TOWARDS AUTOMATIC GENERATION OF REALISTIC WEB QUERY SEQUENCES

Valdís Sigurþórsdóttir
Master of Science
Computer Science
June 2011
School of Computer Science
Reykjavík University

M.Sc. RESEARCH THESIS

ISSN 1670-8539

Towards Automatic Generation of Realistic Web Query Sequences

by

Valdís Sigurþórsdóttir

Research thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

June 2011

Research Thesis Committee:

Björn Þór Jónsson, Supervisor
Associate Professor, Reykjavík University, Iceland

Hrafn Loftsson
Assistant Professor, Reykjavík University, Iceland

Yngvi Björnsson
Associate Professor, Reykjavík University, Iceland

Towards Automatic Generation of Realistic Web Query Sequences

Valdís Sigurþórsdóttir

June 2011

Abstract

The rapid growth of the web, and the increased use of web search engines, places heavy demands on the efficiency of the underlying information retrieval algorithms. Studying long-term efficiency issues, such as buffer management policies is thus important. The major search engines have access to extensive user traces but only short traces have been made publically available. Performance studies require long user traces for efficient testing, leading to growing demand for easier accessability for the research field. We therefore propose a methodology for a) learning user behavior patterns from the available short traces and b) generating extensive query sequences based on those behavior patterns in order to simulate actual search engine users. In a detailed performance study, we show that our methodology simulates the original trace quite well, both in terms of user behavior patterns and in terms of basic search engine performance metrics.

Drög að sjálfvirkri gerð raunsærra veffyrirspurna

Valdís Sigurþórsdóttir

Júní 2011

Útdráttur

Ör vöxtur vefsins, og aukin notkun á vefleitarvélum, leggur miklar kvaðir á skilvirkni í undirliggjandi tækni. Rannsóknir á aðferðum til að tryggja skilvirkni yfir lengri tíma, svo sem biðminnisstjórnun, krefst afar langra fyrirspurnaruna. Fyrirspurnir myndaðar af handahófi skila ekki raunhæfum fyrirspurnum. Starfsmenn helstu leitarvélafyrirtækja geta fengið aðgang að eigin gögnum en einungis stuttar fyrirspurnarunur eru aðgengilegar vísindasamfélaginu. Við leggjum því til aðferðafræði til að a) læra einfalt líkan að hegðunarmynstri notenda út frá stuttum fyrirspurnarunum og b) búa til langar fyrirspurnarunur sem nota þetta hegðunarmynstur til að líkja eftir raunverulegum notendum leitarvéla. Í rannsóknum okkar sýnum við fram á að aðferðafræði okkar líkir allvel eftir upphaflegu fyrirspurnarununni, bæði þegar kemur að hegðunarmynstri og með tilliti til skilvirkni mælinga fyrir leitarvélar.

To my daugther Katla, the sunshine of my life.

Acknowledgements

I want to thank my supervisor Björn Þór Jónsson for ideas, input and patience. I would also like to thank the committee members, Yngvi Björnsson and Hrafn Loftsson, for their invaluable comments which helped to improve the presentation of my work. Valgerður Steinþórsdóttir was invaluable for her encouragement and reviews despite poor health. Thanks to my co-workers, especially Sigurbirna Hafliðadóttir for review and motivation, Kjartan R. Guðmundsson for Python and Sqlite input and Guðmundur Örn Ingvarsson and Sveinn G. Gunnarsson for Unix help. Thanks to Landbankinn for use of hardware. I also want to thank my family and friends, Mom, Dad, Kristján, Aneta, Valgerður, Jóna, Steinunn, Anna Guðný, Anna María, Sigga, Stella and Steina... to name some names; thanks I could not have done this without your help.

Publications

The model and software described in this thesis rest on foundations developed by students at Reykjavík University, as described in Chapter 3. The bulk of the material of the thesis has been submitted for international publication in a manuscript co-authored by Björn Þór Jónsson (Reykjavik University). While Björn contributed to the writing of the submitted document, the main contributions of the thesis, as described in Chapter 1, are entirely my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of the web, and the corresponding increased use of web search engines, places heavy demands on the efficiency of the underlying information retrieval algorithms. Studying long-term efficiency issues, such as buffer management, CPU cache utilization, and distributed processing, is thus important. Doing so, however, requires extremely long query traces in order to reach a processing equilibrium and get meaningful results. In addition, because these search engines are geared towards a general user population, those query traces must be "realistic", i.e., representative of actual usage.

1.1 Query Traces

Query traces are log files of queries and interactions that users have submitted from web search engines or other type of text search systems. Query traces usually consist of information such as identification (typically anonymized so that it cannot be traced back to an actual user), the search query, and the time of submission. Other information may vary depending on the type of system. Obtaining realistic query traces is not trivial. Researchers working for the large search engine companies have the advantage of access to the actual traces from their respective search engines. Due to security reasons, however, only short traces with limited information have been made publically available and can be used by the general academic. These short traces can, of course, be repeated, but such cyclic query traces will not properly exercise some important aspects of query processing, such as buffer management policies. Furthermore, using query traces from a short period may result in homogeneous queries from fairly few users, due to the temporary attention to a particular topic that was in the news at the time.

The obvious alternative is random query generation. Since the query distribution is skewed, however, there are extremely many terms (e.g., rare names and misspellings) which occur only in a few documents, leading to abnormally low buffer utilization and short query execution times. A known method of compensation is to reduce the scope of the random term selection (Tomasic & Garcia-Molina, 1993), but it is difficult to tune the choices of the lower and upper bounds. It is unlikely that randomly generated queries can exercise the query processing engine in a similar manner as actual usage.

There is thus a dire need for a methodology to generate extremely long sequences of realistic queries, in particular for the general academic that does not have access to the long search engine traces. The generated query sequences need not necessarily be realistic in the sense that all the queries would make sense for actual users. Instead, it is more important that the overall characteristics of the queries exercise the query processing engine in a realistic manner, thus resulting in meaningful performance results. In this thesis we propose a methodology for the generation of such query sequences.

1.2 Contributions

In this thesis, we make the following three major contributions:

- First, we describe a methodology for a) analyzing user behavior patterns of the available short search engine traces, and b) using those behavior patterns to generate extensive query sequences in order to simulate actual search engine users. Note that this query generation methodology was originally described informally by E. Tryggvason (2002), but has been formalized as a Markov chain in this thesis.
- Second, we propose three different methods to select the actual terms of the generated queries. The terms are chosen with an aim towards delivering the same query characteristics as in the original traces and towards requiring a similar effort from a search engine.
- Third, in a detailed performance study, we show that our methodology is able to simulate the original trace quite well, both in terms of user behavior patterns and in terms of basic search engine performance metrics.

The methodology does not currently capture all aspects of human behavior. More work is required, for example, to accurately emphasize the post-processing effort of various search engines, such as page ranking, and to simulate the temporal drift of attention between hot topics of the day. We believe, however, that our query generation methodology

can serve as a solid foundation for query generation for large-scale query processing studies. Researchers at the large search engine companies could, for example, tune a query generation engine to match the characteristics of their users and open the generation software to the general academic, rather than the actual query traces.

1.3 Overview of the Thesis

In Chapter 2 we review basic concepts of Information Retrieval and in Chapter 3 we describe some related work. In Chapter 4 we describe the user behavior model proposed in this thesis and in Chapter 5 we describe the proposed term selection policies. In Chapter 6 we evaluate the query generation methodology. We conclude the thesis and propose some future work in Chapter 7.

Chapter 2

Information Retrieval Background

The concept of information retrieval (IR) dates back to the 1940s. The science of IR consists of searching document collections for documents and information within documents. IR also concerns metadata of documents and searching relational databases. IR touches on many areas of science such as computer science, mathematics, statistics and library science among other areas. The first IR systems, however, were introduced in the 1950s and 1960s (Singhal, 2001; Luhn, 1957). Web search engines are the most visible IR applications. When the web search engines became common, in the 1990s, the need for large-scale web information retrieval became obvious. Today, Google is the most popular search engine, but there are many competitors such as Yahoo! and Bing (Comscore.com, 2011).

This chapter reviews some information retrieval (IR) topics that are necessary for the understanding of our proposed query generation methodology. We review how traditional IR systems operate (e.g., see Baeza-Yates and Ribeiro-Neto (1999) for a detailed survey) and briefly discuss some additional processing performed by many of the web search engines.

2.1 Queries and Query Refinement

Information retrieval systems typically use *natural language* techniques (also known as *vector space model*). A natural language query consists of a list of terms (words, implicitly connected by the $\vee$ operator); any document which contains one or more of the

terms is perceived to be relevant.[1] Documents are *ranked* by perceived relevance to the user query (see below). Additionally, most systems restrict the answer to the few most relevant documents.

Query refinement is an important search behavior in IR systems (Fidel, 1991; Koenemann, Quatrain, Cool, & Belkin, 1994; Jansen et al., 2000). When a ranked list of documents does not match what the user had in mind, the user refines the query by adding or removing terms, and resubmits it. This may occur repeatedly, until the user is satisfied with the returned results.

2.2 Ranking of Documents

Many systems accomplish the ranking of documents using *cosine similarity* (or variants thereof). Using the cosine similarity measure, the perceived relevance of document d to query q is:

$$\textit{relevance}_{q,d} = \frac{\sum_t w_{d,t} \cdot w_{q,t}}{W_d}, \tag{2.1}$$

where $w_{d,t}$ is the "weight" of term t in d, $w_{q,t}$ is the weight of t in q, and W_d is the "vector length" of document d:

$$W_d = \sqrt{\sum_t w_{d,t}^2} \tag{2.2}$$

The product $w_{d,t} \cdot w_{q,t}$ is called the *partial similarity* of document d due to term t. The weight of t in d is defined by:

$$w_{d,t} = f_{d,t} \cdot idf_t \tag{2.3}$$

where $f_{d,t}$ is the number of occurrences of t in d and idf_t is the *inverse document frequency* of the term t. An analogous formula applies to $w_{q,t}$, although $f_{q,t}$ is typically 1. The inverse document frequency is defined as:

$$idf_t = log_2(N/f_t) \tag{2.4}$$

where N is the number of documents in the collection, and f_t is the number of documents in which term t appears at least once. The relevance of a document to a query is zero if the document has no relation to the query and would be eliminated from the list of result documents when using a search engine. High value of the relevance, indicates that the document and the query have more terms in common and document would appear high

[1] In many systems, additional operators, such as proximity operators, which restrict the location of terms in the documents, are provided. According to (Jansen, Spink, & Saracevic, 2000) such operators are used in less than 20% of the queries. Therefore, this work does not consider such operators.

up in the list of result documents. The inverse document frequency assigns a high value to terms that are found only in few documents in the collection, but a low value to the more common terms; it is used by many query evaluation algorithms to decide the order in which terms are processed.

2.3 Query Processing

The most commonly used index structure is the *inverted index*. It has one *inverted list* for each term t, in a document collection c where all $(d, f_{d,t})$ entries (required for ranking) are stored. The term "inverted" is drawn from the fact that the document collection is inverted from being a collection of documents that consists of terms to being terms with a collection of documents. Query processing is typically performed by scanning the inverted lists and accumulating (partial) scores for documents. Since IR systems do not return a single correct answer, IR researchers have developed *unsafe* (or *approximate*) query evaluation algorithms, which improve the response time of the system at the cost of a potential degradation in retrieval effectiveness. Some unsafe optimizations achieve significant improvement in response time for individual queries, while maintaining acceptable retrieval effectiveness.

