A Fast Method for Discovering Critical Edge Sequences in E-Commerce Catalogs*

Kaushik Dutta† Debra VanderMeer † Anindya Datta‡ Pinar Keskinocak§
Krithi Ramamritham¶

Abstract

Web sites allow the collection of vast amounts of *navigational data* – clickstreams of user traversals through the site. These massive data stores offer the tantalizing possibility of uncovering interesting patterns within the dataset. For e-businesses, always looking for an edge in the hyper-competitive online marketplace, the discovery of *Critical Edge Sequences* (CESs), which denote frequently traversed sequences in the catalog, is of significant interest. CESs can be used to improve site performance and site management, increase the effectiveness of advertising on the site, and gather additional knowledge of customer behavior patterns on the site.

Using web mining strategies to find CESs turns out to be expensive in both space and time. In this paper, we propose an approximate algorithm to compute the most popular traversal sequences between node pairs in a catalog, which are then used to discover CESs. Our method is both *fast* and *space efficient*, providing a vast reduction in both the run time and storage requirements, with minimum impact on accuracy.

*An early version of this work appeared in the ACM Conference on Electronic Commerce, 2001
†Florida International University
‡Walking Stick Solutions, Inc.
§Georgia Institute of Technology
¶Indian Institute of Technology, Mumbai

1 Introduction

Navigational data – clickstreams of user traversals through web sites – can be used to identify interesting customer behavior. Consider *Books.com*, a fictional online bookseller. *Books.com* tracks visitors as they navigate through the site, and subsequently analyzes the aggregated data. Assume that the analysis reveals the following: (a) if a visitor, in a session, navigates through *Orchid Gardening* (OG) and *Military History* (MH), he tends to eventually make a purchase of a *Music History* book; and (b) the shorter the time a user takes to traverse the site between OG and MH, the higher the likelihood of an eventual purchase. Suppose Books.com could appropriately recognize the popular traversal sequences between OG and MH, and that the analysis of these paths reveals that the most popular sequences are time-consuming to traverse (e.g., if server-side delays or large numbers of images lead to large download times, or if the number of clicks required is large). This would indicate possible areas of improvement on the site.

Finding popular sequences can be useful in several other scenarios, for example in predicting traversal behavior for caching, or in providing preferential resource allocation for heavily-traversed sequences on the site. For example, if the path from OG to MH is traversed often, upon noticing that a user has traversed the OG node, we can prefetch and cache business rules related to MH recommendation content. Then, once the user reaches MH, the relevant recommendations are available from cache, removing the need to query the recommendation engine and reducing the time required to serve pages to the user.

Analysis of these sequences can also reveal valuable patterns that can aid marketing on the site, e.g., for delivering advertisements, e-coupons, or running promotional campaigns [7]. Consider the problem faced by a marketing manager at an online toy store in developing an adaptive promotional campaign during the holiday season. During this period, the site's sales are greatly impacted by fads and trends – toys become popular virtually overnight [30]. If he can place his targeted advertising and promotional material appropriately on the site while the trend is still growing, he can greatly enhance sales for a trendy toy. If, however, he must wait several days to identify popular traversal sequences, he may well miss the peak of the trend, leaving significant revenue opportunities on the table.

To motivate the specific problem addressed in this paper, we return to the example of *Books.com*.

Suppose that an analysis of the popular traversal sequences between OG and MH reveals two distinct sequences:

1. OG → *Home Improvement* → **Anthropology** → **Medicine** → **Mythology** → *Buddhist Literature* → MH
2. OG → *Computer Science* → **Anthropology** → **Medicine** → **Mythology** → *Woodworking* → MH

It is interesting to note that the most popular traversal sequences between OG and MH share a common subsequence, i.e., *Anthropology* → *Medicine* → *Mythology*, indicating that users frequently traverse this subsequence when navigating from OG to MH (regardless of which exact traversal sequence they may have followed). We call such edge sequences *Critical Edge Sequences* (CESs). The focus of this paper is the *fast identification of critical traversal patterns at a web site*. Our work with a major clicks-and-mortar, where the extremely rapid discovery of such CESs is important, motivated this research. Web-sites need to unearth and react to information about CESs rapidly, in matters of minutes and hours as opposed to days or weeks.

The size of traversal data on web sites is non-trivial. Large, high-traffic web sites can generate daily session logs in the terabyte range. While several existing approaches, (particularly sequence mining techniques), can be adapted to find CESs, these algorithms require at least one full scan of the log data, resulting in running times on the order of multiple days or weeks, even for computationally efficient (i.e., log-linear or linear) algorithms running on enterprise class servers.

Therefore, we propose *efficient and approximate* methods for CES discovery. Our approach to dealing with extremely large input datasets is to gather traversal data in *aggregated* form (rather than using web/application server logs), i.e., we use *approximate data* and design *approximate algorithms* for CES discovery, which leads to *approximate results*. Since the output is approximate, the *quality* of the solutions is a concern. To mitigate this concern, we demonstrate that the approximation technique we propose allows us to accurately discover most CESs. Moreover, we provide guidelines for maximizing expected accuracy given a set of input parameters.

In summary, the contribution of this paper is the postulation of practical, applicable solutions to the problem of fast CES discovery. Our case study results show accuracy (i.e., number of CESs correctly found as compared to results generated on full traversal datasets) on the order of 90% obtained in 15 minutes; the corresponding results using full data required 18 hours to obtain.

The remainder of this paper is organized as follows. Section 2 describes our model and defines the CES discovery problem. Section 3 describes our approach to CES discovery and contributions.

Section 4 discusses related work. Section 5 presents our data approximation method, including a description of how the input is prepared, while Section 6 describes our approximate traversal sequence-finding methods. Section 7 describes CES discovery methods, based on the traversal sequences found using the methods described in Section 6. Section 8 presents a case study, demonstrating the reduction in running time and storage requirements as compared to the full-data case. Section 9 presents our experimental results, showing the accuracy of our methods. Section 10 concludes the paper.

2 E-commerce Site Model and Problem Description

We model a site as a graph $G = (V, E)$ similar to [17] and [13], where each node $v \in V$ is assigned a unique node-id and represents a particular concept in the e-commerce site, e.g., a product or a category of products in a hierarchically organized site, and each edge $e = (u, v) \in E$ represents the links on the site, i.e., the possible navigation from node u to node v on the site (where the graph may be cyclic or acyclic). In this model, a customer's traversal through the site can be described as a *clickstream*, i.e., a sequence of nodes the user visited, and the order in which he visited them.

Symbol	Description
$G = (V, E)$	graph with V vertices and E edges
N_i	node in the site graph, identified by node-id i
$\mathcal{Z}$	set of possible unique traversal sequences between a start node and an end node
$s \in \mathcal{S}$	subsequence in the set of unique subsequences computable from $\mathcal{Z}$
$r \in \mathcal{R}$	value in the set of traversal frequency values for edge subsequences in $\mathcal{S}$
T	threshold frequency, greater than which an edge sequence is deemed critical
f	maximum fanout in a site graph
l	maximum traversal length of a user session
d	depth of previous traversal considered
$\mathcal{PV}$	set of vector of probability values associated with traversed edges in G
c	cardinality of $\mathcal{PV}$
$\langle N_x, N_y, N_z \rangle$	a traversable edge sequence in G
$\mathcal{P}(\langle N_i, N_j \rangle)$	an element of $\mathcal{PV}$, associated with a particular traversed edge $\langle N_i, N_j \rangle$ in G
$\mathcal{P}_d(\langle N_i, N_j \rangle)$	a vector of traversal probabilities, where $\lvert\mathcal{P}_d(\langle N_i, N_j \rangle)\rvert$ is the number of d-length paths leading to N_i.
$\mathcal{P}(\langle N_i, N_j \rangle \lvert \langle N_p, N_q, N_i \rangle)$	probability that a user will traverse the edge $\langle N_i, N_j \rangle$, given that he arrives at N_i by traversing the sequence $\langle N_p, N_q, N_i \rangle$
k	number of MPESs to be found
C	count of nodes in a potential CES in N-gram analysis
$\rho_d(\langle N_i, N_j \rangle)$	single value representing the path $\langle N_i \rightsquigarrow N_j \rangle$
$\mathcal{M}$	the set of k MPESs returned
ζ	the complete set of subsequences in $\mathcal{M}$
m	the number of subsequences in $\mathcal{M}$
ϵ	the fraction of CESs found in error
δ	the fraction of CESs missed

Table 1: Summary of Notation

Following [13], our model separates the abstract site content representation, which corresponds to the set of nodes and node-ids described above, from the concrete page delivered to a user. Essentially, we assume that a user requests content associated with a particular node-id, but may receive additional content associated with multiple other nodes in the site graph in the delivered page, allowing limitless combinations of site content to be generated. For example, a user who requests information on the book "Gone With The Wind" may also receive links to other historical fiction titles based on results retrieved from a recommendation engine. This type of model reflects the current practice in Web content delivery, namely dynamic page generation, where the pages served to a user are generated on the fly, based on the site's business logic and the user's state at run-time.

