

# SUSAX: Context-Specific Searching in XML Documents Using Sequence Alignment Techniques

Krunal Patel and Kajal T. Claypool

Department of Computer Science, University of Massachusetts - Lowell  
{kbpatel|kajal}@cs.uml.edu

## Abstract

Keyword searching while very successful in narrowing down the contents of the Web to the pertaining subset of information, has two primary drawbacks. First, the accuracy of the search is closely coupled with the choice of keywords. Second, keywords are limited in their expressibility. In particular, they fail to adequately capture the “contextual information” implicit in most searches done by users. In this paper we present an approach to efficiently address these drawbacks of keyword searching over XML documents. In particular, we present SUSAX a system for approximate contextual querying over XML documents wherein queries are represented as simple XPath. A key contribution of our work is the novel algorithm used to match the XPath-like query with similar paths in the repository. The algorithm is based on sequence alignment algorithms prevalent in life sciences domain for discovering the similarity between genome and protein sequences. In this paper, we show an adaptation of the sequence alignment algorithm for now discovering and cataloging the similarity between two paths.

## 1 Introduction

Over the past several years “google” has become part of the common phraseology used by people of all ages. Coined after the popular Google [7] search engine, the phrase is more generally representative of the *keyword searching* capabilities offered by search engines such as Google [7], Yahoo [11], HotBot [9], and MSN [10]. In all cases the user’s query, a single keyword or a set of multiple keywords (and’ed or or’ed), is matched *exactly* against an inverted index of keywords, and a ranked list of documents (by PageRank [7] for instance) containing the specified keyword are returned to the user.

However, while keyword searching has indeed been very successful in narrowing down the contents of the Web to the

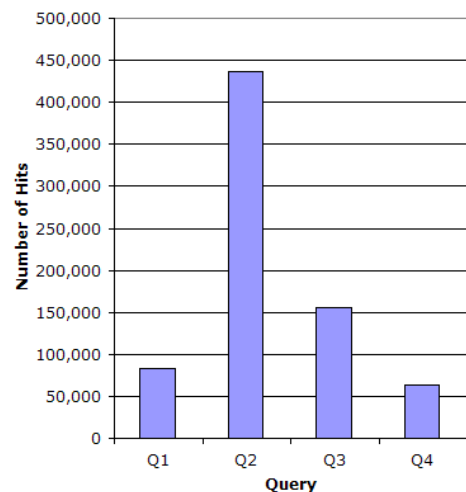


Figure 1. Keyword Search Using Google.

pertaining subset of information, it has two primary drawbacks. First, the accuracy of the search is tightly coupled to the choice of keywords. For example, to get all the *publications of Michael Stonebraker* we must choose the right keywords for conducting the search. Figure 1 shows the results obtained by using the keywords “published work Stonebraker”, “Stonebraker”, “articles Stonebraker” and “publications Stonebraker”. Searches with these keywords yielded different sets of results – in fact non-overlapping results in the first page. Only the keywords “publications Stonebraker” yielded the desired hits shown in Figure 2 in the first page of results. Second, keywords are limited in their expressibility. In particular, they fail to adequately capture the *contextual information* implicit in searches typically conducted by the users. For example, the query *all publications by Stonebraker in 1997* is at best represented by keywords “Stonebraker publications 1997” – a search whose first hit is completely irrelevant.

These drawbacks of keyword searching while

[Top hit]  
StreamBase Systems Mike Stonebraker Selected Publications  
StreamBase Systems Mike Stonebraker selected publications.  
www.streambase.com/www/about/publications\_stonebraker.html - 10k -

[9th hit]  
DBLP: Michael Stonebraker Michael Stonebraker. List of publications  
from the DBLP Bibliography Server - FAQ ... 170, Spyros Potamianos,  
Michael Stonebraker: The POSTGRES Rules System ...  
www.informatik.uni-trier.de/~ley/  
db/indices/a-tree/s/Stonebraker:Michael.html - 101k - Oct 26, 2005 -  
Cached - Similar pages

**Figure 2. The Top 2 Desired Hits Using Keywords.**

formidable to address in the general Web context, are however tractable for a significant chunk of the Web that comprises of semi-structured XML documents. In this paper, we present an approach that extends schema matching techniques [4, 2, 3, 6, 15, 5] to efficiently address the drawbacks of keyword searching over XML documents. In particular, we propose SUSAX – a system for approximate contextual querying over XML documents wherein queries are represented as simple XPath of the form `/publications/Stonebraker/1997`.

A key contribution of our work is the novel algorithm used to match the XPath-like query with similar paths in the repository. The algorithm is based on sequence alignment algorithms prevalent in life sciences domain for discovering the similarity between genome and protein sequences. We show how a commonly used sequence alignment algorithm, the Needleman-Wunsch algorithm [19], can be adapted for discovering and cataloging the similarity between two paths – providing a ranked set of results that takes both approximate matching and contextual information into account. Preliminary results indicate that for XML documents, approximate, path-augmented contextual searching yields better, more relevant results than keyword searching alone.

**Roadmap:** The rest of the paper is organized as follows. In Section 2 we briefly outline the Needleman-Wunsch algorithm. Section 3 presents the core of our approach and sets the context for the remainder of the paper. Section 4 describes the linguistic algorithm that we employ as part of our overall SUSAX algorithm, while Section 5 outlines the adaptation of the Needleman-Wunsch algorithm for searching and aligning simple XPath. In Section 7, we present some preliminary experimental results to show the feasibility of our approach. Section 8 reviews some relevant related work and we conclude in Section 9.