2.4 Web Search Engines

Modern web-search engines, such as Google, are designed to be scalable search engines that avoid all disk accesses (Brin & Page, 1998; Kleinberg, 1998). They typically use a lexicon, containing a list of available terms, along with the inverted index that the lexicon points to, to answer user queries. Pages are downloaded using distributed crawlers (also known as spiders), which store them into a repository. An indexer then stores information about each web page, including the link structure, in an anchor file. This anchor file is then used to calculate page rankings in a post-processing step, where the link-structure of the documents is used to find good candidates for returning to the user. Since these algorithms are complex and usually proprietary, and we are more interested in the user behavior patterns, we have omitted these post-processing steps from our current work. Extending our work to generate query results with a reasonable hyper-structure is a very interesting path for future work.

Chapter 3

Related Work

We are not aware of any work that uses traces to generate query sequences, as we do. Web query traces have, however, been analyzed in various ways. Jansen (2006) provides an extensive overview of a search log analysis. He describes a search log, its existence and the different levels of analysis. He explains why it is collected, how it is used and can be used, along with an overview of what has already been done in this area of research.

A very large search engine log was analyzed by Silverstein, Marais, Henzinger, and Moricz (1999). They show that queries are typically short and seldom modified. They also show that query terms are frequently constituents of phrases, meaning that search engines might consider search terms as parts of phrases even if the user did not explicitly specify them as such.

Ozmutlu, Spink, et Ozmutlu (2002) use Poisson sampling to analyze a search log from Excite (`www.excite.com`), they show how a sample set can represent the characteristics of the entire search log.

A time analysis on a trace was performed by Beitzel, Jensen, Chowdhury, Grossman, and Frieder (2004). The analysis was performed on a very large query log and shows how traffic and topical categories fluctuate between hours within days. Some categories seem to fluctuate considerably while others are stable all day long. The research offers valuable input into the aspect of search engine performance testing regarding indexing, routing and caching algorithms.

Transaction logs from nine different search engines were compared by Jansen and Spink (2006). They compared the difference of usage between search engines. The results were that web searching is performed in a similar way, regardless of what search engine is

being used. We hence conclude that an analyzed trace from one search engine is suitable for generating a query trace to test another search engine.

Web queries can be categorized into three broad categories: informational queries, navigational queries and transactional queries (Jansen, Booth, & Spink, 2008; Rose & Levinson, 2004). Informational queries are queries that cover a broad type and return many relevant results (e.g., Colorado or trucks). Navigational queries seek a single page (e.g., youtube or delta airlines). Transactional queries arise when the user intends to perform an action such as purchasing a car or downloading a program. Zhang, Jansen, and Spink (2009) analyzed a web search engine transaction log and showed that most queries belong to the informational query category, and that the rate of those queries varied during different time periods, while the other two categories were more stable. The proportion among these categories was about 12:2:1 (informational:transactional:navigational).

The study most similar to ours, is the one by Zhang et al. (2009), who performed a time and pattern analysis of a web search engine transaction log with an aim towards building a model of user interactions. They analyze various characteristics of the transaction log but do not look into each detailed action such as adding or dropping a term. They used a transaction log from the Dogpile web search engine (`www.dogpile.com`) and showed that time series analysis can be used to predict some user behavior such as that users who enter the shortest queries are more likely to click the highest rated results. Among other interesting results were that people seem to use the basic web search engines more, rather than using search engine variant that focuses on finding specific file types such as images or audio files. The average query length was about 2.9 terms and remains stable for the duration of the log.

The model presented in this thesis was originally proposed by Tryggvason (2002). We have refined the model with respect to users dropping terms, introduced the formal notation used in this thesis, and added policies for selecting terms for the generated sequences. The software used for the experiments in this thesis was originally created by Böðvarsson (2006) and Berghreinsson (2007). The software was improved, however, as the performance was greatly improved and errors removed, and the new policies and methods were added.

Chapter 4

A Model of User Behavior

The basic search engine user actions are fairly simple. The user starts by writing a few terms, and then submits the query. Based on the outcome, the user may choose to refine the query or to end the session. During query refinement, the user can add terms and/or drop terms, and then submit the refined query. There may, of course, be arbitrarily many refinements. The size (or the length) of the session depends on the number of refinements made by the user. For each time the user clicks the search button, a query is stored in a search engine log and is also known as the query trace as we use in our experiment. Our goal is to simulate the process described by the query trace.

The query generation is performed in two steps. The first step is to generate an *action sequence* that imitates the overall user behavior of adding terms, dropping terms, and submitting queries. The second step is term selection, where the actual terms of the queries are chosen. Splitting the procedure into two steps makes it possible to use different term selection methods for the same action sequence, making comparison and evaluation more precise and efficient.

In this chapter, we first propose a simple Markov chain for modeling user behavior (Section 4.1). We discuss how trace analysis can yield probability values for the parameters of this model (Section 4.2) and how the model is used to generate action sequences (Section 4.3). We propose two refinements to the model to more accurately capture the evolution of model parameters during the query refinement process (Section 4.4). Finally, in Section 4.5 we discuss an improved model to distinguish better between add and $drop$ actions. The improved model was used to partially generate action sequences. Term selection policies are then described in Chapter 5.

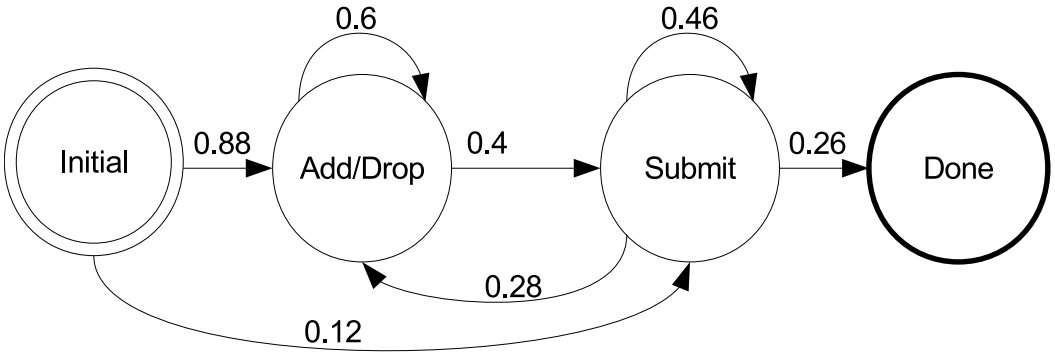

Figure 4.1: The basic Markov chain

4.1 A Basic Model

A Markov chain (MC) is a mathematical framework for modeling transitions from one state to another. The next state depends only on the current state and not on previous states or transitions. The model is often described as a directed graph, where the edges are labeled with the probability of each transition. A MC is a sequence of random variables $X_1, X_2, X_3, ...$ with the Markov property, namely that, given the present state, the future and past states are independent. Formally, $Pr(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = Pr(X_n + 1 = x|X_n = x_n)$. The possible values of X_i form a countable set S called the state space of the chain.

As discussed above, a user can add or drop terms, submit the query and end the session and these are the possible transitions using the MC model. Likewise, in this simple model, the set of states is $S = \{\textit{initial}, \textit{add/drop}, \textit{submit}, \textit{done}\}$. The name of state s indicates the last action made by the user. Figure 4.1 illustrates the states s and possible transitions, along with probability for the user behavior using MC model[1]; surprisingly, a transition

[1] the probabilities shown are actual numbers from our analysis.

Query	User	Query
1	1	program
2	1	certify program
3	2	inform certify require employee
4	3	certify program
5	3	certify employee

Table 4.1: Example of a search engine query trace

is possible from the *initial* state to the *submit* state, resulting in an empty query.[2] The transitions that lead to the state *add/drop* can either be to add or drop a term; MC distinguish between these actions in code. The high re-submitting rate is because users click a link that views the next ten results and by doing that, the same query is re-submitted with different page numbers. These re-submissions could easily be removed from the query trace, but we decided to leave them in as they may be valuable to search engine optimizers. More precisely, the possible transitions are:

$$
\begin{aligned}
Pr(X_{n+1} = \textit{add/drop} \mid X_n = \textit{initial}) \\
Pr(X_{n+1} = \textit{submit} \mid X_n = \textit{initial}) \\
Pr(X_{n+1} = \textit{add/drop} \mid X_n = \textit{add/drop}) \\
Pr(X_{n+1} = \textit{submit} \mid X_n = \textit{add/drop}) \\
Pr(X_{n+1} = \textit{add/drop} \mid X_n = \textit{submit}) \\
Pr(X_{n+1} = \textit{submit} \mid X_n = \textit{submit}) \\
Pr(X_{n+1} = \textit{done} \mid X_n = \textit{submit})
\end{aligned}
$$

4.2 User Behavior Analysis

The probability of each transition is calculated by analyzing an existing query trace, which represents a set of typical user behaviors. The analysis process steps through the existing query trace using the MC. For each term added or dropped a transition is made and also for each submission and an end of a session. During the analysis the transitions and the visitations to each state are counted and proportioned to find the probability of each action. For further illustration we use a simple example.

[2] Most of the empty queries in the search trace used in the experiments arise due to a cleaning process that removes terms that are not in the document collection. Empty queries may, however, arise in practice and search engines treat those queries differently: Google and Bing do nothing, while Yahoo! shows a special page with a demonstration video.

		State X_n			
		initial	*add/drop*	*submit*	*done*
State X_{n+1}	*initial*	0.00	0.00	0.00	0.00
	add/drop	1.00	0.55	0.40	0.00
	submit	0.00	0.45	0.00	0.00
	done	0.00	0.00	0.60	0.00

Table 4.2: Markov chain probability transition matrix

Example 1: Behavior analysis using basic MC

Consider the simple query trace shown in Table 4.1. This trace contains five queries and refinement processes for three different users.

The first user starts by adding a term, thus moving from state initial *to state* add/drop. *Subsequently, the query is submitted, thus making a transition from state* add/drop *to state* submit. *The user then refines the query by adding another term, thus moving from state* submit *to state* add/drop. *The query is now re-submitted, thus moving from state* add/drop *to state* submit. *Finally, the user session ends, moving from state* submit *to state* done. *At this point in time, the user has twice moved from state* add/drop, *in both cases to the state* submit *via the transition* submit, *so* $Pr(X_{n+1} = submit|X_n = add/drop) = 100\%$.

The second user, on the other hand, adds three words from the state add/drop, *and then submits. At that point in time, the state* add/drop *has been visited six times; in half of those cases the query has been submitted, while in the other half of those cases, a term has been added. Therefore, after processing two users, the probability of transitions* $Pr(X_{n+1} = add/drop|X_n = add/drop)$ *and* $Pr(X_{n+1} = submit|X_n = add/drop)$ *are both 50%.*

We leave it as an exercise for the reader to verify that after analyzing the entire trace, the probability transition matrix P *is as shown in Table 4.2.*

A note is in order regarding the choice of *add* and *drop* actions for the third user of Table 4.1. This user adds one term and drops another while preparing the second query. We choose to always select *drop* actions ahead of *add* actions, in order to avoid dropping newly added terms.

Query	State X_n	State X_{n+1}	Terms
1	*initial*	*add/drop*	1
1	*add/drop*	*submit*	1
2	*submit*	*add/drop*	0
2	*add/drop*	*add/drop*	1
2	*add/drop*	*submit*	1
3	*submit*	*add/drop*	2
3	*add/drop*	*submit*	2
3	*submit*	*done*	–

Table 4.3: Example of a generated action sequence

4.3 Generating Actions

The output of the user behavior analysis is the complete probability transition matrix P. Action generation consists of using this probability matrix to guide a random selection of actions. The generation is always started in state *initial* and weighed random function is used to decide the next step where the transition with the higher probability is more likely to occur. A query can be edited by adding and dropping terms but at the time of the action "submit", the query is "written" as it looks like at that time. At the end of each session, the number of visitations to the state *submit* is compared to the number of sequences to be produced, while it is not exceeded, the next session is started in the state *initial*. Example 2 describes the process of generating an action sequence.

