the labeled training sequence length, and $F$ is the number of features. The complexity of CRF crawling algorithm is $O(L + Q \log Q + F^2)$, where $L$ is the average number of outlinks per page, $Q$ is the size of URL queue, and $F$ is the number of features. We use a hash table to implement the operations of inserting and removing URLs from the queue and checking if a newly extracted URL is already in the queue, yielding $O(L + Q \log Q)$ complexity. The CRF inference takes $O(F^2)$ operations, since we must do a matrix-vector multiply for each page. Therefore, for CRF crawling, it takes $O(L + Q \log Q + F^2)$ in the general case.

5.6 Experiments

In this section, we now present the experimental results of our CRF-based focused crawling approach. We used the exactly same topics and target pages as in MEMMs in order to compare their performance described in Chapter 6. The 10 topics are:

/Computers/Software/Operating_Systems/Linux/
/Home/Gardening/Gardens/Wildlife/Butterfly/
/Arts/Performing_Arts/Dance/Ballet/
/Sports/Cycling/Mountain_Biking/Downhill/
/Business/Management/Management_Information_Systems/Call_For_Papers/
/Society/Religion_and_Spirituality/Yoga/
/Health/Nutrition/Disease_Prevention/Heart_Disease/
/Sports/Hockey/Ice_Hockey/Training/
/Science/Astronomy/Amateur/Sky_Maps_and_Atlases/
/Society/Law/Legal_Information/Computer_and_Technology_Law/Internet/

Detailed information about each topic such as target pages, start URLs and the number of training sequences can be found in Appendix A.

5.6.1 Evaluation

We continue to use two methods to evaluate the results. The first one is the number of pages crawled that are relevant. The second one is $Maximum\ Average\ Similarity$
$\sigma$ as discussed in 2.4.

A baseline method Best-First Search (BFS) crawler is used for comparison. BFS crawler assigns priorities to all children of the current page using standard lexical cosine similarity between the set of target pages and the content of the current page. Since we have multiple targets, the visit priority order of a URL here is based on maximal cosine similarity, and the priority queue is sorted by this value.

5.6.2 Results

We have conducted two experiments. One is to compare CRF-based methods with different inference algorithms against BFS crawl. Another one is to test the impact of the multiple features on the performance.

Comparison with Different Inference Algorithms: Viterbi Algorithm and Marginal Mode

This section shows all the results with marginal mode and the Viterbi algorithm for the inference, as discussed in Section 5.4. In the figures, we use $BFS$, $crf$-marginal, $crf$-viterbi to denote Best-First Search crawler, CRF crawl with marginal mode inference and CRF crawl with Viterbi algorithm, respectively. In the text below, we use $BFS$, $CRF$-marginal and $CRF$-viterbi just for clarity.

First we compare CRF-marginal and CRF-viterbi crawls against BFS method. Fig. 5.4 to Fig. 5.13 show some of the results. From the results, we can see that BFS crawl is the worst among the three methods as expected, while CRF-marginal crawl is the best in terms of both evaluation metrics. By comparing CRF-marginal crawl with BFS crawl, we can see that CRF-marginal crawl outperforms BFS crawl on 9 topics, except topic Skymaps as shown in Fig. 5.12, where CRF-marginal crawl has a very close performance to BFS crawl on both the number of relevant pages and the Maximum Average Similarity metric. By comparing CRF-viterbi crawl with BFS crawl, we can see that CRF-viterbi crawl shows better performance on 8 topics than BFS crawl, except topic Skymaps (Fig. 5.12) and topic Balletdance (Fig. 5.13). We can also find that CRF-marginal crawl outperforms CRF-viterbi crawl in terms of both metrics.
One of the possible reasons for the improvement of CRF-marginal crawl and CRF-viterbi crawl over BFS crawl owes to the fact that CRF-marginal and CRF-viterbi use more features for prediction. By incorporating more features, CRF-based methods can fully employ the rich features about the observed pages and produce better prediction in many situations, instead of depending only upon the relevance of the page as BFS crawl does. Each URL in a page is treated individually based on the multiple features, rather than all URLs are treated equally in BFS crawl. Furthermore, CRF-based method is able to handle the inter-page relationship along the page sequence in a better way than BFS method which is a simple classification problem on an individual page. Taking the interactions along the path into consideration is another reason why CRF-based methods outperform BFS crawl in most of these cases. From Fig. 5.4 to Fig. 5.11, we can see the improvement over BFS crawl in terms of either metric on different topics.

