Bulletin of the Technical Committee on

Data Engineering

March 2007 Vol. 30 No. 1

IEEE Computer Society

Letters

Special Issue on Multi-lingual Information Systems

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ipsi.fraunhofer.de/tcde/.

Letter from the Editor-in-Chief

Organizational Changes

This issue contains letters from the incoming and outgoing chairs of the two key organizations within the IEEE Computer Society that are involved in the database area. I urge you to read these letters. The organizations are:

- the Technical Committee on Data Engineering (TCDE), which is the sponsoring organization for both this publication (the Bulletin) and for the International Conference on Data Engineering (ICDE). The outgoing chair is Erich Neuhold of the University of Vienna, the incoming chair is Paul Larson of Microsoft.

- the International Conference on Data Engineering Steering Committee (StC). The outgoing chair is again Erich Neuhold, the incoming chair is Calton Pu of the Georgia Institute of Technology.

I'd like to thank Erich for the enormous role that he has played in chairing so successfully both of these organizations, and to wish the new chairs, Paul and Calton, all the best in their ongoing efforts to build on the foundation that Erich as provided.

The Current Issue

The current issue of the Bulletin is on the topic of multi-lingual support within information systems. This is a topic that is important now and will only continue to grow in importance over time. The world is now ”flat” as Thomas Friedman reminds us in his book. This means there are users and developers of information systems all over the world. Further there is clearly source material in a very wide variety of languages. Accessing this source material via a database or over the web is of increasing importance as people throughout the world expect to be able to find virtually everything ever written anywhere.

What makes this topic an interesting technical area, even a research area, is how extraordinarily difficult it is to deal with the entire gamut of languages. Many languages and language families have idiosyncracies that make things complicated. Simple alphabets are all too frequently the exception rather than the rule. Dealing with these diverse languages within a common framework is a very challenging problem.

Jayant Haritsa has done a fine job of bringing together a some of the work in this area, from a sampling of the languages that we need to be able to deal with. This work comes from both academic and industrial organizations, which indicates to me that there is both great practical interest and real technical challenges in providing multi-lingual support within information systems. I think you will find this issue to be a real eye-opener in terms of both the difficulty of multi-lingual issues and the ingenuity of systems that support it.

David Lomet
Microsoft Corporation

Letter from the Outgoing TCDE Chair

It is time to hand over the chairmanship of the IEEE-CS Technical Committee on Data Engineering. I took over from Betty Salzberg in 2002 and kept the Executive Committee she had selected intact to continue its excellent work.

We have continued the sponsorship of international database related conferences, especially of course our flagship conference, the International Conference on Data Engineering (ICDE). We also have, with the help of our excellent Editor in Chief Dave Lomet, continued the publication of the quarterly Bulletin on Data Engineering and kept working with ACM SIGMOD on keeping the Computer Science Bibliography (DBLP) up to date in the database field. Two years ago we have started our first Workgroup on Self Managing Database Systems. It will present itself at its second workshop just before the ICDE 2007 in Istanbul.

I would like to thank all the members of the Executive Committee for their excellent work and all the Members of the Technical Committee on Data Engineering for their support. I also wish the new Chairman Per-Åke (Paul) Larson all the best for the future and want to thank him again for his willingness to take on the leadership of TCDE.

Erich Neuhold
University of Vienna

Letter from the Incoming TCDE Chair

First of all I would like to thank the members of the TCDE for having elected me chair. On behalf of the TCDE, I would also like to thank Erich Neuhold for his leadership over the last four years.

The activities of the TCDE are led by an Executive Committee whose members are selected by the chair. I am pleased to announce that the Executive Committee for 2007-2008 consists of Karl Aberer (Switzerland), Masaru Kitsuregawa (Japan), Per-Åke (Paul) Larson, chair, (USA), Sam Lightstone (Canada), David Lomet (USA), Erich Neuhold (Austria), Calton Pu (USA), Thomas Risse (Germany). I thank the outgoing committee for their contributions to the TCDE and welcome new and continuing members to the committee.

I do not anticipate any significant changes in the activities of the TCDE. We will continue to sponsor the International Conference on Data Engineering (ICDE), our yearly flagship conference, and strive to maintain and further enhance its quality and reputation. We have also sponsored a number of smaller conferences and workshops in the area of Data Engineering and will continue to do so. Publication of the Bulletin on Data Engineering will continue with Dave Lomet serving as Editor in Chief. As previously, we will cooperate with ACM Sigmod and the VLDB Foundation on issues of common interest to the data management field.

If you have comments, suggestions or ideas for the TCDE, please contact me (paul.larson@microsoft.com) or any member of the Executive Committee. We would appreciate hearing from you.

Per-Åke (Paul) Larson
Microsoft Research

Letter from the Outgoing ICDE Steering Committee Chair

My term as the Chair of the ICDE Steering Committee has ended with 2006 and I would like to thank all the past and current members of the Steering Committee that have served during my time. We have been a great team that worked very well together in selecting future conferences, tracking their planning process and evaluating the final result. Here thanks are also due to Dr. Thomas Risse, my assistant in handling the Chairmans duties.

I am so thankful that we together have been able to grow ICDE from a low point in 1998 to its current size, which is close to 600 participants, including the newly introduced workshops. Of course, the Steering Committee contribution was small, the real work has always been with the officers of the respective conferences and I would like to thank them with this short note also. They are too numerous to list but their enthusiasm and never-ending effort have lead to this result.

I now want to wish all the members of the Steering Committee, but especially its new Chairman Calton Pu, all the success and may ICDE continue to prosper in the future with Cancun, Mexico coming up next year and Shanghai, China in 2009.

Erich Neuhold
University of Vienna

Letter from the Incoming ICDE Steering Committee Chair

First, I want to thank Dr. David Lomet, the editor-in-chief of the Bulletin, for this opportunity for the ICDE Steering Committee(StC) to communicate directly with TCDE members. Second, I want to thank members of the StC for electing me as the new chair. I will do my best to carry out the duties of the StC chair.

The main purpose of this letter is to explain the function of ICDE StC. Foremost is an acknowledgement and expression of gratitude to the past StC chair, Dr. Erich Neuhold. During the last 8 years, Erich has provided firm leadership that brought ICDE to the Modern Era, characterized by a successful democratization of ICDE. The main tenets established by Erich are:

Principle of Law. StC functions, conduct, and member election are governed by written rules, which have been published in the Charter document http://www.informatik.uni-trier.de/ ley/db/conf/icde/bylaws.html.

Principle of Openness. A nurturing environment that encourage the active participation of delegates from all countries and underrepresented groups in all aspects of ICDE, including the StC itself.

Principle of Least-Interference. The best organizers are selected to run the ICDE conferences. They make their own decisions (responsively) and are free to innovate according to their talents.

These principles have had a huge impact by bringing in new talents and ideas into each new ICDE. My goal as the new StC chair is to uphold these principles established by Erich, so the StC can guide the future ICDEs towards both greater intellectual impact and wider participation.

Concretely, the main function of StC is to select the organizers of future ICDE conferences. For details of the proposal and selection process, please contact me at calton.pu@cc.gatech.edu. After a team is selected, the StC also monitors the progress of future ICDE organization as a process and provides support to the organizers to help them make every ICDE a success.

The current StC members are: Calton Pu, Karl Aberer, Rakesh Agrawal, Dimitrios Georgakopoulos, Masaru Kitsuregawa, Per-Åke Larson, David B. Lomet, Kyu-Young Whang.

Calton Pu
Georgia Institute of Technology

Letter from the Special Issue Editor

This issue of the Data Engineering Bulletin describes a spectrum of research projects making the first attempts towards scaling the multi-lingual Tower of Babel confronting today's globalized information systems.

Most industrial-strength database engines, both commercial and public-domain, are extremely well-designed for efficiently storing and processing textual information in languages based on the Latin script. These languages include English, the defacto lingua franca of the world, as well as Western European languages such as French, German, Spanish and Italian. But in a rapidly globalizing universe, where the "world is becoming flat", database engines should ideally support text data processing equally efficiently and seamlessly in the entire panoply of human scripts, including Arabic, Cyrillic, Greek, Brahmi, and Kanji, each used by millions of people in a variety of languages. Apart from ease-of-use, there are also compelling business imperatives – market studies indicate that customers are significantly more likely to purchase a product if it is advertised in their native language – making multilingualism a critical factor in global e-Commerce. Similarly, the importance of multilingual support in e-Governance solutions has been well documented.

In a nutshell, while database engines have by and large successfully become "programming-language-neutral", we must now work towards creating *natural-language-neutral* database systems. This issue contains five articles that provide insight into the technical challenges of developing such systems and propose techniques for addressing several key issues in their development.

The first article, by Sorensen and Roukos of IBM Research, highlights the problems of handling full-text indexing and search for morphologically complex languages, such as Chinese and Arabic. A UIMA-based modular architecture for flexible processing of texts based on language-specific properties is proposed as a roadmap towards organically supporting applications for world markets.

The second article by Sarkar of IIT Kharagpur considers the problem of regular expression matching, as exemplified by SQL's LIKE operator, in the multi-lingual world. Through a detailed analysis of language scripts, a rich set of character classes is proposed to comprehensively support script-specific regular expression matching. A new crosslingual operator is introduced to match regular expressions across languages, and extending the standard character-based matching to higher-order linguistic terms is also discussed.

The third article by Ramanand et al from IIT Bombay investigates the Wordnet world of semantically rich language ontologies and quantitatively shows how under all the seeming diversity, there is significant unity in their structural and semantic properties even across vastly different language families. They also discuss strategies for automatically generating, given the Wordnet of a seed language, the Wordnets of related languages in the family, demonstrating their approach by constructing the Marathi Wordnet from Hindi.

The fourth article by Kumaran and Carlin of Microsoft Research leverages the recently-proposed OrdPath node ordering technique to efficiently implement cross-lingual semantic queries on Wordnet ontologies. Efficiency is of paramount importance in handling such queries since they entail the computationally expensive task of computing transitive closures on the Wordnet hierarchies. The initial performance results suggest that the proposed algorithm can more than halve the time taken by the classical approaches.

Finally, the last article by Pingali and Varma of IIIT Hyderabad, tackles the problem of improving the relevance of answers in multilingual information systems. Specifically, they propose strategies for simultaneously achieving the twin objectives of high precision and high efficiency by developing powerful indexing techniques that are tailored to language models, in contrast to the standard term-based inverted-index approaches.

In closing, I thank the article authors for their painstaking and timely efforts in developing their contributions for this special issue. I also thank Dave Lomet, the Editor-in-Chief, for patiently showing me the ropes. I hope that the work presented here will serve as a stimulus for the academic and industrial research communities to redouble their efforts on multi-lingual information processing.

Jayant R. Haritsa
Indian Institute of Science
Bangalore, India

Rethinking Full-Text Search for Multilingual Databases

Jeffrey Sorensen and Salim Roukos
IBM T. J. Watson Research Center
Yorktown Heights, New York
<sorenj|roukos>@us.ibm.com

Abstract

Textual fields are commonly used in databases and applications to capture details that are difficult to formalize—comments, notes, and product descriptions. With the rise of the web, users expect that databases be capable of searching these fields quickly and accurately in their native language. Fortunately, most modern database systems provide some form of full-text indexing of free text fields. However, these capabilities have yet to be combined with the simultaneous demand that databases provide support for world languages. In this paper we introduce several of the challenges for handling multilingual data and introduce a solution based on an architecture that enables flexible processing of texts based upon the properties of each text's source language. Extending the indexing architecture, and standardizing the query capabilities, are important steps to creating the applications that will serve world markets.

1 Introduction

Text fields capture *unstructured information* such as descriptions, comments, notes and other difficult to formalize information, and are part of most large scale database applications. Standard database tools, and the popular Structured Query Language (SQL), provide little support for applications that demand indexing and searching these fields.

Even so, full-text search indexing is a *standard* non-standard component of every major database system, including the popular open-source databases MySQL and PostgreSQL, through add-ons such as TSearch2 [3,11]. Microsoft SQL Server provides the ability to generate full-text indexes and, as shown in Figure 1, a query syntax for searching the database records.

These three examples illustrate searching for a keyword, searching for the different inflected forms of the word foot (feet, footed, etc.), and searching for documents that contain two terms in close proximity. More traditional databases do not provide such query capabilities. While most databases provide string wildcards and LIKE operator or regular expression matching, using them for search of text fields requires laborious record by record string comparison operations.

In order to implement fast full-text search, a *full-text index* must be built. However, to make a full-text index one must understand issues of tokenization, punctuation handling, normalization stemming and morphology, and word segmentation; these are all properties that are specific to each language and region. Today, most databases are configured to contain only one language, but for international companies this is already changing.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

```
SE AdventureWorks;
GO
SELECT Name, ListPrice
FROM Production.Product
WHERE ListPrice = 80.99
AND CONTAINS(Name, 'Mountain');
GO

USE AdventureWorks;
GO
SELECT Comments, ReviewerName
FROM Production.ProductReview
WHERE CONTAINS (Comments,
    'FORMSOF(INFLECTIONAL, "foot")');
GO

USE AdventureWorks;
GO
SELECT Description
FROM Production.ProductDescription
WHERE CONTAINS(Description,
    'bike NEAR performance');
GO
```

Figure 1: Examples of full-text queries for Microsoft SQL Server 2005 [5]

We distinguish between (1) a *single language database*, (2) a database that contains specific fields in a specific language (as in a system that supports searching Arabic and Chinese documents using English queries), and (3) the most general case, a database with fields that have multiple languages in the same field. We are proposing to use an architecture that supports all three types of databases.

Both Unicode support and full-text indexing are recent additions to traditional relational databases. However, as the Internet becomes truly international, see Figure 2, the ability to index world languages will become essential. Especially note that the size of the "Other" category means that developers will need to handle *many* new languages.

Looking at Microsoft SQL Server, again as representative of the state of the art, Microsoft uses a separate operating system component to implement the full-text search. This component has some support [5] for international text support using "word breakers, stemmers, and filters." However, this fixed architecture, originally designed for English, makes it difficult to handle databases that have fields from multiple languages.

In this paper we will be describing in detail the differences between the processing needed for several languages. By discussing languages that differ greatly in morphological complexity, we hope to demonstrate the need for a flexible, modular architecture for full-text indexing. To address this we propose database developers look to the Unstructured Information Management Architecture (UIMA) project affiliated with the Apache web server and its related projects.

2 Related Work

This paper references a number of papers dealing with various aspects of text processing needed for constructing indexes. However, the details of how to build and maintain indexes are beyond the scope, but well summarized in [29]. More recent developments in building compressed full-text indexes are presented in [16].

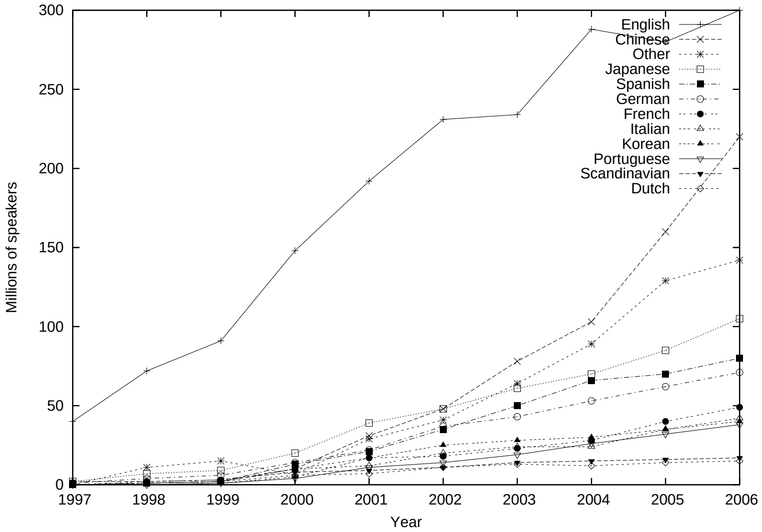

Figure 2: Internet population growth by language [24]

Our focus is primarily on the operations required to extract terms from the text for indexing purposes. However, there are several papers related to the building of indexes and their use in search. In [13], query operators are built into a database engine to allow fuzzy matching of strings from different languages through a mapping to approximate pronunciation differences, and, more abstractly, using a multilingual taxonomy, semantic category matching. Similarly, [32] studies the application of various edit distances seeking to match strings cross-lingually if their pronunciations are similar.

The trade off between building an exhaustive index and using a fuzzy distance metric during search involves trade-offs between computational complexity and memory storage. Several algorithms are compared in [21], matching terms cross lingually based upon string matching functions. Algorithms of this type can be used to reduce the total size of the index by collapsing interlanguage spelling variation. Similarly, [1] presents work that seeks to identify terms in documents that represent names of people or places in different languages.

3 Internationalization of Applications

Traditionally, internationalization efforts have focused on creating applications that can be operated by populations in different languages. This means tailoring fonts, menus, and other application messages often through the modification of global operating system settings. Many systems today determine the precise manner to collate and search strings through global "locale" settings. While these technologies have allowed products to be sold in international markets, this approach to "internationalizing" applications has been largely unsuccessful as an approach to creating multilingual applications. Even today, the standard ANSI C++ language has very limited support for international character representations.

Web browsers represent the leading edge of internationalization technology, as the web rendering technologies are the only software currently capable of rendering multilingual texts irrespective of most operating system settings. To a large extent most users today can effectively retrieve, and even create, texts in most languages, often limited only by their keyboards and the availability of alternate text input methods. This is amply demonstrated by the Wikipedia project [28] which, using a unified set of web tools, has successfully grown to more than 10,000 articles in each of 21 languages.

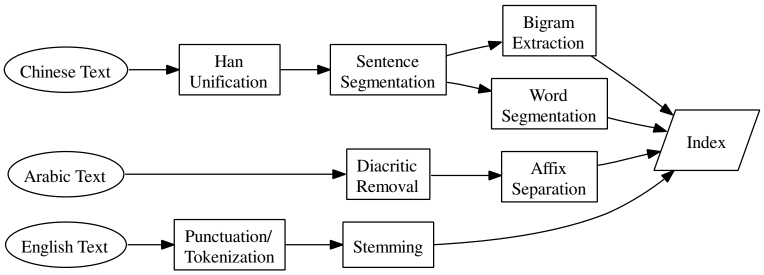

Figure 3: Typical text processing components for indexing

4 Modularization of Text Analysis

Developing international applications is a challenge for any platform. At present, implementing applications with full-text search requires correctly configuring a database server's operating system and a careful design of the database. If full-text searching is to become a standard feature of applications, making it work with multi-lingual texts must be simplified. Some of the current impediments include: a lack of standards for indexing and query specification, small alphabet assumptions in the text-processing components of databases and operating systems, the widespread use of components that do not correctly transfer Unicode data.

We seek to address many of these problems using a text processing architecture that is sensitive to the source language, even if said language is not specified explicitly. We present examples of processing pipelines for several languages that represent some of the widely varying needs.

4.1 Text-Indexing Pipeline

The full-text processing pipeline, as implemented in all of the current indexers for current database platforms, invariably involves a word segmenter, often just synonymous with "whitespace" tokenization. Many systems also make use of *stemmers* and *stop word lists*. Such a pipeline is usually adequate, at least for English texts. This is less true for other European languages which often have much richer word morphologies.

The Unstructured Information Management Architecture (UIMA) is a set of standards [9] and an Apache hosted collection of libraries intended to foster interoperability between developers of semantic analysis tools and other components for document search. While UIMA is intended to integrate analysis tools that reach to higher levels of abstraction than is typical for full-text search [17], it also has considerable strength for internationalization through its integration with Java and the International Classes for Unicode (ICU) [12] project. The capability to chain a sequence of text analysis engines, when appropriate, is a crucial capability when dealing with multiple languages. Figure 3 illustrates several chains of analysis tools used for a variety of languages.

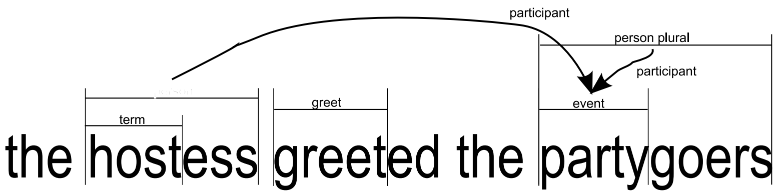

Figure 4: Marking character spans using stand-off notation in UIMA

4.2 Stand-off Notation

UIMA, through the Common Analysis Structure (CAS) represents tokens and other values to be indexed through the use of stand-off notation. Stand-off notation is the separate demarkation of text substrings via data structures that mark the begining and ending character offsets, as opposed to mark-up which uses *in-line* notation which is inserted into the text document. As the original text is unmodified, it can be inspected by any indexing processing subsystem. Stand-off notation is also important when text spans may be overlapping. The case of Chinese is exemplary, as Chinese text must be broken into individual words either using rules or some other automatic mechanism. However, this cannot be done unambiguously. Therefore, a common practice is to index Chinese simultaneously as words and as *overlapping bigrams* [14]. By referring to the text through a sequence of offset pairs instead of a single sequence of tokens, stand-off notation can support indexing of these overlapping spans. Figure 4 illustrates a potential use of stand-off notation to mark an English sentence that contains complex morphology. In addition to finding word stems and labeling verbs with their infinitive form, we may also apply, potentially, overlapping labels of semantic categories like "persons" or "locations" and even hierarchical relationships between these labels (e.g., the PARTICIPANT(PERSON,EVENT) relation in Figure 4).

4.3 Subjects of Analysis

When documents are translated, either by human or machine, or when text processing is used in a manner that modifies text in an irreversible way, the CAS is capable of maintaining multiple parallel representations. Within UIMA, these separate representations are considered differing *views* of the same text. Multiple subjects of analysis can be an important tool when building an index multilingually, as links into a view provide the matching correspondence in the original text. Thus both the document and its translation may be maintained in parallel in this CAS container.

4.4 Unicode Representation

The Unicode standard is an ambitious effort to represent the union of the characters from all written languages into one extended alphabet. Unicode remains controversial as a solution in some quarters, particularly among Chinese and Japanese communities [27]. The controversy is in part due to the complexity of the task of cataloging all of the hundreds of thousands of characters that occur in languages around the world. However, the problem is also due to the difficult experiences that developers have had with incomplete or, often, incorrect implementations of the Unicode Standard.

The Unicode consortium has been very active and the Unicode Standard is a rapidly evolving document with changes in virtually every aspect. The reference, and open-source, implementation [12] of the standard - the International Classes for Unicode (ICU) project has been under constant evolution and is itself a large and complex project that is often included, in whole or in part, in other projects such as Apache and the Java language.

5 Text Analysis Components

Most full-text indexing systems have a fixed architecture for text ingestion. At present, full-text indexing is still considered a special feature with only limited integration into the database. This means, in part, that the database has very limited control over the processing of individual text fields.

We are proposing that a more flexible approach be used. To illustrate why this would be beneficial, we would like to review some of the major components of text analysis to demonstrate how their relative importance changes, depending upon which language is being processed. When these multiple components can be combined flexibly on a record by record basis, perhaps based upon the content of other fields, multilingually indexing and searching can be more easily and effectively accomplished. This dynamic ability to combine text-analysis components is one of the principal design goals of the UIMA project.

5.1 Language Identification

Where possible, applications should allow users to specify unambiguously what language a source text is written in. Typically, users enter texts in the same language as the interface an application is written in. However, this is a brittle assumption for polyglots. For text entry, the ability to check spelling is highly desirable, and this requires specification of not only the language but even the specific regional variant. Well written applications allow the user to make these specifications explicitly, although support for mixed languages within a single text field can pose significant interface challenges. When users cannot specify the language, or in cases where text fields are otherwise uncategorized, statistical techniques can be employed to determine the likely language [2, 23]. The UIMA standoff annotation can effectively encode the language ID of different spans in mixed language fields.

5.2 String Matching

The question of whether two strings, perhaps one typed in by the user and one stored in a database, are equivalent is an essential component of nearly every application. While less complicated than the issue of *near match* or *phonetic match*, there are difficulties that must be faced when a single string has multiple representations that are visually equivalent.

In the English world, many applications make use of lower case comparisons so that “English” and “english” are considered equivalent. However, this is quite problematical for applications comparing “espanol” with “Español” for a number of reasons, including ongoing difficulties associated with the case folding functions and case-insensitive string comparison functions that are based on ASCII character assumptions.

The Unicode standard has a substantial literature on the problem of accented characters, the first to be aware of is the existence of the variety of *normalization forms* [7] where characters can be represented in both composed and decomposed forms. In addition, the Unicode report on *Case Mappings* [8] gives some guidance on how to perform a case-independent string matching.

The growth of web search as a user interface has greatly changed the expectations of users. Most search engines use broad equivalence classes so that distinctions between searches that involve punctuation and accents are often impossible to make. This can be both forgiving when the representation is ambiguous (consider résumé versus resume) but maddening when the difference is essential.