Example 2: Generated action sequence

Table 4.3 shows a typical action sequence based on the analysis of Example 1, containing one session with three queries. The first query contains one term, the second query also contains one term—albeit a different one—and the third query contains two terms, including the term from the second query.

As the example in Table 4.3 shows, user sessions can be longer in the generated action sequence than in the original trace. It may also result in queries containing a particular number of terms that never appeared in the original trace.

4.4 A Refined Model of User Behavior

In Chapter 5 we describe policies to assign the actual terms to the queries. Before that, however, we must address two accuracy issues with this simple model of user behavior. A user submitting the first query should be more likely to refine the query than a user

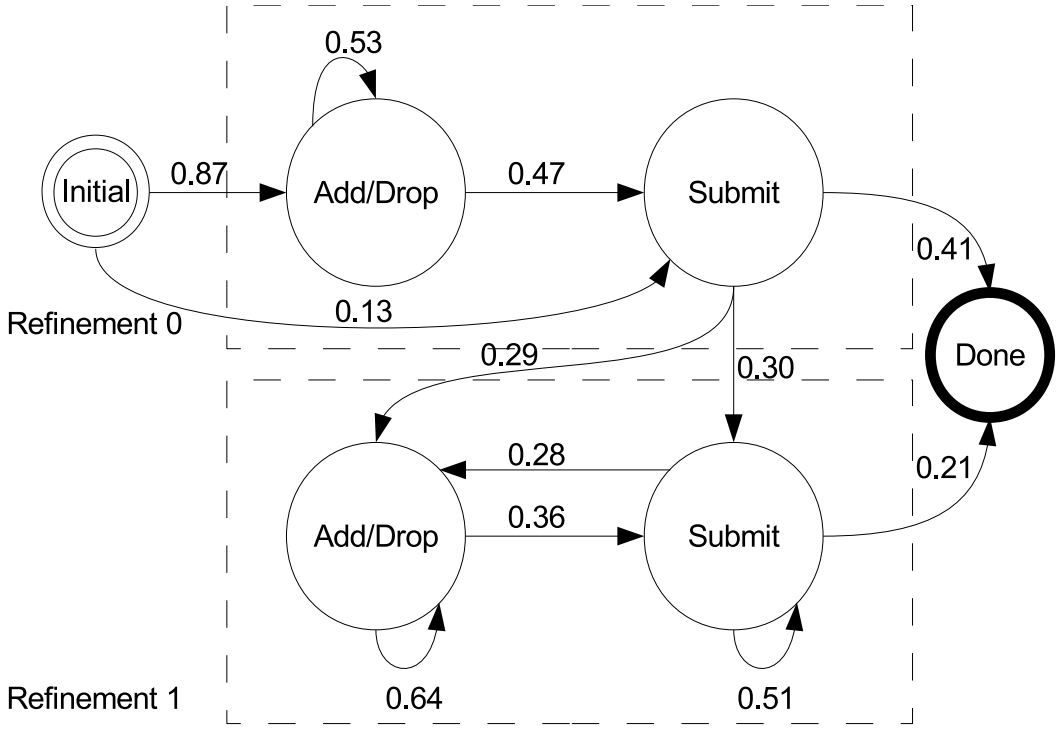

Figure 4.2: A refined MC to account for impact of query refinements

submitting the tenth query and therefore we need to add a refinement dimension to our model. A similar issue arises with the number of terms in a query, as a user with one term in the query should be more likely to add a term than a user that already has five terms, a user with no terms should not have any possibility of a *drop* transition.

What is needed to improve the accuracy of our model is a method to base the probability function on the number of query refinements seen so far and the number of terms in the previous query. Fortunately, however, this turns out to be fairly easy.

4.4.1 Impact of Query Refinements

Consider first the impact of query refinement. Figure 4.2 shows a modified MC which distinguishes between the initial query (here called refinement 0) and subsequent refinements queries (summarized into refinement 1). We can refer to the split states using $add/drop_0$, $add/drop_1$, $submit_0$ and $submit_1$.

Query	State X_n	Action	State X_{n+1}	Terms
1	*initial*	*add*	*add/drop*	1
1	*add/drop*	*drop*	*add/drop*	0
1	*add/drop*	*submit*	*submit*	0
2	*initial*	*add*	*add/drop*	1
2	*add/drop*	*drop*	*add/drop*	0
2	*add/drop*	*submit*	*submit*	0
2	*submit*	*end*	*done*	–

Table 4.4: Illustration of accuracy issues with the simple model of user behavior

Needless to say, it is also likely that the probability of dropping terms on the first refinement (second query) is different from the probability of dropping terms in the second refinement. We propose to address this by extending the model even further to distinguish between states *add/drop*$_0$ through *add/drop*$_n$ and *submit*$_0$ through *submit*$_n$. Note that the states *add/drop*$_n$ and *submit*$_n$ always summarize refinements n and above.

A key question, then, is how large n should be. A good rule of thumb should be to make n sufficiently large to simulate the original trace well, while avoiding overfitting by distinguishing between session lengths that occur very rarely in the original trace. In Chapter 6 we demonstrate how to apply this methodology and we will use the term R_{max} to represent the maximum n.

4.4.2 Impact of Terms in Previous Query

The MC can be extended similarly to account for the terms in the previous query, to make the model more accurate and avoid transitions such as dropping a term with an empty query. In this case, e.g., state *add/drop*$_{i,j}$ refers to refinement i that started with j terms. Note that the original query always starts without terms, so *add/drop*$_{0,j}$ is not a valid state for $j > 0$. The same methodology can be used to avoid overfitting to rare cases as for the number of refinements. In Chapter 6 we use the term T_{max} to represent the maximum j.

4.5 Improved Markov Chain

Our model does not distinguish between the actions *add* and *drop* which causes a problem of dropping newly added terms, this can cause empty queries and an unnecessary work being performed.

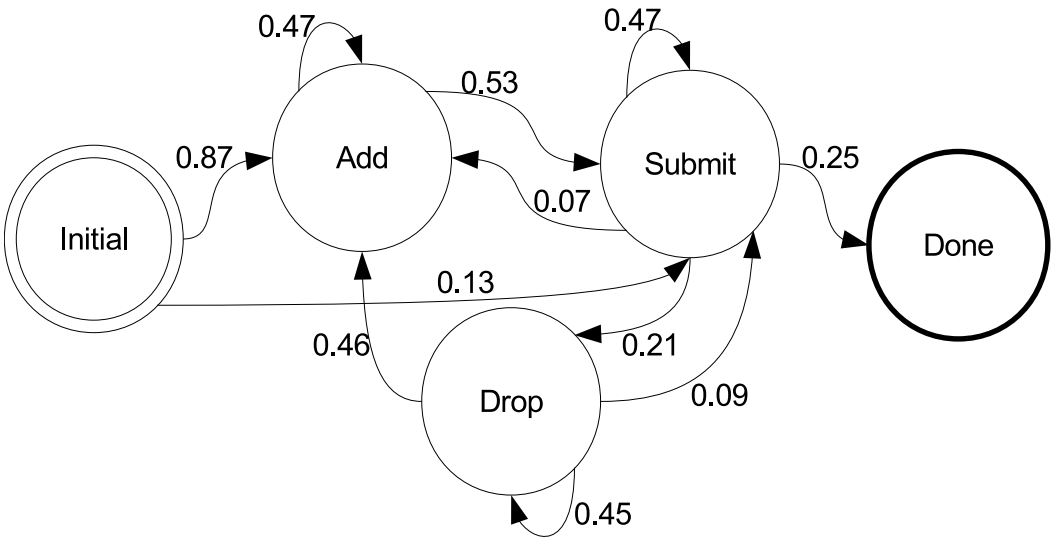

Figure 4.3: The improved basic Markov chain

The problems are better illustrated with the following example.

Example 3: Accuracy issues during generation of an action sequence

Consider the action sequence of Table 4.4. Note that this action sequence is not based on the probability function from the previous example, but rather on a more generic probability function such as might arise with a very long and varied original trace.

In the first query, a term is first added as is typically the case. Subsequently, however, the same term is then dropped, resulting in an empty query being submitted. This sequence is essentially the same as submitting directly from the initial *state, and would never appear in the query trace; the probability of dropping a term in the initial query should be 0%, which is different from the general case.*

In the second query, a similar situation arises, where a newly added term is immediately dropped. Since this is not the first query, however, dropping terms can indeed occur. What should not occur, however, is the immediate removal of newly added terms.

Note that we have coded the generation such that newly added terms are only dropped if there are no older terms, but we did not disallow dropping terms. A better solution

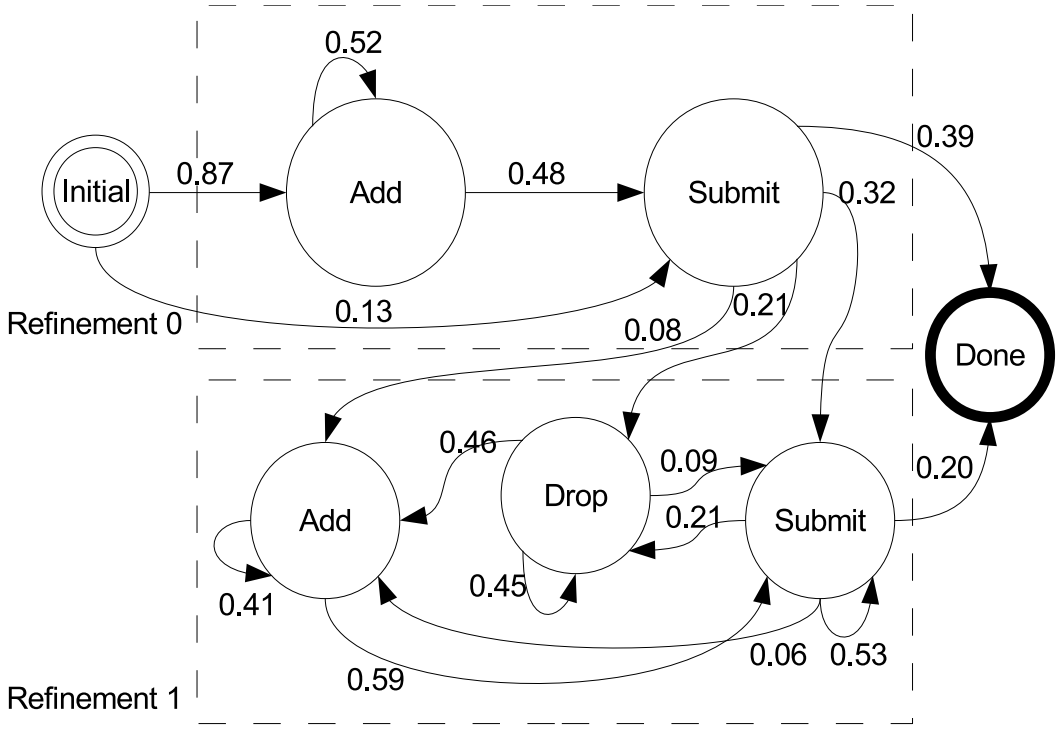

Figure 4.4: The Extended Markov chain model improved

is to split up the *add/drop* state into states add and $drop$. This was discovered late in the process and due to the tight timeframe, it was impossible to run the full experiment using this improved model. In this section we describe the improved model. We managed to generate action sequences using the improved MC model and compare them to the formerly generated action sequences. These initial results indicate little difference from the previously generated action sequences and should therefore not affect the final result. The results and the comparison between action sequences will be discussed further in Chapter 6.