We now compare the results only based on CRF-marginal and CRF-viterbi crawls. We see that CRF-marginal outperforms CRF-viterbi crawl, except the topic Biking shown in Fig. 5.10, and the topic Butterfly with threshold 0.7 shown in Fig. 5.11. The goal of comparing CRF-marginal and CRF-viterbi inference is to study when we want to find the most likely hidden state sequence versus finding the distribution for each individual hidden state. This depends on the different applications. Many applications of CRFs have been shown good performance when applying the Viterbi algorithm to many problems in natural language processing [49, 69], feature induction for NER [48], and bioinformatics [70, 45]. These applications are trying to make predictions based on having the most likely sequence labeling. Thus the Viterbi algorithm makes more sense because the single most likely sequence of hidden states could differ greatly from the sum of a number of possible sequences with a different hidden state, as calculated in marginal probability.

For our focused crawling problem, knowing what the probability of the hidden state a page is in at a particular time makes more sense; therefore finding marginal probability gives better prediction than using the Viterbi algorithm. The experimental results are consistent with the results of MEMMs using marginal mode and Viterbi inference discussed in the last chapter (Section 4.4), in which MEMM with marginal mode also performs better than MEMM with Viterbi algorithm. Therefore, we can
Figure 5.4: Topic *Fitnessyoga*: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

conclude that using marginal mode for inference provides a better way for our Web page focused crawling problem.

**Comparison with Different Features**

In this section, we continue to test the impact of the multiple features on the performance. The previous section shows that CRF-marginal crawl performs better than CRF-viterbi crawl, so in this experiment, we choose to compare CRF-marginal crawl with different features: with all features, with Word feature only, and with the remaining features: Text feature, Description feature, URL token feature and Anchor text feature (See all features in Section 2.2.2). We denote them as *crf-marginal*, *crf-marginal-word*, and *crf-marginal-sim-meta-T* in the figures respectively.

As figures Fig. 5.14 to Fig. 5.18 show, CRF-marginal crawl with all features combination performs constantly better than with Word feature only and with sim-meta-T features on almost all topics. This also further confirms our approach that with multiple features representation, even if only some of them are present, the relevant path can be effectively identified.

One exception is topic *Butterfly* (Fig. 5.19), where CRF-marginal crawl with all
Figure 5.5: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. (a) Topic Linux. (b) Topic Fitnessyoga.

Figure 5.6: Topic Internetlaw: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.
Figure 5.7: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. (a) Topic **Internetlaw**. (b) Topic **Hockey**.

Figure 5.8: Topic **Callforpapers**: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.
Figure 5.9: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the Maximum Average Similarity. (a) Topic Balletdance. (b) Topic Callforpapers.

Figure 5.10: Topic Biking: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.
Figure 5.11: Topic *Butterfly*: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

Figure 5.12: Topic *Skymaps*: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.
Figure 5.13: Topic *Balletdance*: Comparisons of different methods: BFS, CRF-marginal and CRF-viterbi on the number of relevant pages in the set of downloaded pages with different thresholds.

features performs worse than that with Word feature only and with sim-meta-T features overall. In topic *Butterfly*, crawl with sim-meta-T features shows better performance than that with the complex features on thresholds 0.5, 0.6, and 0.7. Word feature gives the best performance with higher threshold 0.8 compared to the other two. This also indicates that choosing useful features is very important to the performance. Such an example is also shown in Fig. 5.15(b), where sim-meta-T features gives a very good performance on topic *Fitnessyoga*. Exploiting more features which are not included in this thesis will be described in the future work in Chapter 7.

5.7 Summary

This chapter described the use of linear-chain Conditional Random Fields (CRFs) for the focused crawling problem. Like MEMMs, the conditional nature of CRFs provides us the flexibility to include multiple overlapping features. Unlike MEMMs, CRFs define a single conditional probability using a global normalization factor. In this case, linear-chain CRFs can be roughly considered as conditionally trained HMMs, with additional flexibility to integrate overlapping features, but without having to worry about the independence assumptions in HMMs. Following the system framework described in Chapter 2, we presented detailed algorithms for training and crawling.
Figure 5.14: Topic Internetlaw: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 5.15: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages. (a) Topic Linux. (b) Topic Fitnessyoga.
Figure 5.16: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages. (a) Topic Balletdance (b) Topic Hockey.

Figure 5.17: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages. (a) Topic Biking. (b) Topic Skymaps.
Figure 5.18: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages. (a) Topic *Hearthealthy*. (b) Topic *Callforpapers*.