The notation used throughout the remainder of the paper is summarized in Table 1.

Having discussed our model, we now formally describe the problem of discovering CESs.

Consider a Web site graph $G = (V, E)$. Suppose we are given a start node $N_a \in V$, and an end node $N_b \in V$, such that N_b is reachable from N_a. Let $\mathcal{Z}$ denote the set of (possibly partially overlapping) unique traversal sequences (paths) in G from N_a to N_b. Let $\mathcal{J}$ denote the binary relation $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ denotes the set of unique subsequences computable from any element in $\mathcal{Z}$, $\mathcal{R}$ is a set of frequency values, and (s_i, r_i) ($s_i \in \mathcal{S}$ and $r_i \in \mathcal{F}$) denotes that the sequence s_i was traversed r_i times. We refer to s_i as an "edge sequence" in G and r_i as the "frequency of traversal" of edge sequence s_i. A CES of N_a and N_b in G is any edge sequence $s_i \in \mathcal{S}$ such that $r_i \geq T$, where T is a configurable, pre-determined *threshold frequency*.

The goal of our approach is to identify the longest possible CESs that satisfy the frequency criteria T. By definition, each subsequence of a CES is itself a CES. In our result set, however, we do not need to enumerate each subsequence-based CES. For example, if the sequences $\langle N_1, N_2, N_3 \rangle$ and $\langle N_3, N_4, N_5 \rangle$ both meet the frequency criteria T, our algorithm would return the sequence $\langle N_1, N_2, N_3, N_4, N_5 \rangle$, rather than the two shorter sequences.

Example: In Figure 1, the sequences marked M_a, M_b, and M_c represent frequently traversed paths between OG and MH. In this example, each path has been traversed 20 times. We consider subsequences – potential CESs – of three nodes each to keep the discussion brief, yet still provide insight into the CES definition.

From Figure 1, we can find $\mathcal{Z}$:

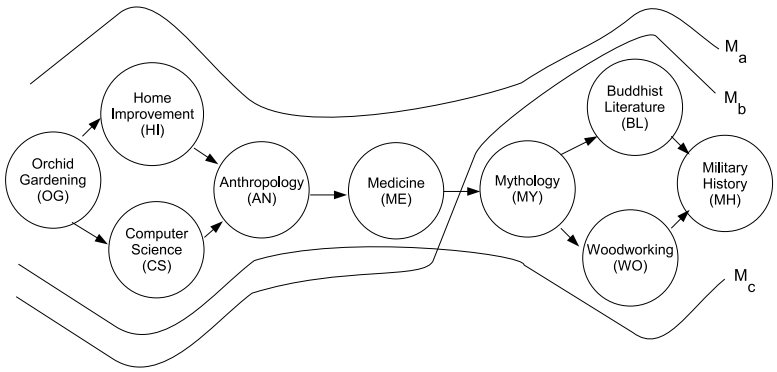

Figure 1: Example Traversal Graph from the Books.com site graph

$$\begin{aligned} \mathcal{Z} &= \{\langle OG, HI, AN, ME, MY, BL, MH\rangle, \\ &\quad \langle OG, CS, AN, ME, MY, BL, MH\rangle, \\ &\quad \langle OG, CS, AN, ME, MY, WO, MH\rangle\} = \{M_a, M_b, M_c\} \end{aligned}$$

In $\mathcal{S}$, there are nine edge sequences consisting of 3 nodes as shown in Table 2. For $T \geq 60$, only the sequence $\langle AN, ME, MY\rangle$ qualifies as a CES. For $T \geq 40$, sequences $\langle OG, CS, AN\rangle$, $\langle CS, AN, ME\rangle$, $\langle ME, MY, BL\rangle$, and $\langle MY, BL, MH\rangle$ would qualify as well.

Traversal Frequency f	**Subsequence** s
20	$\langle OG, HI, AN\rangle$,$\langle HI, AN, ME\rangle$,$\langle ME, MY, WO\rangle$,$\langle MY, WO, MH\rangle$
40	$\langle OG, CS, AN\rangle$, $\langle CS, AN, ME\rangle$, $\langle ME, MY, BL\rangle$, $\langle MY, BL, MH\rangle$
60	$\langle AN, ME, MY\rangle$

Table 2: Subsequences and traversal frequencies in example CES analysis

The CES discovery problem can be formally stated: *given a web site graph G and a mechanism for observing navigation over G, design algorithms to rapidly, efficiently, and accurately discover CESs, given specific start and end nodes.* Our approach can easily be extended to find all CESs in the e-commerce catalog irrespective of specific start and end points.

3 An Overview of Our Approach to Finding Critical Edge Sequences

To discover CESs between the node pair (N_a, N_b) we follow a 3-step approach. (1) *observe user traversals between N_a and N_b and accumulate traversal data*; (2) *based on the data collected in step (1), discover the Most Popular Edge (traversal) Sequences (MPESs) between N_a and N_b*; and (3) *based on the MPESs discovered in step (2), identify CESs.*

3.1 Accumulating traversal data

Due to the prohibitively large data volumes in web logs, analyzing the full log data and applying existing web mining algorithms to exactly identify the MPESs on a site is impractical. Consider the clicks-and-mortar site where we tested our methods. On average, this site experiences 10 million unique user sessions in a day. Each user requests, on average, 10 pages on the site. A log entry on this site consists of some 300 bytes of data, and each user click, on average, generates 15 log entries (since each embedded object requested from the site, e.g., image files, is logged separately). In this case, the size of the log file generated *per day* is $10,000,000 \times 10 \times 300 \times 15$ bytes or 0.45 TB. Over the course of a single year, the data volume grows to almost 164 TB. There exist techniques, such as those described in [14], by which it is possible to store only the meaningful clickstreams (by deleting the web log records of embedded objects and by representing each page by a unique id corresponding to the e-catalog). Typically, such filtering will reduce the daily log file size to 5% of the original size [14], resulting in a daily log size of 22.5 GB and yearly log size of at least 8 TB for this site. Many sites (e.g., Yahoo! [31], which serves more than 600 million pages per day), produce log sizes orders of magnitude larger than those of our clicks-and-mortar site.

These data sets are still too large for rapid analysis. Therefore, we aggregate traversal data as we gather it. Our proposed data-gathering technique does not maintain a record of each traversal of interest, but rather maintains *counts* of such traversals aggregated across all users, which significantly reduces the overall data size over filtering techniques. For example, the aggregated data for the optimal setting for our algorithm produces a reduced size of 45.4 MB as compared to the corresponding filtered log size of 4 GB for 3 hours traversal data. Due to aggregation, this percentage drops further over time. In case of large sites like Yahoo, the aggregated data size for a year would likely be in the low-GB range. For the e-catalog site that motivated this paper, the aggregated data size for a year's worth of data would be a few hundred MB.

To aggregate the traversal data, we store *multiple counts of traversals through each node, based on a limited number of nodes traversed to reach it* (see Section 5 for details). This results in *approximated* traversal data, with significant reductions in storage requirements over the full-storage case (see the case study in Section 8).

3.2 Finding the Most Popular Edge Sequences

Based on the data accumulated using our data-gathering strategy, we next seek to identify the MPESs – the most frequently traversed sequences between a pair of nodes. (With reference to our example in Section 5, the traversal paths M_a, M_b, and M_c represent MPESs between the nodes OG and MH.) We propose algorithms for this purpose (see Section 6 for details), inspired by known path traversal algorithms, designed to provide *accurate* solutions *quickly*. The output is a set of k MPESs, where k is a configurable number of most popular edge sequences to be found.

3.3 Finding Critical Edge Sequences from MPESs

Based on the MPESs found, we need to find the actual CESs, which are the subsequences of MPESs (and, potentially, actual MPESs) that meet the frequency threshold criteria T. To find the CESs, we need to consider the overlapping subsequences across the set of MPESs (see Section 7 for details). Though we specifically concentrate on finding CESs between two nodes, our approach can easily be extended to find CESs between multiple nodes.

4 Related Work

The problem of finding the MPESs, the most difficult part of the CES discovery process, maps nicely to sequence mining discovery [5, 11, 8, 22, 24] which is a standard approach for finding sequential patterns in a dataset. *Web usage mining* (e.g., [8, 22, 26, 18, 23]) is a sub-area within the area of sequence mining particularly applicable to the MPES-discovery problem. Much work on algorithms for web usage mining (e.g., [15, 21, 14, 27, 26, 32]) is based on data mining [1, 2, 4, 19] and sequential mining in particular [3, 29]. Virtually all approaches in area take filtered web logs (i.e., full traversal data) as input, and output exact sequential patterns matching some criteria. For example, [18] finds all repeating subsequences meeting a minimum frequency threshold, while [23] suggests a document prefetching scheme based on a user's next predicted action on a site.