## 2 Background - Needleman Wunsch Algorithm

Pairwise sequence alignments are fundamental to similarity searches conducted in the life sciences domain. Two basic classes of alignment algorithms have been developed – global and local. Global alignment algorithms attempt to align sequences over their entire length, while local alignment algorithms concentrate on discovering and aligning only the conserved motifs. The global alignment algorithm developed by Needleman and Wunsch [19] is one of the more popular algorithms and is the foundation of many of the other algorithms proposed in the bioinformatics literature. In our work on path alignment, we adapt this global alignment algorithm to align and discover the similarity between simple XPath. In this section, we briefly outline the Needleman-Wunsch algorithm.

The Needleman-Wunsch (NW) algorithm takes two input sequences  $S_1$  and  $S_2$ , and generates an alignment together with a score as an output. Figure 3 shows the global alignment of two input sequences,  $S_1$  and  $S_2$  (shown at the top of the figure), calculated using the Needleman-Wunsch algorithm. The input sequences are represented as FASTA [12] sequences – a popular format for sequence representation. The output of the algorithm are the aligned sequences – sequences that have potentially been modified by the introduction of gaps – together with an overall score for the alignment.

```
Score = 33
S1: CTKQADCAEDECC
S2: CGRQASGRLCGNRLCC

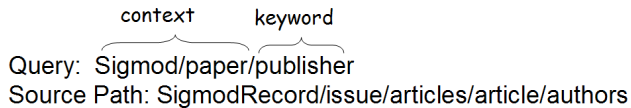
Alignment:

CTKQAD—CAEDECC
CGRQASGRLCGNRLCC
```

**Figure 3. An Example Alignment Calculated Using the Needleman-Wunsch Algorithm.**

The total score of an alignment is calculated based on the sum of terms for each pair of residues, provided by reputed similarity matrices such as BLOSUM and PAM that provide the affinity between two given residues<sup>1</sup>, plus the terms for each gap introduced by the algorithm. Typically, each exact match gets a positive score, each mismatch gets a penalty of  $-s$ , where  $-s$  is the similarity score retrieved from the substitution (BLOSUM or PAM) matrix, and each gap repre-

<sup>1</sup>For example, BLOSUM gives the similarity between A and C as 0, and between C and t C as 9.



**Figure 4. Query Representation in SUSAX.**

senting an insertion or deletion in one of the two sequences gets a penalty of  $-d$ . The Needleman-Wunsch algorithm has a computational complexity of  $O(nm)$ , where  $n$  and  $m$  are the lengths of the two sequences.

### 3 SUSAX - An Overview

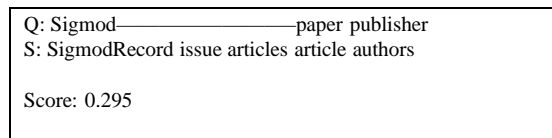
In this section, we give a brief overview of the three key steps that represent the core functionality of SUSAX, and using which we can provide approximate contextual searching of XML documents. In SUSAX, a query  $Q$  is represented as a simple path as shown in Figure 4. Here, the leaf node, `publisher`, represents the *keyword* for the search, while the path from the root to the leaf, `Sigmod/paper` represents the *context* for the keyword search. XML documents are broken down into similarly represented set of source paths. The given query  $Q$  is compared against all sources paths  $P$  in the repository, and a ranked list of documents highlighted with matching source paths is returned to the user. The approximate contextual comparison of the query  $Q$  with a source path  $P$  is conducted via the three primary steps outlined in Figure 5 and briefly described below.

**Step 1 - Tokenization:** In the first step, a given query  $Q$  is converted into a set of tokens, where each token is separated from the next by the `'/'`. For example, the tokenization step generates the tokens `{Sigmod, paper, publisher}` for the query  $Q$  given in Figure 4. A similar tokenization step is carried out for the source paths, one at a time, resulting for example in the tokens `{SigmodRecord, issue, articles, article, authors}` for the source path  $P$  shown in Figure 4.

**Step 2 - Approximate Matching:** Once tokenized, a *similarity* measure is computed for each possible query-source token pair. The similarity between the query and source tokens provides a measure of the approximate match between the two tokens – 1.0 if the two tokens match exactly, and a value between 0.0 and 0.9 for approximate matches. The output of this step is a *similarity matrix* that provides the similarity score for every token pair in the query and source paths. Figure 6 depicts a similarity matrix generated for the query and source sequences given in Figure 3. The similarity measure and the corresponding similarity matrix

	SigmodRecord	issue	articles	article	authors
Sigmod	0.667	0.0	0.0	0.0	0.0
paper	0.167	0.25	0.5	0.5	0.143
publisher	0.061	0.167	0.1	0.1	0.167

**Figure 6. The Similarity Matrix Generated from Step 2 - Approximate Matching.**



**Figure 7. Alignment of the Query and the Source Paths.**

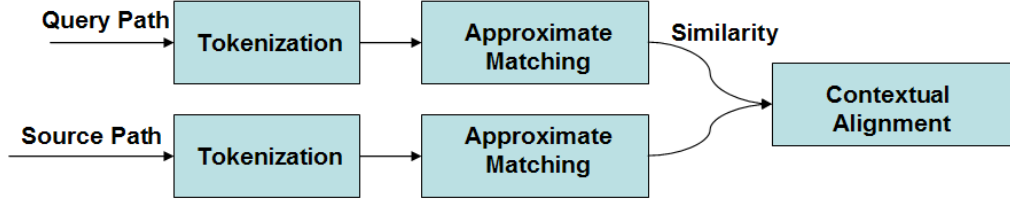
are computed using the *Label Match* algorithm detailed in Section 4.