Figure 4.3 shows how the state *add/drop* has been split into the states *add* and *drop*. Performing any transition that points to the state *drop* tells us that the user was dropping a term. Note that transition from state *add* to state *drop* is not available due to the fact that we choose to drop terms first when user refines the search; as we described earlier.

Figure 4.4 illustrates the same changes made to the refined model of Figure 4.2. The *drop* state is not used in the initial query and therefore it shows up only on refinement level one.

4.6 Summary

In summary, we initially presented a very simple MC for user behavior and presented a methodology for learning the probability transition matrix P and using the MC to generate action sequences that simulate user search behavior. We then extended that model into a two-dimensional array of states, which allow us to more accurately capture the evolution of the probability function during the query refinement process. At a later stage we improved our model to capture the edit actions of users even better to prevent dropping newly added terms and skip unnecessary work. So far, however, we have not created any specific queries; we now turn to the actual term selection policies.

Chapter 5

Term Selection Policies

This chapter describes the second step of query sequence generation, namely term selection, where the actual terms to *add* and *drop* are chosen. The baseline policy is *random selection* (*RN*) where all terms have the equal probability of being added or dropped. As with the probability function of the MC, however, we propose to analyze the properties of the terms in the original trace and use those properties to select the terms of the generated queries.

We propose three such policies: The *List Length* (*LL*) policy uses information about the length of the inverted list (how many documents a term can be found in) to guide the term selection; The *Average Relation* (*AR*) policy uses the average similarity of a term to the documents in the collection; and a combined policy (*AL*) which uses both the list length and the average relation to select the terms.

A lexicon table consists of distinct row for each term found in a document collection. Each row has information relevant to the term such as the list length and in our case we also store information about average relation. The lexicon points to an inverted index which stores in which document the term is found in. Example 4 shows what a lexicon could look like for a small document collection.

Document	Text (Cleaned and Stemmed)
1	require employee inform supervisor inform
2	require
3	program employee certify require
4	program require

Table 5.1: Example document collection

Term	***LL***	***AR***	$\lceil$***AR***$\rceil$
require	4	1.x 10^{-29}	1
employee	2	0.39	1
inform	1	0.57	1
supervisor	1	0.29	1
certify	1	0.50	1
program	2	1.25	2

Table 5.2: Term information summary

Example 4: Lexicon table

Consider the document collection shown in Table 5.1, which will be used as a running example in this chapter. As is typical in IR, the documents in this figure have been cleaned (HTML tags have been removed, as well as common words from a stop list) and the terms stemmed. Table 5.2 shows a lexicon table for this collection. The table includes the inverted list length for each term, as well as the average relation (the calculation of the average relation will be described in detail later in this chapter).

5.1 The List Length Policy

The length of the inverted list indicates how many documents the term occurs in, and is thus a basic measure of how common the term is in the document collection. Table 5.2 shows that most terms in our running example occur in one document, while one term occurs in all four. Such skew is, in fact, common in real-life document collections; Figure 5.1 shows the distribution of list length in the document collection used in our experiments. The figure shows that many terms have low list length and few terms have high list length. The collection is described further in Chapter 6.1.

In order to use the list length to guide the term selection process, we accumulate statistics on the list length distribution in the original query trace, into a probability histogram. A separate distribution is accumulated for each transition in the MC. This distribution is then used in the term selection process to choose terms. When adding a term using a transition such as $Pr(X_{n+1} = \textit{add/drop} | X_n = \textit{initial})$ we have a list of terms (or a list of specific characteristics of a term, in this case **LL**) that have been added using that transition, followed by a number of occurrences. We then use weighted random to choose the **LL** we want to use. Finally a random term is chosen from the terms that fulfill the criteria from the inverted index. Example 5 will explain the *add* transition.

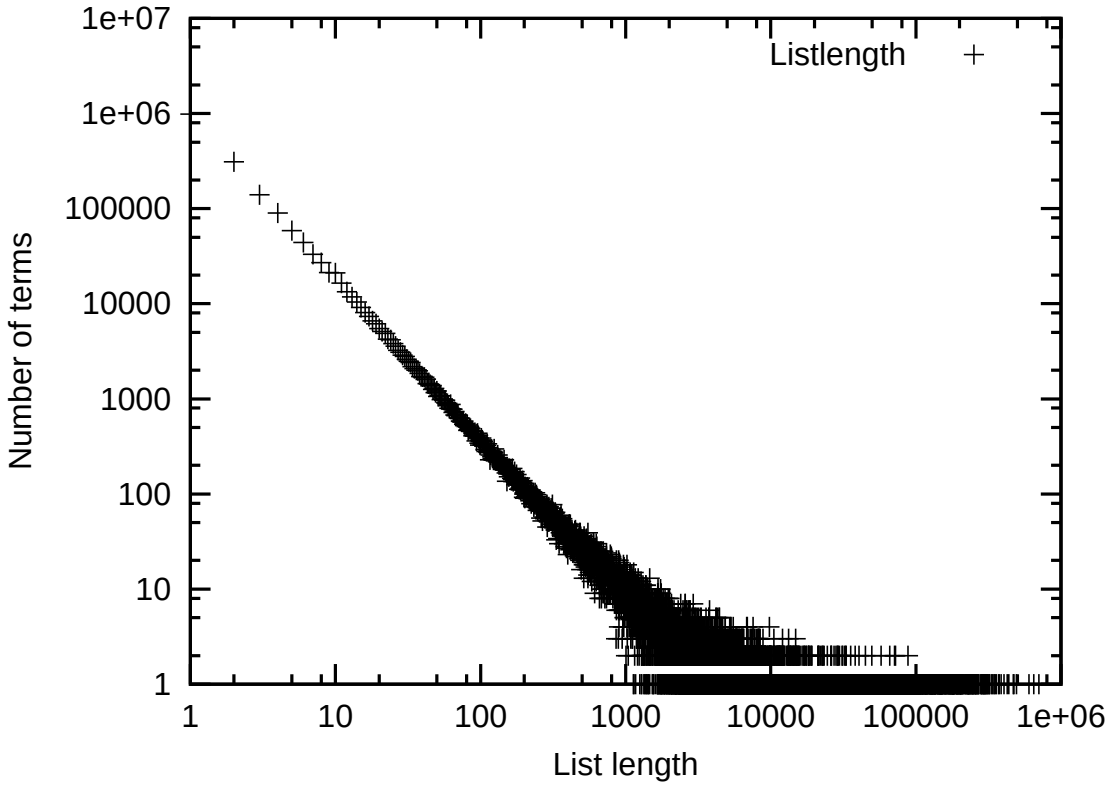

Figure 5.1: Inverted list length distribution

Example 5: Adding a term using LL policy

Consider the original query trace in Table 4.1. Furthermore, let us focus on the first term selected in each user session, which is the transition $Pr(X_{n+1} = add/drop|X_n = initial)$ *(using* add*). This transition occurs three times in the query trace, and the terms selected are: "program", "inform", and "certify". Turning to Table 5.2, we see that "inform" and "certify" have list length of 1, while "program" has list length of 2. The list length distribution, represented by (*list length:count*) pairs, is thus: (1:2); (2:1).*

Turning to the example action sequence in Table 4.3, we see that the first action is indeed to add a term from this distribution. We now have a 67% probability of choosing a term with list length 1 (the candidates, according to Table 5.2 are "inform", "supervisor" and "certify") and 33% probability of choosing a term with list length 2 (the candidates are "employee" and "program").

The first attempt to drop a term is a similar process as adding a term, but the main difference is that we only have the terms in the current query to choose from and we can not count on that we have a match. If there is no match to the analyzed distribution, a random term from the current query is removed. Example 6 describes the drop process.

LL_N	LL_2	LL_1
1	1	1
8	8	8
10	10	10
12	12	10
16	16	20
92	92	90
96	96	100
112	110	100
1,154	1,200	1,000

Table 5.3: Examples of significant digit calculations

Example 6: Dropping a term

Using the original query trace in Table 4.1, we can see only one case of the user dropping a word. The user(3) has submitted the query "certify program" and needs to drop the term "program", before he adds "employee" and submits again. The transition $Pr(X_{n+1} = add/drop|X_n = submit)$ *(using* drop*) is used to drop the word. According to Table 5.2, the word "program" has list length of 2 and therefore we try to use that to select a term to drop. If none of the terms in the current query fits that criteria, a term is chosen randomly from available terms.*

When terms are added, we are guaranteed to find at least one term with the selected list length. For the shorter list lengths, however, many candidates exist and the term selection is thus not restricted to the actual terms used in the original query trace. When terms are dropped, on the other hand, it is not always possible to find a term with the selected list length, since the selection is now restricted to the terms that are already in the query. In this case, the term to drop is chosen at random.

When using a large collection, it is possible to overfit the distribution. Consider, for example, a term that occurs in 4,376 documents. It is unlikely that any other term occurs in that many documents, but there may be several terms that occur in 4,350–4,450 documents and those terms should be equally relevant. In order to avoid overfitting and to increase the flexibility of the term selection process, we propose to vary the number of *significant digits* that are considered in each case.[1] Table 5.3 shows examples of rounding the list length *LL* to 2 or 1 significant digits, indicated by LL_2 and LL_1, respectively.

[1] Significant digits are commonly used in physics to indicate the accuracy of measurement equipment. The list length can indeed be said to be a measurement of frequency.

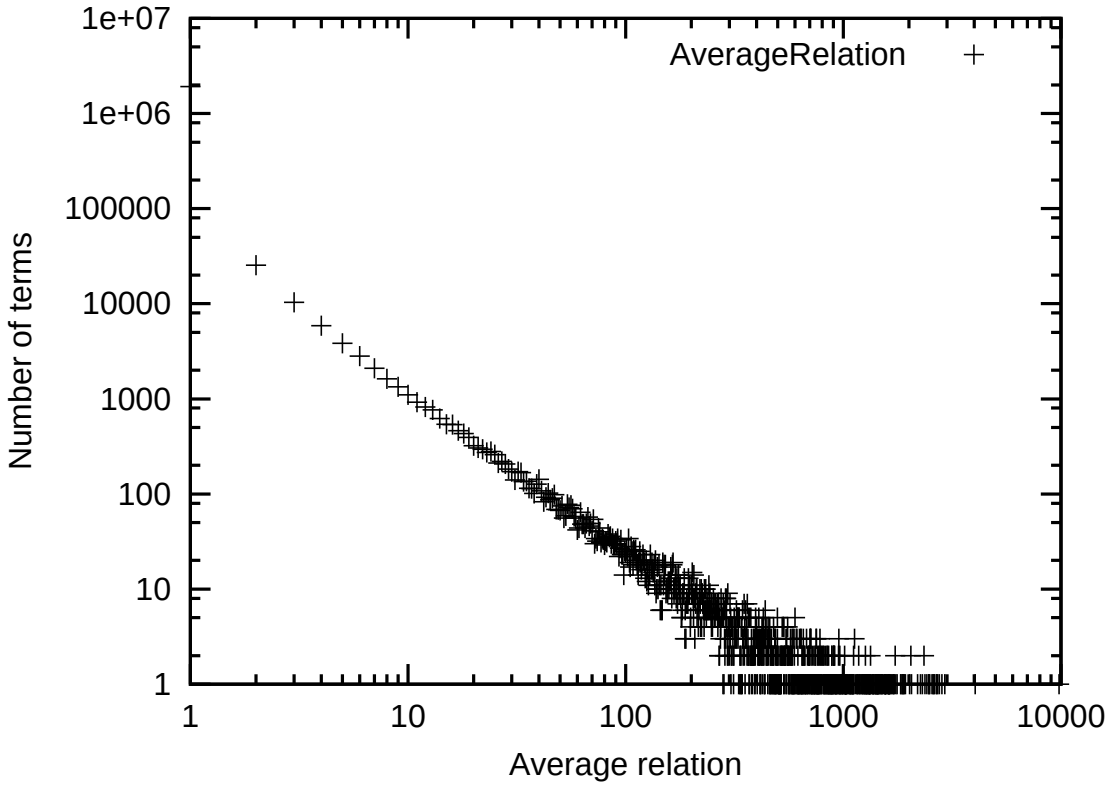

Figure 5.2: Distribution of average relation

5.2 The Average Relation Policy

The list length policy does not take into account the fact that terms may occur multiple times in some documents, which should affect the importance of a term. If a term appears in many documents, for example, we may assign little relevance to that term, whereas a term that occurs rarely but often in a single document might be more important. We propose to use the average relation of each term t to the whole document collection as the basis of a term selection policy. The average relation (*AR*) is defined as the weighted average of the partial similarities[2] of all documents d where term t is found:

$$AR_t = \sum_d \frac{w_{d,t}}{w_d} \tag{5.1}$$

where $w_d = \sum_t w_{d,t}$ represents the total similarity of each document to all terms.