However, when considering languages outside of European contexts, these problems are only the beginning. Although most non-European languages do not have upper and lower case distinctions, some languages have rich morphology that makes simple many-to-one equivalence insufficient for searching indexes. We will consider two specific languages cases as illustrative.

5.2.1 Traditional and Simplified Chinese

For databases and user interfaces, Chinese presents numerous challenges. Beyond the sheer complexity and size of the character set, there also exist several related language variants. In Taiwan and Hong Kong, *Traditional Chinese* characters are the predominant written form. Mainland China and Singapore use the much more recently developed *Simplified Chinese* character set.

The Unihan database is a project that seeks to index the relationship between the characters used in these different representations. While it important to maintain the distinctions between the various forms of the written Chinese language for rendering purposes, for search and indexing the representation of terms should be unified, typically by mapping to simplified Chinese. Of course, what constitutes a “term” in Chinese, where characters are written without any white-space separation, is a matter addressed in Section 5.3.3.

اَلمقر اَل + مقر aal + maqar the location	مقرهم مقر + هم maqaru + hum their location
بِالمقر ب + اَل + مقر bi + aal + maqar by the location	بمقرهم ب + مقر + هم bi + maqari + hum by their location
لِلمقر ل + اَل + مقر li + aal + maqar to the location	لمقرهم ل + مقر + هُم ly + maqari + hum to their location

Figure 5: An illustration of Arabic morphological changes

Many older string handling libraries assume byte or smaller alphabet sizes - in particular libraries that perform compression; many Asian texts clearly violate these assumptions. In addition to the some 70,000 unique ideographs designated in the Unicode specification, there exists extension mechanisms such as *ideographic descriptions* which allow users to describe an individual character using a hierarchical combination of other characters; a description that may be as many as 16 Unicode code points, meaning a single rendered character might be described by some 30 bytes in a file.

Character descriptions that are this long and complex are often chopped into meaningless sequences by naïve components. This is one of the many reasons we advocate, wherever possible, the use of *standoff notation* for text processing components.

5.2.2 Arabic

The Arabic alphabet is more similar in size to English than Chinese. However, it is much richer than either language in morphological complexity. In written Arabic, a single word often represents many terms, sometimes related. Written Arabic does not typically encode short vowel sounds; the reader determines the meaning based upon the context, filling in the missing vowels. Through the use of *diacritics*, or accent marks, the short vowels can be specified, and often are, in contexts where they must be specified unambiguously such as in children's educational books or poetry.

Unicode supports the encoding of diacritics through combining forms and through the use of composite forms for the most common diacritic combinations. That is, there is more than one representation of many accented characters, and they are graphically equivalent. This is true for other languages besides Arabic. For example, the "ü" character may be equivalently encoded using the code point `U+00FC`, or as the ASCII equivalent letter "u" `U+0075` followed by the combining diaeresis (or umlaut) character `U+0308`. Note too that this is also distinct from the stand alone diaeresis character `U+00A8`.

For Arabic, diacritics are not used for searching and indexing because they do not typically appear in the written text. However, a much larger problem exists with Arabic writing, and that involves the complex morphology. Arabic words typically combined with suffixes and prefixes that indicate gender, number, possessives and other relationships typically represented with prepositions in English. Searching typically requires identifying the "root" word or, for more effective searching, the "stem." Figure 5 shows an example of these types of modifications all of which share the same stem word.

5.3 Segmentation

Working with international or multilingual texts also presents problems already familiar to practitioners working with English texts. Consider the issue of *tokenization*, or the practice of segmenting an English text into word or word-like units.

One of the first difficulties that developers encounter is how to correctly handle punctuation. A first approximation is to simply consider words to be all contiguous alphabetic sequences. However, the frequent use of abbreviations, such as "Mr." or "Mrs." would be incorrectly segmented as a sentence boundary by a simple rules based tokenizer that considered periods as end of sentence markers.

In many cases, the solutions used for English tokenization can be applied to other languages directly. We will review some of those techniques in the subsequent sections. However, it is important to also note the ways in which techniques originally developed for other languages are now influencing the processing of English texts.

5.3.1 Rule based segmenters

The highly popular Snowball stemmers [22], originally developed for the English language, are used for a variety of European languages. Snowball is a language for encoding an algorithmic description of a stemming algorithm as well as C++ and Java code generators that convert the algorithmic description into code. The Snowball website includes ample references detailing the effect that stemming has on information retrieval.

Unfortunately, rule based segmenters, at least as they are used today, are limited in the amount of context that can be employed when trying to determine the correct word root. Words which involve spelling changes can be incorrectly processed; "mouse" being the root of "mice" is a well known example. As one begins to integrate texts with increased use of morphological changes, the limitations of rule based systems become increasingly apparent.

5.3.2 Statistical Stemmers

In addition to the rule based systems popular for European languages, the use of statistical models has received considerable attention [15] for languages such as Arabic. In these cases, character or word n-grams are used with a dynamic programming search algorithm to find the most likely segmentation of words into roots, prefixes, and suffixes (affixes).

Arabic segmentation can also depend upon contextual features not captured by n-grams, and in these cases additional features may prove useful [31]. However, it is important to remember that search terms are supplied without context, and determining which index entries match can be quite difficult when compared to the words in context. In these cases, query expansion or n-best segmentation should be applied to make the search more inclusive.

5.3.3 Words

Text processing of documents in multiple languages often entails solving problems that English speaking developers take for granted. For example, in many Asian languages, the concept of a *word* is subtle. Chinese and Thai are written in a style that does not delimit word boundaries.

The Unicode consortium has acknowledged this problem, and has sought to provide applications such as word processors with reasonable equivalents of the "next word" and "last word" commands through the use of dictionary based techniques. However, the current implementation, which is only currently used for Thai, is still considered by the community to be a work in progress.

The segmentation of Chinese into word units, such as those that denote actions, places, or people, is an area of considerable research [26] among the machine learning community. As with segmentation, the use of statistical machine learning techniques show the greatest promise for providing an automatic means of separating words in Chinese and, potentially, other Asian languages.

5.3.4 Transliteration

Representing names of people and locations in non-native languages requires a system of mapping from one representation to another. Usually this is done phonetically, although in many cases place names are simply different words in different languages requiring, in this case, translation dictionaries. Figure 5 includes several such examples of Arabic words rendered to be read phonetically, using the letters of the English language.

Phonetic transliteration is also very commonly used in Chinese names, often resulting in letter sequences atypical of English rules, such as the name "Qian." Transliteration is a process that is amenable to automatic methods [25]. It should also be noted that ICU has support for rule based transliterations [12] even though this is not part of the Unicode standard.

5.4 Translation

Machine translation has been demonstrated [6,10,30] to be effective in providing cross-lingual search. Using the CAS Subject of Analysis capability, it is possible, in a UIMA processing pipeline, to simultaneously maintain multiple representations of the same text. By indexing records in both their source language and one or more translations that can be indexed in parallel, thus making it possible to build cross-lingual search tools.

5.5 Semantic Labeling

The information retrieval community also performs higher levels of annotation of texts, including the identification of "entities" such as people, places, organizations, and companies. The extraction, from text fields, of these types of objects—and the ability to build structured data from free text [4] also promise to greatly enhance future search capabilities. Thus, it is important for database systems to include flexibility in their text analysis architecture so as to not limit future applications' ability to use such deeper analysis.

6 Information Retrieval for International Text

Extracting terms from text fields is only the first step in implementing full-text search, as the terms must be placed into an index that provides a fast reverse map from terms back to the relevant documents. The details of building such an index can range from straightforward to complex, depending upon the quantity of text and the vocabulary. While many of the issues are beyond the scope of this paper, we would like to introduce several relevant ideas from the information retrieval community that can be employed to make index based retrieval more valuable.

6.1 String Comparison

Approximate string matching techniques, such as the use of edit distance or phonetic matching is an important tool for assisting users when the exact spelling of a term is unknown. Efficient computational methods exist [20] for many edit distances. However incorporating fuzzy string matching in the index search has to be carefully throttled to mitigate the increased computational requirements.

6.2 n-gram Indexing

Chinese text, in particular, due to its lack of explicit word segmentation is frequently indexed [14] using character bi-grams. That is, every overlapping two character sequence is placed into the index. Interestingly, as we move

Figure 6: A demonstration of a cross-lingual search engine

to larger n-grams, this technique can also be used for European languages. In [18], character n-grams were found to be comparable to other text indexing techniques. Although similar results were not found in the case of Arabic [19]. Ideally, n-gram enabled search can be combined, in a weighted fashion, with traditional word based search and provide near matches in cases where exact matches are not found, without introducing expensive string comparisons across the entire database contents.

7 Conclusion

Full text indexing typically involves the process of stripping away details that are unimportant in terms of search. However, complete applications must also report results and generate meaningful output. Figure 6 shows an example of output from a cross-lingual search engine that has found relevant Chinese and Arabic documents given search terms in English. The system uses UIMA and many of the components discussed in this paper to provide access to a multilingual database of documents.

Fully indexed text fields in databases often change substantially the way that applications are designed. While structured data, and database normalization are important contributors to efficiency and scaling, many real world problems resist structuring. Text fields are widely used in applications to capture those *ad-hoc* details. Without full-text indexing, these fields are essentially opaque to the application, and, unfortunately, the users.

As applications continue to push into international forums, the need to uniformly handle records regardless of the language is increasingly important. Fortunately, mature standards already exist that database designers should consider when extending these capabilities. Unicode technology is already part of most desktop computers through their web browsers. Modularizing the full-text indexing capability of contemporary databases, and standardization of the analysis would be a substantial step in the right direction.

References

[1] Yaser Al-Onaizan and Kevin Knight. Translating named entities using monolingual and bilingual resources. In *ACL '02: Proc. of the 40th Annual Meeting on Association for Computational Linguistics*, pages 400–408, Morristown, NJ.

[2] Olga Artemenko, Thomas Mandl, Margaryta Shramko, and Christa Womser-Hacker. Evaluation of a language identification system for mono- and multilingual text documents. In *SAC '06: Proc. of the 2006 ACM symposium on Applied computing*, pages 859–860, New York, NY.

[3] Oleg Bartunov and Teodor Sigaev. Tsearch2. PostgreSQL RDBMS Extension http://www.sai.msu.su/ megera/postgres/gist/tsearch/V2/.

[4] Jennifer Chu-Carroll, John Prager, Krzysztof Czuba, David Ferrucci, and Pablo Duboue. Semantic search via XML fragments: a high-precision approach to IR. In *SIGIR '06: Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 445–452, New York, NY.

[5] Microsoft Corporation. SQL server 2005: Full text search architecture. http://msdn2.microsoft.com/en-us/library/ms142541.aspx, April 2006.

[6] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: an architecture for development of robust HLT applications. In *ACL '02: Proc. of the 40th Annual Meeting on Association for Computational Linguistics*, pages 168–175, Morristown, NJ.

[7] Mark Davis. Unicode normalization forms. Unicode Technical Report 15, The Unicode Consortium, San Jose, CA, August 1998.

[8] Mark Davis. Case mappings. Unicode Technical Report 21, The Unicode Consortium, San Jose, CA, November 1999.

[9] David Ferrucci and Adam Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, 2004.

[10] Martin Franz, J. Scott McCarley, and Salim Roukos. Ad hoc and multilingual information retrieval at IBM. In *TREC '89: Text Retrieval Conference*, National Institute of Standards and Technology, pages 104–115, 1998.

[11] J. M. Hellerstein, J. F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *Proc. of the 21st International Conference on Very Large Data Bases*, 1995.

[12] International Business Machines, Inc. International components for Unicode user guide. Software Library http://icu.sourceforge.net/, 2006.

[13] A. Kumaran, Pavan K. Chowdary, and Jayant R. Haritsa. On pushing multilingual query operators into relational engines. In *ICDE '06: Proc. of the 22nd International Conference on Data Engineering*, 2006.

[14] K. L. Kwok. Comparing representations in Chinese information retrieval. In *Research and Development in Information Retrieval*, pages 34–41, 1997.

[15] Young-Suk Lee, Kishore Papineni, Salim Roukos, Ossama Emam, and Hany Hassan. Language model based Arabic word segmentation. In *ACL '03: Proc. of the 41st Annual Meeting on Association for Computational Linguistics*, pages 399–406, Morristown, NJ.

[16] V. Mäkinen and G. Navarro. Compressed full-text indexes. Technical Report TR/DCC-2006-6, Department of Computer Science, University of Chile, April 2006.

[17] Alan Marwick. Text mining for associations using UIMA and DB2 Intelligent Miner. *IBM Developer Works*, http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0602marwick/, February 2006.

[18] Paul Mcnamee and James Mayfield. Character n-gram tokenization for European language text retrieval. *Inf. Retr.*, 7(1-2):73–97, 2004.

[19] Suleiman H. Mustafa and Qasem A. Al-Radaideh. Using n-grams for Arabic text searching. *J. Am. Soc. Inf. Sci. Technol.*, 55(11):1002–1007, 2004.

[**20**] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[**21**] Ari Pirkola, Jarmo Toivonen, Heikki Keskustalo, Kari Visala, and Kalervo Järvelin. Fuzzy translation of cross-lingual spelling variants. In *SIGIR '03: Proc. of the 26th Annual International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 345–352, New York, NY, 2003.

[**22**] M. F. Porter. Snowball: A language for stemming algorithms. http://snowball.tartarus.org, 2001.

[**23**] John M. Prager. Linguini: Language identification for multilingual documents. In *HICSS '99: Proc. of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 2*, page 2035, Washington, DC, 1999.

[**24**] Global Reach. Global Internet statistics (by language). http://global-reach.biz/globstats/evol.html, 2004.

[**25**] Charles Schafer. Novel probabilistic finite-state transducers for cognate and transliteration modeling. In *Proc. of the 7th Conference of the Association for Machine Translation of the Americas*, pages 203–212, Cambridge, MA, August 2006.

[**26**] Richard Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. In *Proc. of the 32nd annual meeting on Association for Computational Linguistics*, pages 66–73, Morristown, NJ, 1994.

[**27**] Suzanne Topping. The secret life of Unicode. *IBM Developer Works*, http://www-128.ibm.com/developerworks/library/u-secret.html, May 2001.

[**28**] Jakob Voss. Measuring Wikipedia. In *Proc. of the 10th International Conference of the International Society for Scientometrics and Informetrics 2005*. International Society for Scientometrics and Informetrics, Stockholm, Germany, July 2005.

[**29**] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[**30**] Jinxi Xu, Alexander Fraser, and Ralph M. Weischedel. TREC 2001 cross-lingual retrieval at BBN. In *TREC '01: Text Retrieval Conference*, National Institute of Standards and Techonology, 2001.

[**31**] Imed Zitouni, Jeffrey S. Sorensen, and Ruhi Sarikaya. Maximum entropy based restoration of Arabic diacritics. In *ACL '06: Proc. of the 41st Annual Meeting on Association for Computational Linguistics*, Sydney, Australia, 2006.

[**32**] J. Zobel and P. W. Dart. Phonetic string matching: Lessons from information retrieval. In H.-P. Frei, D. Harman, P. Schäble, and R. Wilkinson, editors, *Proc. of the 19th International Conference on Research and Development in Information Retrieval*, pages 166–172, Zurich, Switzerland, 1996.

Regular Expression Matching for Multiscript Databases

Sudeshna Sarkar
Computer Science & Engineering Department,
IIT Kharagpur
email: sudeshna@cse.iitkgp.ernet.in

Abstract

Modern database systems mostly support representation and retrieval of data belonging to different scripts and different languages. But the database functions are mostly designed or optimized with respect to the Roman script and English. Most database querying languages include support for regular expression matching. However the matching units are designed for the Roman script, and do not satisfy the natural requirements of all other scripts. In this paper, we discuss the different scripts and languages in use in the world, and recommend the type of regular expression support that will suit the needs for all these scripts. We also discuss crosslingual match operators and matching with respect to linguistic units.

1 Introduction

Language is a defining feature of human civilization, and many languages and scripts have come into existence that are used by people around the world. As multilingual requirements are coming to the forefront in today's world, databases are required to effectively and efficiently support storage, indexing, querying and retrieval of multilingual texts. Various databases now offer some multilingual support, but most of the features are still designed and optimized for the Roman script.

Database languages like SQL use certain string matching primitives. However these primitives are very rudimentary and have been primarily designed with the Roman alphabet in view. But when one considers the diversity of writing systems, firstly there is a need to re-think these primitives for monolingual matching, and evolve patterns that accommodate the diverse requirements for the different scripts. Secondly, we also need to address the requirements of multilingual databases, where information from different languages and scripts are stored in the same database. For multi-script databases, in addition to being able to handle matching requirements in various individual languages, we need to have cross-lingual regular expression matching operators. Thirdly, as text data is becoming increasingly common, databases are being used to store running text data. The use of special matching units in addition to the letter-based matching methods will extend the power and functionality of databases with respect to text data.

In this work we will inspect some of the important types of writing systems, and discuss effective operators keeping in view the requirements of these writing systems. The most popular pattern matching operator in SQL and other popular database languages is LIKE which supports wildcard characters for matching a single

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

character or a sequence of characters. We find that in some writing systems the elementary character units are different from a visual graphemic unit, and the notion of what a character stands for varies across different writing systems. We discuss this diversity, and recommend appropriate matching units for various types of scripts and languages. In this work, we propose different standard character classes which need to be defined for the various languages and script families. The support for this may be built into the database. If not it is possible to specify these using user defined functions. We also propose the introduction of a cross-lingual pattern matching operator, so that pattern matching can be supported across different languages. Further, we recommend that databases provide support to linguistic matching units.

The paper is organized as follows: In Section 2, we discuss briefly about the major types of writing systems, in order to understand the meaningful units with respect to the different systems. In Section 3 we discuss the regular expression support in standard databases including SQL. In Section 4, we make our recommendations about defining character classes to better support regular expressions across different scripts and languages. We also introduce a crosslingual matching operator LexLIKE.

2 Writing systems

A *writing system* [2, 11] is a type of symbolic system used to represent elements or statements expressible in language. All writing systems possess a set of base units, which are called *graphemes*, and collectively they form a *script*. Writing systems differ in many ways including the basic units they use and the direction in which the script is written. We now discuss in brief the major classes of writing systems.

2.1 Types of writing systems

Abjads Abjads [3] which are used by scripts of languages like Arabic, Hebrew and Persian were the first type of alphabet to be developed. These scripts represent only consonants, and not vowels in the basic graphemes. They have one symbol per consonantal sound. In rare cases, the vowel marks may optionally be included by means of diacritics. Most of Abjads are written from right to left.

In the Arabic script, words are written in horizontal lines from right to left, but numerals are written from left to right. The long vowels /aː/, /iː/ and /uː/ are represented by letters. Vowel diacritics, which are used to mark short vowels, and other special symbols appear only in a few religious and children texts.

Alphabets An alphabetic writing system has a set of letters, or graphemes, each of which roughly represents one or more phonemes, both consonants and vowels, of the spoken language. Many languages (e.g., those based on the Latin, Cyrillic, Greek and Armenian alphabets) use multiple letter cases in their written form (e.g., English uses majuscule or upper case and minuscule or lower case). The most widely used alphabets are the Latin or Roman alphabet and the Cyrillic alphabet, which have been adapted to write numerous languages.

In some cases combinations of letters are used to represent single phonemes, as in the English 'ch'. A pair of letters used to write one sound or a sequence of sounds that does not correspond to the written letters combined is called a *digraph*. In some languages, digraphs and trigraphs are counted as distinct letters in themselves, and assigned to a specific place in the alphabet, separate from that of the sequence of characters which composes them, in orthography or collation. For example, the Dutch alphabet includes 'ij' which is usually collated as a single unit. The Croatian alphabet includes some digraphs such as 'dž', 'lj' and 'nj' which are treated as single letters. Other languages, such as English, split digraphs into their constituent letters for collation purposes. Some alphabetic writing systems use extra letters and ligatures. A ligature occurs where two or more letter-forms are joined as a single glyph. French has the ligatures œ and æ. The German 'esszett' represented as ß is a ligature that is replaced by SS in capitalized spelling and alphabetic ordering. In English, æ is treated as a spelling variant and not an independent letter. But languages like Icelandic, Danish and Norwegian, treat æ as a distinct vowel. Many alphabetic systems add a variety of accents to the basic letters, These accented letters can have a number of different functions, such as, modifying the pronunciation of a letter, indicating where the stress should fall in a word, indicating pitch or intonation of a word or syllable, and indicating vowel length. We discuss more about accents in Section 2.2.

Syllabic Alphabets or Abugidas A syllabic alphabet (also referred to as Abugida or alphasyllabary) is an alphabetic writing system which has symbols for both consonants and vowels. Consonants have an inherent vowel. A vowel other than the inherent vowel is indicated by diacritical marks. The dependent vowels may appear above or below a consonant letter, to the left of or to the right of the consonant letter, and in some cases it may have more than one glyph component that surround the consonant letter. A consonant with either an inherent or marked vowel is called an *akshara*. Vowels can also be written with separate letters when they occur at the beginning of a word or on their own. The vast majority of Abugidas are found in South and Southeast Asia and belong historically to the Brahmi family.

Devanagari (used to write Hindi, Sanskrit, Marathi, Nepali, and some other North Indian languages), Bengali, Gurmukhi script, Tamil, Telugu, Tibetan, Burmese, Thai are some examples of Abugidas. The inherent vowel in Devanagari is /a/. Thus the first consonant in Devanagari क stands for 'ka' and not just the consonant sound 'k'. 'ki' is written by using a diacritic along with 'ka' that stands for the vowel sound 'i'. The same diacritic will be used to get 'ti' from 'ta'. Diacritics are also used for nasalizing the vowel in the syllable (e.g., किँ which represents 'ki' nasalized.). A lonely consonant or dead consonant without any following vowel is sometimes indicated by a diacritic e.g. halant or virama in Devanagari (क्). In a few cases, a separate symbol is used (e.g., ৎ in Bengali is the dead form of the consonant ত When two or more consonants occur together, special conjunct symbols are often used which add the essential parts of first letter or letters in the sequence to the final letter. Such consonants are called half consonants. The final character may retain some similarity with the combining consonants or can be different. Some examples from Devanagari are न् + त = न्त, क् + ष = क्ष. The half forms do not have an inherent vowel. A consonant cluster can be composed of partial parts of the constituent consonants or can have a completely different form. Thus the effective orthographic unit in Abugidas is a syllable consisting of *C*V* (*C* stands for a consonant, and *V* for a vowel) in general.

Syllabaries A syllabary is a phonetic writing system consisting of symbols representing syllables. A syllable is often made up of a consonant plus a vowel or a single vowel. Some examples are hiragana (Japanese), katakana (Japanese), Inuktitut, Cherokee (Tsalagi). Some writing systems like Iberian use a script that is a combination of syllabic and alphabetic. Modern Japanese is written with a mixture of hiragana and katakana, plus kanji.

Logographic writing systems In logographic writing systems, each character or logograph represents a morpheme which is the minimal unit in a language that carries meaning. A logogram may represent a word, or part of a word like a suffix to denote a plural noun. Many Chinese characters are classified as logograms. But no logographic script is comprised solely of logograms. All contain graphemes which represent phonetic elements as well. Each Chinese character represents a syllable and also has a meaning. The phonetic elements may be also used on their own.

2.2 Diacritics

A diacritic is a small sign added to a letter. As we have already pointed out, several types of diacritic [9, 10] markers are used in different scripts. It often alters the phonetic value of the letter to which it is added, but in some cases it may modify the pronunciation of a whole word or syllable, like the tone marks of tonal languages. As we have already discussed, Abugidas and, in some cases, Abjads (such as Hebrew and Arabic script) use diacritics for denoting vowels. Hebrew and Arabic also indicate consonant doubling and change with diacritics.

Diacritics are used in Hebrew and Devanagari for foreign sounds. As noted earlier, Devanagari and related Abugidas use a diacritical mark called a halant to mark the absence of a vowel. The Japanese hiragana and katakana syllabaries use the dakuten and handakuten symbols to indicate voiced consonants.

A letter which has been modified by a diacritic may be treated as a new, individual letter, or simply as a letter-diacritic combination, in orthography and collation.

Some of these languages use diacritics that are considered separate letters in the alphabet, a few of which are mentioned below. Estonian has the following distinct letters õ, ä, ö, ü which have their own place in the alphabet, between w and x. Finnish uses dotted vowels ä and ö which are regarded as individual letters. Hungarian uses the acute and double acute accents. In Spanish, the character ñ is considered a letter, and is collated between n and o. Icelandic uses acute accents, digraphs, and other special letters, which each have have their own place in the alphabet.