**Step 3 - Contextual Alignment:** The similarity matrix generated in **Step 2** is used in the final step to compute the optimal alignment between the query and the source paths. The contextual alignment is the adaptation of the Needleman-Wunsch pairwise global alignment algorithm, and aligns, as much as possible, each token in the query path to its best match in the source path. The output of this step is the aligned sequence together with a total normalized score for the alignment. The normalized score for the alignment provides an overall ranking of the query path to all relevant source paths in the repository. Figure 7 shows the alignment of the query and the source path given in Figure 3, together with the alignment score. Note here that two gaps (aligning with the tokens `issue` and `articles` in the source path) are introduced in the query path to provide the best match for each query token (`Sigmod`, `paper` and `publisher`). Details on the contextual alignment step are presented in Section 5.

### 4 Approximate Matching – The Label Similarity Algorithm

Approximate matching computes the similarity for every query-source token pair for a given query and source path, and dynamically generates as output a similarity matrix (shown in Figure 6) with similarity scores for all possible query-source token pairs. To compute the similarity scores, and hence the similarity matrix, we employ a *label match* [23] algorithm that semantically compares two labels using a dictionary and other auxiliary information.

A token, typically representative of natural language, can



**Figure 5. The Primary Steps in Performing the Approximate Contextual Comparison of a Query with a Source Path.**

be classified as either (i) an atomic token - composed of a single word; or (ii) a composite token - composed of multiple words, where the start of each word is distinguished generally by punctuations (for example, *purchase-order*), case distinction (for example, *purchaseOrder*), or numeric digits (for example, *street1*). Typically, no restrictions are applied on the words themselves – they can be a fully-defined dictionary word, an abbreviation, an acronym, or a substring. For example, *qty* is an abbreviation of *quantity*; *uom* an acronym of *unitOfMeasure*; and *addr* a substring of *address*.

The similarity measure between two tokens,  $T_q$  and  $T_s$ , is determined as follows. The two given tokens are first parsed into subtokens. The similarity between the query and source subtokens is then measured using existing linguistic similarity measures such as *lin* [14], *path* and *wup* [24]. Finally, the similarity between the two tokens, query and source tokens,  $T_q$  and  $T_s$ , is computed as the average of the best similarity measure of each query subtoken with a source subtoken. Formally, the similarity measure,  $SM(T_q, T_s)$  is given as:

$$SM(T_q, T_s) = \frac{S_q + S_s}{|T_q| + |T_s|} \quad (1)$$

where

$$S_q = \sum_{t_q \in T_q} [\max_{t_s \in T_s} \text{lingMatch}(t_q, t_s)] \quad (2)$$

$$S_s = \sum_{t_s \in T_s} [\max_{t_q \in T_q} \text{lingMatch}(t_s, t_q)] \quad (3)$$

Here  $t_q$  is a query subtoken of the query token  $T_q$ , *lingMatch* is the linguistic similarity measure (*lin* [14], *path*, or *wup* [24]) for a pair of tokens,  $|T_q|$  is the number of query subtokens for the query token. The variables  $t_s$ ,  $T_s$  and  $|T_s|$  are similarly defined for the source token.

Figure 8 gives the pseudo-code for the algorithm. For each token pair, we first check if there is an identity, acronym, abbreviation, or substring match between the two

tokens using a domain-specific, local dictionary that defines the common set of abbreviations, acronyms, and commonly used substring/short hand notations for a given domain. If such a match can't be determined, we invoke the linguistic similarity algorithm to determine the similarity distance between the two tokens. We use the *path* linguistic similarity measure – a similarity measure based on the path lengths between concepts, and equal to the inverse of the shortest path length between two concepts. To determine the *path* similarity of two words, we use Wordnet::Similarity [21], a freely available tool that measures the semantic similarity and the relatedness between a pair of concepts. The tool provides six measures of similarity including the *path* measure, and three measures of relatedness, all of which are based on the WordNet [16] lexical database. We ran an independent set of experiments [23] and found that the *path* similarity measure had the highest precision and recall for the domains tested.

## 5 Contextual Alignment: Adaptation of the Needleman Wunsch Algorithm

The tokens generated in the **Tokenization** step together with the similarity matrix produced by the **Approximate Matching** step (see Section 3) are the main inputs for this last and final step of **Contextual Alignment** that produces the alignments as shown in Figure 7. In this section, we describe the Contextual Alignment algorithm – an adaptation of the popular pairwise global alignment algorithm from the life sciences domain.

Based on the Needleman-Wunsch algorithm, we split our Contextual Alignment algorithm into three phases: the *initialization* phase, the *fill* phase, and the *trace back* phase.