[2] For the definition of partial similarity, see Formula 2.1 on page 6 and the following text, but note that $w_{q,t}$ is 1 in all cases.

Example 7: Adding a term using AR policy

Consider again the original query trace of Table 4.1 and focus on the first term selected in each user session, which is the transition $Pr(X_{n+1} = add/drop|X_n = initial)$*. This transition occurs three times in the query trace, and the terms selected are: "program", "inform", and "certify". Turning to Table 5.2, we see that "inform" and "certify" have average relation (rounded up) of 1, while "program" has average relation of 2. The average relation distribution is thus (*average relation*:*count*) pairs, is thus: (1:2); (2:1). When selecting terms from this distribution, we therefore have a 67% probability of choosing a term with average relation 1 (the candidates, according to Table 5.2 are "require", "employee", "inform", "supervisor" and "certify") and 33% probability of choosing a term with average relation 2 (the candidate is only "program").*

For the document collection used in our experiments, the results of the average relation calculation ranged from 0 to 9791.49. Figure 5.2 shows the distribution of the average relation; as with list lenghts it is very skewed. It shows that many terms have low average relation and few terms have high average relation. Since matching numbers with decimal points is a slow process, the numbers were rounded up to whole integers to speed up the term selection. As with the list length, we use 2 or 1 significant digits to avoid overfitting and to increase the flexibility of the term selection process.

5.3 The Combined Policy

The combination of average relation and list length (*AL*) term selection policy is similar to the previous policies, except that the probability distribution of terms is now a two-dimensional histogram, rather than a single-dimensional histogram as in the case of the *LL* and *AR* policies. We observe that there is a strong correlation between list length and average relation (the correlation coefficient is about 90%; Figure 5.3 illustrates the correlation between *LL* and *AR*) and the probability distribution is thus very skewed as before. As with the other two policies, we use 2 or 1 significant digits to avoid overfitting and to increase the flexibility of the term selection process.

5.4 Summary

We have proposed three policies for term selection, based on inverted list length (*LL*), average relation to the document collection (*AR*), and a combination of the two (*AL*).

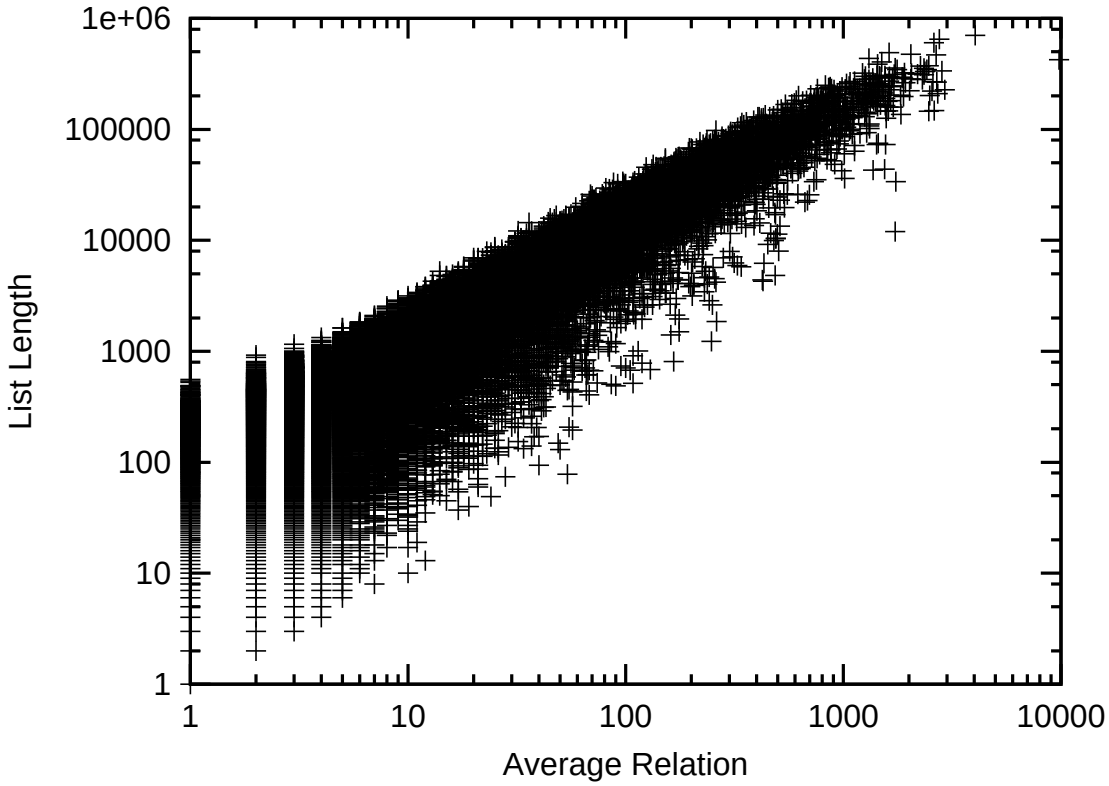

Figure 5.3: Correlation between list length and average relation

Since the list length and average relation are highly correlated, however, all three policies can be expected to give rather similar results.

We also experimented with a policy that used the relationship between individual terms to guide the term selection. This would, of course, best match the term selection process of actual users. Unfortunately, the preparation of the relationship statistics was projected to take months given our limited computing resources, and thus such a policy is left for future work.

Chapter 6

Experimental Evaluation

In this chapter, we study the quality and efficiency of our query generation methodology. We start by describing the experimental environment (Section 6.1). Then we study action sequence generation (Section 6.2) and finally evaluate the term selection policies (Section 6.3). The evaluation is performed by comparing characteristics of the generated query sequences to the original query trace that we intend to simulate.

6.1 Experimental Environment

The experimental environment consists of a query trace and document collection, which are used to generate statistics for the query generation. Furthermore, we developed a simple search engine to gather metrics regarding basic performance parameters.

6.1.1 Query Trace

The query trace we used as a starting point in our query generation is a well-known query log consisting of actual queries submitted to the web search engine `excite.com` in 2001. This query trace has been studied and analyzed in detail by other researchers (e.g., see (Spink, Jansen, Wolfram, & Saracevic, 2002; Spink, Wolfram, Jansen, & Saracevic, 2001)). The query trace consists of 1,025,910 queries submitted in 266,566 sessions. The average session length (number of queries in a session) is thus about 3.8 queries; the shortest session is a single query, while the longest session consists of exactly 100 queries. The average number of terms in a query is about 2.1; the largest query contains 38 terms,

while about 9.3% of the queries contained no term (due to the cleaning process in almost all cases).

6.1.2 Document Collection

The data collection used in this project is the WT10G (Web Track 10Gigabytes) dataset, a widely used benchmark for information retrieval development and evaluation. The dataset is distributed by CSIRO in Australia and was developed by a crawl of the web in 1997. The document collection was cleaned from html-tags, email addresses, and all characters other than English letters (a-z) and all terms were set to lower case and stemmed. After this processing the document collection consist of 1,991,324 terms in 1,681,575 documents. The most common term is *page*, which occurs 1,956,579 times in the collection, while the term *access* is the most common within a single document, occurring 41,938 times in one particular document.

6.1.3 Metrics

To evaluate the quality of the action sequence generation, we created 30 sequences of 100,000 queries each, and compared the query size and session length distributions to that of the original query trace. We define two parameters, R_{max} and T_{max}, to account for the size of the two-dimensional array of $add/drop_{i,j}$ and $submit_{i,j}$ states that distinguish between statistics based on the refinements and number of terms at the start of the query. We show how to select these parameters effectively.

To evaluate the quality of the term selection process, we generated actual queries from each of these 30 sets using the four different term selection policies (random, *LL*, *AR*, and *AL*), as well as three variations of significant digits (unlimited, 2, and 1), for a total of 10 runs (the number of significant digits does not affect random selection).

We then took the generated queries and ran them through our home-made search engine, which kept track of the total list length of all terms in the query and the answers for the query. The total list length indicates the basic processing cost of scanning the inverted index, while the number of documents occurring in more than one term is a measure of the similarity of the terms in the query; the more such documents, the more similar the terms.

6.1.4 Software and Hardware

The software builds on a prototype developed by E. Þ. Böðvarsson (2006) and refined by V. Berghreinsson (2007). The software was developed in Python, using numpy for the numerical calculation, and SQLite database. The timed measurements were performed on a Lenovo T61p laptop with 2 GB of RAM, an 2.40 GHz Intel Dual Core processor, and 320 GB of local disk space. The operating system was Microsoft XP Professional SP3.

6.2 Action Sequence Generation

The action sequence is a sequence of transitions that can be performed after each other, an example of generated action sequence is shown in Table 4.3. Before analyzing the quality and performance of the action sequence generation, it is necessary to set the T_{max} and R_{max} parameters to avoid overfitting to rare cases. We ran multiple experiments, where both T_{max} and R_{max} were varied from 0 to 10, and the performance analyzed for different combinations. As described in Section 4.4 setting $T_{max} = 0$ or $R_{max} = 0$ is not appropriate, but it does give us a baseline to compare to.

6.2.1 Selecting T_{max}

Figure 6.1 shows the impact of the T_{max} parameter on the average number of terms in a query and the corresponding standard deviation. The x-axis shows the value of T_{max} in each case, while the y-axis shows the average across all values of the R_{max} parameter. The values in each case are compared to the average number of queries and the corresponding standard deviation from the original trace that we are trying to simulate. We want to choose the parameter that is closest to the original trace. As the figure shows, changing T_{max} from 0 to 2 affects the average number of terms significantly. Increasing the parameter further does not affect the quality, since both the average number of terms and standard deviation seems to stay stable. The standard deviation shows a similar effect. Finally, the figure shows that $T_{max} = 2$ gives the best fit in terms of both average and standard deviation, and therefore we chose to use this value in our experiments. Note that the T_{max} parameter does not affect the session length significantly.