On the other hand, some languages with diacritics do not produce new letters. For example, French uses *grave*, *acute*, *circumflex*, *cedilla* and *diæresis*. German uses the *umlauted* vowels ä, ö or ü to indicate vowel modification. Thai has its own system of diacritics derived from Indic numerals, which denote different tones.

2.2.1 Tones

Tone is the use of pitch in language to distinguish words. Many of the languages of South-East Asia and Africa are tone languages and the script contains marks to indicate tones. The tone contour represents how the pitch varies over a syllable for a tone in a tonal language. It is usually denoted by a string of two or three numbers, or an equivalent pictogram. Many languages have tones and there are several schemes for representing tones in the orthography of the language: tone letters, tone diacritics, and superscript numbers. In African languages, usually a set of accent marks are used to mark tone. The most common accent marks are: high tone denoted by acute (as in $\acute{a}$), mid tone denoted by macron (as in $\bar{a}$) and low tone denoted by grave (as in $\grave{a}$. Several variations are common, and often one of the tones is omitted. Omotic languages have a more complex tonal system which are indicated by numbers from 1 to 5. Contour tones are indicated by a pair of numbers e.g., 14, 21.

In Thai (an Abugida) and Vietnamese (an alphabet) tones are indicated with diacritics. In a few cases, a script may have separate letters for tones, as is the case for Hmong and Zhuang. Numerical systems to denote tones are more common in Asian languages like Chinese.

2.3 Word segmentation

Apart from differing scripts, writing systems also differ in other dimensions. In many written languages, the word boundary is explicit. For example, many languages like English and Hindi have spaces marking word boundaries. Arabic uses different letter shapes for word initial, medial and final letters. But word segmentation is a problem in several Asian languages that have no explicit word boundary delimiter, e.g. Chinese, Japanese, Korean and Thai. In Sanskrit, words are joined together by *external sandhi* which involves the process of coalescence of the final letter of a word with the initial letter of the following word.

2.4 Writing systems on the computer

Writing systems on the computer make use of a character encoding for a language which assigns a unique value to represent each of the characters in that language. Different ISO/IEC standards are defined to deal with each individual writing system to implement them in computers (or in electronic form). There are several different encodings like ASCII and the ISO Latin family of encodings which use one byte to represent a character. ISO 10646 is an international standard, by ISO and IEC. It defines UCS, Universal Character Set, which is a very large and growing character repertoire, and a character code for it. Since 1991, the Unicode [1] Consortium has worked with ISO to develop The Unicode Standard ("Unicode") and ISO/IEC 10646 in tandem. Unicode adds rules and constraints such as rules for collation, normalization of forms, and so on. In Unicode, each character, from the writing systems of all languages, is given a unique identification number, known as its *code point*.

While Unicode is a standard for representation of various languages, if one considers the Unicode representation, in many cases there is a difference between a grapheme or code represented in the computer and a visual unit or glyph. This is especially evident in the case of Abugidas, where three different approaches are used in Unicode representation. Regular expression support for a script must take into account the model of representation used for that script.

The Devanagari model, used for most Indian languages, represents text in its logical order and encodes an explicit *halant* (*virama*) symbol. For example, the syllable 'ki' is written in Devanagari by placing the diacritic for the vowel 'i' to the left of 'ka' as in कि, but is represented in Unicode as the code for 'ka' followed by the code for 'i'. The diacritic marker for the vowel 'o' in Bengali surrounds the consonant letter to which it is attached: ক ('ka') with vowel 'o' is represented as 'ko' as in কো, but the Unicode representation of 'ko' is U+0995 U+09CB representing 'ka' and 'o' respectively.

The Thai model represents text in its visual display order. For example, consider the syllable 'ke'. In Thai script, the diacritic for the vowel 'e' comes to the left of the consonant symbol 'ko kai' (ก) as in เก. In Unicode this syllable is represented by U+0E40 U+0E01, where the code for the vowel comes before the code for the consonant, but 'kae' as in แกะ is written as U+0E41 U+0E01 U+0E30, where the vowel marker is indicated by two codes, one coming before and one after the code for the consonant.

The Tibetan script is syllabic with consonant clusters written using groups of characters, some of which are stacked vertically into conjunct characters. Vowels (other than the inherent /a/) are indicated using diacritics. Morphemes are separated by a dot (tsheg). The Unicode representation of the Tibetan script [1] uses one set of codepoints which are used to represent either a stand-alone consonant or a consonant in the head position of a vertical stack, and a second set of codepoints to represent consonants in the subjoined stack position. The code for the vowel sign is always placed after the consonants in the Unicode representation. Thus a syllable is represented by a consonant symbol followed by zero or more subjoined consonant symbols followed optionally by vowel diacritics. The syllable སྒྲུ, is represented by U+0F66 U+DF90 U+0FB2 U+0F74 where the four codes stand for the following four characters, ས་ྒ ྲ ུ, the first being a head consonant('sa'), the next two subjoined consonants ('ga' and 'ra'), and the third a vowel diacritic ('u').

3 Regular expression and multilingual support in databases

We will now turn our attention to reviewing the regular expression support in standard database systems, and then discuss their suitability for multilingual handling.

3.1 Multilingual datatype and collation support

Most databases now have support for storing multilingual characters in fixed and variable length formats, and also have support for representation in Unicode. Further, since the lexicographical order may be different in different scripts, the SQL standard allows one to specify collation sequences to index the data, as well as to correctly sort the data for display order. The database server specifies a default collation sequence to be specified during installation. Most database servers allow collation sequences to be specified at the table and individual column level while creating a database. Collation can be changed using alter database. Database systems like Oracle 10g [12], SQL Server 2005 define different collation sequences for many languages including ones that are case insensitive and accent insensitive. Similar to sort order, match can be case insensitive or diacritic insensitive.

^	Matches start of line
$	Matches end of line
?	Matches zero or one occurrence of the preceding element
*	Matches zero or more occurrences of the preceding element
{n}	Exactly n repetitions of the preceding element
[abc]	Character list, matches a, b or c
[b-g]	Matches b, c, d, e, f or g
[.ce.]	Matches one collation element including digraphs e.g., [.ch.] in Spanish.
[:cc:]	Matches character classes e.g., [:alpha:] matches any character in the alphabet class. Some of the defined character classes are [:alnum:] for all alphanumeric characters, [:digit:] for all numeric digits, [:lower:] for all lowercase alphabetic characters, etc.
[=ec=]	Matches equivalence classes. For example [=a=] matches all characters having base letter 'a'. A base letter and all its accented variations constitute an equivalence class.

Table 1: Additional wildcards supported in Oracle

3.2 Regular expression support in databases

Regular expressions are a powerful tool for matching text data. A regular expression comprises one or more character literals and/or metacharacters.

SQL SQL uses the LIKE operator for regular expressions. This allows one to match literals as well as patterns of a single character or multiple characters. The usage of LIKE is as follows:

select "column_name" **from** "table_name" **where** "column_name" **like** PATTERN

PATTERN can include the following wildcard characters: _ which matches a single character, and % which matches a string of 0 or more characters.

Oracle Oracle SQL [8] has introduced several operators including regexp_like and regexp_substr. Oracle's implementation of regular expressions is based on the Unicode Regular Expression Guidelines and meant to encompass a wide variety of languages. regexp_like has the following syntax: regexp_like (string, pattern, [parameters]);. The optional argument parameters can have different values including i (to match case insensitively) and c to match case sensitively. In addition to the _ and % wildcard characters which are used in like, regexp_like uses regular POSIX and Unicode expression matching. Some of the wildcards used are shown in Table 1. However though there is support for multiple scripts, all scripts are treated uniformly.

3.3 Crosslingual matching

In multilingual databases there is very little support for comparison of strings across different scripts. Equality comparison of strings from different languages can be useful for proper nouns and can be done to strings from an equivalent set of languages under equivalence of individual character sets. Kumaran and Haritsa [5,6] proposed a new SQL operator called LexEQUAL for phonetic matching of specific types of attribute data across languages. It is suggested that the LexEQUAL operator be implemented based on parameterized approximate matching in phoneme space. It requires a G2P (Grapheme to Phoneme) for the concerned languages.

select BookTitle, Author **from** Books **where** BookTitle **LexEQUAL** 'Himalaya'
InLanguages English, Arabic, Marathi, Bengali

will return entries where the BookTitle column has strings that are phonetically close to 'Himalaya' and can be in one of the languages specified. The SemEQUAL operator [7] has been proposed by Kumaran and Haritsa for semantic matching of multilingual attribute data. The query

select BookTitle, Author **from** Books **where** Category **SemEQUAL** ‘History’
InLanguages English, Hindi, French

will return entries where category is ‘History’ for English books, ‘Histoire’ for French books, इतिहास for Hindi books. A variation, ‘SemEQUAL ALL’ is used to retrieve additional entries whose category column contains synonyms of ’History’ in a thesaurus.

3.4 Lexical Query processing

In addition to these, with the growing popularity of text processing, the database engines need to build in support for functional units of words. Microsoft SQL Server 2005 has good support for full-text search (SQL FTS) [4]. This allows efficient querying with large amounts of unstructured data. In contrast to the LIKE predicate, which only works on character patterns, full-text queries perform linguistic searches against this data, by operating on words and phrases based on rules of a particular language. Full Text Searching allows for string comparisons, returning both results and a matching score or weight. Typically, full text index may be enabled on some selected columns in a database table. Four T-SQL predicates are involved in full-text searching: freetext, freetexttable, contains, and containstable. The query

select BookTitle **from** Books **where freetext**(BookTitle, ‘database’)

finds a word having ‘database’ as stem anywhere in the BookTitle column. It matches with “Database Systems” as well as with “Principles of Databases”. freetexttable works similarly, but returns its results in a Table object. contains and containstable offer a much more complex syntax for using a full-text indexed column. Some of the capabilities include search for a given prefix term, and searching for different generations of a word using the formsof term. fomsof accepts two arguments inflectional or thesaurus. The inflectional argument causes match with entries containing the same linguistic stem as each word in the search phrase. The thesaurus argument enables a thesaurus based expansion on the search phrase. For each supported language, there will have to exist a single thesaurus file. For example, the following query

select BookTitle, Author **from** Books **where contains**(BookTitle, ’**formsof** (inflectional, design)’)

retrieved those entries where the BookTitle field contains a word which is any inflected form of ‘design’.

contains(*,‘**formsof**(inflectional, climb) and **formsof**(thesaurus, mountain)’)

will find documents containing inflectional forms of ‘climb’ and all words meaning the same as ‘mountain’ (from thesaurus support). During indexing, language options specify how words or tokens are broken from the text stream and stored in the index. At query time, the search phrases supplied in a query are expanded by the parser before searching the full-text index. This expansion process is performed by a language-specific component called a stemmer and the exact expansions that occur will depend on the language-specific rules that have been used.

3.5 Setting up globalization support environment

Databases that support multiple scripts and languages need to provide a support environment for specifying information about the supported languages. As an example, we discuss the support provided in Oracle for setting up locale and language specific support and configuration.

National language support (NLS) in Oracle provides the ability to choose a national language and store, process and retrieve data in native languages. It also ensures that database utilities, error messages, sort order, and date, time, etc. automatically adapt to any native language and locale. Oracle’s globalization support is implemented with the Oracle NLS Runtime Library (NLSRTL). The NLSRTL has language-independent functions that allow proper text and character processing and language convention manipulations. The behavior of these functions for a specific language is governed by a set of locale-specific data that is identified and loaded at runtime.

The locale-specific data is structured as independent sets of data for each locale that Oracle supports. The locale data specifies language, territory, character set, and linguistic sort and can be customized by the user. It is used to define whether matching is to be case or accent insensitive, the collation sequences, the collating elements, etc. Oracle also supports user-defined characters and customized linguistic rules.

NLS parameters in Oracle can be specified by initialization parameters on the server which override the default. It can also be specified as environment variables on the client, with the 'alter session' statement, and in SQL function in reverse order of priority.

4 Recommendations for multilingual and multiscript support

4.1 Units in different writing systems

When we study the adequacy of the regular expressions to handle scripts in multiple languages we note that the notion of what comprises a character varies across languages. Different writing systems use different graphemic units for representation. A single unit may represent a single consonant, a single vowel, a syllable, a consonant and vowel combination, a letter with accent, or a ligature. Different languages have different semantics associated with a character.

1. In Abjads, a grapheme corresponds to a single consonant. A consonant is the main unit along with a few long vowels. The vowel sounds are mostly left out, but in a few cases, they are indicated by diacritics associated with the consonant unit.

2. In alphabetic writing systems, a grapheme corresponds to a single vowel or a single consonant. Consonants and vowels have equal status as letters. Some scripts contain ligatures, some contain digraphs or trigraphs as collating units. Accents or diacritics combine with letters and can be treated as an accented base unit in some cases, and as separate letters in others. In the former case, a character along with its diacritic may be considered as a single unit, or one may consider them as two different units. Ligatures and multi-character collating units can also be treated as one or two units.

3. In Abugidas, a grapheme corresponds to a phoneme unit which represents a consonant vowel combination, or isolated vowel sounds. A consonant letter by itself has an inherent vowel associated with it, usually /a/, which is overridden by diacritic marks for other dependent vowels. In some cases, dead consonants which are not associated with any vowel, and do not combine with the next consonant letter, are used. Dead consonants are usually marked by the diacritic for halant in Devanagari, Bengali, etc. Even though the consonant vowel combination is visually a single unit, this unit is obtained by composing the consonant symbol with diacritic marks. Thus these units are decomposable. Some of the Abugidas also use consonant clusters which are represented by ligatures. Orthographically such ligatures are a single unit, even though they are formed by a sequence of consonants. Half consonants are part of a consonant cluster. For some of these languages, there are four types of units: consonant only (without the inherent vowel), independent vowel, consonant(s) plus vowel unit forming a syllable, and diacritic marker/vowel marker.

4. In syllabic writing systems, a grapheme corresponds to a syllable.

5. In logographic writing systems, a grapheme corresponds to a morpheme. However in most logographic scripts many of the characters have associated phonemes representing syllables.

6. In all languages, the word as a lexical unit plays a very important role. Morphological analysis of a word lets us find the root form which is also called the stem or lemma, and the inflectional affixes.

Based on the above discussion, we identify the following units which are meaningful for different scripts: collating unit, vowel, vowel mark diacritic, consonant, consonant cluster, ligature, single live consonant, dead consonant, half consonant, syllable, morpheme, diacritic, accent diacritic, tone diacritic and tonal symbols. At the lexical level, the important units are word, stem, suffix, and prefix.

A subset of these units are typically meaningful for a given script and language. These more natural units can be defined as the default units for those scripts. Table 2 shows the meaningful units for the different writing systems, and Table 3 shows the composition of some of the character classes for the different types of scripts.

In addition, when we have a multilingual database where different entries may be in different languages, it is important to define crosslingual matching capabilities. The proposed crosslingual operator LexEQUAL discussed earlier does phonological mappings across multiple scripts. Similar to the LexEQUAL operator, it is also useful to design crosslingual pattern matching operators.

4.2 Recommended character classes for monolingual match

On the basis of the units found in different languages, we identify a few symbols that stand for a single instance of a given unit. We use the character class notion used in Oracle to represent the different units. The SQL wildcard "_" matches a single character. We equivalently introduce the character class [:char:], and use either of them to denote a single character unit. The interpretation of a character in different writing systems is recommended to be as follows:

Abjads:	a vowel or a consonant with optional diacritic marks
Alphabets:	a single consonant or vowel letter or a ligature or a multi-character collating unit or a letter with the associated diacritics
Abugidas:	independent vowel or diacritic vowel or consonant or consonant cluster with or without vowel diacritic
syllabaries:	a syllable
logograms:	a morpheme or a phonetic syllable

Sequences of characters are represented in the following ways:

`%` or `[:char:]*`	is taken to be a sequence of zero or more character units.
`[:char:]+`	is taken to be a sequence of one or more character units.
`[:char:]?`	is taken to be zero or one character unit.

In addition to these, we propose the following more specific wildcards:

`[:c:]`	is used to represent a consonant or a consonant cluster, or a collating unit consisting of two or more consonants.
`[:v:]`	is used to represent a vowel or a ligature of vowels in case of Abjads and alphabets, and an independent vowel in case of Abugidas.
`[:hc:]`	is used to represent a half consonant for some Abugidas.
`[:dc:]`	is used to represent a dead consonant for some Abugidas.
`[:syll:]`	is used to represent a syllable. This is meaningful for syllabaries, is used for independent vowels or consonant clusters in case of Abjads and Abugidas, and for phonetic units in logographs.

The following classes are recommended for diacritic symbols:

`[:dia:]`	is used to represent any diacritic symbol, or a set of diacritic marks.
`[:acc:]`	is used to represent an accent diacritic.
`[:dvow:]`	is used to represent a vowel diacritic.
`[:tone:]`	is used to represent a diacritic representing a tone. This is also used to represent a separate tone symbol as well as numeric tone representations.

The following lexical units are also recommended. These units are meaningful in almost all languages, but it is especially useful to identify these entities in languages where the written language does not use word boundaries,

	Abjads	Alphabets	Abugidas	Syllabaries	Logograms
Character Units	consonant, independent vowel, consonant with dia.	consonant, vowel, digraph, accented unit, ligature	consonant, vowel, syllable, vowel dia. cons cluster	syllable	morpheme
independent vowel	long vowels only	yes	yes		
vowel diacritic	yes(rare)		yes		
ligature	yes	yes		yes	yes
consonant	yes	yes	yes		
consonant cluster			yes (dn)		
live consonant dead consonant half consonant			yes (dn) yes (dn) yes (dn)		
syllable	yes		yes	yes	yes
diacritic	yes	yes	yes	yes	
accent dia.		yes			
tone dia.		yes	yes(Thai)		
morpheme	yes	yes	yes	yes	yes

Table 2: Table showing the meaningful units in different writing systems (dn denotes Devanagari)

Abjads	consonant	=	consonant + consonant with vowel dia
	vowel	=	independent vowel
	syllable	=	consonant with optional diacritic + independent vowel
	char	=	syllable
Alphabets	char	=	consonant + vowel + digraph + accented unit + ligature
Abugidas	consonant	=	live consonant + dead consonant + half consonant
	consonant cluster	=	live consonant + half consonant* live consonant
	syllable	=	independent vowel + consonant cluster with optional diacritic
	vowel	=	= independent vowel + vowel dia.
	diacritic	=	vowel dia + tone dia
	char	=	vowel + consonant + consonant cluster + syllable
Syllabary	char	=	syllable
Logogram	char	=	morpheme

Table 3: Rules showing relations between the different units in different writing systems.

as in Thai, Chinese and Japanese. Identifying these units would require a morphological analyzer or a stemmer for the languages, as well as a text segmentation module for languages with no delimiters.

`[:morph:]`	is used to represent a morpheme. This is the basic unit for logographs. But such units are meaningful for all languages.
`[:word:]`	matches a single word
`[:stem:]`	matches the stem of a word
`[:suffix:]`	matches a suffix
`[:prefix:]`	matches a prefix

Apart from the character classes we define the following matching units:

`[=a=]`	matches all letters with the base character 'a' and includes all diacritics of 'a', as well as upper and lower cases of 'a'. It also matches multiple diacritics on a.
`[=a=:v]`	matches all letters with the base character 'a' with optionally vowel diacritics.
`[=a=:acc]`	matches all letters with the base character 'a' with optionally accent diacritics.
`[=a=:tone]`	matches all letters with the base character 'a' with optionally tone diacritics. Tones may also occur as numeric superscripts or as separate tone symbols. In the case where they are separate symbols, we may use [:tone:].

Note that in order to do these matches, the database system has to know not only the code points which correspond to diacritics like accents, vowel markers, etc., but also there will be language specific differences about the position of the diacritic in the text representation with respect to the base letter.

4.3 Monolingual match operators

Most database systems allow the specification of a collation sequence. For example, by appropriately setting the value of nls_sort in Oracle one can do accent or case insensitive matching. PostgreSQL additionally uses a special keyword ILIKE to make the match case insensitive.

We recognize that various types of case and accent insensitive matching are often very useful, and we recommend that apart from being able to specify the matching scheme, databases should have distinct operators for various levels of matching: literal matching, case insensitive matching, diacritic insensitive matching, as well as phonetic matching. Phonetic matching can also be done across languages and is discussed in Section 4.4.

x LIKE y : x matches y according to the current match environment
x ILIKE y : x matches y insensitive to case
x DILIKE y : x matches y insensitive to diacritics

The semantics for these operators must be specified in the locale for the language. Some examples for successful matches in English are Book ILIKE book and naive DILIKE naïve. For languages like Hindi and Bengali, diacritic insensitive matching should match on the consonants only. Some examples of successful matches might be हिरन DILIKE हरिनी, and कर्म DILIKE करम. While defining the match levels we have to take into account the characteristics of the script. For example, in scripts where the accented forms are treated as different characters, accent insensitive matching with respect to these characters is not common.

4.4 Crosslingual operators

The LexEQUAL operator [5] has been proposed for crosslingual matching. We wish to extend crosslingual matching to pattern matching. Our proposed operator LexLIKE is modeled based on LexEQUAL, and can be used for phonological matching between two strings or between a string and a pattern belonging to one language or two different languages. In case of crosslingual match, the user has to specify the set of target languages, or * to specify any language, and an optional Threshold parameter can help tune the degree of match between the string and the pattern. The syntax is proposed to be x LexLIKE y [Threshold α] [InLanguages ...].

Suppose one is looking for books in different languages whose titles reference 'Manas Sarovar' (a holy lake in Tibet variously transliterated in English as 'Manas Sarobar', 'Manas Sarovar', 'Mansarovar', 'Manasarovar', 'Manasarowar', etc., written in Hindi usually as मानसरोवर, and in Bengali as মানস সরোবর). The following query

select BookTitle **from** BOOKS **where** BookTitle **LexLIKE** ”Man% Sa[:syll:][vb]ar” Threshold 0.75
InLanguages Bengali, Hindi, English

should match the different renderings of 'Manas Sarovar' in the language sets indicated. Some examples of match using the LexLIKE operator for monolingual match in Bengali are পাখি LexLIKE পাখী, and পাখি LexLIKE পক্ষীThreshold 0.7.

The LexLIKE operator can be implemented if the grapheme to phoneme mappings (G2P) for the languages are specified. The G2P is used to convert the pair of expressions being compared to a canonical form. Approximate string matching can be performed in the phonemic space akin to the LexEQUAL operator [6].

In addition to crosslingual matching on exact words, crosslingual semantic matching is very useful for which the SemEQUAL operator had been proposed. We propose an equivalent notation for a crosslingual semantic matching operator which allows us to search a part of a field by using 'contains'. This is illustrated by the following example.

select BookTitle **from** Books **where contains**(BookTitle, [=rain=:syn])
InLanguages English, Bengali, German

will match with all BookTitles that contain synonyms of 'rain' from the documents in the specified languages. For example, this will match titles containing the word বৃষ্টি in Bengali, वर्षा in Hindi, and Regen in German.

4.5 Lexical processing

For lexical processing, we use the contains [4] predicate from Microsoft SQL server full text search. This allows us to match some of the words in a field. The contains predicate has the basic syntactic form **contains** (col_list, '<search condition>'). This predicate is true if any of the indicated columns in the list col_list contains terms which satisfy the given search condition. A search condition can be a regular expression denoting a word, a phrase, or can be a prefix, suffix, or stem. We propose the following lexical units that can be used:

`[:stem=book]`	matches all words with stem as 'book'.
`[:prefix=anti]`	matches all words with a prefix 'anti'.
`[:suffix=ment]`	matches all words with a suffix 'ment'.
`[:syn=book]`	matches all words synonymous to 'book' from the thesaurus.

Example:

select BookTitle **from** Books **where contains**(BookTitle, [:stem=discovery])

will match with all BookTitles that contain inflections of 'discovery', e.g., 'discoveries'.

select BookTitle **from** Books **where contains**(BookTitle, [:syn=discovery])

will match with all BookTitles that contain synonyms of 'discovery' in the thesaurus, like 'breakthrough' and 'uncovering'.

select BookTitle **from** Books **where contains**(BookTitle, [:stem=ভাষা] and [:suffix=কে])

will match with all BookTitles that contain a word which has stem ভাষা and has কে as one of its suffixes, and thus will match with a title containing the word ভাষাটাকেই, a word whose stem is ভাষা and which has three suffixes, টা, কে and ই .