There exists other research that applies sequence mining techniques in mining web logs, some exploiting multi-threaded and multi-processor systems. All these approaches require at least one full scan of the log data, resulting in long run times for large log datasets. For example, SPADE [21] requires three full scans of the log data. Both webSPADE [14] and ASIPATH [15] are parallel-

algorithm variations of SPADE developed for web logs, and require one full scan of the log data and multi-processor servers. As we demonstrate in Section 8, compared to these algorithms, our approach gives near real-time performance on a standard single processor server.

In [20], Kum, et. al., propose an approximate method for sequence mining. However, their approach considers a different, "looser", definition of a sequence than we do. Specifically, if there exists a sequence $\langle N_1, N_2, N_3, N_4, N_5\rangle$, [20] will consider $\langle N_1, N_3, N_4\rangle$ and $\langle N_1, N_2, N_4\rangle$ as valid subsequences. By this definition of a subsequence, most of the subsequences used in analysis will likely not have occurred consecutively in the original sequences. In contrast, mining critical edge sequences requires such consecutiveness, otherwise the results may yield non-existent paths on the web. For this reason, we cannot directly apply their work. *The difficulty of applying existing methods to our problem was the primary motivator in designing an approximate solution for MPES discovery.*

5 Gathering Approximate Data

For efficient CES analysis, it is impractical to store and manipulate large volumes of data. Therefore we develop a data approximation strategy which results in a significantly smaller input data set.

One simple approach to approximate data storage is to store only the edges in the graph (i.e., paths with two nodes, or 2-length paths). However, the specific path a customer follows before coming to a particular node usually impacts which path he will follow leaving the node. Storing only 2-length paths does not allow us to capture such information. An intermediate solution is to store information about d-length paths (i.e., paths with d edges and d+1 nodes) that lead to a particular node, and their frequencies. For example, in Figure 2 we have a total of 7 paths that pass through node N_2 and 3 3-length paths arriving at node N_2. We have the following count on the traversals of these 3-length paths: 3 customers traversed $\langle N_3, N_1, N_2\rangle$, 3 customers traversed $\langle N_4, N_1, N_2\rangle$, and 1 customer traversed $\langle N_5, N_1, N_2\rangle$.

We want to store the probability that a customer will traverse a particular edge after arriving at node N_2, given that the customer came to N_2 following a particular 3-length path. For example, the probability that a customer will traverse $\langle N_2, N_{11}\rangle$ given that he arrived at N_2 through $\langle N_3, N_1, N_2\rangle$ is $\mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_3, N_1, N_2\rangle) = 0.67$, since two out of three paths that arrive at N_2 through $\langle N_3, N_1, N_2\rangle$ continue to N_{11}. Similarly, the probability that a customer will traverse $\langle N_2, N_{11}\rangle$ given that he

arrived at N_2 through $\langle N_4, N_1, N_2\rangle$ is $\mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_4, N_1, N_2\rangle) = 0.67$. Finally, the probability that a customer will traverse $\langle N_2, N_{11}\rangle$ given that he arrived at N_2 through $\langle N_5, N_1, N_2\rangle$ is 0.

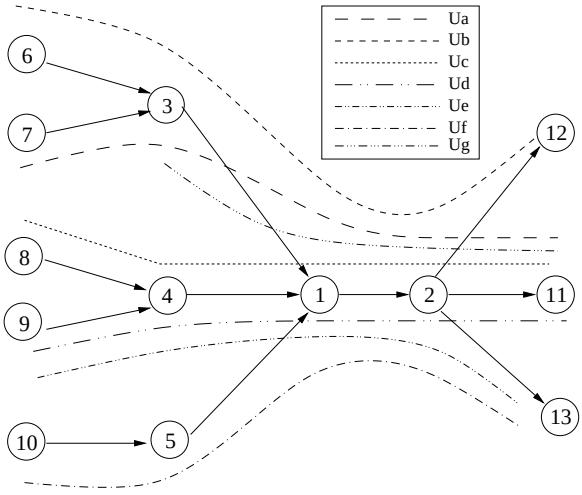

Figure 2: Example Traversal Graph

More formally, our approximate input data, in this context, may be thought of as a binary relation $(E, \mathcal{PV})$, with tuples of the form (e_i, pv_i) where e_i is a traversed edge in G, i.e., $e_i \in E$ and pv_i is a probability vector $\langle p_1, p_2, \ldots, p_c\rangle$, $p_i \in \mathcal{PV}$. The cardinality of $\mathcal{PV}$, i.e., the value of c, is the number of possible d-length traversal paths in G leading into e_i, and p_j denotes the probability that a visitor in G will traverse the edge e_i, given that he has arrived at the start node of e_i by traversing the j^{th} d-length path incident on this start node.

Gathering and aggregating input data consists of two distinct steps: (a) session information is accumulated while visitors are traversing the site (described in detail in the Appendix); and (b) probability vectors are computed in batch mode from the collected session information obtained in step (a). New clickstreams are periodically integrated via batch mode processing into the aggregated traversal dataset, i.e., *the pre-processing step takes place in small increments over time; the algorithm never considers the full traversal data as input.*

In general, for some $d > 1$, say, $d = 3$, we consider dependence up to 3 nodes (2 edges), i.e., we have information about sequences of length 3, and assume that the probability of traversing an edge is dependent on only the three most recently traversed nodes; all previous traversals are assumed to be independent. When $d = l$, where l is the maximum user session length, we have complete information about the sessions. Another way of looking at *depth* d is to model a user's

future traversal behavior as a Markov chain, where $d = 1$ corresponds to a first-order Markov Model, and $d = i$ corresponds to an $(i)^{th}$ order Markov Model.

Clearly, the cost and accuracy of the result is a function of d: larger d values correspond to more detailed information about sessions, larger storage requirements and more accurate results. We are interested in exploring the cost/accuracy tradeoff of varying d (the only tunable parameter in our approach). We address two questions: (a) *to what degree does this independence assumption affect accuracy?* (b) *how does accuracy scale, i.e., does increasing depth provide marginal returns?* We explore these questions in detail in Section 9.2.2, in the context of our experimental results.

6 Algorithmic Approximation

We present the *Configurable-Depth Algorithm* (CDA) for finding MPESs, given varying degrees of information (varying values of d) about user traversals and hence varying dependence assumptions regarding prior traversal. CDA is a variant of Dijkstra's Single-Source Shortest-Path (SSSP) algorithm[1], Shier's Single-Source k Shortest-Paths algorithm [25] or Eppstein's k Shortest-Paths algorithm [16]. The similarities between the problems underlying both SSSP and CDA are clear: both consider paths through a graph and seek to find the "best" paths – SSSP considers "best" to be the least sum of edge-weights, while "best" for CDA is the path with the highest probability of traversal.

The differences between CDA and Shier's k-SSSP lie in the values and functions used in finding the "best" paths. Specifically, CDA varies from Shier's k-SSSP in the following ways: (a) the meaning of the edge weights; (b) the fitness function for a path; and (c) the function used to compare paths to one another. We discuss each of these differences in turn.

6.1 Edge Weight in CDA

Shier's k-SSSP uses a positive integer value, representing the length (weight) of the edge, whereas CDA uses the *probability of traversal of an edge* (see Section 5) as the weight of the edge. For $d = 1$, this probability is a scalar value, whereas for $d > 1$, we have a vector of probabilities for traversing a particular edge, one probability value for each unique path of length d leading up to the source node of the edge.

[1]We also considered Floyd's All-Pairs Shortest Path algorithm, but found that it would compute many more sets of MPESs than needed, since most Web sites have only a small set of high-frequency start nodes.

6.2 Path Weight Function in CDA

In Shier's k-SSSP, the weight of a path is described as the sum of edge weights of that path, whereas in CDA, the path weight is the product of the probabilities of the edge weights in the path. For $d = 1$, we can use scalar multiplication. For $d > 1$, we define a path weight function, CDA-MULT, which handles vectors of probability values for each edge.