Figure 5.19: Topic *Butterfly*: Comparisons of different methods: with Word feature only, sim-meta-T feature only and the all features combination on the number of relevant pages within the set of downloaded pages with different thresholds.
Experimental results showed that CRF-based crawl performs better than BFS crawl, and also confirmed our findings in MEMMs that using marginal mode is better than using Viterbi algorithm, and the crawler using the combination of all features performs constantly better than the crawler that depends on just one or some of them.
Chapter 6

Comparison of Different Methods

We have applied MEMMs in Chapter 4 and linear-chain CRFs in Chapter 5 for the focused crawling problem. Although they are different models - MEMMs are directed discriminative models, whereas CRFs are undirected graphical models - they have many properties in common. Both of them assume the first-order Markov assumption, i.e. $p(s_{t+1} | s_t) = p(s_{t+1} | s_t, s_{t-1})$, and calculate the conditional probability $p(s | o)$ directly. Both of them use the same training data and incorporate the same multiple features. MEMMs can be seen as a localized version of CRFs since the important difference between the two is that MEMMs use a pre-state local normalizer while CRFs assume a global normalizer over the whole sequence. Therefore, it is fair to compare these two models on the focused crawling problem. In the following sections, we also provide an estimation of computational costs and summarize our discussion on the advantages and disadvantages of each method.

6.1 Comparison of the Performance with MEMM and CRF

Fig. 6.1 to Fig. 6.7 show the performance comparison results on all topics. Since both CRFs and MEMMs showed better performance with the marginal mode in previous chapters, we compare CRF-marginal and MEMM-marginal against BFS crawl in the figures. As the figures show, CRF-marginal crawl outperforms MEMM-marginal crawl on seven topics Biking, Hearthealthy, Hockey, Fitnessyoga, Skymaps, Calloforpapers and Internetlaw in terms of both evaluation metrics, and MEMM-marginal crawl shows better results on three topics Linux, Butterfly, Balletdance.

Generally speaking, the performance with CRFs is better than with MEMMs on these topics. It exhibits slight improvement for predicting the next important links to follow. Global normalization seems to help since this is the only difference between MEMMs and linear-chain CRFs.
Figure 6.1: Topic *Linux* and Topic *Butterfly*: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal on the number of relevant pages in the set of downloaded pages.

Figure 6.2: Topic *Balletdance*: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages in the set of downloaded pages. (b) Maximum Average Similarity.
Figure 6.3: Topic *Biking* and Topic *Hockey*: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal on the number of relevant pages in the set of downloaded pages.

(a) Topic *Biking*  
(b) Topic *Hockey*

Figure 6.4: Topic *Fitnessyoga*: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity.
Figure 6.5: Topic Hearthealthy and Topic Internetlaw: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal on the number of relevant pages within the set of downloaded pages with different thresholds.

Figure 6.6: Topic Callforpapers: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity.
Figure 6.7: Topic Skymaps: Comparisons of different methods: BFS, MEMM-marginal and CRF-marginal. (a) The number of relevant pages within the set of downloaded pages. (b) Maximum Average Similarity.

6.2 Comparison of Computational Costs with BFS, MEMM, and CRF

Perhaps the most serious problem with our approach is the computational costs when we apply graphical models for focused crawling. For a supervised learning problem, there are two main steps: learning/training and prediction/inference. Compared to the BFS crawl, which has no extra costs on training and low computational costs on prediction during crawling, graphical models-based methods need higher computational costs on both training and crawling.

During the training, since MEMM calculates the normalization factor over each position (Section 4.1.3), in theory it reduces dramatically the computational costs compared to CRFs, which need to compute a global normalizer (Section 5.3). However, as shown in Section 5.3, the Equation 5.17 can be implemented efficiently using dynamic programming from Equation 5.19 to Equation 5.21; therefore the computational cost differences between MEMMs and CRFs are minor. The left side of Table 6.1 shows the computational costs on each topic for MEMMs and CRFs. In our experiments using L-BFGS for training, described in Section 4.2.2, the training stops either when the number of iterations reaches 100, which is the maximum number of iterations over the training data, or when it converges with the convergence criteria for finding optimum solution $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$, 0.001.
Table 6.1: Computational Costs

<table>
<thead>
<tr>
<th>Topics</th>
<th>Training Time (s)</th>
<th>Crawling Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BFS</td>
<td>MEMMs</td>
</tr>
<tr>
<td>Linux</td>
<td>-</td>
<td>917</td>
</tr>
<tr>
<td>Butterfly</td>
<td>-</td>
<td>1021</td>
</tr>
<tr>
<td>Biking</td>
<td>-</td>
<td>962</td>
</tr>
<tr>
<td>Hearthealthy</td>
<td>-</td>
<td>1167</td>
</tr>
<tr>
<td>Hockey</td>
<td>-</td>
<td>288</td>
</tr>
<tr>
<td>Fitnessyoga</td>
<td>-</td>
<td>807</td>
</tr>
<tr>
<td>Balletdance</td>
<td>-</td>
<td>1021</td>
</tr>
<tr>
<td>Callforpapers</td>
<td>-</td>
<td>727</td>
</tr>
<tr>
<td>Skymaps</td>
<td>-</td>
<td>726</td>
</tr>
<tr>
<td>Internetlaw</td>
<td>-</td>
<td>722</td>
</tr>
</tbody>
</table>