5 Conclusion

In this paper we have examined different scripts and proposed different character classes to facilitate construction of regular expressions for different scripts. We also study the support that some databases provide for linguistic matching with respect to the morphology of the words in free text, as well as semantic matching operators. Based on the study and the characteristics of different languages, we recommend that lexical classes for these units be used which can be the basis of matching. We have also discussed crosslingual matching operators that compare patterns across different languages. To check for semantic equivalence, semantic matching needs to be supported with the help of a thesaurus. Cross lingual semantic matching would require multi-language dictionaries and thesauri. However we have not discussed about the implementation of these operators in this paper. Many of these can be implemented on top of some existing database systems. For example, corresponding to every language, the appropriate character classes can be defined in the style file for the language. But the efficient implementation of some of the operators can be facilitated by intrinsic support by database engines, so that efficient index structures are created.

6 Acknowledgment

The author gratefully acknowledges the role of Professor Jayant Haritsa for the conception of this paper, as well as his help in all stages of preparing this paper.

References

[1] The Unicode Consortium. *The Unicode Standard Version 5.0.* Addison Wesley, 2006.

[2] Florian Coulmas. *The Writing Systems of the World.* Oxford, Blackwell, 1991.

[3] Peter T. Daniels and William Bright, editors. *The World's Writing Systems.* Oxford University Press, 1996).

[4] James R. Hamilton and Tapas K. Nayak. Microsoft sql server full-text search. *IEEE Data Engineering Bulletin*, 24(4):7–10, 2001.

[5] A Kumaran and Jayant R. Haritsa. Lexequal: Multilexical matching operator in sql. In *Proceedings of the 2004 ACM SIGMOD*, pages 949–950, Paris, France, 2004.

[6] A Kumaran and Jayant R. Haritsa. Lexequal: Supporting multiscript matching in database systems. In *9th International Conference on Extending Database Technology, EDBT 2004*, pages 292–309, Greece, 2004. Springer.

[7] A. Kumaran and Jayant R. Haritsa. Semequal: Multilingual semantic matching in relational systems. In *Database Systems for Advanced Applications, 10th International Conference, DASFAA*, pages 214–225, Beijing, China, 2005. Springer.

[8] Kevin Loney. Oracle database 10g: The complete reference, 2005.

[9] J.C. Wells. Orthographic diacritics and multilingual computing. In *Language Problems and Language Planning*, volume 24.3, Feb 2001.

[10] Wikipedia. Diacritic. http://en.wikipedia.org/wiki/Diacritic.

[11] Wikipedia. Writing system. http://en.wikipedia.org/wiki/Writing_system.

[12] Weiran Zhang. Pattern matching with multilingual regular expressions. In *24th Internationalization & Unicode Conference*, Atlanta, 2003.

Mapping and Structural Analysis of Multi-lingual Wordnets

J. Ramanand, Akshay Ukey, Brahm Kiran Singh, Pushpak Bhattacharyya
ramanand@it.iitb.ac.in, {akshayu,brahm,pb}@cse.iitb.ac.in
Indian Institute of Technology, Bombay

Abstract

In this paper, we present observations on structural properties of wordnets of three languages: English, Hindi, and Marathi. Hindi and Marathi, spoken widely in India, rank 5th and 14th respectively in the world in terms of the number of people speaking these languages. The observations suggest the existence of the 'small world' property in wordnets and also lend credence to the belief that the world of concepts, which the words are manifestations of, share common properties across languages. These concepts are represented by synsets (sets of synonymous words) in wordnets. Therefore, it makes sense to link the synsets of wordnets of different languages to create a global wordnet grid. In fact, the EuroWordnet project is already doing so for a number of European languages. We too report our work on linking English, Hindi and Marathi synsets. The first task – linking of English and Hindi wordnets – requires clever ideas on mapping accurately the synsets of the two languages. The second task – relatively easier due to the close correspondence between Hindi and Marathi – reduces to programmatically borrowing lexico-semantic relations from the Hindi wordnet into the Marathi wordnet, since the Marathi wordnet is largely aligned with the Hindi wordnet. To the best of our knowledge, ours is the first in-depth investigation into lexical knowledge networks of multiple languages with a view to evaluating them, and also the first attempt to create a multiwordnet involving two major Indian languages and English.

1 Introduction

Princeton wordnet ([MB90]) is a lexical knowledge network of English words. Its growing popularity as a useful resource for English and its incorporation in natural language tasks has prompted the creation of similar wordnets in other languages as well. The English wordnet is maintained manually by a team of lexicographers and computer scientists. This manual method sacrifices speed for quality and may be impractical for efforts to bootstrap new wordnets in other languages, especially if there is a lack of linguistic support. This has therefore motivated research into using existing wordnets in Indian languages like Hindi to create data for languages like Marathi which are part of the same language family. Also, matching synsets from wordnets in unrelated languages such as English and Hindi provide benefits in multi-lingual contexts.

The increasing number of wordnets has also sparked off interest in understanding the structure and properties of these wordnets with a view to characterising and comparing them. A starting point is to study some statistical properties by considering a wordnet as a graph. This yields interesting observations on degree distribution,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

nature of clustering, and shortest path lengths of the graphs. These help understand the nature of relatedness of words in languages, enhance the usefulness of wordnets and could also help design these databases better.

The roadmap of this paper is as follows: Section 2 discusses the related work. Section 3 introduces the basics of wordnet relevant to this discussion. Section 4 is on the structural properties of wordnets in English, Hindi, and Marathi. Section 5 outlines an automatic algorithm for bridging synsets of the Princeton English wordnet (EWN) and the Hindi wordnet (HWN). Section 6 gives a programmatic way of establishing relations in the Marathi wordnet (MWN) using HWN. Section 7 discusses the results and observations and suggests directions for future work.

2 Related work

Our study of the structure of wordnets has been motivated by the desire to compare and evaluate many different lexical knowledge networks, *viz.* wordnets (Princeton wordnet [MB90], EuroWordnet [VR98], Hindi wordnet [NC02] *etc.*), Conceptnet [LS04], Hownet [ZQ00], Mindnet [VR98], VerbNet [KS05], IEEE SUMO [NP01] and ontologies [GU04]. Another source of motivation has been the 'small world' property observed in complex networks [WA06, WC03] (do wordnets possess such properties?).

Mapping/Linking/Bridging of knowledge networks has always been a problem of great interest. Quite a few wordnets in Eurowordnet [VO98] have been created by leveraging existing wordnets. [PB02] gives details of methodologies used for this purpose. [SR06] is interesting from the point of view of linking wordnets of two sister languages (Hindi and Marathi).

Mapping synsets of one wordnet to the synsets of another involves word sense disambiguation (WSD) [YA92] as an important sub-task. [LL00] discusses a set of automatic WSD techniques for linking Korean words collected from a bilingual machine readable dictionary (MRD) to English WordNet synsets. An example of work on aligning one wordnet (Italian) with another (Princeton) is in [PB02]. Many of these mapping efforts have used the idea of the Lesk algorithm [LE86].

The mapping of Princeton wordnet to other knowledge networks has also received attention in the lexical knowledge network community [NP03].

A multi-lingual linked structure of wordnets is the goal of the Global Wordnet effort. Wordnets of many languages of the world can be found at the website (http://www.globalwordnet.org) of this endeavour.

3 A primer on wordnet

A wordnet for a language is a linked structure of concept nodes represented by sets of synonymous words called *synsets*, which are connected through lexico-semantic relations. For the user, the wordnet is a rich lexicon-like database that is queried using an API or a browser to obtain information about words. The basic idea of a wordnet can be presented through the lexical matrix. The rows in the matrix represent concepts, while the columns stand for words. Thus, a particular column represents the polysemy of a word, while a particular row depicts synonymy. The entries of a row define a synset. An example is shown in Table 4. Here, *board* is a polysemous words with several meanings. The synset for the sense *'a stout length of sawn timber'* consists of the words *board* and *plank*.

For the discussions that follow, we give examples from Hindi and Marathi wordnets – with adequate translations in English – to keep the multi-lingual flavour.

Synsets are the building blocks of wordnets. The principles of *minimality, coverage* and *replaceability* govern the creation of the synsets:

1. **Minimality**: Only the minimal set that uniquely identifies the concept is used to create the synset, *e.g.*

S.No.	(Senses, Words)	**Board**	**Plank**	**Table**	**Card**	**Gameboard**
1.	A committee having supervisory powers	x				
2.	A stout length of sawn timber	x	x			
3.	A surface for board games	x				x
4.	A table for meals	x		x		
5.	An endorsed policy of a political party		x			
6.	A set of data arranged in rows and columns			x		
7.	A printed circuit board	x			x	

Table 4: Example of entries in the English wordnet lexical matrix

{`ghar`, `kamaraa`, `kaksh`}[1] (*room*).
The Hindi word `ghar` is ambiguous and cannot by itself uniquely denote the concept of a *room*. For instance, it could also mean *house*, *native country*, or *family*. The addition of `kamaraa` and `kaksh` (also meaning *room*) to the synset brings out this unique sense.

2. **Coverage**: The synset should contain all the words denoting a concept. The words are listed in order of (decreasing) frequency of their occurrence in the corpus. *e.g.* {`ghar`, `kamaraa`, `kaksh`} (*room*)

3. **Replaceability**: The words forming the synset should be mutually replaceable in a specific context. Two synonyms may mutually replace each other in a context C, if the substitution of the one for the other in C does not alter the meaning of the sentence. Consider,
   {`svadesh`, `ghar`} (*motherland*)
   `amerikaa meN do saal bitaane ke baad shyaam svadesh/ghar lauTaa`
   Literal translation: America in two years stay after Shyam motherland returned
   'Shyam returned to his motherland after spending two years in America'
   The replaceability criterion is observed with respect to synonymy (semantic properties) and not with respect to the syntactic properties (such as subcategorization).

To explicate the meaning, a synset is associated with a gloss of definition and an example sentence.

3.1 Lexico-Semantic relations

A wordnet incorporates semantic and lexical relationships among synsets.

3.1.1 Semantic Relations

Semantic relations link two synsets. Examples of these are:

1. **Hypernymy and Hyponymy** encode semantic relations between a more general term and specific instances of it.
   {`belpatra`, `belpattii`, `bilvapatra`} *'a leaf of a tree named `bel`'* → {`pattaa`, `paat`, `parN`, `patra`, `dal`} *'leaf'*
   Here, `belpatra` (*a leaf of a tree named `bel`*) is a kind of `pattaa` (*leaf*). `pattaa` (*leaf*) is the hypernym of `belpatra` (*a leaf of a tree named `bel`*), and `belpatra` (*a leaf of a tree named `bel`*) is a hyponym of `pattaa` (*leaf*).

[1]For transliterations of the Devanagari script used in Hindi and Marathi, refer to http://www.aczoom.com/itrans/ and http://en.wikipedia.org/wiki/ITRANS

2. **Meronymy and Holonymy** express the part-of relationship and its inverse.
   {`jaR`, `muul`, `sor`} *'root'* → {`peR`, `vriksh`, `paadap`, `drum`} *'tree'*
   Here, `jaR` (*root*) is the part of `peR` (*tree*), implies `jaR` (*root*) is the meronym of `peR` (*tree*) and `peR` (*tree*) is the holonym of `jaR` (*root*).

3. **Entailment** is a semantic relationship between two verbs. Any verb *A* entails a verb *B*, if the meaning of *B* follows logically and is strictly included in the meaning of *A*. This relation is unidirectional. For instance, *snoring* entails *sleeping*, but *sleeping* does not entail *snoring*.
   {`kharraaTaa lenaa`, `naak bajaanaa`} *'snore'* → {`sonaa`} *'sleep'*

4. **Troponymy** is a semantic relation between two verbs when one is a specific 'manner' elaboration of another. For instance,
   {`dahaaRanaa`} *'to roar'* is the troponym of {`bolanaa`} *'to speak'*.

5. **Cross-linkage between different parts of speech**: Some wordnets like the HWN also link synsets across different parts of speech.

3.1.2 Lexical Relations

Lexical relations link two specific words in two different synsets. Examples of these are:

1. **Antonymy** is a lexical relation indicating 'opposites'. For instance,
   {**`moTaa`**, `sthuulkaay`} (*'fat'*) → {**`patlaa`**, `dublaa`} *'thin'*.
   `patlaa` (*thin*) is the antonym of `moTaa` (*fat*) and vice versa.

2. **Gradation** is a lexical relation that represents possible intermediate states between two antonyms. *e.g.*, {*`jawaanii`*} *'youth'* between {*`shaishav`*} *'childhood'* and {*`buDaapaa`*} *'old age'*.

4 Structural properties of wordnets

Wordnets can be represented as graphs and studied for graphical properties like *average shortest path*. 'Small World' properties ([WA06]), which have been observed in complex and large networks ranging from the likes of citation graphs to the web graph to biological oscillators, are also seen to occur in wordnets.

The *Small World* nature of these graphs means that despite the formidable size of the graphs, the average shortest path between nodes is small. Another statistic is that of *cluster coefficient* which measures whether "friends" of a node are also "friends" of each other. Results show that this is indeed true of wordnets, indicating a grouping of concepts that could suggest the presence of "clouds" or "cores" in wordnets. The last measure is that of the shape of the degree distribution graph of these nodes. Each wordnet shows a characteristic power-law distribution in its degree distribution, which indicates the presence of a few highly-connected hubs and a majority of nodes with much lesser connectivity. This also explains why the average path length is small – this can be attributed to the 'popular' hub concepts.

The observations were carried out on EWN, HWN, and MWN. Only semantic relations were considered for the small world properties as these relations link one synset to another. In contrast, lexical relations such as antonymy link two specific words within different synsets and do not apply to the other words in those synsets. All the links in the graphs are directed. For EWN, nouns and verbs were studied individually, as each set is a large graph in its own right and is separately provided in the wordnet database. The data is summarised in Table 5.

Wordnet	No. of Nodes	No. of Edges	Edges per Node
English WN v2.1 (Nouns)	84709	226483	2.674
English WN v2.1 (Verbs)	13769	29883	2.167
Hindi WN v1.0	24041	80138	3.333
Marathi WN v1.0	21116	39136	1.853

Table 5: Surface level statistics of wordnets

4.1 Degree Distribution

We compute a distribution function $P(k)$ which is the proportion of total number of nodes that have exactly k edges emanating from them ([WC03]). The function was calculated as follows:

1. Get degree k for each node
2. For each unique k, count the total number of nodes whose degree is k
3. For each unique k, $P(k)$ = (degree occurrences/total number of synsets)

Plotting $P(k)$ vs. k shows a power-law characterised by an exponent γ. A log-log plot shows a straight-line, indicating the scale-free nature of the graph. This shape was seen repeated for all wordnet graphs. A sample graph (for HWN) is shown in Figure 1.

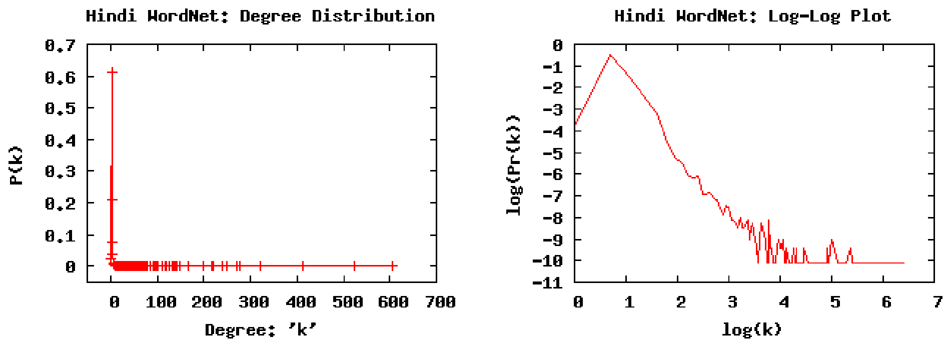

Figure 1: Degree Distribution and Log-Log plot for HWN

By measuring the slope of the line in the log-log plot, we obtained exponents γ as shown in Table 6.

Wordnet	English WN (Nouns)	English WN (Verbs)	Hindi WN	Marathi WN
Exponent(γ)	-2.063	-2.224	-2.592	-2.841

Table 6: Exponents for the Degree Distributions

These results show that while most of the nodes in the graphs have very low degree, a few nodes with very high connectivity exist. These concepts are abstract or common concepts that tend to have many specific instantiations and are richly connected to other concepts. For instance, in EWN, the synset for the concept $\{$*person, individual, someone, somebody, mortal, soul*$\}$ has 403 relations, the synset for $\{$*city*$\}$ has 666 relations, while nodes such as $\{$*tour-de-force*$\}$ or $\{$*oversight, inadvertence*$\}$ have just one relation (to their parent). Similarly, the synsets equivalent to {*person, individual, ...*} in HWN and MWN have 607 and 626 relations respectively, the highest in each wordnet. The lower exponents (by absolute value) in the English wordnet show that the gap between proportions of degree-poor nodes and degree-rich nodes is lower than in the newer wordnets. This is possibly due to the relative maturity of EWN. Wordnet building involves identifying new synsets and creating appropriate links among synsets, which remains an ongoing task. A new synset will at least be linked to its parent hypernym. A synset in a more mature database is likely to have greater 'richness' by being linked to more synsets, whereas in a new wordnet, a greater proportion of synsets will only have the parental link. In the newer wordnets, the hubs are much more important and vital to the network than in the older database. This can be one indication of the maturity of a wordnet.

4.2 Cluster Coefficient

Cluster Coefficient C_i for a node i (with degree k_i) of a directed graph is defined as follows [WA06]:

$$C_i = \frac{|E(\Gamma_i)|}{2 \times \binom{k_i}{2}}$$

where Γ_i is the subgraph made of the neighbours of i, $|E(\Gamma_i)|$ is the number of edges of the subgraph, and $2 \times \binom{k_i}{2}$ is the total number of possible edges in Γ_i.

One extreme is where no neighbour of a node is connected to other neighbours of that node giving $C_i = 0$, whereas at the other end, each neighbour is adjacent to every other neighbour, thus forming a clique and giving $C_i = 1$. The cluster coefficient for the entire graph is found by averaging cluster coefficients for its nodes.

For the wordnets, the results are shown in Table 7. The results show that the coefficient is much higher than would be possible for a random graph, where it would be closer to $1/N$ (where N is the number of nodes). In EWN, the nodes with smaller degrees (usually ≤ 5) tend to have a higher C_i, while the degree-rich hubs have very low C_i as it is very unlikely that many of their neighbours will be related to each other. In fact, diverse groups connect to each other via these hubs. It is also seen that synsets pertaining to a specific domain such as the synset for {*American_football*} tend to have greater C_i. The newer wordnets have lower clustering coefficients as the relations structure among synsets is not very rich.

Wordnet	English WN (Nouns)	English WN (Verbs)	Hindi WN	Marathi WN
Cluster Coefficient	0.526	0.632	0.268	0.358

Table 7: Cluster Coefficients

4.3 Shortest Path

The shortest path length between two vertices i and j in a graph is the smallest number of edges required to traverse from i to j. Further, the shortest lengths between all pairs in the graph are averaged to produce the average length for the graph. The results are summarised in Table 8. The average length is fairly small for graphs of these sizes. The hubs of high degree are responsible for these short distances by being well-connected. The length in HWN and MWN is smaller, primarily because of the relatively smaller size of the graphs.

4.4 Link Distribution in wordnets

Wordnets have a collection of relations of different types that connect synsets. These are not standardised, but the nature of relations is fairly similar across different wordnets as the expectations from them are common. From the results (Figure 2), it can be seen that the taxonomic relations *i.e.* hypernymy/hyponymy usually dominate.

Wordnet	Average Shortest Path	Median Avg. Shortest Path	Std. Dev. Shortest Path	Maximum Shortest Path
English WN (Nouns)*	8.878	8.779	7.174	20
English WN (Verbs)	9.611	9.399	7.997	27
Hindi WN	4.378	4.339	2.639	15
Marathi WN	4.255	4.132	0.187	20

(*A 10 % sample was used for calculation)

Table 8: Average Shortest Path for the wordnets

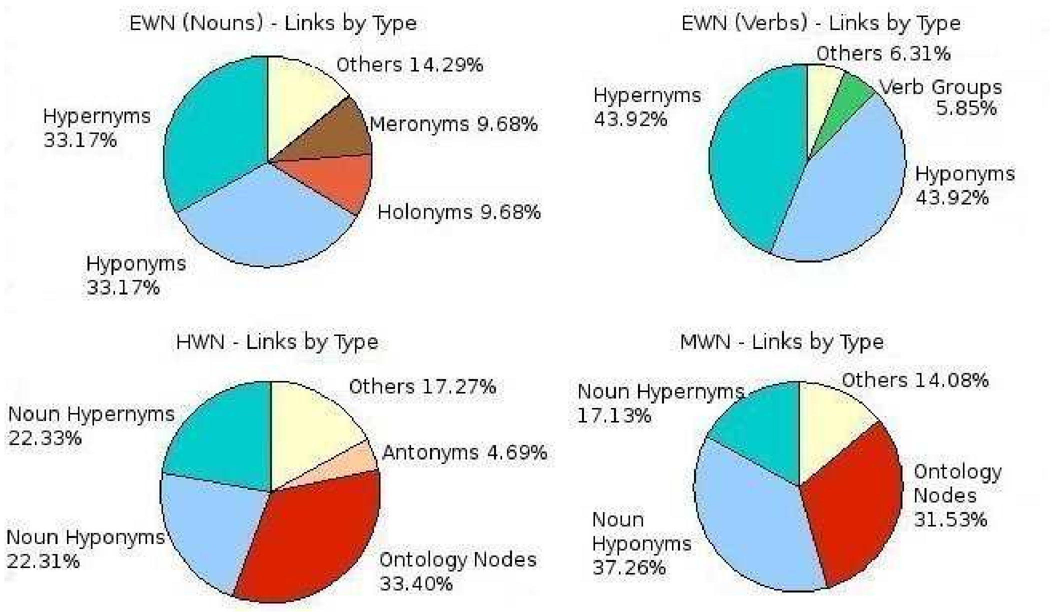

Figure 2: Distribution of Link Types for wordnets

4.5 Synset Sizes

Each synset entry has one or more words in it, which are (near) synonyms for each other and have that specific concept as their meaning. Figure 3 shows the distribution of the number of nodes with different synset sizes. This can be taken to be a distribution of synonymy in the languages. We see that all the wordnets show very similar graphs (the English wordnet has a greater number of short synsets just because of the size of the database), indicating that there seems to be no significant linguistic differences in this distribution.

5 Mapping English and Hindi wordnet synsets

5.1 Intuition

The algorithm takes as input an English synset and produces as output the best matching Hindi synset. First, a set of *candidate synsets* is obtained by finding the Hindi translations of the first word in the input synset and then finding the Hindi synsets that contain one or more of these translations in them. The first word in an English synset best represents the sense of the synset ([MB90]). Hence, the Hindi synsets which denote senses closely related to that denoted by the input synset are likely to contain some translation of this first word.

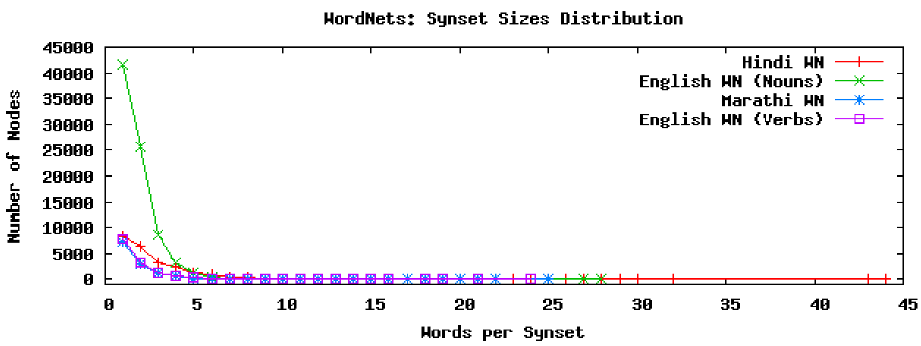

Figure 3: Synset Sizes Distributions for wordnets

After finding the *candidate synsets*, the actual weighting procedure starts. This involves the following steps:

1. Find the hypernymy hierarchies of the *candidate synsets* and call them the *candidate hierarchies*.
2. Traverse the hypernymy hierarchy of the input synset and for each synset in the hierarchy, find the Hindi translations of the words occurring in the synset.
3. Find the synsets occurring in the *candidate hierarchies*, which contain any of these translations. Increase the weights of these hierarchies if such a match is found.

At the end of the weighting procedure, the *candidate synset* corresponding to the hierarchy with the maximum weight is returned as output. In case there is no *candidate synset*, the string "No match found" is returned as output.

The hypernymy hierarchy is employed for many word sense disambiguation endeavours ([LE86]). Hence it was the natural choice for this algorithm too.