We define a path probability $\mathcal{P}(\langle N_1, N_2, ..., N_k\rangle)$ for path $\langle N_1, N_2, ..., N_k\rangle$ in terms of the edge probabilities $\mathcal{P}_d(\langle N_i, N_j\rangle)$, for edges $(N_i, N_j) \in E$ as the vector multiplication of probabilities of edges on a path. Specifically, we define the multiplication of edge probability vectors $\mathcal{P}_d(\langle N_i, N_j\rangle) = \mathcal{P}_d(\langle N_i, N_q\rangle) \times \mathcal{P}_d(\langle N_q, N_j\rangle)$, where N_q is an intermediate node for path $N_i \rightsquigarrow N_j$, as follows, given that the path $\langle N'_1, N'_2, ..., N'_{(d-1)}\rangle$ is on the path $\langle N_1, N_2, ..., N_{(d-1)}, N_i \rightsquigarrow N_q\rangle$ and $N'_{(d-1)}$ is the node just before node N_q on the path $N_i \rightsquigarrow N_q\rangle$:

$$
\begin{array}{rc}
\text{if} & \mathcal{P}(\langle N_i, N_j\rangle|\langle N_1, N_2, ..., N_{(d-1)}, N_i\rangle) \in \mathcal{P}(\langle N_i, N_j\rangle) \\
\text{and} & \mathcal{P}(\langle N_i, N_q\rangle|\langle N_1, N_2, ..., N_{(d-1)}, N_i\rangle) \in \mathcal{P}_d(\langle N_i, N_q\rangle) \\
\text{and} & \mathcal{P}(\langle N_q, N_j\rangle|\langle N'_1, N'_2, ..., N'_{(d-1)}, N_q\rangle) \in \mathcal{P}_d(\langle N_q, N_j\rangle) \\
\text{then} & \mathcal{P}(\langle N_i, N_j\rangle|\langle N_1, N_2, ..., N_{(d-1)}, N_i\rangle) = \\
\multicolumn{2}{c}{\langle \mathcal{P}(\langle N_i, N_q\rangle|\langle N_1, N_2, ..., N_{(d-1)}, N_i\rangle) \times \mathcal{P}(\langle N_q, N_j\rangle|\langle N'_1, N'_2, ..., N'_{(d-1)}, N_q\rangle)\rangle}
\end{array}
\tag{1}
$$

To clarify these notions, we consider the example of finding the probabilities of paths ending at $\langle N_2, N_{11}\rangle$ in Figure 2 with depth $d = 3$. Before calculating path probabilities, we need to first find the edge probabilities for $\mathcal{P}_3(\langle N_2, N_{11}\rangle)$, $\mathcal{P}_3(\langle N_1, N_2\rangle)$, and $\mathcal{P}_3(\langle N_1, N_{11}\rangle)$. For example, the edge probability vector for $\mathcal{P}_3(\langle N_2, N_{11}\rangle)$ is as follows:

$\mathcal{P}_3(\langle N_2, N_{11}\rangle) = \langle \mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_3, N_1, N_2\rangle), \mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_4, N_1, N_2\rangle), \mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_5, N_1, N_2\rangle)\rangle$

The edge probability vectors for $\mathcal{P}_3(\langle N_1, N_2\rangle)$, and $\mathcal{P}_3(\langle N_1, N_{11}\rangle)$ are generated similarly.

Using these edge probabilities and vector multiplication, we can find path probabilities. For example, we find the path probability:

$$
\mathcal{P}(\langle N_1, N_{11}\rangle|\langle N_7, N_3, N_1\rangle) \quad = \quad \mathcal{P}(\langle N_1, N_2\rangle|\langle N_7, N_3, N_1\rangle) \times \mathcal{P}(\langle N_2, N_{11}\rangle|\langle N_3, N_1, N_2\rangle) \tag{2}
$$

Path probabilities for $\mathcal{P}(\langle N_1, N_{11}\rangle|\langle N_6, N_3, N_1\rangle)$, $\mathcal{P}(\langle N_1, N_{11}\rangle|\langle N_8, N_4, N_1\rangle)$, $\mathcal{P}(\langle N_1, N_{11}\rangle|\langle N_9, N_4, N_1\rangle)$, and $\mathcal{P}(\langle N_1, N_{11}\rangle|\langle N_{10}, N_5, N_1\rangle)$ are computed in a similar fashion.

6.3 Path Comparison Function in CDA

In order to find the k most-traversed paths, we need a means of comparing paths to one another. In CDA, instead of the minimization function found in shortest paths algorithms, we use a maximization function. In order to do this, we define a function for path comparison, *CDA-MAX*. If we have two probability vectors $\mathcal{P}'$ and $\mathcal{P}''$, then we can define the *CDA-MAX*$(\mathcal{P}', \mathcal{P}'')$ as follows:

$$\textit{CDA-MAX}(\mathcal{P}', \mathcal{P}'') = \begin{cases} \mathcal{P}' & \textit{if } \sum_{\forall p' \in \mathcal{P}'} p' \geq \sum_{\forall p'' \in \mathcal{P}''} \mathcal{P}'' \\ \mathcal{P}'' & \textit{otherwise} \end{cases}$$

That is, between two paths for $N_i \rightsquigarrow N_j$ with their corresponding probability vectors $\mathcal{P}'$ and $\mathcal{P}''$ we choose the path which has the highest sum of probability values in the probability vector.

In order to use *CDA-MAX*, we must be able to represent a path as a single value, rather than as a vector of values. To achieve this, we define $\rho_d(\langle N_i, N_j \rangle)$ as the sum over the d-depth probabilities in the path $N_i \rightsquigarrow N_j$: $\rho_d(\langle N_i, N_j \rangle) = \sum_{\forall p \in \mathcal{P}(\langle N_i, N_j \rangle)} p$.

7 Finding Critical Edge Sequences

We next describe how to actually derive CESs from a set of k MPESs (the output of the CDA algorithm described in Section 6). Given a set of k unique paths with at most l nodes, our goal is to find all matching edge sequences across those k paths, where the length of a matching sequence can range from 1 to $l-1$ (a matching sequence of length l would indicate a non-unique set of k paths).

Fortunately, existing work in pattern matching can be applied to this problem. Specifically, we make use of work in *N-gram analysis* [9, 6] from the areas of Natural Language Processing (NLP) and Information Retrieval (IR).[2] N-gram analysis is used in NLP to find the Markovian probability of the n^{th} phoneme (i.e., sound) following a sequence of $n-1$ phonemes – one application of this is in speech recognition. IR uses a similar approach with words instead of characters to support text-retrieval applications.

Consider a scenario with input paths $\langle N_a, N_b, N_c, N_a, N_e \rangle$ and $\langle N_a, N_b, N_d, N_a, N_e \rangle$, i.e., $k = 2$. Figure 3 shows a tree representation of the unique sequences within these two paths. To find the frequency of particular sequences, we maintain a count C with each node in the tree. For example,

[2] We note here that other sub-string matching algorithms, e.g., those described in [10], can also be applied.

the sequence $\langle N_a, N_b \rangle$ appears twice in the input paths; thus, the count 2 this node ($C = 2$). *If a particular sequence has a value of $C \geq T$, that sequence is a CES.*

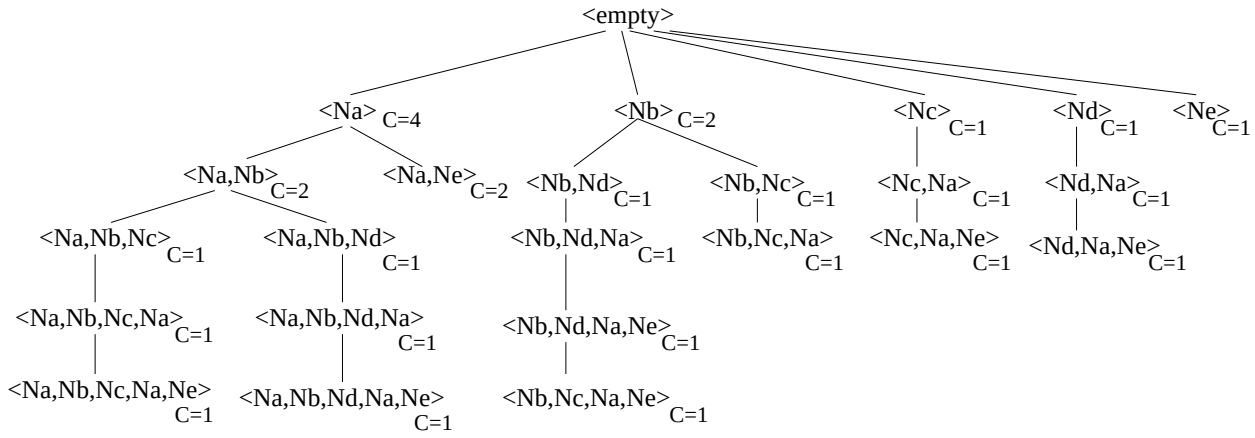

Figure 3: Example Tree Representation for CES data

We can traverse the tree to filter sequences based on their characteristics (e.g., frequency or length) using standard breadth-first or depth-first search strategies with pruning. For example, to find CESs with frequency $T \geq 2$ and length ≥ 2, we would prune branches of the tree upon discovering a tree node with $C = 1$, since lower nodes cannot have higher count values.

The values for k and l are small integers; typically, $k \leq 10$ and $l \leq 25$. Here, the size of the input dataset for CES analysis is extremely small in comparison to the input dataset for finding the k most traversed paths. Thus, rather than discussing the time and space complexity of N-gram analysis in depth, we refer the reader to [9].