In the *initialization* phase, a matrix  $F$  indexed by the two paths, query path  $P_q$  and source path  $P_s$  is constructed. The initial value for the matrix is set as:

$$F(0, 0) = 0, \\ F(0, i) = -i * d, \text{ and}$$

		SigmodRecord	issue	articles	article	authors
	0.0	-0.15	-0.3	-0.45	-0.6	-0.75
Sigmod	-0.15	0.667	0.517	0.367	0.217	0.067
paper	-0.3	0.517	0.917	1.017	0.867	0.717
publisher	-0.45	0.367	0.767	1.017	1.117	1.034

**Figure 9. The Alignment Matrix  $F$  Generated Based on Equation 4 for Query and Source Paths in Figure 4. The values were computed using a gap penalty value  $d = 0.15$ .**

```

double labelMatch (String:  $L_s$ , String:  $L_t$ )
{
  tokens = tokenizer ( $L_s$ )
  tokent = tokenizer ( $L_t$ )
  for each  $t_s \in$  tokens
    for each  $t_t \in$  tokent
      if isIdentical ( $t_s, t_t$ )
        match[ $t_s$ ][ $t_t$ ] = 1.0
      else if isSubstring ( $t_s, t_t$ )
        match[ $t_s$ ][ $t_t$ ] = 0.9
      else if isAbbreviation ( $t_s, t_t$ )
        match[ $t_s$ ][ $t_t$ ] = 0.9
      else if isSimilarInDomain ( $t_s, t_t$ )
        match[ $t_s$ ][ $t_t$ ] = 0.7
      else
        match[ $t_s$ ][ $t_t$ ] = linguisticMatch ( $t_s, t_t$ )
  totalWeight = computeMaxMatch (match)
   $QoM_L$  = totalWeight + (|tokens| + |tokent|)

  return  $QoM_L$ 
}

```

**Figure 8. The Label Match Algorithm.**

$$F(j, 0) = -j * d$$

where  $d$  represents the gap penalty – that is the penalty incurred if there is no possible match between the two tokens<sup>2</sup>. Typically, the gap penalty is a tunable parameter and can be adjusted automatically by the system or set by the user. Here  $i$  and  $j$  represent the indices for the matrix  $F$ .

In the next phase, the *fill* phase, the algorithm proceeds to fill the matrix, starting at the top left corner and proceeding down to the bottom right corner, with the best score  $F(i, j)$ . The best score of  $F(i, j)$  of two tokens  $T_q, T_s$  is given as:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + SM(T_q, T_s) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases} \quad (4)$$

where the value  $F(i-1, j-1) + SM(T_q, T_s)$  is obtained if  $T_q$  aligns with  $T_j$ ;  $F(i-1, j) - d$  is obtained if  $T_q$  is aligned to a gap; and  $F(i, j-1) - d$  if  $T_s$  is aligned to a gap. Here,  $SM(T_q, T_s)$  is the similarity measure obtained from the similarity matrix generated in **Step 2** of the overall SUSAX process. This term corresponds to the  $s$  term in the original Needleman-Wunsch algorithm described in Section 2. Figure 9 depicts the matrix  $F$  as computed by Equation 4 for the query and source paths given in Figure 4.

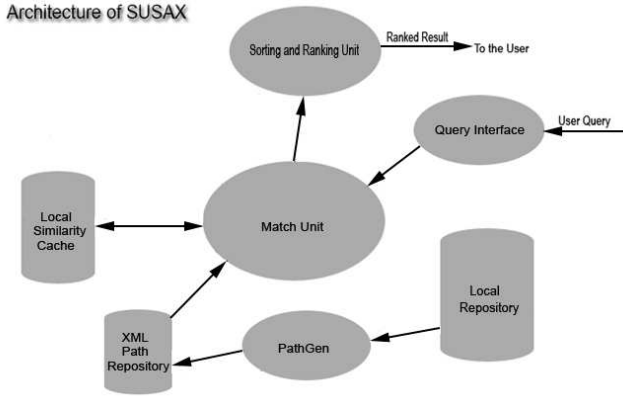
As the matrix  $F$  is filled, a back pointer is kept in each cell specifying the link to the cell from which the  $F(i, j)$  value was derived. The value in the last cell,  $F(n, m)$ , of the matrix  $F$  is by definition the best score for the global alignment between the query path  $P_q$  and the source path  $P_s$ . Figure 10 depicts the alignment matrix  $F$  with the back pointers for the query  $P_q$  and the source paths  $P_s$  given in Figure 4.

		SigmodRecord	issue	article	article	authors
	0.0	←-0.15	←-0.3	←-0.45	←-0.6	←-0.75
Sigmod	↑-0.15	0.667 ↖	←0.517	←0.367	←0.217	←0.067
paper	↑-0.3	0.517 ↑	↖0.917	↖1.017	↖0.867	↖0.717
publisher	↑-0.45	0.367 ↑	↖0.767	↖1.017	↖1.117	↖1.034

**Figure 10. The Alignment Matrix  $F$  Shown with Back Pointers.**

In the *traceback* phase, the actual alignment between the paths  $P_q$  and  $P_s$  is constructed. The traceback phase is initiated at the last cell  $F(n, m)$  in the matrix. At each step the algorithm backtracks from the current cell  $(i, j)$  to one of the cells  $(i-1, j-1)$ ,  $(i-1, j)$  or  $(i, j-1)$  dependent on

<sup>2</sup>The gap penalty corresponds to the term ‘ $-d$ ’ given in Section 2.