During crawling, all methods follow the flow chart shown in Fig. 2.11. The two pink rectangles (marked Prediction and Analyze Page) are the only differences among the different methods. BFS crawl has fewer computational costs since it only involves text preprocessing, extracting links, calculating similarity value between the targets and the newly-seen page, and assigning visit priority based on the sorted similarity values of URLs in the queue. MEMMs and CRFs crawls, on the other hand, require additional feature extraction (Section 2.2.2) and MEMMs/CRFs inference based on parameters learned from training data (Section 4.3.1/Section 5.4). In our implementation, we use the queue size equal to 500 and there are parts of the code that intentionally slow down the crawlers; They are not without purpose. The code respects the Robot Exclusion Protocol and spreads the crawling load over many remote Web servers. For example, the focused crawler follows the top URLs in the queue; if two consecutive URLs in the queue have the same server domain, the crawler will wait ten seconds to avoid constant and continuous burden on the same server.

The right side of Table 6.1 gives the summary of computational costs for different crawl strategies on all topics. In general, BFS crawl has less crawl time than MEMMs and CRFs crawls, and MEMMs and CRFs crawls have no large differences based on these topics. As the results show in the previous chapters, MEMMs and CRFs give better crawling performance than BFS, but at the expense of relatively higher...
computational costs.

6.3 Comparison of MEMM and CRF with HMM, BFS, and CGS

In Chapter 3, we described the focused crawler system using HMMs to learn from user's browsing patterns, and compared the performance with BFS and CGS crawls (see Section 3.4.2). As described in Section 2.1.1, training data is created from user-visited pages, and target pages are user-marked pages during the browsing. HMMs showed better results than CGS crawl and BFS crawl, which encouraged us to move forward, that is, to exploit multiple features with MEMMs and CRFs to find relevant pages in a broader extent. To have sufficient training data to learn from multiple features with MEMMs and CRFs, we chose to employ Web Data Collection, and target pages on the topics are chosen directly from the Open Directory Project (ODP), as described in Section 2.1.2. As a supervised learning problem, the effect of the training data is very important. We have shown in Section 3.4.3 of Chapter 3 that HMM-based crawl designed for learning from user's data gives bad performance when applied on the Web graph constructed using all nodes. Therefore, we did not apply HMMs on Web Data Collection.

The original intention was to compare the performance of different methods for focused crawling. However, HMM-based crawl was applied under different assumptions and on different data sets, therefore, there is no way to compare them directly. Instead, the methods were compared using Average Precision Relative to BFS (APR) measure, which indirectly compared the performance of CGS, HMMs, MEMMs, and CRFs against the baseline method, BFS. Average Precision Relative to BFS (APR) is defined as follows:

\[
R = \frac{\text{Average precision using the method to be evaluated}}{\text{Average precision using the baseline method, BFS}}
\]  

That is, we compare the relative performance of different methods over BFS crawl: the larger the APR, the better performance the method has. We first calculate the precision relative to BFS for each threshold 0.5-0.8, then average them together. Fig. 6.8 shows the four topics we used in all methods. CRFs and MEMMs crawls
show better APR than HMMs and CGS crawls on topic *Biking* (Fig. 6.8(a)) and Topic *Callforpapers* (Fig. 6.8(b)), especially with the increase in the number of downloaded pages. In Topic *Hockey* (Fig. 6.8(c)), CRFs and MEMMs give mixed APR values compared to HMMs: MEMMs gives better performance than HMMs in the beginning of the crawl, while CRFs shows dramatic improvement during the later stages of the crawl. Topic *Linux* is a very different topic in our experiments, as shown in figure (Fig. 6.8(d)), HMMs gives the best APR value over other methods, and CRFs shows the worst performance. This is because during the crawl using HMMs and BFS in the Chapter 3, the BFS crawl gives extremely poor performance on the number of the relevant pages, making the APR ratio gap of HMMs and CGS much bigger.

### 6.4 Discussion

We modeled the focused crawling problem as a sequential task and have applied HMMs, MEMMs and CRFs models. The following table briefly summarizes the advantages and disadvantages of three graphical models.