8 CDA Case Study

We present a case study, based on data gathered from traversal logs over a 150,000-product e-commerce site, demonstrating the space and running time savings of CDA over full-data analysis.

For each of four time periods, we collected a *full-traversal* data set and two *approximate* data sets, resulting in 12 sets of data. The full-data set stores complete traversal sequences, while the other two data sets store approximate data for $d = 3$ and $d = 4$. We analyzed each data set for $k = 3$ using a Dell PowerEdge server with a single 3.0 GHz CPU, 2 GB RAM, and 120 GB of hard disk space.

To perform the full-data analysis, we chose the *Web Utilization Miner* (WUM) tool [27] because it incorporates a query processor, allowing the specification of arbitrary patterns of interest (e.g., arbitrary start and end points) in the query language MINT [28], and is known to implement efficient algorithms and provide fast running times [27]. In Section 9, we use WUM as a benchmark, comparing its performance to that of our approach in terms of accuracy.

To use WUM to find the MPESs between two specified nodes, N_a and N_b, on a web site, we provided web log data and the following MINT query to the WUM tool:

- **select** `t` **from** *node* as a b, **template** $a * b$ as t **where** $a.url =$ "N_a" and $b.url =$ "N_b" and $(b.support/a.support) > 0.1$

In this query, $a * b$ represents a regular expression pattern to be matched, where a is the start node and b the end node, while the $*$ matches any sequence of zero or more nodes between the start and end points. The restriction $b.support/a.support > 0.1$ restricts the number of MPESs from N_a to N_b, retaining only those sequences where at least 10% of the users who arrived at N_a traversed to N_b. Once we have the results from the MINT query, we can easily sort on the support restriction to find the k MPESs.

Time Period	Data Type	Space Required	Running Time
50 minutes	Full	1 GB	2 hours
50 minutes	Approximate, $d = 3$	10.1 MB	4.2 minutes
50 minutes	Approximate, $d = 4$	34 MB	12 minutes
1.5 hours	Full	2 GB	6 hours
1.5 hours	Approximate, $d = 3$	17.3 MB	5.3 minutes
1.5 hours	Approximate, $d = 4$	54.2 MB	18.2 minutes
2.5 hours	Full	3 GB	18 hours
2.5 hours	Approximate, $d = 3$	31.9 MB	9.2 minutes
2.5 hours	Approximate, $d = 4$	99.3 MB	33 minutes
3 hours	Full	4 GB	38 hours
3 hours	Approximate, $d = 3$	45.4 MB	15 minutes
3 hours	Approximate, $d = 4$	112.5 MB	45 minutes

Table 3: Case Study Comparing Space and Running Times for CDA and Full-data Analysis

Table 3 shows the time period each data set represents, as well as the storage and running times. Clearly, the approximated datasets result in significantly less storage and lower running times than the full-data set (at least two, in some cases three orders of magnitude improvement). We find that incrementing d from 3 to 4 has adds roughly a 3x increase in running time and storage requirements. We describe the variation in accuracy between these two cases in Section 9.2.

9 Accuracy Evaluation

To evaluate the accuracy of our algorithms, we compare the quality of the output CDA to that of WUM (where WUM produces fully accurate results because it considers the full data as input). We would like to determine to what extent varying the independence assumption (as encoded by d) affects accuracy. Next, we describe our accuracy metrics and the results of our experiments.

9.1 Accuracy Metrics

To measure the performance of our algorithms, we first describe how one can compare the two sets of k paths returned by WUM and CDA. Let $\mathcal{M}_{exact}$ and $\mathcal{M}_{approx}$ be the set of paths returned by WUM and CDA, respectively. By definition, $\mathcal{M}_{exact}$ represents the no-error case. Intuitively, the more the subpaths (i.e., CESs) of $\mathcal{M}_{approx}$ overlap with the subpaths of $\mathcal{M}_{exact}$, the more accurate the result of CDA. More formally, we define an m-edge sequence set $\zeta_m(\mathcal{M})$ of $\mathcal{M}$ as follows: $\zeta_m(P) = \{p : |p| = m$, *and* p *is a subsequence in path* P, *and* $P \in \mathcal{M}\}$. That is, $\zeta_m(P)$ is the set of all subpaths p of length m in $\mathcal{M}$.

Recall that CESs can be derived from a set of k popular paths by filtering subsequences of those k paths based on application-dependent criteria, such as a minimum probability threshold or frequency of occurrence in the k paths. For the purposes of error checking, however, we compare the *unfiltered* results in $\mathcal{M}_{approx}$ to those in $\mathcal{M}_{exact}$. Note that this places CDA at a distinct disadvantage: low-probability subsequences in WUM are likely sources of error in our approximate methods, yet these subsequences have the same priority as high-probability subsequences in our accuracy metrics. In this way, we are considering the worst-case scenario in terms of the accuracy of our methods.

We propose two error metrics, ϵ and δ, based roughly on the *recall* and *precision* measures from the Information Retrieval (IR) literature. δ measures the proportion of CESs found in error, in much the same way that *precision* measures error in a set of returned items. Intuitively, δ tells us what proportion of the results *should not have been returned, but were.*

ϵ measures correct CESs missed by the approximating algorithm. This is similar to *recall.* Intuitively, ϵ tells us what proportion of the correct result set *should have been returned, but weren't.*

We define ϵ as a similarity function:

$$\epsilon = \frac{\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{exact} - (\mathcal{M}_{exact} \bigcap \mathcal{M}_{approx}))|}{\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{exact})|}$$

The greater the overlap of edge sequences between $\mathcal{M}_{exact}$ and $\mathcal{M}_{approx}$, the closer the result is to the ideal solution, and, thus, the lower the error. The multiplication by l gives more weight to longer matching subpaths than to shorter ones. Intuitively, longer matching subpaths between the exact and approximate results indicate higher accuracy than shorter ones, since they indicate a stronger overall correlation between $\mathcal{M}_{exact}$ and $\mathcal{M}_{approx}$. If all edge sequences of $\mathcal{M}_{exact}$ match those of $\mathcal{M}_{approx}$, then all CESs have been found. This is the no-error case, where $\epsilon = 0$. If none of the edge sequences match, $\epsilon = 1$.

For example, consider the traversals shown in Figure 4. The path $\mathcal{M}_{exact} = \langle N_1, N_2, N_3, N_4, N_5 \rangle$ was returned by WUM, while the path $\mathcal{M}_{approx} = \langle N_1, N_2, N_3, N_6, N_7, N_8, N_4, N_5 \rangle$ was returned by CDA with depth $d = 1$. Here, $m = 7$, and the denominator, $\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{exact})| = 20$. To find the numerator, we count the number of CESs of each length from $\mathcal{M}_{exact}$ that do not appear in $\mathcal{M}_{approx}$: one CES of length 1, two CESs of length 2, two CESs of length 3, and one CESs of length 4 from $\mathcal{M}_{exact}$ fail to match CESs in $\mathcal{M}_{approx}$, making $\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{exact} - (\mathcal{M}_{exact} \bigcap \mathcal{M}_{approx}))| = 15$. By our expression, then, $\epsilon = 0.75$, i.e., the error is quite high. This occurs primarily because our example (for simplicity) considers only two path sequences leading from N_1 to N_5; presumably, in a realistic scenario considering a site graph with a large average fanout, there would be more than two paths leading from the start node to the end node.

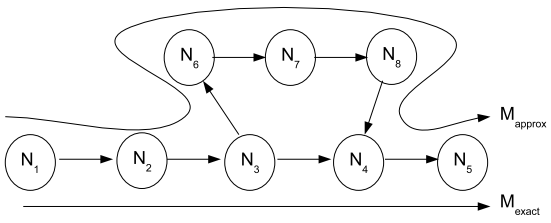

Figure 4: Similarity and Difference of Paths

While ϵ measures the similarity between $\mathcal{M}_{exact}$ and $\mathcal{M}_{approx}$, we also would like to measure the difference in the results. There is clearly quite a bit of difference between the two paths shown in Figure 4: the segment $N_3 \rightsquigarrow N_4$, $\mathcal{M}_{approx}$ contains only the sequence $\langle N_3, N_4 \rangle$, while $\mathcal{M}_{exact}$ contains

a much longer sequence, $\langle N_3, N_6, N_7, N_8, N_4 \rangle$. This difference is not necessarily dependent on ϵ, since ϵ considers only matching subsequences. We measure this difference with δ:

$$\delta = \frac{\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{approx} - (\mathcal{M}_{approx} \bigcap \mathcal{M}_{exact}))|}{\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{approx})|}$$

where $\zeta_m(\mathcal{M}_{approx} - (\mathcal{M}_{approx} \bigcap \mathcal{M}_{exact})) = \zeta_m(\mathcal{M}_{approx}) - (\zeta_m(\mathcal{M}_{approx} \bigcap \zeta_m(\mathcal{M}_{exact}))$. Ideally, $\delta = 0$, i.e., the paths in $\mathcal{M}_{exact}$ do not differ at all from the paths in $\mathcal{M}_{approx}$. Similarly, if the paths in $\mathcal{M}_{exact}$ and $\mathcal{M}_{approx}$ are disjoint, then $\delta = 1$. Due to multiplication by l, the longer the non-matching subpath, the greater the error, and the higher the δ value.