**Figure 11. The Architectural Overview of SUSAX.**

the cell from which the value  $F(i, j)$  was derived. Based on the backtracking, a pair of tokens is added onto the front of the current alignment. The tokens  $t_q$  and  $t_s$  are added if the the backtrace is to  $(i - 1, j - 1)$ ; a gap character “-” is added at  $t_q$  if the backtrace is to  $(i - 1, j)$ ; and “-” is added at position  $t_s$  if the backtrace is to  $(i, j - 1)$ . When the first cell of the matrix is reached, the traceback step terminates. The output of this step is an alignment between the two given sequences. Figure 7 illustrates the final alignment produced by the traceback phase for the example query and source paths shown in Figure 4.

## 6 Architectural Overview of SUSAX

Figure 11 gives an architectural overview of the SUSAX system used for the preliminary experimental results presented in this paper. The **Local Repository** is the data store of XML documents collected from various sources on the Web. We do not consider, for this version of SUSAX, issues pertaining to web crawling and refreshing of the documents in the repository in event of updates. The **PathGen** component generates all unique paths from the XML documents in the Local Repository. The PathGen unit, executed only when the Local Repository is updated, also embeds useful information, such as the source document for each path. The XML paths generated by the PathGen are stored in the **XML Path Repository**. While currently the XML Path Repository is represented by a single XML document, we plan on adding an XML indexing facility to speed up the approximate contextual search outlined in this paper.

Users input simple path queries into the system via the **Query Interface**. The Query Interface both captures the user query as well as pre-processes the query removing

empty tokens and making the queries case-independent. The *user query* is passed on to the **Match** unit, the core component of SUSAX. The Match unit performs three primary tasks. First, the Match unit retrieves the set of paths from the XML Path Repository against which the query path must be compared. Second, for every query and source path pair it follows the three key steps outlined in Figure 5 – namely the **Tokenization**, **Approximate Matching** and **Contextual Alignment**. Tokenization generates a set of tokens corresponding to the query and source paths as outlined in Section 3. The Approximate Matching unit uses the Label Match Algorithm outlined in Section 4 to produce the similarity matrix. Finally, the Contextual Alignment unit uses the similarity matrix as input and produces the final alignment between the query and source paths. Last, the Match unit performs book-keeping and updates a **Local Similarity Cache** of similarity values for a token pair. The Local Similarity Cache maintains the similarity values between two given tokens, and is used primarily as a faster alternative to the Label Match algorithm. Thus, if a pair of tokens exists in the Local Similarity Cache, the similarity value is used for the similarity matrix generation and the Label Match algorithm is not invoked.

The last unit of the SUSAX is the **Sorting and Ranking** component. The Sorting and Ranking unit takes the pairs of all query and source paths, their alignments and alignment scores as input. As a first step, it normalizes the alignment scores by dividing the alignment score with the total number of query tokens. The source (aligned) paths are then sorted based on the normalized score. Documents corresponding to the ranked list of source (aligned) paths are displayed in decreasing order of scores to the user.

## 7 Experimental Evaluation

We have conducted a preliminary set of experiments to evaluate the effectiveness of our approach in terms of (1) the overall precision and recall of the approximate, contextual alignment; and (2) the accuracy of the SUSAX results when compared to local Google search performed on the same XML repository (Local Repository).

### 7.1 Experimental Setup and Methodology

The SUSAX system was implemented in Java (J2SE Runtime Environment Version 5.0) and evaluated on a Dell 2.8 GHz Pentium 4 workstation with 512 Mb of RAM. The **Local Repository** for this set of experiments was setup using XML documents obtained from the University of Washington’s XML Repository [22], and included XML documents from 321gone, ebay, book, customer, SigmodRecord, ubid and yahoo. All unique XML paths were generated from these documents resulting in a

total of 306 paths in the XML Path Repository. Queries, while targeted towards these domains, were variants, in length and token labels, on the paths in the repository. The **Label Match** algorithm used the Perl interface to the WordNet-Similarity, version 0.15, that internally utilized WordNet 2.1.

## 7.2 Preliminary Results

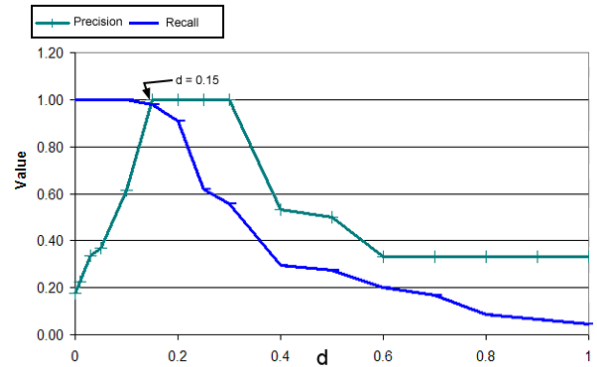
To evaluate the quality of our approach, we compared the manually determined real matches ( $R$ ) for a given match task with the matches  $P$  returned by the match algorithm. We determined the true positives, that is the correctly identified matches  $I$ ; the false positives, that is the false matches,  $F = P \setminus I$ ; and the false negatives, that is the missed matches,  $M = R \setminus I$ . Based on the cardinalities of these sets, the *Precision* and *Recall* of the SUSAX algorithm was computed.