<table>
<thead>
<tr>
<th>Models</th>
<th>First-order Markov</th>
<th>Label Bias</th>
<th>Flexibility of Features</th>
<th>Global Solutions</th>
<th>Computational Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMs</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>MEMMs</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CRFs</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

Based on the assumptions and definition of HMMs discussed in Chapter 3, this model lacks the flexibility to use arbitrary features and the ability to obtain global solutions. Therefore, we apply HMMs to learn from the user’s browsing data, since the user-visited data contains strong sequence structures about the topic when the user seeks to fulfill his/her topic-specific information need. MEMMs are more expressive models, they make fewer assumptions about the data than HMMs, and let the data speak for itself, that is, they allow arbitrary and overlapping features. However, MEMMs use local normalization factor at each position, so that they actually form a chain of local models, that is, each position contains a “local classifier”. As a result, MEMMs may suffer the label bias problem [39], i.e. the total probability
Figure 6.8: Average Precision Relative to BFS (APR) by CRFs, MEMMs, HMMs, CGS crawling methods. Precision relative to BFS is calculated by Precision using the method to be evaluated over Precision using BFS.
received by $s_{t-1}$ must be passed on to the state labels at next position $t$ even if $o_t$ is completely incompatible with $s_{t-1}$, since the distribution sums to 1 across all next state $s_t$ at position $t$. Therefore, MEMMs also give local optimal solutions rather than global solutions. Linear-chain CRFs, on the other hand, have all the advantages of MEMMs without the label bias problems. They use a global normalizer which results in a global solution and avoids the label bias problem. Like MEMMs, CRFs use multiple features from the data to solve the problem directly, rather than using an intermediate step, as in HMMs. Experimental results show that CRFs are able to outperform MEMMs and HMMs with slight improvement overall, and the properties of CRFs make it possible to capture the topical relations better along the hyperlinks for the focused Web crawling.

From Chapter 3 to Chapter 4, we have described some experimental results using different models: HMMs, MEMMs and CRFs, and compared them with BFS crawl. The main purpose of these experiments is to crawl pages on different topics and study the three graphical models for focused web crawling. Within these results (see Figures 6.3 and 6.5), we can clearly see some flat lines or jumps on the number of relevant pages. This reflects the normal behaviours when the crawler crawls on the real Web, due to the nature of the Web. Pages appear clustered in the Web graph, which indicates that Web pages on a given topic are more likely to link to those on the same topic. At the same time, it can be a long way from one relevant page to the next. The flat lines or jumps could happen when there are no good out-links from current page or when the crawler enters a large topical cluster where topical coherence persists, especially for a single-threaded crawler.

Although in our experiments the topics and target pages are specified by the user or extracted from DMOZ, it is easy to use more flexible ways and apply the method to a new topic. For example, bookmarks usually are the favorite Web page URLs that a user has visited or will visit, which can be used directly as the target pages on the topic if they are well organized within a folder, or can be further clustered into different topics before applying the underlying models to train and crawl.
Chapter 7

Conclusion

This chapter summarizes the thesis contribution and discusses potential future work directions.

7.1 Contribution

The goal of this research has been to design a robust method for the focused crawling problem, capable of collecting as many pages as possible that are relevant to the given topics. This is not an easy task, since focused crawlers can only use information gleaned from previously crawled pages. To address this challenge, we proposed a new approach for focused crawling to capture sequential patterns along paths leading to targets based on probabilistic models. To the best of our knowledge, our approach is the first work to apply graphical probabilistic models on focused web crawling.

The major contributions of this thesis work are summarized as follows:

1. We have proposed a novel approach for focused Web crawling to search for relevant pages on a specific topic from the Web under a unified framework, where the focused crawling problem is modeled as a sequential task. Our approach is to use a combination of content analysis and link structure of paths leading to targets to learn from training data and emulate them to find more relevant pages in the rapid growth Web graph. The system is unique in that it models the process of crawling by a walk along an underlying chain of hidden states, defined by hop distance from target pages, from which the actual topics of the documents are observed. When a new document is seen, prediction amounts to estimate the distance of this document from a target. In this way, good performance depends on powerful modeling of context as well as current observations.

2. Three probabilistic models have been explored: HMMs, MEMMs, and linear-chain CRFs. For each of the models within this framework, we developed an
inference and a learning algorithm accordingly. With Hidden Markov Models (HMMs), we focused on semantic content analysis with sequential patterns learned from the user’s browsing behavior on specific topics. HMM crawler trained from User Data has been shown to be an effective crawler over Context-Graph crawler and Best-First crawler. It also showed that HMM crawler trained using User Data performs better than using all the links between the nodes in the graph.