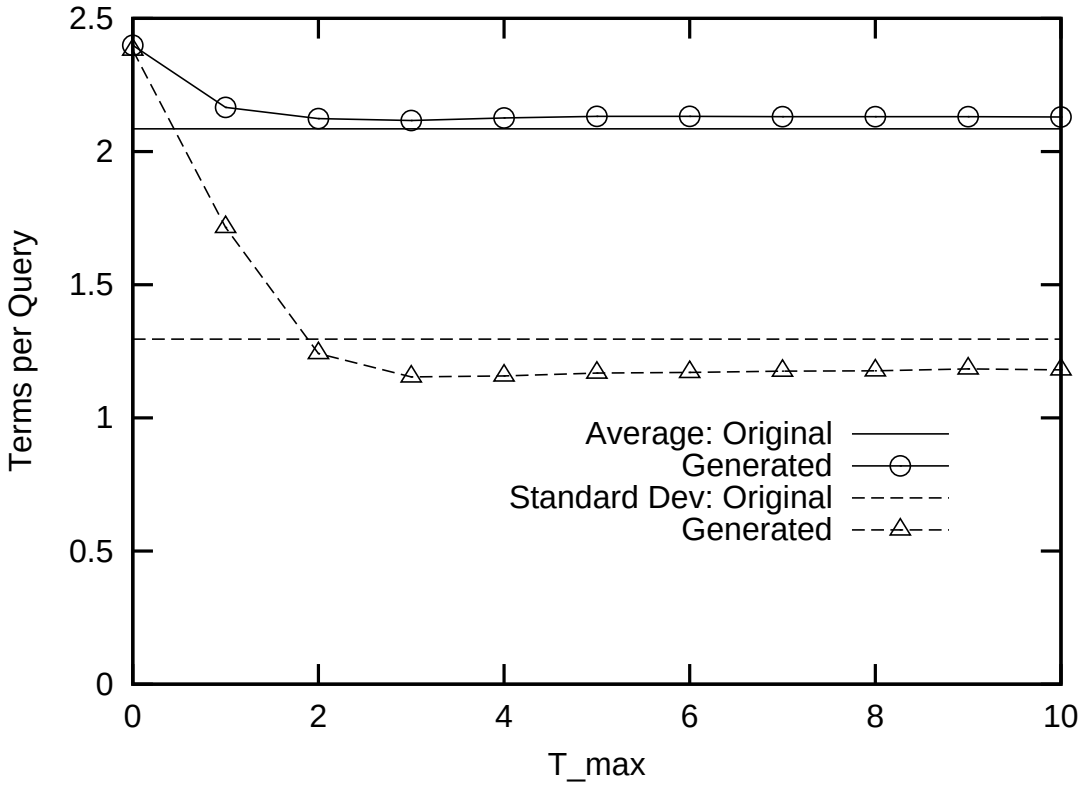

Figure 6.1: Number of terms in queries, varying T_{max}

6.2.2 Selecting R_{max}

Figure 6.2 shows the impact of the R_{max} parameter on the average number of terms in a query and the corresponding standard deviation. The x-axis shows the value of R_{max} in each case, while the y-axis shows the average across all values of the T_{max} parameter. The numbers are compared to the average number of terms and corresponding standard deviation from the original trace that we want to imitate. As the figure shows, the average is fairly stable across all values of R_{max}. The standard deviation, on the other hand, approaches that of the original trace as R_{max} grows. We chose to use $R_{max} = 9$, as only 8% of all queries occur in sessions longer than 9 queries. We also compared accumulated distribution of number of queries and $R_{max} = 9$ was closest to the original query trace. Note that the R_{max} parameter does not affect the query size significantly.

A smaller set of comparable action sequences were generated using the improved MC model described in Section 4.5. The results were nearly the same and the selection of T_{max} and R_{max} would be the same.

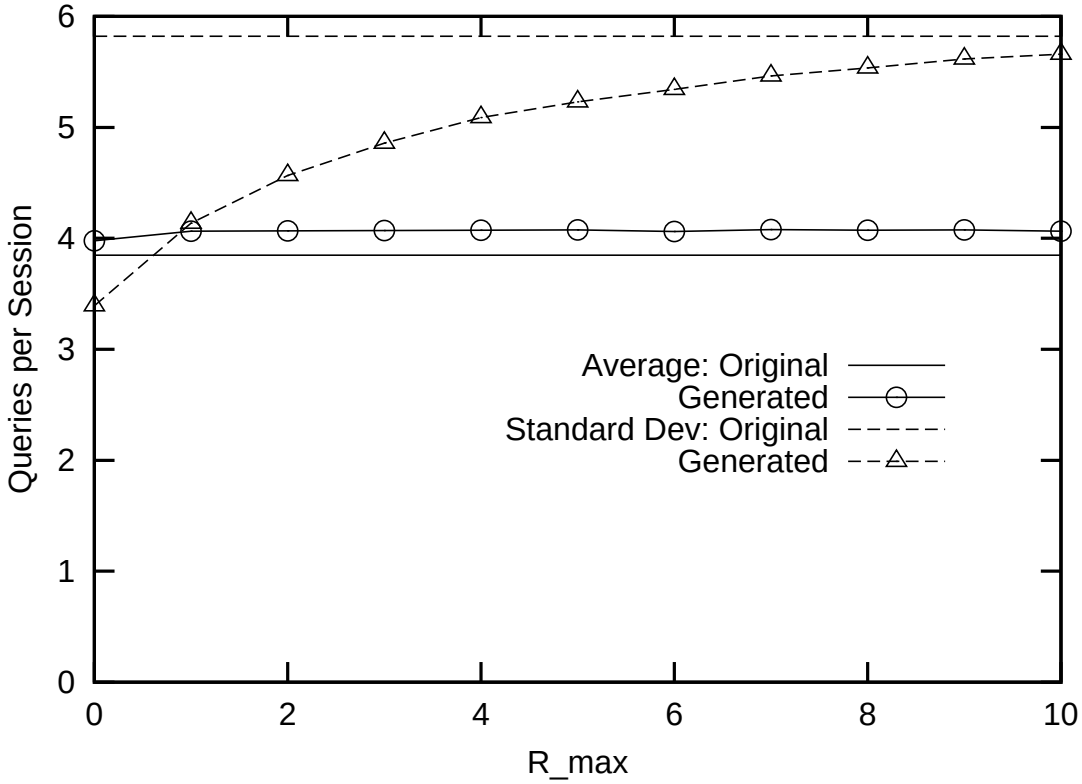

Figure 6.2: Number of queries in sessions, varying R_{max}

6.2.3 Quality

Figure 6.3 shows the distribution of query sizes for the generated sequences compared to the original trace. The two generated sequences were generated using two different models, the MC model and the improved MC model. The x-axis shows the number of terms per query, while the y-axis shows the proportion of queries that are of each size. All traces were generated using $T_{max} = 2$ and $R_{max} = 9$ as discussed above. As the figure shows, the generated sequences match the original trace quite closely. The generated sequences have sligthly more queries with only one term, but slightly fewer with no terms and two terms. We observe also that the generated trace has fewer queries with very many terms. Overall, however, as we saw in Figure 6.1, the difference between the generated sequences and the original trace is very small.

Turning to the number of sessions, Figure 6.4 shows the distribution of session lengths for the generated sequences compared to the original trace. Similar to the figure above, two different models were used to generate the action sequences, the MC model and the improved MC model. The x-axis shows the number of queries in a session, while the y-axis shows the proportion of a sessions that are of each length. The traces are generated using $T_{max} = 2$ and $R_{max} = 9$ as before. As the figure shows, the generated sequences

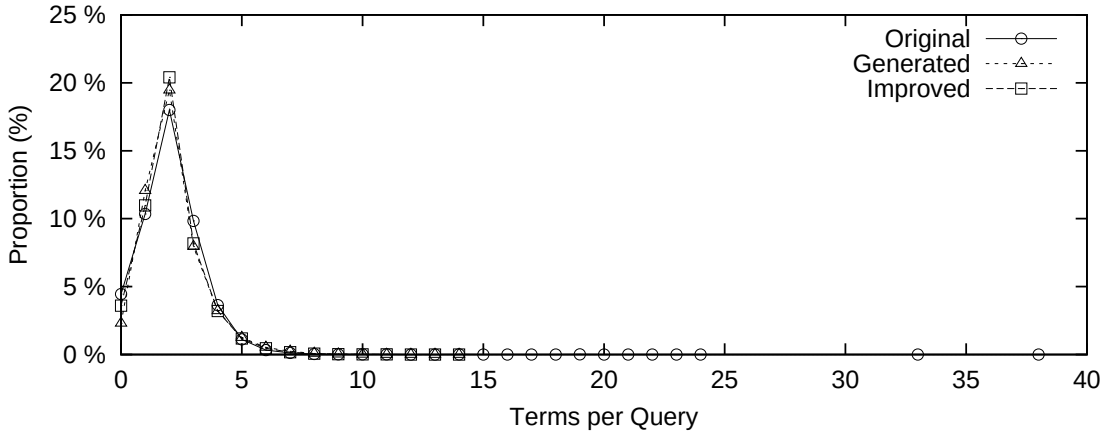

Figure 6.3: Distribution of query sizes

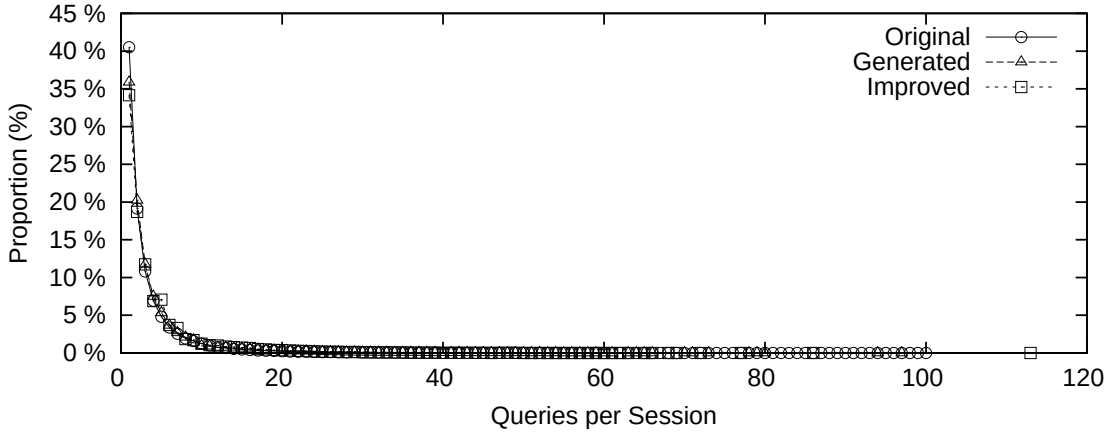

Figure 6.4: Distribution of session lengths

also match the original trace quite closely in terms of session lengths. The generated sequences have sligthly fewer sessions with only one query, but slightly more with a few queries. The improved model seems to generate a little longer sessions by average than the former model. The longest generated session using MC is 97 queries and 113 using the improved MC model, compared to 100 in the original trace. The 113 query session seems to be a extreme case, because looking at the data, a common maximum size is 75-80 queries. The session length does not affect the results of list length and answers. Overall, however, as we saw in Figure 6.2, the difference is negligible.

6.2.4 Time

Analysis of the million query trace takes around 30 seconds. It then takes approximately three seconds to generate an action sequence that consists of 100,000 queries.