5.2 Algorithm

Following are the steps to find a match for a given English synset:

1. The first word of the given English synset is extracted and all the Hindi translations of this word are found out from a bilingual English-Hindi dictionary.
2. All the Hindi synsets which contain any of these translations are determined. They are called the *candidate synsets*.
3. The hypernymy hierarchies of these *candidate synsets* are obtained. They are called the *candidate hierarchies*.
4. The hypernymy hierarchy of the given English synset is obtained.
5. For each synset occurring in the English hypernymy hierarchy, the Hindi translations of all the words occurring in it are found out.
6. These resulting Hindi words are then searched for matches in the *candidate hierarchies*. If a match is found, the weight of the *candidate hierarchy* is increased. Initially, the weights of all the *candidate hierarchies* are set to zero. The increment is a function of:

(a) The level of the Hindi synset in the *candidate hierarchy*, where the match is found. The level of a synset in its hierarchy is defined as the number of synsets by which the synset is separated from the *candidate synset* in the hypernymy hierarchy. The level of the original *candidate synset* is set to 1, that of its hypernym is 2 and so on.

(b) The level of the English synset being considered in the English hierarchy. The level is defined in a similar manner as above: it is the number of synsets by which the present synset is separated from the original synset. The level of the original synset is set to 1.

7. The total weight of each *candidate hierarchy* is computed depending on the number of matches thus found.

8. The *candidate synset* whose *candidate hierarchy* has the maximum weight is mapped to the input English synset.

The following points are to be kept in mind:

1. The increment awarded for a match at lower levels in the hypernymy hierarchy is more than that awarded when the synsets involved are at higher levels. This is because the synsets having higher levels are farther off – in terms of the *sense denoted by them* – from the original synset.

2. The increment should depend on the levels of the English and Hindi synsets in consideration, and that too in a symmetric manner.

3. Due to the limitations of the lexical resources, the algorithm employs substring matching technique.

The function used to increment the weightage of the *candidate hierarchy* is:

$$\textit{Increment} = \frac{[(15 - m) + (15 - n)]}{2}$$

where,

m: The level of English synset in its hierarchy, whose translation has found a match in some *candidate hierarchy*.

n: The level of the Hindi synset in the *candidate hierarchy*, where the match is found.

Justification: As described above, a symmetric function was required, which decreased in value as the levels of the synsets increased. The above menioned function was heuristically chosen. The number 15 was chosen since of all the English synsets whose matches were found, the maximum depth of the hypernymy hierarchy was found to be 15. Also, the maximum shortest path between any 2 synsets in the Hindi wordnet is 15 (Table 8).

Figure 4 illustrates the working of the algorithm. The input synset is {*substance*, *matter*}. Two of the *candidate synsets* are shown: {`arth, abhipraay, aashay, matalab, bhaav, maane, taatpary`} and {`padaarth`}, along with their *candidate hierarchies*. The *candidate hierarchy* corresponding to the synset {`padaarth`} has more number of matches as compared to the number of matches for the other *candidate hierarchy*. The final weight of the *candidate hierarchy 1* is 40.5 and that of the *candidate hierarchy 2* is 108.5. Hence, the synset {`padaarth`} is mapped to the given English synset. Please note that for the purpose of matching, substring matching technique is employed.

5.3 Results

We have mapped the complete EWN V2.0 to the HWN. Approximately 6500 mappings have been checked manually. If we insist on exact matches, we get about 10% accuracy[2], whereas matching with near synonyms

[2] The accuracy is based on the judgement of a human expert, who was involved in building the Hindi WordNet itself.

(i.e. match with immediate hypernyms are accepted) yield about 25% accuracy. The reasons for the low accuracy are (i) a much larger number of English synsets, (ii) the relative immaturity of the Hindi wordnet, and (ii) the deficiencies in the English to Hindi dictionaries. We believe our approach is powerful and interesting, and with the improvement and enrichment of Hindi lexical resources, it will yield much better results.

6 Indo Wordnet Creation - Linking of Wordnets in different Indian Languages

We have, for long, been engaged in building lexical resources for Indian languages ([NC02]) with focus on Hindi and Marathi (http://www.cfilt.iitb.ac.in). The HWN and MWN more or less follow the design principles of the Princeton wordnet for English, paying particular attention to language specific phenomena (such as complex predicates meaning verbs with incorporated nouns and compound verbs) whenever they arise.

While HWN had been created from first principles by looking up listed meanings of words from different dictionaries, MWN has been created derivatively from HWN. That is, the synsets of HWN are adapted to MWN via addition or deletion of synonyms in the synset. For example, the synset in HWN for the word `peR` (meaning 'tree') is {`peR, vriksh, paadap, drum, taru, viTap, ruuksh, ruukh, adhrip, taruvar`}. MWN deletes {`peR, viTap, ruuksh, ruukh, adhrip`} and adds `jhaaR` to it. Thus, the synset for 'tree' in MWN is {`jhaaR, vriksh, taruvar, drum, taru, paadap`}. Hindi and Marathi being close members of the same language family, many Hindi words have the same meaning in Marathi – especially the `tatsam` words (directly borrowed from Sanskrit).

6.1 Relation Borrowing in Marathi wordnet

The process of setting up lexico-semantic relations in one wordnet using the corresponding information from another wordnet is called *Relation Borrowing* ([SR06]). Described below are the different kinds of *Relation Borrowing* from HWN to MWN:

1. **When the meaning is found in both Hindi and Marathi**: This is the most common case, since Hindi and Marathi are sister languages and exist in almost identical cultural settings. The relations are established in MWN for that meaning, using the procedure explained in Figure 5.

2. **When the meaning is found in Hindi but not in Marathi**: Relation borrowing is not possible. For instance, {`daadaa`, `baabaa`, `aajaa`, `daddaa`, `pitaamaha`, `prapitaa`} is a synset in Hindi for 'paternal grandfather'. There are no equivalents in Marathi.

3. **When the meaning is found in Marathi, but not in Hindi**: The relations must be set up manually. For example, {`gudhipaadvaa`, `varshpratipadaa`} is a synset in Marathi for 'New Year' which does not have any equivalent in Hindi.

The algorithm for *Relation Borrowing* is given in Figure 5.
Following data structures are used for the linking purpose:

1. A table called *tbl_all_words*, which, for each word, stores the part of speech (PoS) and an array of ids for synsets in which the word participates. Table 9 illustrates this for the Hindi word *kara* (meaning *to do*).

2. A table called *tbl_all_synsets* which stores the synset ids, the synsets, and the glosses for the various meanings.

3. A table *tbl_<PoS>_<Relation>* for each PoS and Relation combination. For example, *tbl_noun_hypernymy* is the table for the hypernymy semantic relation.

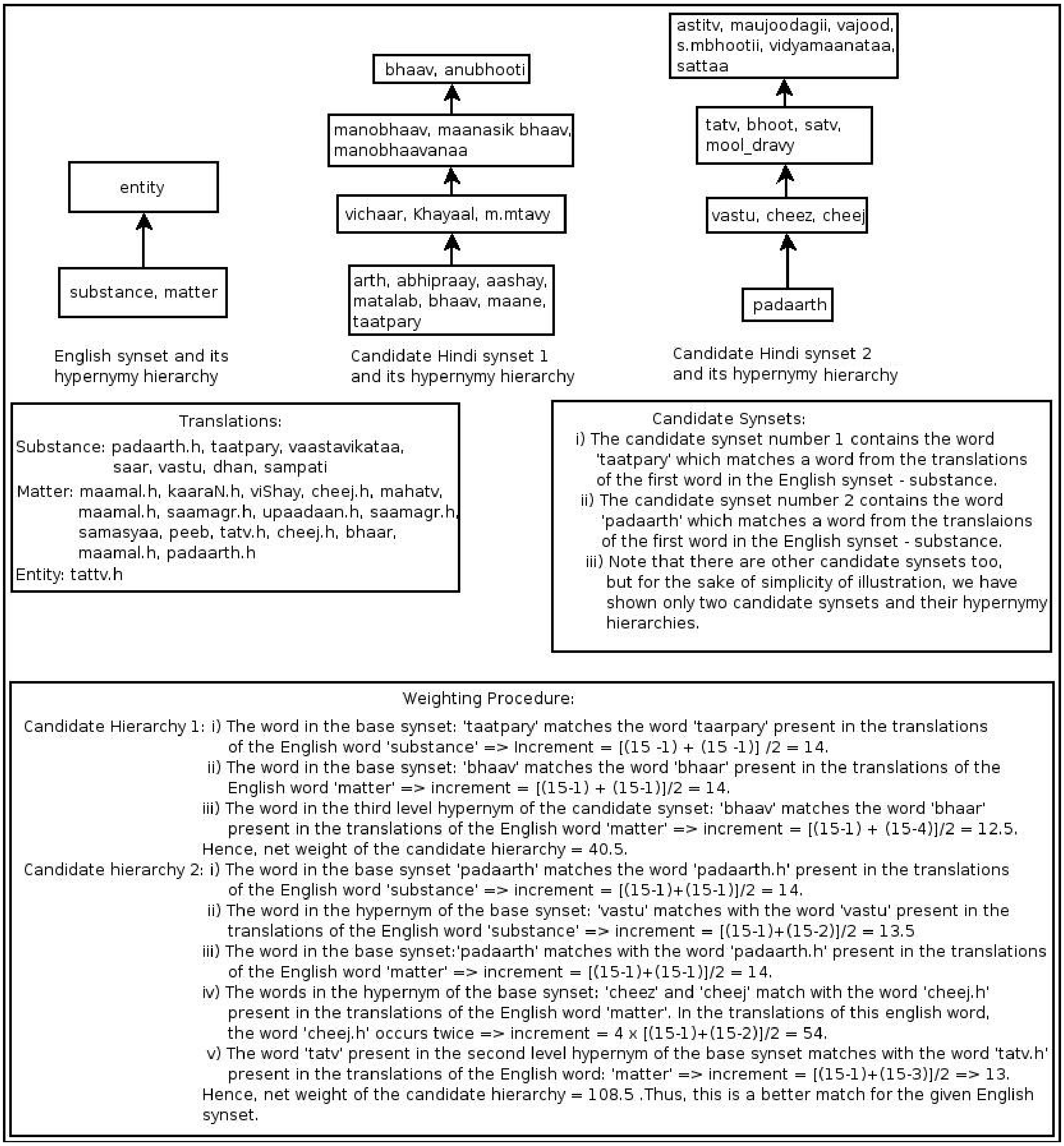

Figure 4: Block Diagram for English to Hindi synset Mapping

```
for each synset identity marathi_synset_id in Marathi WordNet do
  if (marathi_synset_id == hindi_synset_id) do
    for each relation r pointed by hindi_synset_id do
      if (relation type of r is semantic) do
        clamp the synset identity linked by relation r into marathi_synset_id
      end if
      else
        clamp the synset identity linked by relation r in hindi_synset_id to marathi_synset_id
        AND manually insert corresponding lexical elements
      end else
    end for
  end if
end for
```

Figure 5: Algorithm for *Relation Borrowing* between HWN and MWN

hindi_synset_id	*word*	*category*
491	*kara*	noun
3295	*kara*	verb
3529	*kara*	noun

Table 9: Example of *tbl_all_words* entries

Using the basic ideas outlined above, the synsets of MWN are completely linked with semantic and lexical relations. This saves a lot of manual labour. An interface has been designed to facilitate the simultaneous browsing of HWN and MWN. The input to this browser is a search string in any of the two languages. The search results for both the languages are displayed simultaneously.

6.2 Results on HWN and MWN linking

The Marathi wordnet obtained after establishment of relations using the above methodology was evaluated manually by lexicographers. Out of more than 12500 synsets created for Marathi wordnet, the program established relations for around 9000 synsets. The number of synsets for which relations were established is less because of the third case explained in section 6.1. Out of this number, the lexicographers qualitatively evaluated sets of 15% synsets sampled from each part of speech. We find that on an average, about 75% of the synsets of the Marathi wordnet are linked correctly. Considering only the case where synsets are aligned in both the wordnets, the average accuracy is 93%. The inaccuracy is a reflection of the incorrect links between synsets in the Hindi wordnet which is induced in the Marathi wordnet. The incorrect links are due to human error in the development of the wordnet. A huge dataset of more than 25,000 synsets cannot be checked manually. Only an application or a tool scanning the whole wordnet for a specific task reveals such errors. The error-cum-incomplete cases are mainly due to the absence of a complete repository of synsets in the Marathi wordnet which is still growing.

7 Conclusions and Future Work

We have presented our work on statistical measurements on wordnet structures in a trilingual setting – English, Hindi and Marathi. Many kinds of observations on EWN, HWN and MWN have been tabulated, graphically depicted and interpreted. These observations draw attention to the close correspondence and largely similar structural properties of different wordnets. It is interesting to note that just like the English wordnet, wordnets in Hindi and Marathi also exhibit the small-world nature. We plan to use this kind of information for evaluating wordnets. This can have implications for design choices in database implementations of such lexicons since an overwhelming majority of nodes have very low out-degrees. For instance, [KH05] use degree distribution data to optimise the schema of a multi-lingual database to improve performance.

Addressing the problem of matching of English and Hindi synsets revealed the challenges in such linking, the most significant of which is the disparity in the sizes of the wordnets. We would like to improve upon the algorithm presented for the task. The heuristic function used produces promising but not very good results. We would like to improve upon the matching quality by keeping the basic approach same and changing the heuristic function to not just depend on the levels of the synsets where the match was found, but also on the difference between the levels. This might lead to a better heuristic since the closely related synsets across the languages should have matches between the synsets which are at nearly the same levels.

The creation of the Marathi wordnet is being led by the wordnet of its sister language, *viz.* Hindi. The programmatic borrowing of semantic relations has dramatically cut down on the manual efforts of constructing this valuable resource. The method will prove more and more effective as the coverage of the Marathi wordnet increases.

All these issues as described above are – to our mind – valuable steps towards (i) wordnet evaluation and (ii) creation of the multi-lingual Indo wordnet linked with the global wordnet grid.

Acknowledgements: We thank the lexicographers at the Centre for Indian Language Technology (CFILT), IIT Bombay, for their evaluation of results and for their valuable suggestions.

References

[**FR98**] X. Farreres, G. Rigau, H. Rodriguez. *Using WordNet for Building WordNets*. Proceedings of the COLING-ACL Workshop on Usage of WordNet in Natural Language Processing Systems, Montreal, Canada, 1998.

[**GU04**] N. Guarino. *Towards a Formal Evaluation of Ontology Quality – (Why Evaluate Ontology Technologies? Because It Works!)*. IEEE Intelligent Systems, Vol. 19, No. 4:74-81, 2004.

[**KH05**] A. Kumaran, J. Haritsa. *SemEQUAL: Multilingual Semantic Matching in Relational Systems*. Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA), Beijing, China, 2005

[**KS05**] K. Kipper-Schuler. *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D. Thesis. University of Pennsylvania, 2005.

[**LE86**] M. Lesk. *Automatic sense disambiguation using machine readable dictionaries: How to tell a pinecone from a ice cream cone*. Proceedings of the SIGDOC '86, 1986.

[**LL00**] C. Lee, G. Lee, S. JungYun. *Automatic WordNet Mapping using Word Sense Disambiguation*. Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC 2000), 2000.

[**LS04**] H. Liu, P. Singh. *Commonsense Reasoning in and over Natural Language*. Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, 2004.

[**MB90**] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, K. Miller. *Introduction to Wordnet: an on-line lexical database*. International Journal of Lexicography Vol. 3, No. 4, pages 235-244, 1990.

[**NC02**] D. Narayan, D. Chakrabarty, P. Pande, P. Bhattacharyya. *An Experience in Building the Indo-WordNet – A WordNet for Hindi*. Proceedings of the First International Conference on Global WordNet (GWC 02), Mysore, India, 2002.

[**NP01**] I. Niles, A. Pease. *Towards a Standard Upper Ontology*. Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001). Chris Welty and Barry Smith, eds., Ogunquit, Maine, 2001.

[**NP03**] I. Niles, A. Pease. *Linking Lexicons and Ontologies: Mapping WordNet to the Suggested Upper Merged Ontology*. Proceedings of the 2003 International Conference on Information and Knowledge Engineering (IKE 03), Las Vegas, Nevada, 2003.

[**PB02**] E. Pianta, L. Bentivogli, C. Girardi. *MultiWordNet: Developing an Aligned Multilingual Database*. Proceedings of the First International Conference on Global WordNet (GWC 02), Mysore, India, 2002.

[**SR06**] M. Sinha, M. Reddy, P. Bhattacharyya. *An Approach Towards Construction and Application of Multilingual Indo-WordNet*. Proceedings of the 3rd Global Wordnet Conference (GWC 05), Jeju Island, Korea, 2006.

[**VO98**] P. Vossen. *EuroWordNet: a multilingual database with lexical semantic networks*. Kluwer Academic Publishers, 1998.

[**VR98**] L. Vanderwende, S. Richardson, W. Dolan. *MindNet: acquiring and structuring semantic information from text*. Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics, 1998.

[**WA06**] D. Watts. *Small Worlds – The Dynamics of Networks between Order and Randomness*. Princeton University Press, 2006.

[**WC03**] X. F. Wang, G. Chen. *Complex Networks: Small-World, Scale-Free and Beyond*. IEEE Circuits and Systems Magazine, 2003.

[**YA92**] D. Yarowsky. *Word Sense Disambiguation using statistical model of Roget's categories trained on large corpora*. Proceedings of the 14th International Conference on Computational Linguistics (COLING-92), pages 454-460, Nantes, France, 1992.

[**ZQ00**] Zhendong Dong, Qiang Dong. *An Introduction to HowNet*. Available from http://www.keenage.com, 2000.

Multilingual Semantic Matching with OrdPath in Relational Systems

A Kumaran **Peter Carlin**
Microsoft Corporation
{*kumarana,peterca*}*@microsoft.com*

Abstract

The volume of information in natural languages in electronic format is increasing exponentially. The demographics of users of information management systems are becoming increasingly multilingual. Together these trends create a requirement for information management systems to support processing of information in multiple natural languages *seamlessly. Database systems, the backbones of information management, should support this requirement effectively and efficiently. Earlier research in this area had proposed multilingual operators [7, 8] for relational database systems, and discussed their implementation using existing database features.*

In this paper, we specifically focus on the SemEQUAL *operator [8], implementing a multilingual semantic matching predicate using WordNet [12]. We explore the implementation of* SemEQUAL *using OrdPath [10], a positional representation for nodes of a hierarchy that is used successfully for supporting XML documents in relational systems. We propose the use of OrdPath to represent position within the Wordnet hierarchy, leveraging its ability to compute transitive closures efficiently. We show theoretically that an implementation using OrdPath will outperform those implementations proposed previously. Our initial experimental results confirm this analysis, and show that the OrdPath implementation performs significantly better. Further, since our technique is not specifically rooted to linguistic hierarchies, the same approach may benefit other applications that utilize alternative hierarchical ontologies.*

1 Introduction

The volume of information in *natural languages* in electronic format is increasing exponentially [9] and the demographics of users of information management systems are becoming increasingly multilingual [2, 11]. Together these trends create a requirement for information management systems to support processing of information in multiple *natural languages* seamlessly. Database systems, the backbones of information management, should support this requirement effectively and efficiently. The minimal requirement is that the underlying database engines (typically relational), provide similar functionality and efficiency for multilingual data as that associated with processing unilingual data, for which they are well-known. Earlier research in this area had proposed multilingual functions [7, 8] for relational database systems, and discussed ways of implementing them using existing features provided by the database systems. We specifically focus on the SemEQUAL function,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

as defined in [8], implementing a boolean predicate *SemEQUAL*($word_1$,$word_2$), which is true if $word_1$, or any of its synonyms, when translated into the language of $word_2$, are in the semantic transitive closure of $word_2$ or its synonyms. In order to implement this operator, the WordNet [12] ontological hierarchy is used, along with the semantic relationships between its component multilingual word forms.

In this paper, we explore the implementation of SemEQUAL using OrdPath [10], a representation of position in a hierarchy that is used successfully for supporting XML documents in relational systems. We propose the use of OrdPath to represent position within the Wordnet hierarchy, leveraging its ability to compute transitive closures efficiently. We show theoretically that an implementation using OrdPath will outperform those implementations proposed previously. Our initial experimental results confirm this analysis, and show that the OrdPath implementation performs significantly better. Further, since our technique is not specifically rooted to linguistic hierarchies, the same approach may benefit other applications that utilize alternative hierarchical ontologies.

1.1 Organization of the Paper

The paper is organized as follows: Section 2 outlines the semantic matching problem and provides a brief overview of WordNet lexical resources and OrdPath. Section 3 outlines the implementation approaches considered. Sections 4 and 5 theoretically and experimentally compare the various approaches and Section 6 concludes the paper, outlining our future research directions.

2 Multilingual Semantic Matching Problem

In this section, we state the problem of multilingual semantic matching of word-forms in different languages. We then provide a brief introduction to the resources used: a standard linguistic resource – the WordNet [4, 12], an ontological operator as defined in [8], and OrdPath, a hierarchy-numbering scheme [10]. Finally we outline simplifications to the problem for purposes of analysis and implementation.

2.1 Problem Definition

Consider a hypothetical multilingual portal, *Books.com*, with a sample product catalog [10] as shown in Figure 10, where the *Category* attribute stores the classification of the book in the original language of publication.

Table 10: **Sample *Books.com* Catalog**

Author	Author_ FN	Title	Price	Category	Language
Durant	Will	History of Civilization	$ 149.95	History	English
Descartes	Renè	Les Méditations Metaphysiques	€49,00	Philosophie	French
Franklin	Benjamin	Ein Amerikanischer Autobiography	€19,95	Autobiography	German
Gilderhus	Mark	History & Historians	$ 19.95	Historiography	English
Nero	Bicci	Il Coronation del Virgin	€99,00	Arti Fini	Italian
Nehru	Jawaharlal	Letters to My Daughter	£15.00	Journal	English
$\Sigma\alpha\rho\rho\eta$	$\kappa\alpha\tau\epsilon\rho\upsilon\alpha$	$\Pi\alpha\chi\upsilon\delta\alpha\sigma\tau\sigma\Pi\alpha\upsilon o$	€12,00	$M o\upsilon\sigma\kappa\eta$	Greek
Lebrun	François	L'Histoire De La France	€75,00	Histoire	French
Franklin	Benjamin	Un Américain Autobiographie	€19,95	Autobiographie	French

In today's database systems, a query with a selection condition of (Category = 'History'), would return *only* those books that have *Category* as `History` in English, although the catalog also contains history books in French, Greek and German. A multilingual user may be better served, however, if all the history books in all languages (or in a specified set of languages) are returned. A query using the SemEQUAL function of [8] as given below,

```
SELECT Author,Title,Category FROM Books
WHERE Category SemEQUAL 'History'
InLanguages {English, French}
```

and a result set, as given in Table 11, would therefore be desirable.

Table 11: **Multilingual Semantic Selection**

Author	**Title**	**Category**
Durant	History of Civilization	History
Lebrun	L'Histoire De La France	Histoire
Franklin	Un Américain Autobiographie	Autobiographie
Gilderhus	History & Historians	Historiography

It should be noted that the SemEQUAL function shown here is generalized to return not just the tuples that are equivalent in meaning, but also with respect to *specializations*, as in the last two tuples that are reported in the output. `Historiography` (*the science of history making*) and `Autobiography` are specialized branches of `History`. To determine semantic equivalence of word-forms across languages and to characterize the SemEQUAL functionality, we take recourse to WordNet [12], a standard linguistic resource that is available in multiple languages and, very importantly from our perspective, features *interlingual* semantic linkages.

2.2 A Brief Introduction to WordNet

In this section, we provide a brief introduction to WordNet [4, 12]. WordNet arranges the concepts of a language using psycho-linguistic principles, using word-forms as a canonical representation.

2.2.1 Word Form and Word Sense

A word may be thought of as a lexicalized concept; simply, it is the written form of a mental concept that may be an object, action, description, relationship, etc. Formally, it is referred to as a *Word-form*. The concept that it stands for is referred to as *Word-sense*, or in WordNet parlance, *Synset*. The defining philosophy in the design of WordNet is that a synset is sufficient to identify a concept for the user. For example, the word-form `bird` corresponds to several different synsets, two of which are {*a vertebrate animal that can typically fly*} and {*an aircraft*}; each of these two synsets is denoted differently with subscripts in Figure 1. Two words are said to be synonymous, *or semantically the same*, if they have the same synset and hence map to the same mental concept. The synsets are divided into five distinct categories and we explore below only the *Nouns* category. WordNet contains a *lexical matrix* that converts a *word form* (lexicographic representation) to a *word sense* (the semantic atom of the language, namely, the Synset).