- *Precision* =  $\frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$  estimates the reliability of the results;
- *Recall* =  $\frac{|I|}{|R|}$  specifies the share of real matches (the real source paths) that is discovered by the algorithm; and
- *Overall Accuracy* =  $1 - \frac{|F|+|M|}{|R|} = \frac{|I|-|F|}{|R|} = \text{Recall} * (2 - \frac{1}{\text{Precision}})$  represents a combined measure of quality, taking into account the post-match effort needed for both removing false matches and adding the missed matches.

**Computing Optimal 'd' Value.** The gap penalty  $d$  introduced in Section 5 plays an important role in determining the quality – the precision and recall – of the matched source paths returned to the user. Recall that  $d$  is the cost of introducing a *gap* in the source or the query path – to indicate the lack of a match for a given query token  $t_q$  with respect to the source tokens  $t_s$  and vice versa. Thus, lower  $d$  values enable a query token  $t_q$  in the  $i^{th}$  position to match with a source token  $t_s$  in the  $j^{th}$  position,  $i \neq j$ , by introducing gaps to align the tokens. We hypothesize (1) the recall of the SUSAX algorithm is inversely proportional to the gap penalty  $d$ ; and (2) the precision of the SUSAX algorithm, while dependent on multiple factors, also has a loose proportionality relationship with the gap penalty.

We ran an experiment to test our hypothesis as well as to determine the optimal range of the  $d$  value for high precision and recall. For this experiment, we ran three different queries against our Local XML Path Repository, and computed the precision and recall of the SUSAX algorithm for varying values of the gap penalty 'd'. To compute the precision and recall for each query, we a priori had a domain

!h



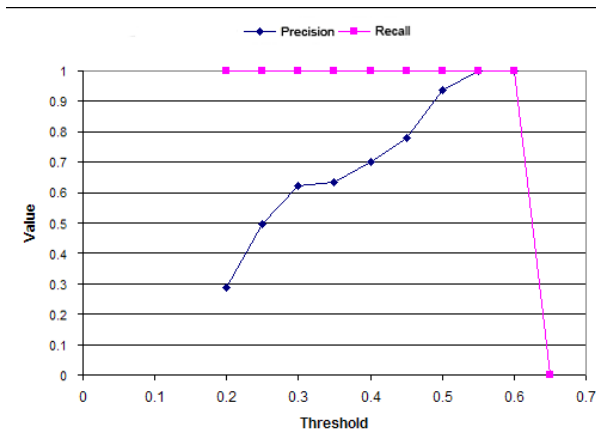
**Figure 12. Average Precision and Recall for Varying  $d$  Values.**

expert manually determine the matches she would expect from the Local XML Path Repository, and then used that to compare the results of the SUSAX algorithm. Figure 12 shows the average precision and recall of the three queries for different  $d$  values. The x-axis is the  $d$  value ranging from 0.0 to 1.0, and the y-axis is the normalized precision and recall value and ranges from 0.0 to 1.0. We found that for the queries tested the optimal range of the  $d$  value is between 0.1 and 0.2. For all subsequent experiments we set the  $d$  value to be 0.15.

**What is a Match? Computing the Threshold.** The SUSAX algorithm returns the aligned query and source paths together with a normalized score value that ranges from 0.0 to 1.0. Typically, a higher score value is representative of a better (approximate) fit with the user's query, while a really low score is indicative of a poor approximation to the user's query. To minimize the noise in the final results presented to the user, we ran a set of experiments to determine the optimal threshold. A normalized score above the threshold is shown to the user, while results with scores below the threshold may be discarded.

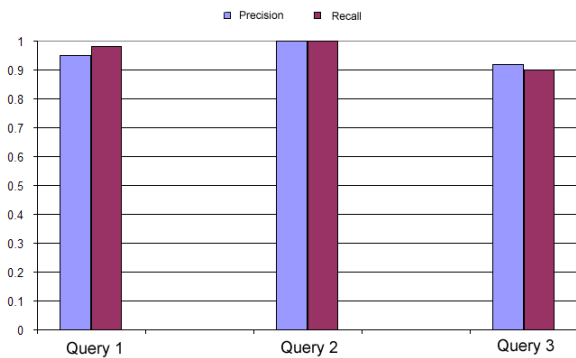
For this experiment we set the gap penalty  $d = 0.15$ . We measured the precision and recall of the SUSAX algorithm for a varying number of thresholds for four independent queries. Figure 13 depicts the average precision and recall measured for the four queries. As can be seen a threshold in the range 0.50 - 0.6 provides the optimal precision and recall. For the remainder of the experiments, we set the threshold of the SUSAX algorithm to be 0.6.

**Precision and Recall.** To validate the previous results (Figure 12) as well as to validate the choice of the  $d$  value,



**Figure 13. Computing the Threshold of the SUSAX Algorithm for a Fixed  $d$  Value. Here  $d = 0.15$ .**

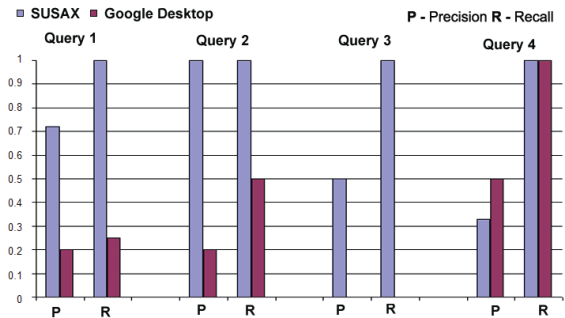
we ran an independent set of experiments to evaluate the precision and recall of SUSAX for a fixed value of  $d$ . For this experiment, we selected three queries and used the SUSAX algorithm to determine matching source paths from the Local XML Path Repository. The  $d$  value for this experiment was set to 0.15, and the threshold was kept at 0.6. The manual matches were determined apriori by a domain expert. Figure 14 depicts the precision and recall that was obtained for the three different queries. The x-axis shows the queries, while the y-axis depicts the normalized value for precision and recall.