3. We further extended our work to take advantage of richer representations of multiple features extracted from Web pages. The advantages and flexibility of MEMMs and CRFs fit our approach well and are able to represent useful context. With Maximum Entropy Markov Models (MEMMs), we exploit multiple overlapping features, such as anchor text, to represent useful context and form a chain of local classifier models. With Linear-chain Conditional Random Fields (CRFs), a form of undirected graphical models, we further focus on obtaining global optimal solutions along the sequences by taking advantage not only of text content, but also of linkage relations. The experimental results showed that focused crawling using MEMMs/CRFs is a very competitive crawler in general over Best-First crawling on Web Data.

4. In our exploration, we have demonstrated the effectiveness of the graphical probabilistic models for focused Web crawling problem. We have studied the impact of different combination strategies, and the results showed that using marginal mode performs better than using Viterbi algorithm, and the crawler using the combination of all features performs consistently better than the crawler that depends on just one or some of them. We also introduced Average Precision Relative to BFS (APR) measure to compare different models and found that the graphical models for focused crawling are consistently better than the standard BFS approach, and CRF crawler performs the best among the three graphical models.

5. We have analyzed the time complexity of crawlers and found that MEMMs and CRFs give better crawling performance than BFS, but at the expense of relatively higher computational costs. Although our discussion is focused on
modeling the focused crawling problem, the methodologies and experience are valuable and applicable to other applications. The framework is the first work to apply graphical probabilistic models on focused crawling. It has the potential to provide a useful application for more effective navigation and access to online information, and aid the studies on graph-based relational learning.

7.2 Future Work

In this thesis, we have applied three probabilistic models for focused crawling. The motivation is to capture statistical dependencies that exist between the entities and also integrate a rich set of features, since the pages' text provides much information and hyperlinks define linkage relationships between pages. MEMMs and linear-chain CRFs showed the promising success of the crawling performance in the experiments, but there are still many aspects that can be further investigated.

In our current work, we have extracted some features from each page along the sequences the crawler visits, and we think that these features represent the important observed information from the pages, since the focused crawler can only have information from its ancestors. These features are mostly extracted from the content of the pages directly. However, other attributes that reflect page link structure may also be very useful. We plan to exploit more complicated features which have already been proved useful for focused crawling in the literature, such as Backlink Counts and PageRank and Forward Link Count from [20], Hub score, Authorities score, link community score used in [38], and DOM tree feature [16]. It is very interesting to incorporate other features and investigate the impact on the performance.

On the other hand, CRFs provide much potential to further extend our work. The original CRFs only allow linear combinations of features. They can be extended using non-linear forms such as Kernel conditional random fields (kCRFs)[40]. In addition, currently we are using linear-chain CRFs, since we are modeling focused crawling as a sequential problem to predict the next important hyperlinks to follow. However, the Web graph is connected in a more complicated way, and focused crawling can not only use page information gleaned from its ancestors, but also from its neighbors or siblings, in which the dependencies can be a tree or graph structure. General CRFs can be defined over arbitrary undirected graphs, not just sequences. We are considering
exploring the possibilities of applying CRFs with more general graphical structures, such as factorial CRF or hierarchical CRF, to the focused crawling problem.

Another direction in which to continue is to extend our research to other domains. Besides the Web itself, many Web-like information spaces exist in the world, such as social networks, phone call graphs, and internet traffic graphs. Many graph-based relational learning problems could be solved using graphical models, since they allow both graphically representing dependencies and including multiple features. The important issue is how to define the problem and find the solution using the appropriate underlying model structure. For example, we can apply CRFs on the Behavioral Monitoring problem. Behavior of information access involves "WHO is doing WHAT to WHICH and HOW MUCH critical information, WHEN and from WHERE", and the Behavioral Monitoring problem can be modeled as a time-based sequential pattern analysis problem based on the important elements of the behavior profile of a user or program to determine if the behavior of an individual's information access is anomalous compared to his/her normal access behavior. Furthermore, our approach can be extended to looking for node clusters with particular connectivity patterns and related content based on link and content analysis, for example, detecting Web spammers which are often machine-generated with regular link structures and common page features.

Figure 7.1: Behavior Modeling Problem as a Time-based Sequential Analysis Problem
Bibliography


Appendix A

Topics

A.1 Topic Linux

- Target Pages:
  
  http://www.freeos.com/
  http://www.linux.org/info/
  http://www.kernel.org/
  http://www.linuxplanet.com/linuxplanet/tutorials/
  http://en.wikipedia.org/wiki/Linux
  http://linux.about.com/od/linux101/
  http://www.comptechdoc.org/os/linux/
  http://cbbrowne.com/info/linux.html
  http://www.linux.com/article.pl?sid=02/03/09/1727250
  http://www.qsl.net/kd2bd/linux.html
  http://www.topology.org/soft/linux.html
  http://whatis.techtarget.com/definition/0,289893,sid9_gci212482,00.html
  http://www.linuxhotbox.com/
  http://www.oracle.com/technology/tech/linux/htdocs/
  oracleonlinux_faq.html
  http://www.cs.huji.ac.il/~nomad/linux.html