2.2.2 Noun Taxonomical Hierarchy

WordNet organizes all relationships between the concepts of a language as a semantic network between synsets. In particular, the nouns in English WordNet are grouped under approximately twenty-five distinct *Semantic Primes* [4], covering distinct conceptual domains, such as *Animal*, *Artifact*, etc. Under each of the semantic primes, the nouns are organized in a taxonomic hierarchy, as shown in Figure 1, with *Hyponyms* links signifying the `is-a` relationships (shown in solid arrows). Efforts similar to the English WordNet are underway [4] in several languages, including Indian, Chinese and European languages. A common feature among such efforts is that they all strive for a taxonomic hierarchy in a respective language that has synsets that may be mapped to a set of English synsets. Further, inter-linking of semantically equivalent synsets between WordNets of different languages are being designed in some languages. Figure 1 shows a simplified interlinked hierarchy (shown as dotted arrows) in English and German.

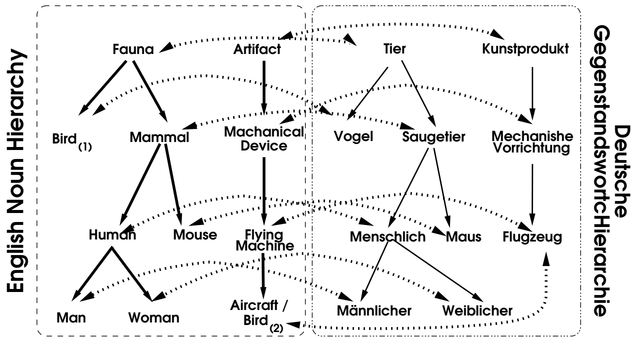

Figure 1: **Sample Interlinked *WordNet* Noun Hierarchy**

2.3 Implementing SemEQUAL using WordNet

WordNet enables mapping word forms to synsets in other languages and comparing them. Denoting an interlinked taxonomic hierarchy of the multilingual strings by $\mathcal{H}_{\mathcal{ML}}$, the SemEQUAL operator is formally defined [8] as follows:

Definition: Given two multilingual noun strings w_i and w_j, and the interlinked multilingual taxonomical hierarchy $\mathcal{H}_{\mathcal{ML}}$, $(w_i \text{SemEQUAL} w_j) \iff (w_i \cap \mathcal{T}_{\mathcal{H}_{\mathcal{ML}}}(w_j) \neq \phi)$, where $\mathcal{T}_{\mathcal{H}_{\mathcal{ML}}}(x)$ computes the transitive closure of x in $\mathcal{H}_{\mathcal{ML}}$.

The basic skeleton of the algorithm to semantically match a pair of multilingual strings is outlined in Figure 2. Here, the SemEQUAL function takes two multilingual strings w_1 and w_2 as input. It returns true if the string w_2 is a member of the transitive closure of w_1 in the multilingual taxonomic hierarchy $\mathcal{H}_{\mathcal{ML}}$. Note that the w_2 could be the values from the column Category in the Catalog table, and w_1 could be the user specified category, say `History`.

SemEQUAL (w_1, w_2)
Input: *Multilingual Strings* w_1, w_2, Taxonomic Hierarchy $\mathcal{H}_{\mathcal{ML}}$ (*as a resource*)
Output: true or false
1. $\mathcal{TC}_{\mathcal{Q}} \leftarrow$ TransitiveClosure$(w_2, \mathcal{H}_{\mathcal{ML}})$;
2. **if** $(\{w_1\} \cap \mathcal{TC}_{\mathcal{Q}})$ **return** true **else return** false;

Figure 2: **The SemEQUAL Operator Algorithm**

SemEQUAL as defined in Figure 2 may be implemented with the following 3 steps:

1. All synonyms of $word_1$ in the language of $word_2$, yielding a set of words, $\mathcal{W}_1$.
2. Find the semantic transitive closure of $word_2$ in the taxonomic hierarchy, $\mathcal{H}_{\mathcal{ML}}$, yielding $\mathcal{W}_2$.
3. Output true if $\mathcal{W}_1 \cap \mathcal{W}_2 \neq \phi$.

Though WordNet hierarchies are directed acyclic graphs, we simplified them into trees, by duplicating all shared nodes and their descendants, with multiple parents. Since the number of such shared nodes is not high [3], the additional storage overhead is not significant.

2.4 A Brief Introduction to Ordpath

OrdPath is a tree numbering schema designed for storage and query of XML data [10]. It is one of a number of novel tree numbering schemes devised in recent years [6]. Ordpaths are formed by concatenating bitstrings, each bitstring representing the position of that Ordpath at one level of the tree. Each bitstring is Huffman encoded so small values (and thus tree nodes with small fanout) take less space than large values. For example, using integers to represent the bitstring for each level and delimiting them by periods, an OrdPath 1.5.3 represents the 3^{rd} child of the 5^{th} child of the root node. An interested reader is referred to [10], for details on Ordpath. Ordpath encoding of hierarchies has the following three key properties:

1. **Encoding of position in the hierarchy** If a and b are two OrdPaths, and the binary comparison $a < b$ is true, then a precedes b in a depth-first traversal in the tree represented by the OrdPaths. This property allows us to test easily whether one OrdPath is in the transitive closure of another: namely, if $a = prefix(b)$, then b exists in the closure of a. This can be easily encapsulated in a function *IsDescendant(path1,path2)* which is true if *path2* is in the closure of *path1*. *IsDescendant(path1,path2)* can be expressed as the conjunctive predicate (*path2* $>=$ *path1 and path2* $<$ *DescendantLimit(path1)*), where *DescendantLimit(path1)* is the largest possible value a descendant of *path1* can have. *DescendantLimit(path1)* can be computed easily.

2. **Small size** Each level of each node in the hierarchy is represented by a variable-length bit string. This results in OrdPaths being quite compact. For instance, WordNet has about $75,000$ nodes, with a maximum depth of 19, a maximum fanout of 398, an average depth of 9.2 and an average fanout of 1. The average Ordpath value representing a node in such a tree will take 30 bits.

3. **Support insert and delete** OrdPaths numbering allow additions and deletions of nodes in the hierarchy, while maintaining the other properties. While we do not anticipate frequent updates to the noun hierarchy, this property allows any domain-specific local updates to the hierarchy to be done efficiently.

3 Implementation Approaches Considered

In this section, we discuss alternative approaches for implementing SemEQUAL in relational database systems. Some of these approaches have been tried in the research literature [8], but are presented here for comparison with the proposed method. Specifically, the following four ways of implementing SemEqual are analyzed. They primarily differ in the representation of the WordNet hierarchy in the relational system, and how the transitive closure is computed.

Parent-child In this representation, the hierarchy relationships are represented as Parent-Child relations. Hence, each row in the table represents a link in the hierarchy. For a given node n, there may be n_{pc} rows in the hierarchy table, each representing a child relationship with n as the parent.

Pre-computed All the transitive ancestor-descendent relationships associated with a node are represented as a set of rows explicitly. Hence, every node n will have a set of n_{ad} relationships, with each row representing one ancestor-descendent relationship between n and a node in its transitive closure.

Inlined pre-computed All the ancestor-descendent relationships associated with a node are represented as one row in the inlined table (assuming that the number of descendents for the node n does not exceed a specified large limit). The inlined descendents are stored in a variable array, associated with the root of the transitive closure.

WordPath The position of a node in the hierarchy is represented by an OrdPath. This representation stores the WordNet hierarchy implicitly.

3.1 Schema Representation for Analysis

In this paper, for the analysis and comparison of different transitive closure computing methodologies, we use a standard Information Retrieval (IR) scenario, where a collection of documents is searched for occurrences of a user-specified specific search term. While a standard search considers an inverted index (or equivalently a standard B+ index structure), SemEQUAL searches the specified search term, and also all elements in its transitive closure with respect to WordNet noun hierarchy. In addition, by appropriate use of multilingual WordNet hierarchies, a query term may be searched across languages, as outlined in [8]. Note that the analysis outlined in this paper applies to other scenarios as well; for instance, SemEQUAL may be used as a join condition between two document collections. However, for clarity of explanation, we restrict our discussion to the simple scenario of searching in a single document collection.

We define the following four tables in this scenario, as explained in detail below. The primary key of a table is indicated with underlining of the appropriate key columns, and the order in which they are defined.

- **Words(<u>Language</u> int, <u>Word</u> string, <u>WordId</u> int)** This table maps words to integers, providing a reference key to be used in other tables, thus reducing storage space. *WordId* is unique across all languages. A secondary index on *WordId* enables translating from *WordId* back to string representation efficiently.
- **Corpus(<u>WordId</u> int, DocumentIdList binary)** This table represents the document collection (referred to as corpus) being searched. Corpus is stored as an inverted index; with a *WordId* as the primary key, accompanied by a compact list of documents that it occurs in. Here we assume that the list of *DocId*s is represented as a list of deltas from one docid to the next, as described in [1].
- **Synset(<u>SourceLanguage</u> int, <u>SourceWordId</u> int, <u>TargetLanguage</u> int, <u>TargetWordId</u> int)** This table stores the set of synonyms of a word, irrespective of the languages, thereby extending the traditional definition of synsets in WordNet, to include the interlingual links between synsets in the languages. This table contains all inter- and intra-language synonyms of words, where intra-language synonyms have the same source and target languages, and inter-language synonyms have different ones.
- **TcWordNet(<u>WordId</u> int, <u>ClosureId</u> int)** This view stores the transitive closure of all of WordNet. Each row of this table contains a *WordId* and one node its closure, represented by *ClosureId*. This is a view because the storage of the transitive closure varies between the approaches.

3.2 Common Query

We use a common IR query of searching a corpus for a given word, for our analysis. The common search query will use SemEQUAL, thus including multilingual semantic search based on the taxonomic hierarchy defined by the interlinked WordNets. It has been shown in [8] that the complexity of *SemEQUAL*(w_1, w_2) is linearly proportional to the number of languages used in the hierarchy. It is reasonable to assume that the set of languages in which the documents in the Corpus table occur is known, or the languages of interest are specified by the user. Therefore, we add a third argument to SemEQUAL, namely *SearchLanguageList*, that specifies the list of languages in which to consider semantic matches in. Such an assumption greatly reduces the effort to generate the transitive closure of the query terms in all languages. The query examined is thus:

```
Select C.DocumentIdList from Corpus C
where SemEqual(C.WordId, @SearchWord, @SearchLanguageList)
```

The parameter `@SearchWord` indicates the search term and `@SearchLanguageList` provides the list of languages in which to look for semantic matches. This query has an algebraic, non-cost-based, rewrite against the above schema, to:

```
Select C.DocumentIdList from Words W where W.Word=@SearchWord
Inner join Synset S
  where S.SourceLanguage = W.Language
  and S.SourceWordId = W.WordId
  and TargetLanguage IN @SearchLanguageList
Inner join TcWordNet T where S.TargetWordId = T.WordId
Inner join Corpus C where C.WordId = T.ClosureId
```

The generated query is used in subsequent analysis, to compare the efficiency of each of the proposed methods to compute the transitive closure.

3.3 Detailed Overview of Approaches

3.3.1 Parent-Child: WordNet(WordId int, ParentWordId int)

In this approach WordNet is stored as a row per parent-child relationship. The transitive closure is computed via a standard SQL:1999 recursive query. Table 12 shows a sample portion of the table.

Table 12: Parent-Child Hierarchy Table

WordId	ParentWordId
Mammal	Animal
Aquatic Mammal	Animal
Dog	Mammal
Tiger	Mammal
Dolphin	Aquatic Mammal

3.3.2 Pre-Computed Closure (PreComputed): WordNetPC (WordId int, ClosureWordId int)

In this approach WordNet is stored with the pre-computed transitive closure in the hierarchy. Each element in a word's transitive closure requires one row. Table 13 shows a sample portion of the table.

Table 13: Pre-Computed Closure

WordId	ClosureWordId
Animal	Mammal
Animal	Aquatic Mammal
Animal	Dog
Animal	Tiger
Animal	Dolphin

3.3.3 Inlined Pre-Computed Closure (Inlined): WordNetIL (WordId int,ClosureIdList array(int))

Assuming an efficient implementation of variable arrays is available in the database system, the duplication of WordId in the pre-computed closure can be eliminated. Alternatively, in database systems that support prefix compression on keys, this methodology may be viewed as the pre-computed closure with prefix compression on WordId. Either way, ordering the closure list on *WordId* maximizes the efficiency of subsequent processing, in particular searching the closure for a particular word. Table 14 shows a sample portion of the table.

Table 14: Inlined Closure

WordId	**ClosureIdList**
Animal	{Mammal, Aquatic Mammal, Dog, Tiger, Dolphin, ...}
Mammal	{Dog, Tiger, ...}
Aquatic Mammal	{Dolphin, ...}

With variable array the in-lined table is unrolled to compute the transitive closure, as follows:

```
Create view TcWordNet as
   Select * from WordNetIL W cross apply unnest(W.ClosureIdList)
```

3.3.4 WordPath

Under this approach, the position of a word in the WordNet hierarchy is encoded by an OrdPath, called a *WordPath* subsequently in this paper. The *WordId* used in other approaches is replaced by *WordPath*. This does not cause an increase in storage size. The key advantage is the elimination of any table representing the WordNet hierarchy. This is possible because the only operation the WordNet table is used for is to enable determining, for any pair of words, whether the first is in the transitive closure of the second. The information necessary to determine this is encoded in the *WordPath* itself. Another advantage is that with the Corpus table having a primary key of *WordPath*, all words in the transitive closure of a given word are co-located. Assuming a type *path* that encapsulates OrdPath functionality, the resulting schema is as follows:

- Words (Language int, Word string, WordPath path)
- Corpus (WordPath path, DocumentIdList binary)
- Synset (SourceLanguage int, SourceWordPath path, TargetLanguage int, TargetWordPath path)

Using the above schema, the standard IR query is rewritten as:

```
Select C.DocumentIdList from Words W where W.Word=@SearchWord
Inner join Synset S
   on S.SourceLanguage=W.Language
   and S.SourceWordPath=W.WordPath
   and TargetLanguage IN @SearchLanguageList
Inner join Corpus C on IsDescendant(C.WordPath, S.TargetWordPath)
```

Comparing this to the query in Section 3.2, the WordPath query eliminates the join with TcWordNet. In addition, the final join between Synset and Corpus is on the *IsDescendant(column,path2)* predicate; as outlined in Section 2.4 this is a conjunctive predicate and the join condition becomes a range seek.

4 Theoretical Analysis

In this section, we present a theoretical analysis of the performance of the *WordPath* approach in comparison to the other approaches outlined in Section 3. Specifically, we compare key parameters affecting relational query performance: the storage size, logical IO, and CPU usage.

4.1 Definitions and Assumptions

First, we define the following symbols and terms that are used in subsequent analysis:

- W - Number of words in WordNet noun hierarchy. For English WordNet, this is 75,000, and it is assumed to be of similar order in other WordNets [4].
- A - Average number of words in the transitive closure for a given word. For the noun hierarchy of English WordNet, this is 9.2.
- S - Average number of synsets for a given word. For English WordNet, s is 1.23, and it is expected to be similar in other WordNets [4].
- M - Average number of corpus documents with a given WordId.
- L - Number of languages for which WordNets are stored and used for processing. Currently, approximately 30 WordNets exist at various stages of completion. [5]
- l - Number of languages involved in a SemEQUAL query. Here assumed to be 3, the average number of languages a user may be interested in (per query).
- C - Average length of a word. For English Wordnet this is 11 characters.
- $PageSize$ - Number of bytes per database page. This number varies from 4K to 64K in various commercial DBMS systems. We assumed 8K for our analysis.

Next, the following simplifying assumptions are made, to make the analysis easier. The first two assumptions affect all approaches proportionately, and hence will not affect the comparisons. The last two assumptions affect WordPath negatively compared to the other approaches, and hence the results provide a lower bound on the relative benefit of WordPath.

- **Non-leaf B-tree page costs are assumed to be 0**. In most schemas the non-leaf tree pages are a small percentage of the database size. Navigation of these pages usually takes a small percentage of the IO and CPU for a given query.
- **Per-row overhead costs are assumed to be 0**. In most database systems the per-row storage overheads are relatively low, typically, 1-2 bytes per row.
- **It is assumed that all IO - Sequential and Random IO - has the same cost**. Clearly, for logical IO this is true, while for physical IO, this assumption is not. However, sequential IO is only possible in the WordPath case or when the storage required for the *DocId*s for a given word exceed a page - in other words, for extremely large Corpus sizes.
- **The average Wordpath size is assumed to be 4 bytes**, the same as that for WordId. Note that the actual average WordPath size for WordNet is less than this.

Given the above definitions, *M*, the number of documents with a given word, determines the scope of a scenario. In the following table, we present some scenarios to indicate approximately the scope of the databases involved. We consider in our analysis document collections up to that corresponding to $M = 100,000$.

Table 15: Values of M and associated scenarios

M	Scenario	Notes
10	Company Product Catalog	10^5 products, 10 words/title
100	Large County Library	10^6 book titles, 10 words/title
1,000	U.S. Library of Congress	10^7 book titles
10,000	PubMed abstract index	10^7 abstracts, 1000 words/abstract
100,000	PubMed article index	10^7 article texts, 10000 words/article

4.2 Storage Size Comparison

Based on the tables described in Sections 3.1 and 3.3, the total storage size of each table, is as shown in Table 16.

Table 16: Table Storage Sizes

Table	Size	Other Structures
Words	$(8 + C) * W * 2$	2 indices
Synsets	$16 * W * S * l$	
WordNet, Parent-Child	$8 * W$	
WordNet, Pre-computed	$8 * A * W$	
WordNet, Inlined	$(4 * A + 4) * W$	
Corpus	$W * M$	

Using the above formulae, the total schema storage size for each of the approaches, and for different scenarios, are computed and shown in Table 17.

Table 17: Storage Sizes (MB)

M	ParentChild	PreComputed	Inlined	WordPath
10	45.4	54.1	49.7	44.8
100	51.8	60.5	56.2	51.2
1,000	116.2	124.9	120.5	115.6
10,000	759.9	768.6	764.3	759.3
100,000	7,197.2	7,205.9	7,201.6	7,196.6

The corpus size eventually becomes the most dominant factor. Since in all approaches the corpus table has the same number rows (though not in the same order in each approach), the storage size for all approaches eventually converges. For smaller collections, the Parent-Child and WordPath approaches have approximately equal storage, and have a slight advantage, space-wise.

4.3 Logical IO

For cold queries, defined as those that are run when pages needed for answering the queries are not in-memory, all unique logical IOs become physical IOs and dominate response time. For warm queries, logical IO is still, in general, the leading factor in response time of a query. However, how much logical IO an approach takes depends on the query plan followed. In all scenarios considered here, the best plan is to first lookup relevant synsets, compute their respective closures and then lookup each closure member in the corpus. In the WordPath approach, the best plan is to compute the synsets and lookup each synset in the corpus. Table 4.3 outlines formulas for the IO required at each stage for every approach.

Approach	GetId	Get Synsets	Get Closures	Corpus Lookup
ParentChild	1	$16 * SL/PageSize$	$ASl + \lceil 4 * ASl/PageSize \rceil$	$ASl * \lceil M/PageSize \rceil$
PreComputed	1	$16 * SL/PageSize$	$Sl * \lceil 8A/PS \rceil$	$ASl * \lceil M/PageSize \rceil$
Inlined	1	$16 * SL/PageSize$	$Sl * \lceil (4A + 4)/PS \rceil$	$ASl * \lceil M/PageSize \rceil$
WordPath	1	$16 * SL/PageSize$	0	$Sl * \lceil AM/PageSize \rceil$

Below, we present a detailed examination of each of the stages presented in Table 4.3:

- **GetId:** In all alternatives, this is a single IO to translate from string to WordId or WordPath, as appropriate in an approach.
- **Get Synsets:** In all alternatives, this is a lookup of the query language and word Id/Path, followed by a scan of the synsets in each relevant target language. We assume that the l/L ratio (languages of interest to the query vs total languages represented in the system) can vary and all L languages are fixed and co-located. Further, it is assumed that the l languages are distributed such that all pages for a given path are touched. Note that since $S \simeq 1$ and $L < 30$, this is usually a single IO.
- **Get Closures: Basic Parent-Child** Assuming no co-location due to lack of correlation between WordId and position in the hierarchy, each word in the transitive closure will be a separate logical IO. There are $S * l$ closures to compute and A words in the average closure, resulting in $A * S * l$ co-located rows, each 4 bytes long. In addition, there is IO associated with temporary storage of intermediate/final closures.
- **Get Closures: Pre-Computed** These are all co-located for a given synset, but not across synsets or languages. Each closure takes $8 * A$ bytes of space.
- **Get Closures: Inlined** The analysis is same as that of Pre-Computed approach, other than each closure takes $4A + 4$ bytes of space.
- **Get Closures: WordPath** No closure computation is required.
- **Corpus Lookup: (Basic, PreComputed and Inlined)** With no co-location, each of the $A * S * l$ closure elements can be assumed to require separate IO. Each of these lookups requires $M/PageSize$ IOs.
- **Corpus Lookup: WordPath** Synsets and languages are not necessarily co-located, but the A elements of each of these closures are. Hence, each closure takes $A * M/PageSize$ bytes of space.

Using the above formulae, Table 18 shows the total logical IOs required by each approach for different scenarios, and for different implementation methodologies.

Table 18: Logical IO for each Approach in each Scenario

M	ParentChild	PreComputed	Inlined	WordPath
10	27	15	15	4
100	27	15	15	4
$1,000$	27	15	15	5
$10,000$	38	26	26	20
$100,000$	159	151	151	143

It may be observed that the WordPath approach shows a significant IO advantage in most of the modeled scenarios for the given set of parameters and assuming frequency of words in the corpus follows a continuous distribution.

4.4 CPU Complexity

Beyond logical IO, CPU complexity is most strongly tied to rows processed and relational operators selected in the execution tree. In Table 19, we compare these factors for each approach.

Table 19: Rows Processed and Relational Operators for each Approach

Approach	Rows Processed(formula)	Rows Processed(number)	Operators
ParentChild	$1 + 2 * Sl + 2 * ASl$	73	$3 + A$
PreComputed	$1 + 2 * Sl + 2 * ASl$	73	4
Inlined	$1 + 2 * Sl + 2 * ASl$	73	4
WordPath	$1 + Sl + ASl$	39	3

We note that the WordPath approach processes only about 50% of the rows, compared with that of other approaches. This is because rows processed is dominated by the $A*S*l$ term, for which WordPath has a leading coefficient of 1 while the others have a 2. We also note that rows processed is independent of Corpus size. Other than Parent-Child which requires several scans of the hierarchy table, the differences in plan complexity are not significant among the other approaches.

5 Experimental Results

We implemented the ParentChild, PreComputed and WordPath approaches as outlined in this paper in SQL Server 2005 using the noun hierarchy of Wordnet Version 1.5. Two simplifications were made to ease implementation: First, the ordpath type in SQL Server is under development, so a static depth-first numbering scheme was used as a substitute. Second, it was assumed that S=1 rather than 1.23. Neither simplification is expected to affect results significantly. Tables 20 and 21 show the results for small corpus sizes M=10 and M=100. In both, search terms were picked so that transitive closure size was equal to the Wordnet average of 9.

Table 20: Performance Comparison (M=10)

Approach	CPU time	Logical IO	Physical IO	Elapsed time(cold)
Parent-Child	107413	2442	16	291050
Pre-Computed	1764	26	11	140625
WordPath	1209	8	7	88320

Table 21: Performance Comparison (M=100)

Approach	CPU time	Logical IO	Physical IO	Elapsed time(cold)
Parent-Child	108643	2451	19	331926
Pre-Computed	2088	35	14	189983
WordPath	1397	10	8	139406

The above figures indicate that the actual performance is in line with that expected from the theoretical analysis, confirming our claim that OrdPath methodology significantly outperforms earlier implementation methodologies. OrdPath's efficient encoding scheme eliminates the need for explicit computation of transitive closures, and thus exhibits superior performance for SemEQUAL query. We are currently experimenting with larger database sizes (M values up to 100,000) and with a variety of hierarchies. We hope to report a full study in due course.

6 Conclusion & Future Directions

In this paper, we explored the implementation of a multilingual semantic matching function, SemEQUAL, using OrdPath [10] to represent position in the WordNet [12] hierarchy. OrdPath enables efficient determination of membership in transitive closures, a key step in implementing SemEQUAL. We analyzed theoretically the performance of existing implementations, and with proposed OrdPath methodology, and showed that our implementation would incur significantly less IO and CPU costs, resulting in more efficient processing of SemEQUAL. We implemented the various approaches in SQL Server, and the performance figures for small database sizes confirm that the performance of OrdPath is in line with that predicted by our analysis, and significantly better than the exisitng implementations. We are currently undertaking a through study of OrdPath for different sizes of hierarchies, query types and database sizes, and we hope to report the results in due course. Further, since our technique is not specifically rooted to linguistic hierarchies, the same approach may benefit other applications that utilize alternative hierarchical ontologies.