In the example in Figure 4, $m = 7$, and $\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{approx} - (\mathcal{M}_{exact} \bigcap \mathcal{M}_{exact}))| = 79$, since there are 4 CESs of length one in $\mathcal{M}_{approx}$ but not $\mathcal{M}_{exact}$, as well as 5 CESs of length two, 5 CESs of length three, 4 CESs of length four, 3 CESs of length five, 2 CESs of length six, and 1 CES of length seven. The denominator, $\sum_{l=1}^{m} l \times |\zeta_l(\mathcal{M}_{approx})| = 84$, since there are 84 possible CES matches. Thus, $\delta = 0.94$ in this example, which is quite high. This occurs for the same reasons as in the case of ϵ – the small number of paths compared.

9.2 Experimental Results

In this section, we report the results of our experiments. In particular, we are interested in how well the algorithms find the CESs given varying assumptions of dependence (depth d), as well as the CDA algorithm's sensitivity to fanout in the graph and traversal locality.

9.2.1 Baseline Experimental Results

Our baseline experiment considers the effect of the depth d on the accuracy of the results. While it is possible to show results for our baseline experiments using real data (as described in our case study in Section 8), we cannot easily modify the characteristics of a live site graph, as is needed for our sensitivity experiments. Thus, we generate a synthetic dataset closely resembling our real dataset for the purposes of our sensitivity experiments. The details of how we generated the synthetic data set are presented in the Appendix.

Figure 5 shows ϵ for WUM and CDA for a set of baseline parameters, *MeanFanout*= 8 and *TraversalLocality*= 0.2, where *MeanFanout* is the average number of outgoing edges from a node in

the graph and *TraversalLocality* represents the percentage of nodes that are frequently traversed (i.e., 0.20 for the 80/20 rule). The CDA curves are plotted for four different depths, $d = 1$, $d = 2$, $d = 3$, and $d = 4$. On the X-axis, k is varied between 1 and 5. We include WUM for completeness; since this method uses full information and always produces exact results, ϵ for WUM is always 0.

In Figure 5, the ϵ results portray an increasing trend with progressively diminishing slope. At low values of k, ϵ values are low, signifying that the algorithms do relatively well, but do progressively worse as k increases. The increasing trend is easy to explain – as fewer traversals are recorded over less popular paths (compared to more popular paths), our algorithms have a harder time recognizing frequently traversed sequences. Intuitively, it is easier to distinguish between the most popular and the 3^{rd} most popular paths, than between the 3^{rd} and 5^{th} most popular paths.

The diminishing rate of slope change occurs because the error values of any k^{th} most popular path computation already includes the error effects of the $k-1$ most popular paths. For instance, the ϵ value of the 5 most popular paths will automatically factor in the error effect of the 4 most popular paths, as these 4 paths will be included in the set of 5 most popular paths. Thus, the only additional error that is introduced is due to the k^{th} path. Clearly, as k increases, the error effect of the first $(k-1)$ paths increasingly dominate the overall error effect of all k paths.

It is interesting to note that all the CDA experiments yielded exact results for $k = 1$. In other words, when considering only the single most traversed path leading from the start node to the end node, CDA found the same most traversed path as WUM. This occurs because of the locality in the data; we observed similar results (also due to locality) in our real-data sets.

9.2.2 Looking for an Optimal Depth

An ineresting question to consider is whether or not there exists an optimal depth d, such that the cost/accuracy tradeoff is optimized. If such an optimal depth exists, knowing its value would be of significant practical importance for an analyst in discovering CESs.

In order to explore this issue, we have graphically shown how cost in running time and storage requirements and accuracy vary with depth (Figure 6). The x-axis shows varying values of d, while the y-axis represents normalized values for storage size, time, and ϵ. The metrics are normalized on a scale from 0 to 1 in order to aid in graphing. To normalize the values, we took the maximum experimental values for each metric (based on the baseline experiments for $k = 3$), and divided all

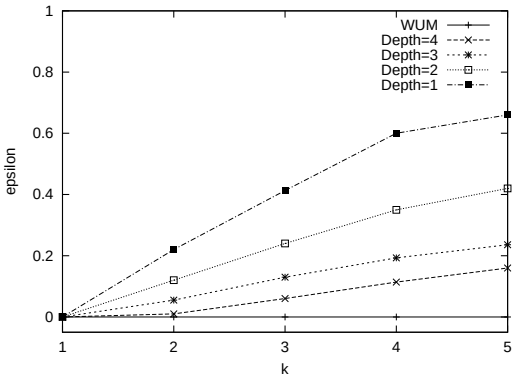

Figure 5: Accuracy ϵ as d varies over a synthetic dataset

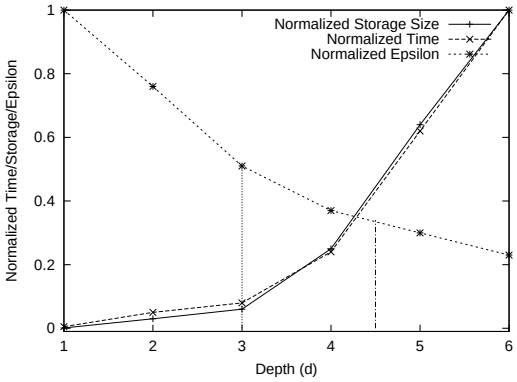

Figure 6: Cost of accuracy

experimental values for that metric by that maximum value. For instance, consider the normalized ϵ values. We obtained the maximum value of ϵ, i.e., 0.43, for the $d = 1$ case in Figure 5, and used it as the normalized maximum value in Figure 6. The normalized ϵ values for larger d values were obtained by dividing the corresponding ϵ values by the maximum values. The maximum storage size for the $d = 6$ case was 450 MB, and the corresponding running time was 3 hours. The remaining normalized storage sizes and running times in Figure 6 were obtained by dividing the actual values by these maximum values.

The first notable feature in Figure 6 is the similarity of the curves for time and storage size. These curves are exponential, i.e., as depth increases, the rate of increase of costs for storage and time increases. Intuitively, increasing d requires storing larger vectors of probability values for each edge (increasing storage costs), which in turn increases the time required to run CDA on the data. The requirements do not increase significantly up to $d = 3$. However, beyond $d = 3$, the rate of slope change begins to increase more quickly with each incrementally larger value of d.

The curve for ϵ, on the other hand, trends downward as d increases, the amount of previous path traversal information considered in calculating the probability of traversing an edge increases, which leads to decreasing error. The curve is asymptotic, i.e., ϵ decreases rapidly at low depth values, but the rate of decrease slows as d increases. At low d values, much accuracy can be gained by increasing d. However, as d approaches l, the maximum user session length, the rate of accuracy gain decreases, since each incremental increase of d provides only marginal gains in accuracy.

We see that up to $d = 3$, the incremental increases in storage and time requirements are minimal. However, error decreases dramatically in this range. This trends continue through $d = 4$, though the

rates of change of the slopes for storage, time and error are beginning to increase between $d = 3$ and $d = 4$. Beyond $d = 4$, the storage and time requirements increase significantly, without significant decreases in error. Thus, we can conclude that, in the experiments reported, the optimal value of depth d is between 3 and 4, the range in which error, time and storage are all within acceptable ranges. Since d must be an integer value, there is a tradeoff decision to be made: for higher accuracy, one can choose $d = 4$ at the cost of higher storage (roughly 3 times the storage requirement of $d = 3$, according to our case study in Section 8). However in most realistic scenarios we have seen, $d = 3$ gives the desired accuracy with lower storage requirements.

Parameter	**Baseline**	**Minimum**	**Maximum**
Average Fanout	8	3	15
Locality	0.2	0.1	0.3

Table 4: Parameters of Synthetic Data

Next, we discuss the results of our sensitivity experiments. We consider the sensitivity of ϵ to two characteristics of the site graph, fanout and traversal locality. For these experiments, we vary the parameters of the synthetic data as shown in Table 4.

9.2.3 Sensitivity to Fanout

Intuitively, one would expect lower (higher) fanouts to result in higher (lower) accuracy due to the smaller (larger) number of possible paths through the site. Figure 7A shows the results of experiments varying fanout f. The curve for Fanout=8 is the baseline curve with *TraversalLocality* of 0.2 and $d = 3$, shown in Figure 5. The curve for Fanout=3 is similar in shape to the baseline curve, but the overall rate of increase and value of error are much lower, because a lower fanout in the graph affords fewer possibilities for error. Similarly, the curve for Fanout=15 has a shape similar to that of the baseline curve, but with a much higher rate of increase of error and higher overall error values.