**Figure 14. The Precision and Recall for Three Different Queries. The  $d$  value here was fixed at 0.15 and threshold was set to 0.6.**

Query 1: root/listing/auction\_info  
 Query 2: item\_info/cpu  
 Query 3: sigmod/issue/number  
 Query 4: book/person/age

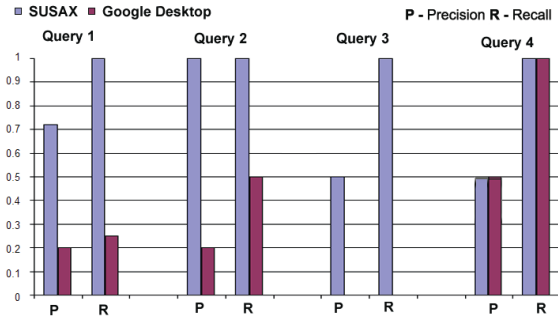
**Figure 15. Four Queries Used for the Comparison Between SUSAX and Desktop Google.**



**Figure 16. Comparing the Precision and Recall of SUSAX with Keyword Searching on Google Desktop.**

**Comparison with Keyword Searching.** To compare and contrast our approach with the popular keyword searching algorithms, we ran experiments to compare the precision and recall of the SUSAX algorithm with the precision and recall of keyword searching. For this experiment we setup the Beta version of the Google Desktop [8], and used it to search the Local XML Path Repository. We selected four queries, shown in Figure 15. Each of the queries was executed using both the Google Desktop as well as the SUSAX algorithm, and the precision and recall was computed for both methods. As before, we had a domain expert who apriori determined the expected set of results. For the SUSAX algorithm, we set  $d = 0.15$ , and  $\text{threshold} = 0.6$ .

Figure 16 depicts the results that were obtained. The x-axis shows the queries that were executed, while the y-axis depicts the normalized values for precision and recall ranging from 0.0 to 1.0. As can be seen, in general the SUSAX algorithm provided higher precision and recall than the Google Desktop. We attribute this to (1) the approximate matching. For compound words such as `auction_info`, while the SUSAX algorithm was able to compute a similarity match (an approximate match) for the compound word as a whole, Google Desktop broke the search down to keyword search `info` or `auction`. Similarly, Google Desktop was unable to match `sigmod` in the query with `SigmodRecord` in the XML Path Repository,



**Figure 17. Comparison of SUSAX with Google Desktop. The threshold in this case was set to 0.8 and the  $d$  value was 0.15.**

while SUSAX was able to perform a similarity match between the two tokens; and (2) the contextual information. SUSAX was able to provide a higher precision by utilizing the context information implicit in the path query. For example, while Google Desktop returned results that contained the keywords *sigmod*, *issue* or *number* or all three, SUSAX was able to return the paths that matched the context *sigmod/issue* and the keyword *number*.

In general, we found that the approximate match in SUSAX has a tendency to lower the precision of the overall results, while raising the recall. On the other hand, the context matching tends to improve both the precision and recall of the algorithm. For example, the results for query Query 4 suggests the precision of SUSAX is hampered by the approximate matching. To affirm these results, we ran another experiment where we set a higher threshold. We set the threshold to be 0.8, thereby raising the approximate matching to a near-exact match. The  $d$  value was fixed at 0.15 as before. Figure 17 presents the results of this experiment. As can be seen, with a near exact match SUSAX provides higher precision – confirming our hypothesis.

## 8 Related Work

Many schema matching algorithms [4, 2, 3, 6, 15, 5] have been proposed in literature to address the problem of schema integration. Many of the algorithms developed thus far [3, 15, 1, 6, 4, 2, 5] rely on two primary factors to detect similarities between the schema entities: the *label* and the *structure* of the involved entities. For example, Madhavan et al. have proposed **CUPID** [15] that combines linguistic and structural matching to establish correspondences between the schema entities. Matching in **CUPID** is carried out in three different phases, with the first phase a linguistic match based on a domain-specific dictionary, the second phase, a bottom up structural matching, resulting in

a structural similarity between pairs of elements, and the third phase providing a weighted match based on the linguistic and structural similarity of pairs of elements. **Tran-Sem** [17], on the other hand, performs schema matching at the granularity of elements using a set of predefined rules to match node by node on the graph representation of the input schemas. **SKAT** [18] focuses on identifying the articulation over two ontologies by comparing input schemas based on their ontological knowledge sources and user defined rules which are utilized to increase precision. Two nodes are once again matched based on their labels. Structural similarity is established based on the similarity of their hierarchical structure. Neirman et. al [20], on the other hand, have proposed a structure-based similarity algorithm for XML documents, based on measuring the edit distance for rooted trees. No linguistic match algorithms are employed as part of their match algorithm.