References

[1] Brewer, E. Combining Systems and Databases: A Search Engine Retrospective. *Readings in Database Systems: Fourth Edition*. Joseph M. Hellerstein and Michael Stonebreaker (*eds.*, MIT Press, Cambridge, MA. 2005). *http://www.cs.berkeley.edu/ brewer/papers/SearchDB.pdf*.

[2] The Computer Scope Ltd. *http://www.NUA.ie/Surveys*.

[3] Devitt A. and Vogel, C. The Topology of WordNet: Some Metrics. *Proceedings of the Second Global WordNet Conference*, 2004. *http://www.fi.muni.cz/gwc2004/proc/119.pdf*.

[4] Fellbaum, C. and Miller, G. A. WordNet: An electronic lexical database (language, speech and communication). *MIT Press*, Cambridge, MA, 1998.

[5] Global WordNets in the World. *http://www.globalwordnet.org/gwa/wordnet_table.htm*.

[6] Koch, C., Processing queries on tree-structured data efficiently. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Chicago, IL. *http://www-dbs.cs.uni-sb.de/ koch/download/pods2006.pdf*.

[7] Kumaran, A. and Haritsa, J. R. LexEQUAL: Multilexical Matching Operator in SQL. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004. *http://portal.acm.org/citation.cfm?id=1007715*.

[8] Kumaran, A. and Haritsa, J. R. SemEQUAL: Multilingual Semantic Matching in Relational Systems. *Proceedings of the* 10^{th} *International Conference on Database Systems for Advanced Applications*, 2005. *http://www.springerlink.com/index/YKKAWFYBB22CGBPF.pdf*.

[9] Lyman, P. and Varian, H. R. How Much Information. *http://www.sims.berkeley.edu/research/projects/how-much-info/*.

[10] O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G. and Westbury, N. ORDPATHs: insert-friendly XML node labels. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004. *http://portal.acm.org/citation.cfm?id=1007686#references*.

[11] The WebFountain Project. *http://www.almaden.ibm.com/WebFountain.*

[12] The WordNet. *http://www.cogsci.princeton.edu/w̃n*.

Multilingual Indexing Support for CLIR using Language Modeling

Prasad Pingali, Vasudeva Varma
{pvvpr, vv}@iiit.ac.in
International Institute of Information Technology
Hyderabad, India.

Abstract

An indexing model is the heart of an Information Retrieval (IR) system. Data structures such as term based inverted indices have proved to be very effective for IR using vector space retrieval models. However, when functional aspects of such models were tested, it was soon felt that better relevance models were required to more accurately compute the relevance of a document towards a query. It was shown that language modeling approaches [1] in monolingual IR tasks improve the quality of search results in comparison with TFIDF [2] algorithm. The disadvantage of language modeling approaches when used in monolingual IR task as suggested in [1] is that they would require both the inverted index (term-to-document) and the forward index (document-to-term) to be able to compute the rank of document for a given query. This calls for an additional space and computation overhead when compared to inverted index models. Such a cost may be acceptable if the quality of search results are significantly improved. In a Cross-lingual IR (CLIR) task, we have previously shown in [3] that using a bilingual dictionary along with term co-occurrence statistics and language modeling approach helps improve the functional IR performance. However, no studies exist on the performance overhead in a CLIR task due to language modeling. In this paper we present an augmented index model which can be used for fast retrieval while having the benefits of language modeling in a CLIR task. The model is capable of retrieval and ranking with or without query expansion techniques using term collocation statistics of the indexed corpus. Finally we conduct performance related experiments on our indexing model to determine the cost overheads on space and time.

1 Introduction

Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which describe documents, or searching within databases, whether relational stand-alone databases or hypertext networked databases such as the Internet or World Wide Web or intranets, for text, sound, images or data. Searching for the relevant information also may involve the notion of a ranking function, which denotes the relevance of a retrieved item for the given query. It should be noted that ranking the search results in the order of their relevance to query is an important aspect in most of the IR systems that

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

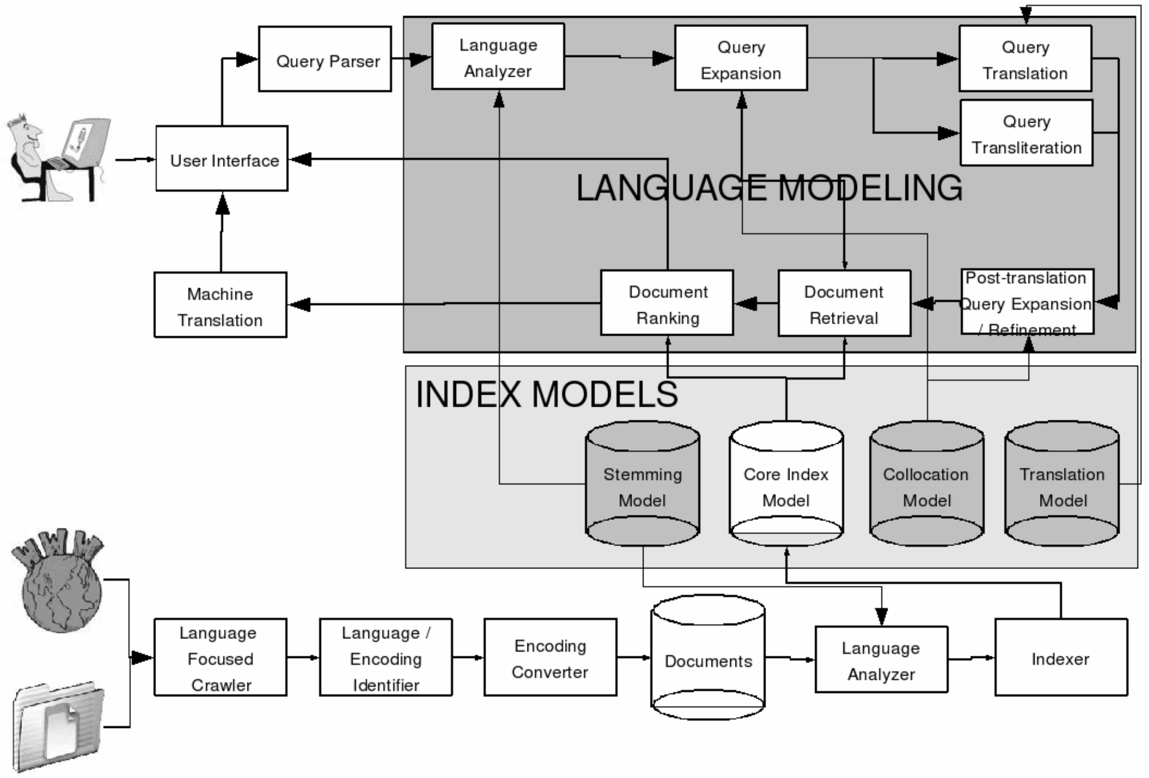

Figure 1: A Typical Information Access Framework

deal with text document retrieval. Boolean IR systems are an exception to this observation, where all the search results that satisfy the boolean constraints of the query are presented to the user in the same order as found in the database. However, boolean retrieval systems were found to be insufficient in building usable applications in many domains. In this paper we deal with the problem of text document retrieval and more specifically Cross-lingual Information Retrieval (CLIR).

CLIR has been an important area of research in the IR community and the need is felt to extend today's monolingual techniques to be able to simultaneously handle content of multiple languages. Assume a query Q_s in user's language (also known as a source language). The core cross-lingual IR problem is defined as retrieving and ranking of relevant documents which may occur in the same source language as that of Q_s and each of a set of target languages $TL_1, TL_2....TL_k$. This problem can be first viewed in terms of a single target language (say the cross language document collection is D_t) and then scaled to multiple languages using the same technique. However, such a solution will not try to exploit the availability of same or similar (comparable) information in multiple languages. In CLIR and Multi-lingual IR (MLIR) research, studies exist which address the translingual IR problem by considering a single target language [4–7], and also by exploiting similar information in multiple target languages [8, 9]. In this study we consider solutions for one target language and then scale such a solution for other languages. This would enable us *not* to assume any comparable data or resources available in multiple target languages and at the same we would not try to do anything special in the ranking function when multiple target languages exist. If both the query and documents were in the same language, all the standard IR techniques (such as weighted keywords, vector space models, probabilistic IR models) can be directly applied to retrieve and rank the documents with slight adaptations to Indian language content. However these techniques may not be directly applicable for the cross-language retrieval problem. In order to achieve the CLIR, some of the possible approaches could be:

- *Translate the document collection* - Manually or automatically translate all the documents in D_t into the source language document set, say $D_s^{'}$ and retrieve and rank the documents using the same language query Q_s using any of the monolingual IR techniques
- *Translate the query* - Convert the source language query Q_s into the target language either manually, automatically or semi-automatically with user interaction (say as $Q_t^{'}$) and retrieve and rank the documents present in the same language (i.e. D_t) using any of the monolingual IR techniques
- *cross-lingual relevance feedback* - If a parallel corpus exists between the query and document languages, i.e., for every source language document in D_s there exists an equivalent target language translated document in D_t which is identifiably annotated. In such a setup, the source language query Q_s can be used to query documents of the same language first, and the retrieved documents from source language are used to obtain their equivalent target language documents from D_t. The terms collected from these documents are then used to retrieve target language documents.

While these different approaches have been studied to address CLIR problem, each of them has its own disadvantages. Translating the entire document collection might work if the document collection is static and is locally available. However these may be unrealistic assumptions if a CLIR system were to be built for an environment like web where the document collection is very huge and is always changing at a very rapid rate. On the other hand translating the query seems promising, but queries are typically too short to provide enough context to precisely translate into a different language. Despite this drawback, CLIR researchers have argued and demonstrated that a precise translation may not be required, since the translated query is not for human consumption and is meant to obtain the gist of the information need. The third approach of relevance feedback has been shown to perform very well [6] but it requires a parallel corpus which is a difficult resource to obtain for many language pairs. Since we are looking at Indian languages as the query language, it is difficult to find any such resources. Therefore we limit our scope to using dictionary based translation of the query, with query expansion using monolingual corpus (i.e. the second approach).

2 System Overview

Having defined the cross-lingual IR problem in general, let us now look at each of the sub-problems to achieve the stated larger goal. This understanding is essential to appreciate some of the decisions made in designing the structure of the index. In order to illustrate some of the major problem areas within CLIR we provide a typical system overview as shown in Figure 1 and use an example information need represented as a query. Consider a user trying to locate documents discussing about various "Nestle's products". This query would be written as "नेस्ले के उत्पादन" in Hindi and as "నెస్లే యొక్క ఉత్పత్తులు " in Telugu [1]. As evident from the Figure 1, there are two broad logical divisions in this CLIR sub-problems. The first logical division consists of offline processing modules to collect and index the document collections. The second logical division deals with runtime processing of query processing, retrieval, ranking and presentation of information to the user. Some of the modules in the shown in Figure 1 are not in the scope of this paper and hence will not be discussed in detail. Let us now look at the functions of various sub-problems or modules that are part of the above mentioned framework that would be dependent on the underlying design of database. The following sections 2.1 to 2.4 discuss offline processing mainly involving data collection and indexing. Sections 2.5 to 2.10 discuss online processing of the IR system at the time of issue of a query by the user.

[1] Hindi and Telugu are two major Indian languages geographically corresponding to Northern and South-Eastern India, respectively.

2.1 Language / Encoding Identifier

As mentioned in the previous example query, our system is expected to accept queries from multiple Indian languages. Similarly, the document collections that are to be retrieved can be present in multiple languages and multiple encodings in each language. A number of language encodings are in usage on the world wide web and other document repositories. Especially in Indian language document repositories, these document collections are available in a set of national and international standards for Indian language character set encodings, while a number of publishers use proprietary non-standard encodings. In order to be able to index such content, it is very important to identify the language and such encodings in order to achieve a better recall in retrieval by having much broader coverage. For language identification a set of statistical and non-statistical (rule based) techniques can be used to identify the language of a document with a certain precision. Script based recognition for Unicode character sets works to an extent, but still ambiguities might exist. For example, while most of the major Indian languages have a script of their own, Marathi, Hindi, Nepali and Sanskrit all occur in the Devanagari script, in which case script based heuristics may not work. Therefore for ambiguous scripts a better set of heuristics are required or statistical language identification techniques can help. In our work, we use a set of heuristic based techniques using fonts and character ranges to recognize character encodings such as UTF-8, ISCII or other proprietary encodings from HTML and PDF documents [10].

2.2 Encoding Converter

Once languages and character encodings are identified for a particular document or a particular piece of text within a document, such content needs to be converted into a standard encoding so that it can be indexed and retrieved. Unless all the character encodings of a single language are converted into a single encoding, it will not be possible to compare strings and match queries for retrieval. Therefore an automatic encoding converter is essential to enable information access. Again similar to language identification, a set of heuristic based, or statistical methods can be used to convert content of one encoding into another. For example, one could come up with a mapping table of various glyphs or glyph sequences of one encoding into another which can be then used to automatically convert one character encoding into another. Since Indian languages are syllabic in nature most of these are rarely one to one glyph mappings and end up being one to many or many to one. An example implementation of such encoding conversion into UTF-8 is discussed in [10].

2.3 Language Analyzer

A language analyzer program performs a set of natural language tasks on the indexable content and queries before passing it to the indexer or retrieval engine. Some of the standard and broad tasks of a language analyzer w.r.t IR would be to tokenize the input text and identify valid tokens of language, identify which of the valid tokens are worth indexing and which are not (also known as stop word identification) and performing some level of morphological processing on these tokens such as stemming. For instance, the example queries of "नेस्ले के उत्पादन" and "నెస్లే యొక్క ఉత్పత్తులు " may be converted into "नेस्ले उत्पाद" and "నెస్లే ఉత్ప " after stop-word elimination and stemming. In order to perform these tasks, a language analyzer specific to each language may be needed to be built or on the other hand a generic solution can be devised which may work for all the languages. Tokenization of string into tokens is a fairly standard heuristic based process, however stop word identification and stemming may have to be language specific if heuristics or word lists were used [11, 12]. Some research has been done on this problem using statistical stemming techniques [13, 14].

2.4 Indexer

An indexer program builds an index database. This index could be an inverted index or some other data structure suitable for IR with some meta-data useful for ranking and presentation of information to the user. Each of the indexable tokens obtained after stop-word removal and stemming are used by the indexer module to store information such as the documents in which the token has occurred and frequencies and position of such occurrences in these documents etc. Advanced indexers also have capabilities to store fielded indices. A fielded index would store the index information per field per document. For example a search engine application may want to provide ability to search a particular portion or meta-data of a document, such as the title of the document or the body text or web-based fields such as search within a given website etc. An indexer module should be capable of building data structures which enable fast retrieval of relevant documents and also enable ranking of these documents. The design of data-structure for index should also be sensitive to transaction aspects such as locking of indices for writes, updatability of index without difficulty etc.

2.5 Query Expansion

A query expansion module is an optional module for a search engine, which is used to add/rephrase/refine some keywords to the query. A query is viewed as user's expression of information need in the form of a set of keywords. For the user to express information need as keywords, the user has to guess the language model (keywords) occurring in the documents relevant to his/her information n.eed. Usually the users may not be able to guess the right keywords that may be found in the relevant documents thereby resulting in a poor user satisfaction. A query expansion module would automatically try to add more keywords to the user's query resulting in better user satisfaction. Query expansion could be achieved by adding synonymous words or semantically related words from a thesaurus or by using collocation statistics from a monolingual corpus. For example, a few terms can be added to the example query previous mentioned using co-occurrence statistics to add terms such as "मिल्कमेड, मेगि"(Milkmaid, Maggi) which co-occur with "नेस्त्ले" (Nestle) in a large Hindi corpus.

2.6 Query Translation

A query translation module is required to achieve CLIR. Query translation can be achieved using a bilingual lexicon or translation models built using parallel corpus. Bilingual lexicon can be obtained by digitizing and converting human readable dictionaries or can be obtained by manually building them with the help of linguists. Two types of bilingual lexica exist, one which just lists all the possible translations for a given source language word, the second one which not only lists the possible translations, but also comes with some meta-data for the translation, such as the synonyms, the probability of translation etc. Lexicon with translation probabilities are sometimes referred to as statistical lexicon. Statistical lexicon can be automatically built using parallel corpora. In some cases the direction of translation may be different from the direction of the CLIR task. For example, bilingual lexicon may provide translations for words of language L_1 into L_2, while the CLIR system might take in queries of L_2 to retrieve documents of L_1. In such situations proper set of heuristics need to be used in order to translate queries using a reverse lookup of the dictionary. In case of a statistical lexicon, a likelihood estimate for the translations needs to be computed. In the previous example query, "उत्पाद" would get translated as "product, produce, production, producer" etc., since we store only the stems and hence translate stems. Storing the complete words may result in higher translation precision, but would lead to lower recall since most of the word formations may not be found in the dictionary.

2.7 Query Transliteration

Not all queries can be translated into the target language using a bilingual lexicon. One of the important issues for query translation in CLIR systems is the handling of out of vocabulary words (or OOVs). Words such as proper nouns, words borrowed from a foreign language fall under this category. Such words are directly transliterated into the script of the target language. Such transliterations are not always exact transliterations and result in being influenced by the dialect and morphology of the target language. It is very common to find such keywords in the queries of CLIR systems where the source and target languages share a high number of cognates. Handling OOVs for CLIR systems across various Indian languages becomes very important since various Indian languages share higher proportion of cognates when compared with English. Heuristic based approaches using phoneme mappings or probabilistic transliteration models [15] can be used to address the problem of OOVs. This module is responsible to convert "नेस्ले" (written as Neslay in Roman script) and "నెస్లే " (written as Neslay in Roman script) as "Nestle".

2.8 Post-translation Query Expansion/Refinement

We differentiate post-translation query expansion from the previous query expansion module, since this module not only adds more keywords to the translated query, but also performs query refinement by eliminating noisy translations from the previous modules. Query refinement can be achieved by disambiguation from any context information during the query translation process or using techniques such as pseudo relevance feedback.

2.9 Document Retrieval

Document retrieval forms the central problem of IR. The problem of document retrieval is to retrieve documents relevant to the user's information need from a given document collection. The concept of relevance is very difficult to define. In naive terms, all the documents containing the keywords from the user's queries may be treated as relevant, however, such a definition will always have exceptions. A number of relevance definitions exist which not only are a function of queries and documents but a number of other parameters such as the user and the context in which the query was issued. However, most of the IR research ignores these other dimensions and largely tries to define relevance as a function of user's query and the documents from which a relevant subset needs to be identified.

2.10 Document Ranking

Document ranking problem is a function performed after retrieval of relevant documents. The goal of document ranking module is to find the best way to order the set of retrieved documents such that the items/documents are ordered in the decreasing order of their relevance. In many IR models ranking is achieved using the same relevance function as mentioned in the previous module. However in some relevance models such as boolean IR model [16] such a ranking is not inherent in the relevance function.

While each of the above mentioned sub-problems in a typical CLIR framework is a research worthy problem in itself, a set of frameworks / solutions can be defined in such a way that they solve more than one of the above sub-problems. Such frameworks could be, for example, a vector space framework, probabilistic framework or ontology based frameworks. In other words this entire problem of information retrieval can be viewed in two dimensions. One is the vertical dimension which is that of each of the sub-problems mentioned above. The other is a horizontal dimension of an approach or framework within which generalized solutions to an extent can be achieved which address more than one of the above mentioned sub-problems.

Traditionally, the language modeling technique for IR as described in [1] addresses only document retrieval and ranking modules. We extend the scope of our problem to also include query translation and query expansion modules.

3 Problem Definition

The problem being addressed in this paper is to design an underlying indexing mechanism that can support easy retrieval and ranking of documents for a cross-lingual query using the language modeling based ranking functions mentioned in this section. Before looking at the indexing solution, we need to look at the document retrieval and ranking functions in order to understand the computation that is required during the retrieval time for a cross-lingual query. This would in turn lead to the requirements for the indexing module.

We propose a language modeling (horizontal) approach to CLIR as shown in Figure 1 which cuts across a number of information access sub-problems (verticals). Statistical language models are probability distributions defined on sequences of elementary units, $P(u_1...u_n)$. Language modeling has been used in many NLP applications such as part-of-speech tagging, parsing, speech recognition, machine translation and information retrieval. Estimating sequences can become expensive in corpora where phrases or sentences can be arbitrarily long (data sparseness problem), and so these models are most often approximated using smoothed N-gram models based on unigrams, bigrams and/or trigrams.

In speech recognition and in most of the other applications of language modeling, these models refer to a probabilistic distribution capturing the statistics of the generation of a language, and attempt to predict the next word in a speech or text or state sequence. However the language models used in information retrieval may be viewed in a slightly different way. When used in information retrieval, a language model is associated with a document in a collection. With query Q as input, retrieved documents are ranked based on the probability that the document's language model (M_d) would generate the terms of the query, $P(Q|M_d)$. And such a language need not be sensitive to term sequence most of the times, since IR applications typically assume bag-of-words model for documents and queries.

The elementary unit in our IR models is a term. We define a term to be a sequence of characters which are either words or conflated words (stems). We propose to build and apply term based language models to various CLIR functions such as query enrichment, query translation, document retrieval and ranking. The language models being proposed here are trying to model semantically related words and unigram language models rather than actually modelling the sequences of terms as in other NLP applications such as speech recognition or part-of-speech tagging. In next few sub-sections we present term based language models which need to be supported by the underlying indexing mechanism.

3.1 Query Expansion

To achieve query expansion, it is required to automatically predict the keywords in the relevant documents to the given information need of the user. In order to achieve this in the absence of any further user context, one can add semantically related keywords to the query to enable retrieval of relevant documents with high precision and recall. Motivated by the fact that the meaning of a new concept can be learned from its usage with other concepts within the same concept [17], we automatically compute the dependencies of a word w on other words based on their lexical co-occurrence in the context of w in a sufficiently large corpus. A term collocation matrix is constructed by taking a window of length k words and moving it across the corpus at one term increments. All words in the window are said to co-occur with the first word with a given weight. Such a weight can be determined to be a function of the distance between the co-occurring words or can be assumed to be of equal weight. The weights assigned to each co-occurrence of terms are accumulated over the entire corpus. That is, if $n(w, k, w')$ denote the number of times word w' occurs k distance away from w when considered in a window of length K, then

$$\mathrm{P}(w', w) \;=\; \frac{\sum_{k=0}^{K} n(w, k, w')}{|C|}$$

where $|C|$ is the size of the corpus. This equation is equal to bigram probability distribution if the window length $K = 1$.

Once such term-by-term language models are built, it becomes very easy to expand given queries or documents using such language models.

3.2 Query Translation

Query translation functionality for the purpose of cross-language retrieval can be achieved in multiple ways. In this paper our focus is on using bilingual lexicon for this task and we describe how a bilingual lexicon can be used to build a probabilistic component which can be embedded into a retrieval model. The embedding of this translation component becomes easier since we assume the process of translation is independent of the actual retrieval and ranking and we perform translations at the term level.

We define translation probability of a target language term t_k from source language term s_i as $P(t_k|s_i)$. This model works well for CLIR since the system need not zero down onto a single possible translation for a given query. Therefore we compute the translation distribution for a given query and only refine the noisy translations with whatever evidence is available. In essence, this probabilistic translation model results in a probabilistic graphical model, where each source language term can get translated to multiple target language terms with different probabilities. This essentially gives us the graphical structure of dependencies (source/target dependencies) and the conditional probability distribution which constitutes our model.

Calculation of such a model can be again achieved in different ways. Ideally, the use of a bilingual parallel corpus would provide the best way to estimate the conditional probabilities and enables building of a statistical bilingual lexicon. However in [18] it was shown that, even in the absence of a bilingual parallel corpus, a conditional probability distribution can be achieved by assuming uniform probabilities of all possible translations to a given term. Apart from these we also propose assuming some underlying distribution based on the position of a given meaning in the lexicon, such as picking only one translation or assume the ordering of translations in the lexicon to bear an importance. Therefore, the query translation model should be capable of handling all possible term translations with weights associated with the translation.

3.3 Document Retrieval and Ranking

In [1], a language modeling framework was defined to retrieve and rank documents for the given information needs (queries). The model is non-parametric and does not assume any underlying classes such as relevance or irrelevance for the items to be retrieved. Every document/item is retrieved by computing the probability of its language model emitting the given query.

Therefore, the ranking function $R(Q, d)$ is defined as

$$\begin{aligned} R(Q,d) &= P(Q|M_d) \\ &= \prod_{w_j \in Q} P(w_j|M_d) \cdot \prod_{w_j \notin Q} (1 - P(w_j|M_d)) \end{aligned} \tag{1}$$

where, Q is the user's query containing a sequence of terms q_i and M_d is the document's language model which can emit a set of terms w_j.