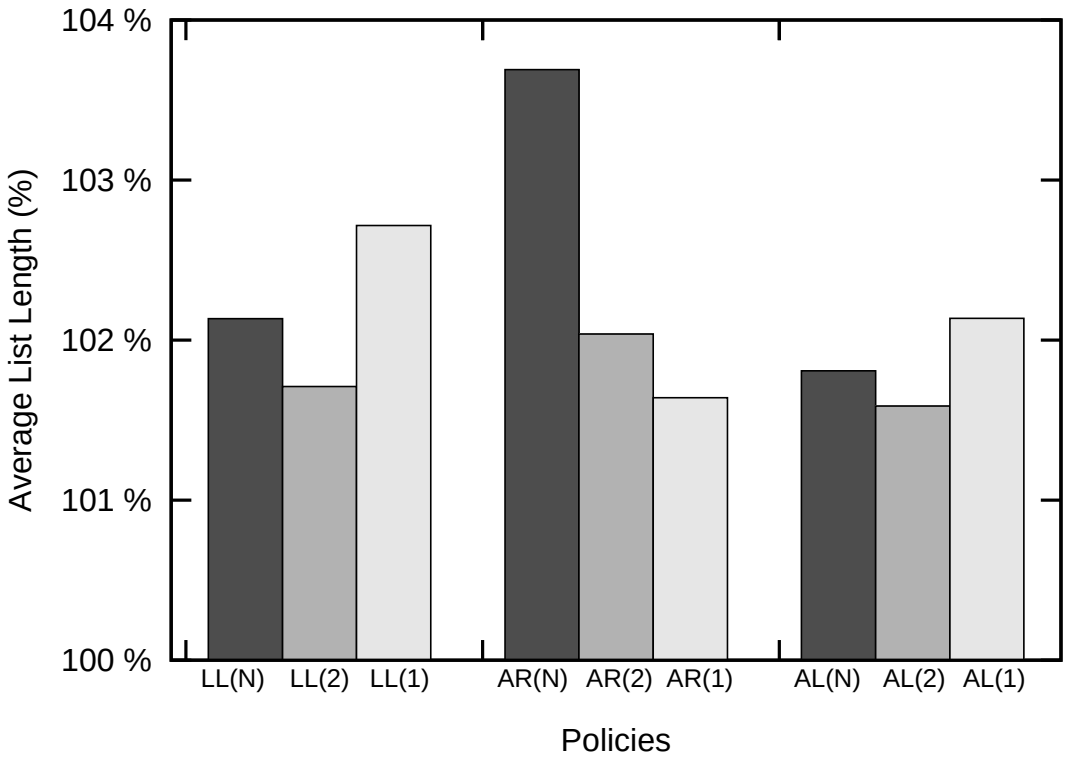

Figure 6.5: Average list length for each query

6.3 Term selection

We have seen that the action sequences match the original trace quite closely. We now evaluate the quality of the term selection policies, by comparing them to the original query trace. The home-made search engine processes all the queries in each sequence (or trace) and captures performance metrics such as the total list length per query, and the number of answers per query.

Note that, unsurprisingly, random selection of terms performs extremely poorly; due to the skew of the list length distribution, the average list length of random queries is only about 0.22% of the average list length of the original trace. More than half of the terms have a list length of less than three and are therefore likely to be chosen half the time with the random policy. In real life, however, such terms are selected much more rarely. As a result, we do not consider random term selection further, but focus instead on the three proposed policies: *LL*, *AR*, and *AL*.

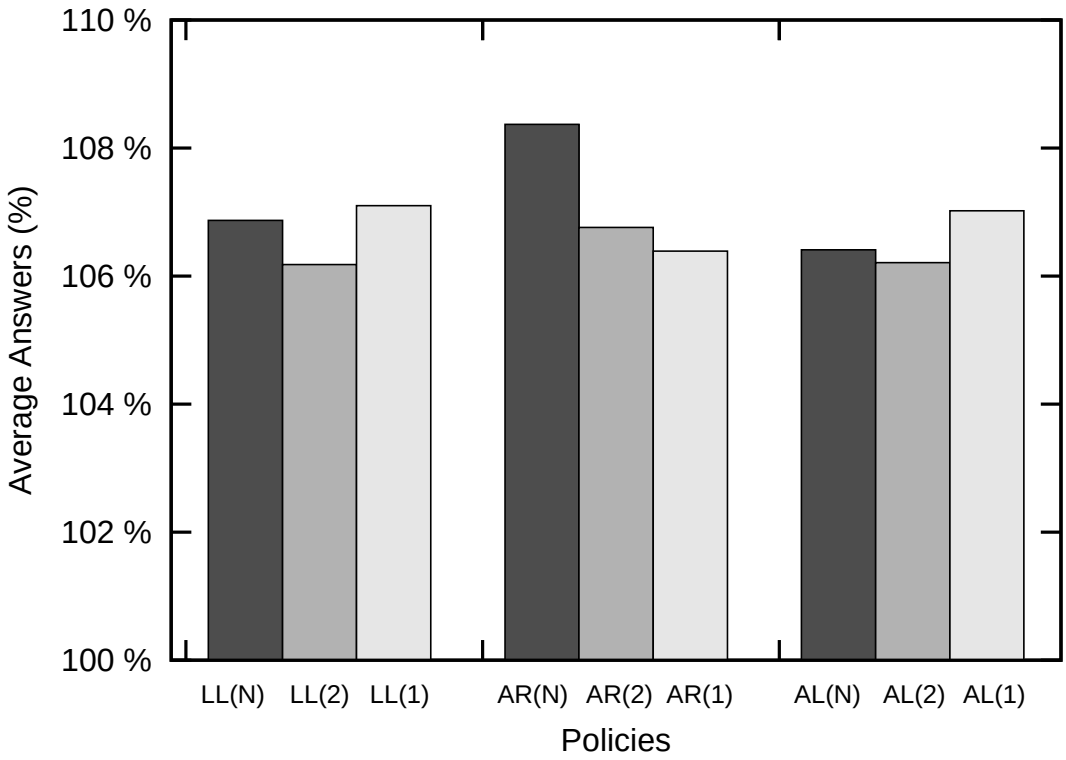

Figure 6.6: Average number of answers to each query

6.3.1 Quality

Consider first the total list length of all the terms in each query. This metric captures the work done to read all inverted lists, regardless of whether they are stored in memory or on disk. All policies have slightly longer inverted lists than the original trace, so Figure 6.5 shows the average list length as the percentage of the original trace (set to 100%). For each of the three policies, the impact of the number of significant digits is also shown. As the figure shows, all variations yield an average list length which is within a 4% difference from the original trace, and most are within 2%. For this metric, LL_2 and AL_2 are the best policies by a small margin.

Turning to the relationship between query terms, Figure 6.6 shows the average number of answers (distinct documents) found relative to the original trace. This metrics is relatively worse than the list length metric, because terms that actual users enter in the same query are typically more related. The difference, however, is still only 6–8%. For this metric also, LL_2, AR_1 and AL_2 are the best policies by a small margin.

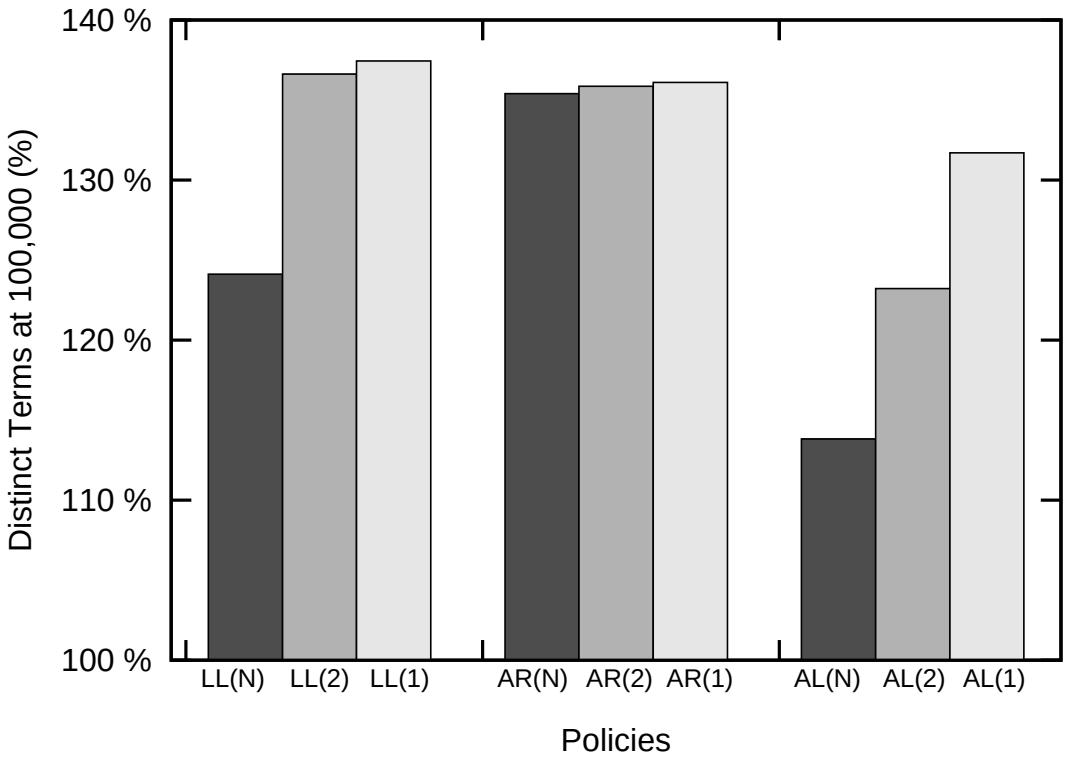

Figure 6.7: Distinct terms in 100,000 query sequences

6.3.2 Number of Distinct Terms

Figure 6.7 shows the number of distinct terms used in the 100,00 queries of the generated sequences, relative to the first 100,000 queries from the original trace. This is a metric that may affect buffer management significantly. All policies return a significantly greater variety of terms than the original trace. This was to be expected, since the choice is only guided by numerical values rather than semantics. The advantage of the generated sequences is that they are not restricted by the terms of the original trace and will thus exercise buffer management policies. The AL_N policy is the most similar to the original trace, as it has the tightest restrictions on the term selection.

Figure 6.8 analyses the number of distinct terms in the query sequences over a longer period. For this figure, we generated sequences of 1,000,000 queries and compared to the original trace. The x-axis shows the number of queries in the sequence, and the y-axis shows the ratio of distinct terms compared to the original trace. The figure focuses on two significant digits, but the results are similar for the other cases. As Figure 6.8 shows, the number of distinct terms is similar for the very first queries, and then diverges quickly and evenly from the original.