9.2.4 Sensitivity to Traversal Locality

Locality of reference refers to the magnitude of popularity of items in a dataset, in this case paths through an e-commerce site. Intuitively, one would expect increased (decreased) locality within a site to lead to greater (lower) accuracy. Figure 7B shows the results of experiments varying *TraversalLocality*. Here, the curve for Locality=0.2 is the same as the baseline curve with *MeanFanout* of 8 and $d = 3$, as shown in Figure 5. The curve for Locality=0.3 shows the same general shape

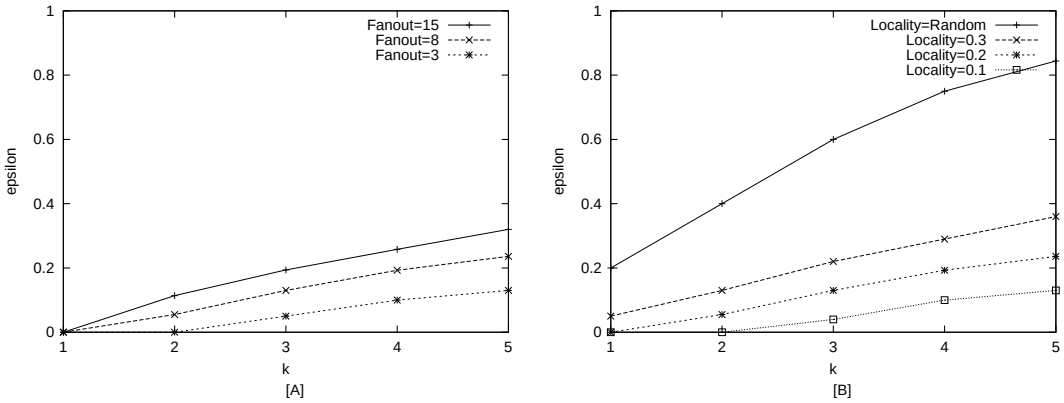

Figure 7: Sensitivity to [A] Fanout and [B] Locality

as the baseline curve, with higher overall error rates. This occurs because, for a fanout of 8, the difference between 70% of users choosing 30% of the links at a given decision point is not all that different from 80% of users choosing 20% at the same decision point.

One would expect, given the similarity between the Locality=0.3 and the baseline curves (Locality=0.2), that the curve for Locality=0.1 would also be similar. While the general shape follows the same trend, the overall ϵ values are much lower. This occurs because the Locality=0.1 experiments all found the 2^{nd} most popular path, thus reducing the cumulative error for $k > 2$ (see the discussion on cumulative error in Section 9.2.1).

The curve for Locality=Random shows the same general shape as the baseline curve as well, but with higher error rates than Locality=0.3 and the baseline case. The increased error rates occur because lower locality of reference leads to lower frequencies of traversal for the most traversed paths, which in turn leads to increased error rates in finding them.

9.3 Summary of Accuracy Evaluation

We now consider the accuracy results described above in light of the question: *Do approximate algorithms provide sufficient accuracy to merit their use?* Our experiments show an optimal depth of $d = 3$ or $d = 4$. For $d = 4$, $\epsilon \leq 10\%$ up to $k = 4$, i.e., our algorithms are 90% accurate for the four most-traversed paths. This kind of accuracy is more than sufficient for the uses of CES, e.g., caching, targeting advertising, and delivering e-coupons. In fact, 90% accuracy in a caching environment would allow for pre-fetching – while the cost of a prefetching error may be high, the probability of error is low.

10 Conclusion

In this paper, we presented approximate methods for finding the Critical Edge Sequences (CESs), i.e., frequently traversed sequences of edges on the paths leading from a source node N_a to a destination node N_b on a web site in a *fast*, *space efficient* and *accurate* manner.

Our CES analysis approach consists of three steps: (a) gathering an approximate traversal dataset based on a depth d; (b) finding the MPESs between two nodes using the CDA algorithm; and (c) finding the actual CESs from the MPESs using N-gram analysis.

We compare our approach in terms of accuracy, running time, and storage requirements to that of an exact CES-finding method based on the well-known sequential mining tool, WUM. We find that CDA provides multiple orders of magnitude improvements in runtime and storage requirements over the full-data approach.

We explore the accuracy of the CDA algorithm using synthetic datasets, using metrics based on the familiar IR notions of precision and recall. In particular, we are interested in how varying the independence assumption affects the accuracy of CDA, as well as CDA's sensitivity to fanout in the graph, traversal locality, and depth of conditional probability.

Based on our results, we gain the following insights: (a) as the value of k increases, the error in the i^{th} ($i <= k$) path in the result increases as well, i.e., the error for $k = 1$ is less than the error for $k = 2$, and so on; (b) as the depth d of conditional probability increases, the error in the result decreases; (c) there exists an optimal depth d, with the best possible tradeoff between cost and accuracy – in our tests, the optimal depth was between $d = 3$ and $d = 4$, at which we obtained results with accuracy up to 90%; (d) as the fanout f in the graph increases, the error in the result increases; and (e) as the locality of reference increases, the error in the result decreases. Our future plans include modifying our algorithms to support user-defined goals of frequency/rarity, path properties and the possibility of considering traversal sequences spanning multiple sites.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Databases*, pages 487–499, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering*, pages 3–14, March 1995.

[4] R.J. Bayardo and R. Agrawal. Mining the most interesting rules. In *Proceedings of the ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 145–154, 1999.

[5] J. Borges and M. Levene. Data mining of user navigation patterns. In *Proceedings of the Workshop on Web Usage Analysis and User Profiling*, 1999. http://www.acm.org/sigs/sigkdd/proceedings/webkdd99/toconline.htm.

[6] M.K. Brown, A. Kellner, and D Raggett. Stochastic language models (n-gram) specification(w3c working draft). http://www.w3c.org/TR/ngram-spec/, January 2001.

[7] P. Chatterjee, D.H. Hoffman, and T.P. Novak. Modeling the clickstream: Implications for web-based advertising efforts. *Marketing Science*, 22(4), Fall 2003.

[8] M-S Chen, J.S. Park, and P.S. Yu. Data mining for path traversal patterns in a web environment. In *Proc. of the Int'l Conf. on Distributed Computing Systems*, 1996.

[9] K.W. Church. Ngrams. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 1995.

[10] T.H. Cormen, C.E. Lieserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw Hill, 1998.

[11] C.R.Cunha and C.F.B. Jaccoud. Determining www user's next access and its application to pre-fetching. In *Proceedings of the IEEE Symposium on Computers and Communications*, pages 6–11, July 1997.

[12] A. Datta, K. Dutta, D. VanderMeer, K. Ramamritham, and S.B. Navathe:. An architecture to support scalable online personalization on the web. *VLDB Journal*, 10(1):104–117, 2001.

[13] A. Datta, D. VanderMeer, K. Ramamritham, and S. Navathe. Toward a comprehensive model of the content and structure of, and user interaction over, a web site. In *Proceedings of the VLDB Workshop on Technologies for E-Services*, Cairo, Egypt, September 2000.

[14] A. Demiriz. webspade: A parallel sequence mining algorithm to analyze web log data. In *Proceedings of the Second IEEE International Conference on Data Mining (ICDM, 2002)*, pages 755 – 758, 2002.

[15] A. Demiriz. Asipath: A simple path mining algorithm. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, pages 9 – 11, Nov 2004.

[16] D. Eppstein. Finding the k shortest paths. In *SIAM Journal on Computing*, volume 28, pages 652–673, 1998.

[17] D. Florescu, A. Levy, D. Suciu, and K. Yagoub. Optimization of run-time management of data intensive web sites. In *Proceedings of the VLDB Conference*, pages 627–638, September 1999.

[18] W. Gaul and L. Schmidt-Thieme. Mining web navigation path fragments. In *Proceedings of the WEBKDD Workshop*, August 2000.

[19] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Quantifiable data mining using ratio rules. *VLDB Journal*, pages 254–266, 2000.

[20] H. Kum, J. Pei, W. Wang, and D. Duncan. Approxmap: Approximate mining of consensus sequential patterns. In *Proceedings of the SIAM International Conference on Data Mining*, May 2003.

[21] M.J.Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42:31–60, Jan-Feb 2001.

[22] B. Mosbasher, N. Jain, E. Han, and J. Srivastava. Web mining: Pattern discovery from world wide web transactions. Technical Report 96-050, University of Minnesota, Dept. of Computer Science, Minneapolis, 1996.

[23] A. Nanopoulos, Dimitrios Katsaros, and Y. Manolopoulos. Effective prediction of web-user accesses, a data mining approach. In *Proceedings of the WEBKDD Workshop*, August 2001.