Other systems such as **LSD** [1] and **SemInt** [13] use machine learning and neural networks frameworks respectively as an approach for combining different match techniques and user feedback to ultimately improve the accuracy of schema matching. LSD employs and extends current machine learning techniques to semi-automatically find mappings between input schemas. LSD uses user-supplied semantic mappings for a small set of data sources together with the sources to train a set of learners. Each learner exploits different information in the input schemas. Once they are trained, LSD finds semantic mappings for a new schema by applying the learners and then combining their predictions using a meta-learner. **SemInt** trains neural networks to find matches between two input schemas. **SemInt** provides a match procedure using a classifier to categorize attributes according to their field specifications and data values and, then trains a neural network to recognize similar attributes. It does not support any linguistic matching. Our work now investigates the possibility of applying schema matching techniques to provide approximate and context specific keyword searching.

## 9 Conclusions and Future Work

**Paper Summary.** As the Web moves progressively towards more semi-structured information and as the number of XML documents on the Web increase, we have at our disposal more information that can be harnessed to provide more effective and accurate searching (than pure keyword searching), for these documents. We propose in this paper SUSAX – an algorithm for *approximate, context-specific* searching over XML documents. The SUSAX algorithm is a unique blend of linguistic similarity techniques prevalent in information retrieval and schema matching domains with global sequence alignment techniques, in particular the Needleman-Wunsch global alignment algorithm, that are

indispensable in the life sciences domain. Our preliminary results suggest there is great potential for such similarity searching techniques and that they provide both higher precision and recall than keyword searching. While approximate searching does have the potential to lower the precision of the results, we found that context-specific searching more than compensated for this drop in precision – in fact raising the overall precision.

**Future Work.** While our experiments have shown the potential of our approach, our experiments are preliminary. Our immediate future work includes validation of our experimental findings against larger repositories and larger query sets that represent realistic query workloads.

In this paper our concentration has been on the development of the approximate, context-specific search algorithm, SUSAX. There are, however, two components to the overall SUSAX approach. The first, is the pairwise comparison of the query  $Q$  with a source path  $P$ . This pairwise comparison is potentially repeated for all possible pairings of the query  $Q$  with source path  $P_i$ ,  $\forall P_i \in Repository$ . The second step is the *ranking* of all the source paths with the highest scoring source path ranked as the top hit and the lowest scoring source path as the lowest hit. While we had anticipated using the normalized score of the algorithm to produce the ranked list of source paths, we found that this basic assumption did not hold. Many paths with more accurate keyword matches but a lower context match were ranked lower than source paths with lower, in some cases no, keyword match but higher contextual match. For example, for the query `sigmod/paper/publisher` the source path `/SigmodRecord/issue/articles/article/authors` was ranked lower (based on normalized score) than the source path `/SigmodRecord/issue/articles/article`. This result was counter-intuitive to what we expected. As part of the future work, our goal is to explore heuristics for the ranking of the source paths and hence the XML documents containing the relevant information.

## References

- [1] D. AnHai, P. Domingos, and A. Haley. Reconciling Schemas of Disparate Data Sources: A Machine-Learning approach. In *SIGMOD*, 2001.
- [2] S. Bergamaschi, S. Castano, M. Vincini, and D. Benevenuto. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36(3):215–249, 2001.
- [3] J. Berlin and A. Motro. AutoPlex: Automated Discovery of Content for Virtual Databases. In *CoopIS*, pages 108–122, 2001.
- [4] M. Bright, A. Hurson, and S. H. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *TODS*, 19(2):212–253, 1994.
- [5] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *vldb*, 2002.
- [6] L. Haas, R. Miller, B. Niswonger, M. Roth, P. Schwarz, and E. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.
- [7] G. Inc. Google: A Web Search Engine. [www.google.com/](http://www.google.com/).
- [8] G. Inc. Google Desktop, Beta Version. <http://desktop.google.com>, 2005.
- [9] L. Inc. Hotbot: A Web Search Engine. [www.hotbot.com/](http://www.hotbot.com/).
- [10] M. Inc. MSN: The Microsoft Search Engine. [www.msn.com/](http://www.msn.com/).
- [11] Y. Inc. Yahoo: A Web Search Engine. [www.yahoo.com/](http://www.yahoo.com/).
- [12] E. B. Institute. FASTA Format Description.
- [13] W.-S. Li and C. Clifton. Semantic Integration in Heterogeneous Databases Using Neural Networks. In *vldb*, 1994.
- [14] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.
- [15] J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *vldb*, pages 49–58, 2001.
- [16] G. Miller. Wordnet: A Lexical Database for English Language. [cogsci.princeton.edu/~wn/](http://cogsci.princeton.edu/~wn/), 2002.
- [17] T. Milo and Z. Sagit. Using Schema Matching to Simplify Heterogeneous Data Translation. In *vldb*, 1998.
- [18] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *fusion*, 1999.
- [19] S. B. Needleman and C. D. Wunsch. A General Method Application to the Search of Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
- [20] A. Neirman and H. Jagadish. Evaluating Structural Similarity in XML Documents. In *webdb*, 2002.
- [21] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet: : Similarity - measuring the relatedness of concepts. In *AAAI*, pages 1024–1025, 2004.
- [22] D. Suciu and G. Miklau. XML Data Repository. <http://www.cs.washington.edu/research/xmldatasets/>, 2002.
- [23] N. Tansalarak and K. Claypool. Qmatch – a hybrid match algorithm for xml schemas. *Elsevier Data and Knowledge Engineering, to appear*, 2006.
- [24] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133–138, New Mexico State University, Las Cruces, New Mexico, 1994.