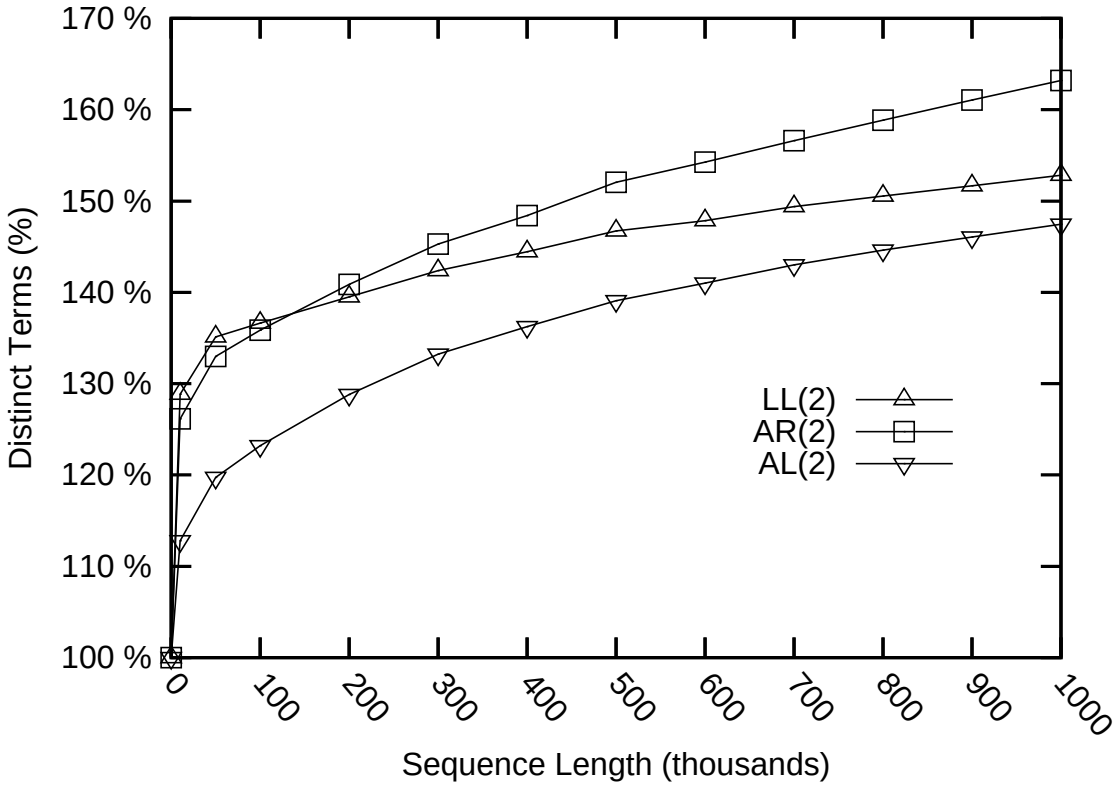

Figure 6.8: Distinct terms in query sequences

6.3.3 Time

Turning to the efficiency of the query generation, Figure 6.9 illustrates the time in seconds that it takes to select terms into 100,000 queries using different policies and significant digit variations. For detailed analysis, the time is split into four categories of work: a) the time to load the term table (about two million entries); b) the time to load the statistics used for term selection; c) the time to actually generate the queries; and d) the time to write the queries to disk. In the fastest methods, the majority of the time is spent on loading data into memory. In the slower methods more time is spent on the actual term selection, because the statistical data is much more complicated. The time is, in general, acceptable but the *AL* policies are the slowest.

6.4 Discussion

The term selection policies all return rather similar results. Most of our methods generate 100,000 queries in less than 50 seconds. The policies LL_2, AR_1 and *AL* yield average list length of 2% or less above the original trace. The *AL* policy is closest to the original

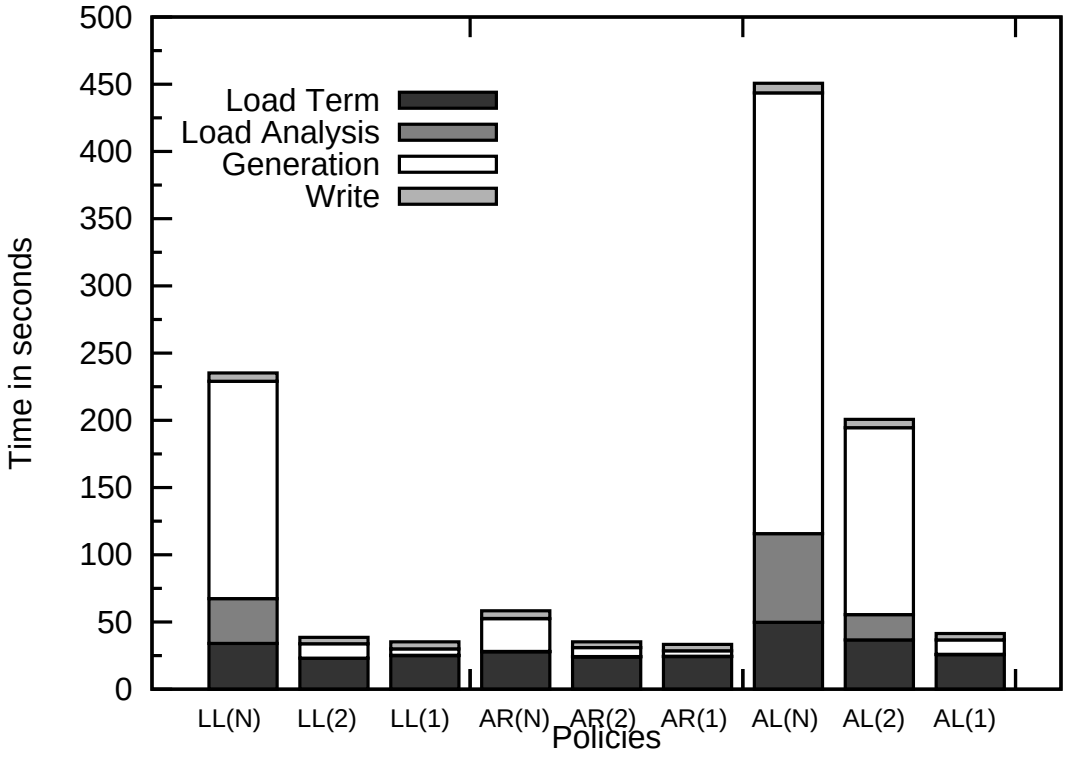

Figure 6.9: Generation time for 100,000 queries

trace in the number of distinct terms but it is also the slowest. The LL_2 policy is fast and appears to be the best option.

Chapter 7

Conclusion

We have argued that there is a dire need for a methodology to generate extremely long sequences of realistic queries, in particular for the general academic that does not have access to long search engine traces. The generated query sequences need not necessarily to be realistic in the sense that all the queries would make sense for actual users. Instead, it is more important that the overall characteristics of the queries exercise the query processing engine in a realistic manner, thus resulting in meaningful performance results.

In this thesis, we have proposed a novel methodology for analyzing user behavior patterns of the available short search engine traces, and using those behavior patterns to generate extensive query sequences in order to simulate actual search engine users. We have also described three different methods to select the actual terms of the generated queries. The terms are chosen with an aim towards delivering the same query characteristics as in the original traces and towards requiring a similar effort from a search engine. In a detailed performance study, we have showed that our methodology is able to simulate the original trace quite well, both in terms of user behavior patterns and in terms of basic search engine performance metrics.

The methodology does not currently capture all aspects of human behavior. More work is required, for example, to accurately emphasize the post-processing effort of various search engines, such as page ranking, and to simulate the temporal drift of attention between hot topics of the day. Many search engines allow proximity operators, which our methodology does not yet support. Implementing a term selection policy that takes into account term-to-term similarity would also be an important direction.

An interesting extension to our model would be to simulate different types of users. A model could be created for each type of users and during the generation, the models could be used to create different query sequences that could then be merged into one.

The different models could consist of both different probility of actions and also different term selection policies. The timing of the queries is also an important part of buffer management, as many queries may be submitted at the same time from many different users. Time could be added to the generated sequences by maintaining statistics about query intervals from the original trace. By combining different models for different users, and generating queries from different models in the appropriate proportions and with the appropriate interval distributions, very complex situations could be simulated using the same simple methodology. The most difficult aspect, in fact, will be discovering and separating the different classes of users.

We therefore believe that our query generation methodology proposed in this thesis can serve as a solid foundation for query generation for large-scale query processing studies. Researchers at the large search engine companies could, for example, tune a query generation engine to match the characteristics of their users and open the generation software to the general academic, rather than the actual query traces.

Bibliography

Baeza-Yates, R. A., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Beitzel, S. M., Jensen, E. C., Chowdhury, A., Grossman, D., & Frieder, O. (2004). Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (p. 321 - 328). Sheffield, United Kingdom.

Berghreinsson, V. (2007). *Bætt afköst við gerð hermifyrirspurna* [Improved performance, during generation of simulated queries]. B.Sc. Thesis. Reykjavik University, Reykjavik, Iceland.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th World Wide Web conference (WWW7)* (p. 107 - 117). Brisbane, Australia.

Böðvarsson, E. T. (2006). *The WWG Software Library for Web Workload Generation.* B.Sc. Thesis. Reykjavik University, Reykjavik, Iceland.

Comscore.com. (2011). [Webpage]. `http://www.comscore.com/Press_Events/Press_Releases/2011/3/comScore_Releases_February_2011_U.S._Search_Engine_Rankings`.

Fidel, R. (1991). Searchers' selection of search keys: III. Searching styles. *Journal of the American Society of Information Science (JASIS)*, *42*(7), 515–527.

Jansen, B. J. (2006). Search log analysis: What it is, what's been done, how to do it. *Library & Information Science Research*, *28*(3), 407 - 432.

Jansen, B. J., Booth, D. L., & Spink, A. (2008). Determining the informational, navigational, and transactional intent of web queries. *Information Processing & Management*, *44*(3), 1251 - 1266.

Jansen, B. J., & Spink, A. (2006). How are we searching the world wide web?: a comparison of nine search engine transaction logs. *Information Processing & Management: an International Journal - Special issue: Formal methods for information retrieval archive*, *42*(1), 248–263.

Jansen, B. J., Spink, A., & Saracevic, T. (2000). Real life, real users, and real needs: A study and analysis of user queries on the web. *Information Processing & Management*, *36*(2), 207–227.

Kleinberg, J. (1998). Authoritative sources in a hyperlinked environment. In *Proceedings of the ninth annual ACM-SIAM symposium on discrete algorithms* (p. 668 - 677). San Francisco, CA, USA.

Koenemann, J., Quatrain, R., Cool, C., & Belkin, N. (1994). New tools and old habits: The interactive searching behavior of expert online searchers using INQUERY. In *Proceedings of the third TExt Retrieval Conference (TREC-3).* Gaithersburg, MD, USA.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, *1*(4), 309–317.

Ozmutlu, H. C., Spink, A., & Ozmutlu, S. (2002). Analysis of large data logs: an application of poisson sampling on excite web queries. *Information Processing & Management*, *38*(4), 473–490.

Rose, D. E., & Levinson, D. (2004). Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web (WWW'04)* (pp. 13–19). New York, NY, USA.

Silverstein, C., Marais, H., Henzinger, M., & Moricz, M. (1999). Analysis of a very large web search engine query log. *SIGIR Forum*, *33*(1), 6–12.

Singhal, A. (2001). Modern information retrieval: a brief overview. *Bulletin of the IEEE Computer Society Technical Commitee on Data Engineering*, *24*(4), 35–43.

Spink, A., Jansen, B. J., Wolfram, D., & Saracevic, T. (2002). From e-sex to e-commerce: Web search changes. *Computer*, *35*(3), 107-109.

Spink, A., Wolfram, D., Jansen, B. J., & Saracevic, T. (2001). Searching the Web: The Public and Their Queries. *Journal of the American Society for Information Science and Technology (JASIST)*, *52*(3), 226–234.

Tomasic, A., & Garcia-Molina, H. (1993). Query processing and inverted indices in shared-nothing document information retrieval systems. *The VLDB Journal*, *2*(3), 243–275.

Tryggvason, E. (2002). *Workload Generation.* B.Sc. Thesis. Reykjavik University, Reykjavik, Iceland.

Zhang, Y., Jansen, B. J., & Spink, A. (2009). Time series analysis of a Web search engine transaction log. *Information Processing & Management*, *45*(2), 230–245.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539