Inspired by this model we extend this model to serve the various functions of CLIR. To include query expansion as part of the retrieval and ranking function, we obtain an expanded query $Q^{'}$ and rewrite the ranking function as

$$\begin{aligned} R(Q,d) &= P(Q^{'}|Q).P(Q^{'}|M_d) \\ &= \prod_{w_j \in Q'} P(w_j, Q).P(w_j|M_d) \cdot \prod_{w_j \notin Q'} (1 - P(w_j|M_d)) \end{aligned} \tag{2}$$

where $P(w_j, Q)$ gives the weight of the expanded term w_j in the context of the given query Q. Similarly query translation probabilities can be included as part of the ranking function, where $Q^{'}$ becomes the translated query or translated and expanded query as the sequence of the modules are plugged in. The actual ranking of documents happens on the probability that the document's language model emits the transformed query, while accounting for the joint probability of the transformation itself.

4 Multilingual Index Implementation

Given the language modeling framework described in the previous section, our task is to design an indexing mechanism to support such a CLIR system. For this purpose, we use a traditional inverted index concept and see how it can be extended for this task. We use Lucene's [2] inverted index mechanism and modify it to suit our model. Before analyzing Lucene's index file structure, we should understand the inverted index concept. An inverted index is an inside-out arrangement of documents in which terms take center stage. Each term points to a list of documents that contain it. On the contrary, in a forward index, documents take the center stage, and each document refers to a list of terms it contains. You can use an inverted index to easily find which documents contain certain terms. Lucene uses an inverted index as its index structure while a forward index facility also exists which can be optionally created. From Equations 2 and 3 it can be observed that, not only the query terms, but also all the terms contained in a document are required to be accessible while computing the rank of a given document. Therefore we will be using both inverted and forward index as our core index model as depicted in the Figure 1. The description of this Lucene core index model is given in the Section 4.1, followed by our modifications to it in Section 4.4.

4.1 Lucene Index Structure Overview

The fundamental concepts in Lucene are index, segments, document, field and term.

An *index* contains a sequence of *segments* which contain documents. Each *document* is a sequence of fields. A *field* is a named sequence of terms and a *term* is a string. The same string in two different fields is considered a different term. Thus terms are represented as a pair of strings, the first naming the field, and the second naming text within the field.

Segments: Lucene indexes may be composed of multiple sub-indexes, or segments. Each segment is a fully independent index, which could be searched separately. Indexes evolve by creating new segments for newly added documents or by merging existing segments. Searches may involve multiple segments and/or multiple indexes, each index potentially composed of a set of segments.

Each segment index maintains the following:

- *Field names*. This contains the set of field names used in the index.
- *Stored Field values*. This contains, for each document, a list of attribute-value pairs, where the attributes are field names. These are used to store auxiliary information about the document, such as its title, url, or an identifier to access a database. The set of stored fields are what is returned for each hit when searching. This is keyed by document number.
- *Term dictionary*. A dictionary containing all of the terms used in all of the indexed fields of all of the documents. The dictionary also contains the number of documents which contain the term, and pointers to the term's frequency and proximity data.
- *Term Frequency data*. For each term in the dictionary, the numbers of all the documents that contain that term, and the frequency of the term in that document.
- *Term Proximity data*. For each term in the dictionary, the positions that the term occurs in each document.

[2] http://Lucene.apache.org

Value	First byte	Second byte	Third byte
0	00000000		
1	00000001		
2	00000010		
...			
127	01111111		
128	10000000	00000001	
129	10000001	00000001	
130	10000010	00000001	
...			
16,383	11111111	01111111	
16,384	10000000	10000000	00000001
16,385	10000001	10000000	00000001
...			

Table 22: VInt Encoding Example

- *Normalization factors*. For each field in each document, a value is stored that is multiplied into the score for hits on that field.
- *Term Vectors*. For each field in each document, the term vector (sometimes called document vector or the forward index) is stored. A term vector consists of term text and term frequency.
- *Deleted documents*. An optional file indicating which documents are deleted.

4.2 Primitive Data Types

Lucene uses *Byte*, *UInt32* (4 bytes), *UInt64* (8 bytes), *VInt* and *Chars* as primitive data types. The *UInt* data types are used based on the address space requirements of 32-bits or 64-bits. A variable-length format for positive integers (*VInt*) is defined where the high-order bit of each byte indicates whether more bytes remain to be read. The low-order seven bits are appended as increasingly more significant bits in the resulting integer value. Thus values from zero to 127 may be stored in a single byte, values from 128 to 16,383 may be stored in two bytes, and so on.

It can be observed from Table 22 that the *VInt* datatype provides compression while still being efficient to decode, by using only required number of bytes to encode an integer address. Lucene writes unicode character sequences using Java's "modified UTF-8 encoding". A complete string is written as a *VInt* representing the length, followed by the actual character data.

4.3 Lucene Index Storage

The following describes the main index files in Lucene. Some of the file structures described below might not include all of the columns, but it won't affect the reader's understanding of the index file.

Segments file: A single file contains the active segments information for each index. This file lists the segments by name, and it contains the size of each segment. Table 23 describes the structure of this file.

Fields information file: Documents in the index are composed of fields, and this file contains the fields information in the segment. Tables 24 shows the master field structure. The fields master file has two slave files, namely field index file (structure described in Table 25) and field data file (described in Table 26).

Fields are numbered by their order in this file. Thus field zero is the first field in the file, field one the next, and so on. Note that, like document numbers, field numbers are segment relative.

Column name	Data type	Description
Version	UInt64	Contains the version information of the index files.
SegCount	UInt32	The number of segments in the index.
NameCounter	UInt32	Generates names for new segment files.
SegName	String	The name of one segment. If the index contains more than one segment, this column will appear more than once.
SegSize	UInt32	The size of one segment. If the index contains more than one segment, this column will appear more than once.

Table 23: Structure of Segments file

Column name	Data type	Description
FieldsCount	VInt	The number of fields.
FieldName	String	The name of one field.
FieldBits	Byte	Contains various flags. For example, if the lowest bit is 1, it means this is an indexed field; if 0, it's a non-indexed field. The second lowest-order bit is one for fields that have term vectors stored, and zero for fields without term vectors.

Table 24: Structure of Fields information file

Column name	Data type	Description
FieldValuesPosition	UInt64	This is used to find the location within the field data file of the fields of a particular document. Because it contains fixed-length data, this file may be easily randomly accessed. The position of document n's field data is the Uint64 at n*8 in this file.

Table 25: Structure of Fields Index file

Column name	Data type	Description
FieldCount	VInt	
FieldNum	VInt	
Bits	Byte	Only the low-order bit of Bits is used. It is one for tokenized fields, and zero for non-tokenized fields.
Value	String	

Table 26: Structure of Fields Data file

Column name	Data type	Description
TIVersion	UInt32	Names the version of this file's format.
TermCount	UInt64	The number of terms in this segment.
Term	Structure	This column is composed of three subcolumns: PrefixLength, Suffix, and FieldNum. It represents the contents in this term.
DocFreq	VInt	The number of documents that contain the term.
FreqDelta	VInt	Points to the frequency file.
ProxDelta	VInt	Points to the position file.

Table 27: Structure of Term information file

Column name	Data type	Description
DocDelta	VInt	It determines both the document number and term frequency. If the value is odd, the term frequency is 1; otherwise, the Freq column determines the term frequency.
Freq	VInt	If the value of DocDelta is even, this column determines the term frequency.

Table 28: Structure of the Frequency file

Column name	Data type	Description
PositionDelta	VInt	The position at which each term occurs within the documents

Table 29: Structure of the Position file

Column name	Data type	Description
TermDelta	VInt	It determines both the term number and joint frequency. If the value is odd, the joint frequency is 1; otherwise, the Freq column determines the joint frequency.
Freq	VInt	If the value of TermDelta is even, this column determines the joint frequency.

Table 30: Structure of the Collocation file

Column name	Data type	Description
TermDelta	VInt	It determines both the term number and conditional probability of translation.
TransProbability	VInt	

Table 31: Structure of the Term Translation file per language pair

Stored fields are represented by two files, one is field index and the other is field data.

Text information file: This core index file stores all of the terms and related information in the index, sorted by term. Table 27 shows the structure of this file. The pointers to term expansion file are stored in this file.

This file is sorted by Term. Terms are ordered first lexicographically by the term's field name, and within that lexicographically by the term's text.

Term text prefixes are shared. The PrefixLength is the number of initial characters from the previous term which must be pre-pended to a term's suffix in order to form the term's text. Thus, if the previous term's text was ”bone” and the term is ”boy”, the PrefixLength is two and the suffix is ”y”.

Frequency file: This file contains the list of documents that contain the terms, along with the term frequency in each document. If Lucene finds a term that matches the search word in the term information file, it will visit the list in the frequency file to find which documents contain the term. Table 28 shows the primary fields of this file. TermFreq entries are ordered by increasing document number.

DocDelta determines both the document number and the frequency. In particular, DocDelta/2 is the difference between this document number and the previous document number (or zero when this is the first document in a TermFreqs). When DocDelta is odd, the frequency is one. When DocDelta is even, the frequency is read as another VInt.

For example, the TermFreqs for a term which occurs once in document seven and three times in document eleven would be the following sequence of VInts: 15, 10, 3.

Position file: This file contains the list of positions at which the term occurs within each document. You can use this information to rank the search results. Table 29 shows the structure of this file.

For example, the TermPositions for a term which occurs as the fourth term in one document, and as the fifth and ninth term in a subsequent document, would be the following sequence of VInts: 4, 5, 4

4.4 Additional Meta-Index Files

We create three additional meta-index files, namely a collocation file to store co-occurrence frequencies of two terms (structure described in Table 30), a query translation file to store weighted term-based translations between languages (shown in Table 31) and a modified text information file to provide pointers to the first two meta-index

Column name	Data type	Description
TIVersion	UInt32	Names the version of this file's format.
TermCount	UInt64	The number of terms in this segment.
Term	Structure	This column is composed of three subcolumns: PrefixLength, Suffix, SuffixLength and FieldNum. It represents the contents in this term.
DocFreq	VInt	The number of documents that contain the term.
FreqDelta	VInt	Points to the frequency file.
ProxDelta	VInt	Points to the position file.
CollocationDelta	VInt	Points to the collocation file.

Table 32: Structure of modified Term information file

files. The text information file mentioned in the core Lucene index model is overridden to accommodate CLIR functions such as stemming and query translation. We first of all modify the term structure to include the SuffixLength to determine the length of the suffix that needs to be stemmed. Storing the suffix as part of the term data structure instead of actually indexing the conflated variants allows the CLIR system to be able to search with or without stemming at the same time. These modifications to the term datastructure can be seen from Table 32. Having customized the underlying index structure, we conducted CLIR experiments for Indian language - English language pairs which are discussed in the next section.

5 Index Performance Evaluation

The modified Lucene indexer was evaluated for indexing time and retrieval time. It is compared against the core Lucene index model which forms our baseline. The evaluations were conducted for 3 runs for both core Lucene model and language model based index. An average of the 3-runs is being reported. Table 33 shows the times taken by the indexer to index 100,000 documents belonging to Hindi and Telugu in UTF-8 format. For the baseline run we used the Lucene's standard analyzer by modifying it to accept UTF-8 content, while for the modified Lucene index we used our own language analyzer which has an Indian language rule based stemmer. In both these runs the JVM was given a runtime memory of 1GB. Similarly, Table 34 shows the times taken for document retrieval for both the runs. We issued a set of 25 Hindi and 25 Telugu queries in a sequence with a single thread and show the average time for retrieval and ranking. In the case of monolingual retrieval runs, only same language documents are retrieved while for cross-lingual run, the queries were also translated into the other language and both the language documents were retrieved. The results of our indexer with and without query expansion module for monolingual and cross-lingual tasks are being reported in Table 34.

5.1 Evaluation Setup

An Intel Xeon 3.0 GHz machine with 4GB RAM and an IDE disk with 7,200 RPM speed was used for our experiments. We used Lucene's version 1.4 as our baseline and extended it as described in Section 4.4. The programming was done using Sun's Java 1.4.2 on a Fedora Core 3 operating system. We indexed a set of 50,000 Hindi and 50,000 Telugu news article HTML documents which were encoded in UTF-8. The total size of the document collection was approximately 700MB, with an average file size of 7.14KB.

We also performed a load-test on our language modeling based indexer for monolingual retrieval, by simulating load with 200, 300, and 400 simultaneous users with a ramp-up time of 1 second. It can be seen from Figure 2 that the index is able to provide a good throughput. The load test for 200 users showed 0% drop rate[3], while the drop rates for 300 and 400 users were 1% and 2.75% respectively.

It can be observed from Tables 33 and 34 that the language modeling based index is slower than the ordinary TF.IDF based indexing model both during indexing and retrieval. However, functionally the language modeling based index provides superior search results than the vector space model both for monolingual retrieval [1] as well as cross-lingual retrieval [3]. This improvement is even higher in languages where resources are difficult to obtain. To give a perspective of the kind of functional improvement, we present the evaluation results of the experiments mentioned in [3] in Table 35. These experiments were conducted using CLEF 2006[4] dataset for CLIR evaluation of Hindi-English and Telugu-English tasks where the queries are in Hindi, Telugu and the information is in English. HNTD and TETD runs represent the baseline experiments using the TF.IDF based retrieval using dictionary based translation, while HNLM and TELM runs represent language modeling based retrieval for Hindi and Telugu queries respectively. Measures such as Mean Average Precision (MAP),

[3]A HTTP request which could not be served due to load is treated as a dropped request. *Drop rate* determines the frequency of failure of the system for a given load.

[4]Cross Language Evaluation Forum. http://www.clef-campaign.org

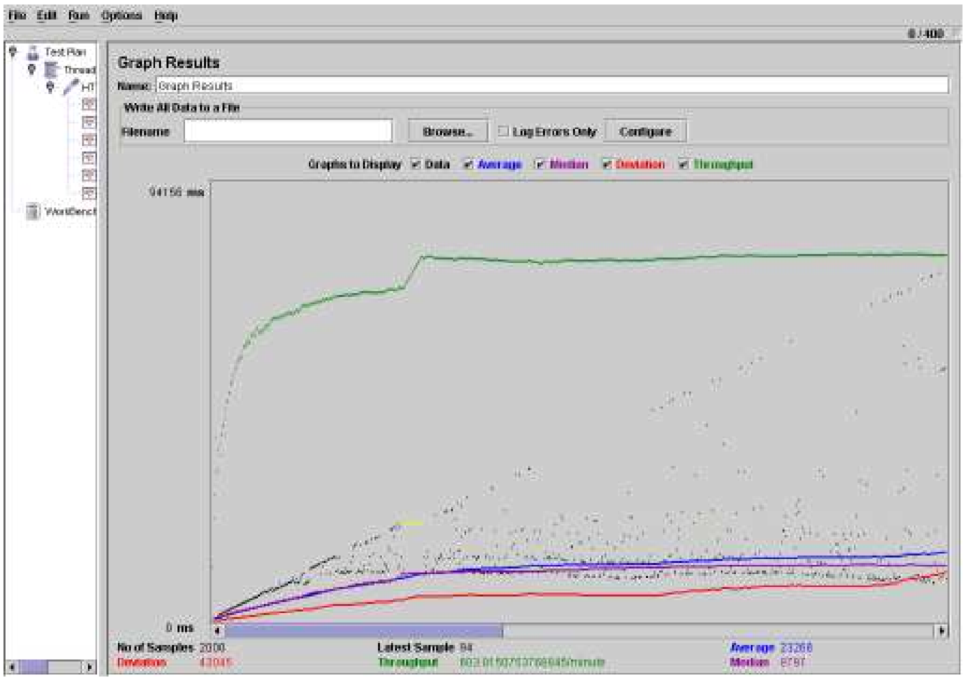

Figure 2: Load Testing on Our Index Model using 300 simultaneous users with a rampup of 1 second

Run	**Index time/1000 docs**	**Memory**
Baseline	89 sec	1024MB given to JVM
Lang. Model	359.71 sec	1024MB given to JVM

Table 33: Indexing Statistics

R-Precision and Geometric Average Precision (GAP) clearly show significant improvement. The performances of HNLM and TELM runs are comparable with the top performing systems at CLEF 2006.

6 Conclusion

In this paper we presented an indexing model for cross language information retrieval using language modeling technique. We used Lucene's inverted index model as our base model and extended it to support language modeling based CLIR. We conducted a set of experiments to test the performance of the new index model during index time as well as retrieval time. We found that consistently the new index model takes longer time to index and retrieve documents when compared to the vector space models. However, it has been shown in information

Run	**Retrieval type**	**Avg. Retrieval Time/Query**
Baseline	monolingual	667ms
LM	monolingual	941ms
LM + query expansion	monolingual	1009ms
LM	cross-lingual	1156ms
LM + query expansion	cross-lingual	1521ms

Table 34: Retrieval Statistics

Run-Id	total Relevant	Relevant-Retrieved	MAP (%)	R-Prec.(%)	GAP (%)	B-Pref.(%)
HNTD	1,258	650	12.52	13.16	2.41	10.91
TETD	1,258	554	8.16	8.42	0.36	7.84
HNLM	1,258	1051	26.82	26.33	9.41	25.19
TELM	1,258	1018	23.72	25.50	9.17	24.35

Table 35: Summary of average results for various CLIR runs

retrieval research that the language modeling based techniques have shown considerable improvement in the quality of search results provided in comparison to TFIDF algorithm. Therefore, we conclude that while there is an increase in time taken by the new index, it can provide better search results when compared to the traditional vector space based IR models.

Acknowledgment

We would like to thank the Department of Science and Technology, Government of India for partially funding this research.

References

[1] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1998, pp. 275–281.

[2] G. Salton and C. Buckley, "Term-weighting Approaches in Automatic Text Retrieval," *Information Process. Management*, vol. 24, no. 5, pp. 513–523, 1988.

[3] P. Pingali, K. K. Tune, and V. Varma, "Hindi, Telugu, Oromo, English CLIR Evaluation," *Lecture Notes in Computer Science: CLEF 2006 Proceedings*, 2007.

[4] L. Ballesteros and W. B. Croft, "Resolving ambiguity for cross-language retrieval," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1998, pp. 64–71.

[5] W. B. Croft and L. Ballesteros, "Phrasal translation and query expansion techniques for cross-language information retrieval," in *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1997, pp. 84–91.

[6] P. Clough and M. Sanderson, "Measuring pseudo relevance feedback & clir," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2004, pp. 484–485.

[7] D. He, D. W. Oard, J. Wang, J. Luo, D. Demner-Fushman, K. Darwish, P. Resnik, S. Khudanpur, M. Nossal, M. Subotin, and A. Leuski, "Making miracles: Interactive translingual search for cebuano and hindi," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 2, no. 3, pp. 219–244, 2003.

[8] L. Ballesteros and M. Sanderson, "Addressing the lack of direct translation resources for cross-language retrieval," in *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2003, pp. 147–152.

[9] J. Mayfield and P. McNamee, "Triangulation without translation," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2004, pp. 490–491.

[10] P. Pingali, J. Jagarlamudi, and V. Varma, "Webkhoj: Indian language ir from multiple character encodings," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*. Edinburgh, Scotland: ACM Press, 2006, pp. 801–809.

[11] L. S. Larkey, L. Ballesteros, and M. E. Connell, "Improving stemming for arabic information retrieval: light stemming and co-occurrence analysis," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2002, pp. 275–282.

[12] C. Fox, "A stop list for general text," *SIGIR Forum*, vol. 24, no. 1-2, pp. 19–21, r 90.

[13] J. Goldsmith, *Unsupervised Learning of the Morphology of a Natural Language*. USA: MIT Press, 2001.

[14] J. Mayfield and P. McNamee, "Single n-gram stemming," in *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. New York, NY, USA: ACM Press, 2003, pp. 415–416.

[15] L. S. Larkey, M. E. Connell, and N. Abduljaleel, "Hindi CLIR in thirty days," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 2, no. 2, pp. 130–142, 2003.

[16] W. Waller and D. H. Kraft, "A Mathematical Model of a Weighted Boolean Retrieval System," *Information Processing and Management*, 1979.

[17] K. Lund and C. Burgess, "Producing high-dimensional semantic spaces from lexical co-occurrence," in *Behavior Research Methods, Instrumentation, and Computers*, 1996, pp. 203–208.

[18] V. Lavrenko, M. Choquette, and W. B. Croft, "Cross-lingual relevance models," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2002, pp. 175–182.

The 23rd IEEE International Conference on Data Engineering (ICDE 2007)

April 15-20, 2007, Istanbul, Turkey

Sponsored by The IEEE Computer Society and Microsoft Research

http://www.icde20007.org

CALL FOR PARTICIPATION

ICDE 2007 Highlights

We have an exciting program designed to appeal both to researchers and to data engineering professions. It includes:

- 3 keynote speeches;
- 5 Advanced technology seminars (no additional fee);
- Presentations of 111 research papers and 11 industrial papers out of 659 submissions;
- 55 poster papers and 28 demos;
- 15 workshops in conjunction with the main conference;
- Banquet

Keynote Speeches

- Yannis Ioannidis, *Emerging Open Agoras of Data and Information,* April 17
- Ricardo Baeze-Yates, *Challenges in Distributed Web Retrieval,* April 18
- Laura M. Haas, *Information for People,* April 19

Advanced Technology Seminars

- Ehud Gudes, *Graph Mining - Motivation, Applications and Algorithms*
- Hanan Samet, *Techniques for Similarity Searching in Multimedia Databases*
- Ashraf Aboulnaga, Christiana Amza, Ken Salem, *Virtualization and Databases: State of the Art and Research Challenges*
- Yannis Ioannidis, Georgia Koutrika, *Personalized Systems: Models and Methods from an IR and DB perspective*
- Limsoon Wong, *An Introduction to Knowledge Discovery Applications and Challenges in Life Sciences*

Planned Workshops

- First International Workshop on Data Engineering and Social Networks, *April 15*
  http://www.bayareabrains.com/~Ensundaresan/icde2007-snworkshop.htm
- Workshop on Data Processing in Ubiquitous Information Systems, *April 15*
  http://www.cs.technion.ac.il/~ranw/DPUbiq/
- The Second IEEE International Workshop on Multimedia Databases and Data Management, *April 15*
  http://www.cs.fiu.edu/mddm07/right.htm
- The Third IEEE International Workshop on Databases for Next-Generation Researchers, *April 15*
  http://zakuro.fuis.fukui-u.ac.jp/SWOD2007/
- First International Workshop on Scalable Steam Processing Systems, *April 15*
  http://research.ihost.com/ssps07/
- Workshop on Data Mining and Business Intelligence, *April 15*
  http://www.dmbi2007.itu.edu.tr/
- Third International Workshop on Privacy Data Management, *April 16*
  http://www.ceebi.curtin.edu.au/PDM2007/
- First International Workshop on Ranking in Databases, *April 16*
  http://www.cs.uwaterloo.ca/conferences/dbrank2007/
- Second International Workshop on Services Engineering, *April 16*
  http://www.hrl.uoit.ca/~seiw2007/
- Second International Workshop on Self-Managing Database Systems, *April 16*
  http://db.uwaterloo.ca/tcde-smdb/
- Workshop on Text Data Mining and Management*, April 16*
  http://www.personal.psu.edu/qzz1/TDMM/
- Third International Workshop on Web Personalisation, Recommender Systems and Intelligent User Interfaces, *April 16*
  http://www.doc.ic.ac.uk/~gu1/WPRSIUI/WPRSIUI07/WPRSIUI07.html
- International Workshop on Ambient Intelligence, Media, and Sensing, *April 20*
  http://aria.asu.edu/aims07/
- Workshop on Spatio-Temporal Data Mining, *April 20*
  http://stdm07.uhasselt.be/
- First International Workshop on Security Technologies for Next Generation Collaborative Business Applications, *April 20*
  http://www.cs.cmu.edu/~jinghai/SECOBAP/

Registration Fees

W: Workshops only registration
CW: Conference and Workshops registration
C: Conference only registration
(until Feb 16, 2007/after Feb 16, 2007)

	W ($)	CW ($)	C ($)
Member	300/350	650/800	500/610
Nonmember	350/425	850/1050	625/760
Member Student	200/250	350/450	225/275
Nonmember Student	240/300	450/550	290/360

Welcome to Istanbul

The former capital of three successive empires Roman, Byzantine and Ottoman, the city Istanbul is a fascinating mixture of the past and present, old and new, modern and traditional. The museums, churches, palaces, mosques, and bazaars, and the sights of natural beauty seem inexhaustible. As you recline on the shores of the Bosporus at the sunset contemplating the red evening light reflected in the windows and slender minarets on the opposite shore, you understand, suddenly and profoundly, why so many centuries ago settlers chose to build on this remarkable site.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398