- Start URLs:
  
  http://www.nonags.com/
  http://tldp.org/
http://www.computerhope.com
http://comptechdoc.org/os
http://www.informationweek.com/techcenters/sw/
http://about.com/compute

• Training Sequences: 11394

A.2 Topic Biking

• Target Pages:
  f0 : http://gardenersnet.com/birds/butterflies.htm
  f1 : http://www.wvu.edu/agesten/wildlife/butterfly.htm
  f2 : http://www.floridalplants.com/gard.butterfly.htm
  f3 : http://www.suite101.com/welcome.cfm/butterfly_gardening/
  f4 : http://forums.gardenweb.com/forums/butterfly/
  f5 : http://garden-gate.prairienet.org/bflyplnt.txt
  f6 : http://butterflygardeners.com/
  f7 : http://hgic.clemson.edu/factsheets/HGIC1701.htm
  f8 : http://mdc.mo.gov/nathis/insects/butterfly/
  f9 : http://www.butterflygardeningandconservation.com/
  f10 : http://www.thebutterflysite.com/gardening.shtml
  f12 : http://www.uky.edu/Ag/Entomology/entfacts/misc/ef006.htm
  f13 : http://www.geocities.com/marl_insect/
  f14 : http://www.milkweedcafe.com/bflygarden.html
  f15 : http://www.monarchwatch.org/garden/index.htm
  f16 : http://www.lepidopterology.com/directory/w_res_02.htm

• Start URLs:
  http://www.thebutterflysite.com/
  http://gardening-today.info/source/butterfly-garden-plant.html

• Training Sequences: 9790
A.3 Topic *Butterfly*

- Target Pages:
  
f0 : http://gardenersnet.com/birds/butterflies.htm
f1 : http://www.wvu.edu/agexten/wildlife/butterfly.htm
f2 : http://www.floridaplants.com/gard_butterfly.htm
f3 : http://www.suite101.com/welcome.cfm/butterfly_gardening/
f4 : http://forums.gardenweb.com/forums/butterfly/
f5 : http://garden-gate.prairienet.org/bflyplnt.txt
f6 : http://butterflygardeners.com/
f7 : http://hgc.clemson.edu/factsheets/HGIC1701.htm
f8 : http://mdc.mo.gov/nathis/insects/butterfly/
f9 : http://www.butterflygardeningandconservation.com/
f10 : http://www.thebutterflysite.com/gardening.shtml
f12 : http://www.uky.edu/Ag/Entomology/entfacts/misc/ef006.htm
f13 : http://www.geocities.com/marl_insect/
f14 : http://www.milkweedcafe.com/nightgarden.html
f15 : http://www.monarchwatch.org/garden/index.htm
f16 : http://www.lepidopterology.com/directory/w_res_02.htm

- Start URLs:
  
  http://www.aboutallsports.com/sports/
  http://cyclinglinks.tripod.com/

- Training Sequences: 12172

A.4 Topic *Heart Healthy*

- Target Pages:
  
f0 : http://www.healthy.net/scr/article.asp?ID=1917
f1 : http://www.healthyfridge.org/
f2 : http://www.howstuffworks.com/heart-diagnosis.htm
f3 : http://www.health-heat.org/
A.5 Topic **Hockey**

- Target Pages:
  - f0 : http://www.nhl.com/laceemup/howtoplay/index.html
  - f1 : http://en.wikipedia.org/wiki/Ice_hockey
  - f3 : http://www.hockeycanada.ca/index.cfm/ci_id/6698/la_id/1.htm
  - f4 : http://www.cbc.ca/olympics/sports/icehockey/
  - f5 : http://www.hockey-fans.com/teams.php
  - f6 : http://www.thecanadianencyclopedia.com/index.cfm?
    PgNmTCE&ParamsA1ARTA0003794
  - f7 : http://www.olympic.org/uk/sports/programme/index_uk.asp?
    SportCode=IH
  - f8 : http://www.birthplaceofhockey.com/evolution/evequipchron-1.html
  - f10 : http://www.safety-council.org/info/sport/hockey.htm
  - f11 : http://www.howstuffworks.com/ice-rink.htm
  - f12 : http://www.hockeyplayer.com/artman/publish/cat_index_2.shtml
  - f13 : http://www.sirlinksalot.net/icehockey.html
  - f14 : http://www.americanhockeycenter.com/nhl.htm
  - f16 : http://www.mcq.org/hockey/aaindex.html
  - f17 : http://www.hockeyology.com/videogames.php3
f18 : http://www.whockey.com/int/