[24] J. Pitkow and P. Priolli. Mining longest repeating subsequences to predict world wide web surfing. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, October 1999.

[25] D.R. Shier. On algorithms for finding the k shortest paths in a network. *Networks*, 1979.

[26] M. Spiliopoulou, L.S. Faulstich, and K. Winkler. A data miner analyzing the navigaitional behavior of web users. In *International Conference of Workshop on Machine Learning in User Modelling*, 1999.

[27] M. Spiliopoulou and C. Pohle. Data mining for measuring and improving the success of web sites. *Journal of Data Mining and Knowledge Discovery,Special issue on applications of data mining to electronic commerce*, 5:85–114, January-April 2001.

[28] M. Spilioupoulou, L.C. Faulstich, P. Atzeni, A. Mendelzon, and G. Mecca. Wum: A tool for web utilization analysis. In *International Workshop on World Wide Web and Databases*, pages 184–203, 1998.

[29] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database Technology*, pages 3–17, March 1996.

[30] A. Throne. Why he who dies with the hottest toy wins. http://www.brandera.com/features/01/03/14/trendset.html, March 2002.

[31] Yahoo! Yahoo media relations: General corporate faq. http://docs.yahoo.com/docs/info/faq.html, March 2006.

[32] M.J. Zaki, N. Lesh, and M. Ogihara. Planmine: Sequence mining for plan failures. In *Proceedings of the Intl. Conference on Knowledge Discovery and Data Mining*, pages 369–373, 1998.

Appendix

Collecting Session Information

We gather traversal information for each user traversal on the site directly, without utilizing the web server log. For each user click, we store a node-id within the user's session state on the web site's application server infrastructure. We concatenate node-ids clicks from each user session together to form a clickstream of node-ids associated with a particular session (as described in [12]). For

example, in Figure 2, user U_a visits nodes N_7, N_3, N_1, N_2, and N_{11}. Thus, the session for U_a's visit can be represented as $\langle N_7, N_3, N_1, N_2, N_{11} \rangle$.

When a session ends (this point is typically configurable on the Web site or can be explicitly identified by user logoff for registered users), the session's clickstream is stored temporarily in a file, where each line of the file represents the traversal information from a single session (each line representing a different user). *This representation is significantly smaller than the web log representation of site activity.* For instance, in the web log representation, each click on any given node, e.g., N_3, would be an independent and full entry, containing a timestamp, URL, and the IP address of the request origination, among other information.

From the data in the temporary clickstream file, we extract the edge probability vectors by a simple two-step method: (a) we first extract all $(d+1)$-length traversal sequences; (b) we then load these subsequences and the frequency of traversal over these subsequences into an AGGREGATE_TRAVERSAL table. A fragment of this table is shown in Table 5 for depth $d = 3$, based on traversals shown in the graph fragment in Figure 2.

Node 1	**Node 2**	**Node** d	**Node** $(d+1)$	$Count_d$	$Count_{(d+1)}$	*Probability*
N_6	N_3	N_1	N_2	1	1	1.0
N_7	N_3	N_1	N_2	1	1	1.0
N_8	N_4	N_1	N_2	1	1	1.0
N_9	N_4	N_1	N_2	2	2	1.0
N_{10}	N_5	N_1	N_2	1	1	1.0
N_3	N_1	N_2	N_{11}	3	2	0.67
N_3	N_1	N_2	N_{12}	3	1	0.33
N_4	N_1	N_2	N_{11}	3	2	0.67
N_4	N_1	N_2	N_{13}	3	1	0.33
N_5	N_1	N_2	N_{13}	1	1	1.0

Table 5: Traversal Table

In each tuple of the AGGREGATE_TRAVERSAL table, we store a $(d+1)$-length sequence, as well as two counts associated with each $(d+1)$-length edge subsequence, and a probability value. $Count_d$ represents the number of traversals of the first d nodes in the sequence, while $Count_{(d+1)}$ represents the number of traversals of the entire $(d+1)$-length sequence. The *Probability* field denotes the probability that a user will traverse the edge represented by the values in the **Node** d and **Node** $(d+1)$ fields, given that he has arrived at the node stored in **Node** d by traversing the nodes stored in **Node 1** and **Node 2**. We can calculate this probability by dividing $Count_{(d+1)}$ by $Count_d$. For example, consider the first record in the AGGREGATE_TABLE fragment, shown in Table 5. This

tuple indicates the follwing three facts:

1. A single user has traversed the path $\langle N_6, N_3, N_1 \rangle$ (shown by a $Count_d$ value of 1);
2. A single user has traversed the path $\langle N_6, N_3, N_1, N_2 \rangle$ (shown by a $Count_{(d+1)}$ value of 1); and
3. If a user traverses the path $\langle N_6, N_3, N_1 \rangle$, then he will navigate to N_2 next, with probability 1.0 (shown as a *Probability* value of 1.0 in Table 5).

Generating the experimental synthetic data set

We model a web site as a graph $G = (V, E)$. In our experiments, G consists of $|V|$ nodes and $MeanFanout \times |V|$ edges, where *MeanFanout* is the average number of outgoing edges from a node in the graph. *NumUsers* users traverse G, each starting at a node chosen from a uniform distribution across all nodes in G. Note that this confers a significant disadvantage to our algorithms – in the real world, users begin traversing the site at a limited number of start nodes (e.g., the home page, and perhaps the first-level category pages). Had we used a limited set of starting nodes in these experiments, our accuracy results would be significantly better than those reported here.

Users traverse the site G for *SessionLength* clicks, and choose links according to a *TraversalLocality*. Here, we represent *TraversalLocality* by the percentage of nodes that are frequently traversed (i.e., 0.20 for the 80/20 rule). We implement *TraversalLocality* by assigning 20% of outgoing edges from a node a high probability of traversal, where the probability of traversal for each of these 20% of outgoing edges is divided equally to sum to 80%. The remaining 80% of outgoing edges have equal probabilities summing to 20%.

In order to generate the synthetic data, we observed characteristics of the real life catalog graph and visitor sessions as shown in Table 6 (where the average values are based on the site graph and the 50-minute full-data dataset described in Table 3). Based on these characteristics, we generated a synthetic graph, and simulated 300,000 visitor sessions using a Zipfian distribution based on the *TraversalLocality* probability assigned to each edge.

Parameter	**Average**	**Minimum**	**Maximum**
Fanout	8	2	10
SessionLength	15	4	20
Number of Users	300,000	-	-
Number of Nodes	150,000	-	-
Traversal Locality	0.2	-	-

Table 6: Characteristics of our dataset

We first validate the synthetic dataset versus the real-life dataset in terms of accuracy. Our work with a commercial site has provided access to a dataset of 300,000 user sessions across the site. We ran the exact method WUM and the approximate method CDA over the real world and synthetic data sets (Figures 8 and 5). In comparing the plots in Figures 8 and 5, we find that the accuracy for our approximate solution for the synthetically-generated data is similar to that of real life data. Thus, we conclude that our synthetic session log generation accurately emulates the real data as far as our approximate algorithms are concerned. We note here that the trends of δ curves (shown in Figure 8[B]) closely resemble those of the trends for ϵ curves in Figure 8[A], for very similar reasons. Thus, we report only ϵ results in this paper.

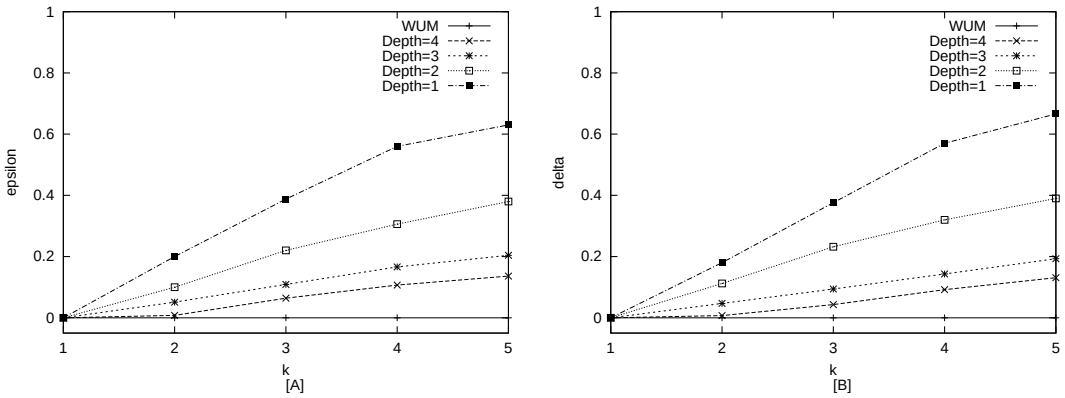

Figure 8: Accuracy [A] ϵ and [B] δ as d varies over a real-life dataset