- Start URLs:
  http://www.cbc.ca/sports/hockey/
  http://news.bbc.co.uk/sport/

- Training Sequences: 4368

A.6 Topic **Fitnessyoga**

- Target Pages:
  f0 : http://www.americanyogaassociation.org/general.html
  f1 : http://indiaexpress.com/mind/yoga/
  f2 : http://www.introductiontoyoga.com/
  f3 : http://www.yogamovement.com/resources/howyogacalms.html
  f4 : http://yoga.org.nz/benefits/benefits_personal.htm
  f5 : http://www.yogasite.com/postures.html
  f6 : http://www.holisticonline.com/Yoga/hol_yoga_pos_step.htm

- Start URLs:
  http://www.yogamotion.info/
  http://altmedicine.about.com/od/mindbodytechniques/
  MindBody_Techniques.htm
  http://exercise.about.com/

- Training Sequences: 11680

A.7 Topic **Balletdance**

- Target Pages:
  f0 : http://www.artofballet.com/
  f1 : http://www.balletinstructor.com/
  f2 : http://www.geocities.com/Vienna/Strasse/5503/home.html
  f3 : http://www.artofballet.com/lessons.html
  f4 : http://en.wikipedia.org/wiki/Ballet
f5 : http://www.ballet.co.uk/
f6 : http://www.dance4it.com/ballethistory.htm

- Start URLs:
  http://www.danzadance.org/
  http://www.misslisasdanceexpress.com/links.php
  http://www.danceenc.com/

- Training Sequences: 12317

A.8 Topic *Skymaps*

- Target Pages:
  f0 : http://www.astro.wisc.edu/dolan/constellations/
  f1 : http://www.projectrho.com/starmap.html
  f2 : http://domeofthesky.com/foyer.html
  f3 : http://www.theskytonight.com/localsky.htm
  f4 : http://astroclub.tau.ac.il/skymaps/monthly/
  f6 : http://www.lindahall.org/events.exhibit/exhibit/exhibits/stars/index.html
  f7 : http://skymaps.com/
  f8 : http://www.fourmilab.ch/solar/
  f9 : http://maps.jpl.nasa.gov/
  f10 : http://www.solstation.com/
  f11 : http://starmap.causeway.co.uk/
  f12 : http://virtualsky.org/
  f13 : http://server2.wikisky.org
  f14 : http://space.about.com/library/weekly/blskymaps.htm
  f15 : http://www.fourmilab.to/yoursky/

- Start URLs:
  http://www.phys-astro.sonoma.edu/people/faculty/tenn/
Educational.htm
http://jrscience.wcp.muohio.edu/html/Astronomy.html
http://sthigpen.freeshell.org/astronomy-links.html

- Training Sequences: 12073

A.9 Topic Callforpapers

- Target Pages:
  f0 : http://www.academic-conferences.org/eckm/eckm2007/eckm07-call-papers.htm
  f2 : http://www.icesacc.org/call.html
  f3 : http://i-know.know-center.tugraz.at/author_guidelines_and_dates
  f7 : http://www.em2i.org/index.files/Page531.html
  f8 : http://edoc.mitre.org/callpa.html
  f9 : http://www.iima.org/New
  f10 : http://business.queensu.ca/icis/callpapers.htm

- Start URLs:
  http://www.informatik.uni-trier.de/~ley/db/conf/index.a.html
  http://directory.google.com/Top/Computers/Computer_Science/Conferences/
  http://dir.yahoo.com/Science/Computer_Science/Conferences/

- Training Sequences: 9632

A.10 Topic Internetlaw

- Target Pages:
  f0 : http://www.aclu.org/privacy/speech/index.html
  f1 : http://cyber.law.harvard.edu/
f2: http://www.bsa.org/usa/antipiracy/
f3: http://www.chillingeffects.org/ecom/
f4: http://www.fplc.edu/tfield/copynet.htm
f5: http://www.angelfire.com/stars/tkchang/
f6: http://www.cybertelecom.org/security/privacy.htm
f7: http://www.cli.org/X0025_LBFIN.html
f8: http://www.eplaw.us/news/
f10: http://grep.law.harvard.edu/
f11: http://www.internetcases.com/
f13: http://legal.web.aol.com/ip/ipguide/licensin.html
f14: http://www.opennetinitiative.org/
f15: http://www.perkinscoie.com/casedigest/
f16: http://www.spamlaws.com/federal/108s877.shtml
f17: http://www.netatty.com/privacy/index.html

• Start URLs:
  http://www.legalsearchmarketing.com/
  http://www.blawgcast.com/
  http://www.legal-forms-now.com/
  http://www.alllaw.com/topics/comp_and_tech/

• Training Sequences: 12210