IMPLEMENTATION OF A TOPIC MAP DATA MODEL FOR A WEB-BASED INFORMATION RESOURCE

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Mustafa Kutlutürk
August, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

IMPLEMENTATION OF A TOPIC MAP DATA MODEL FOR A WEB-BASED INFORMATION RESOURCE

Mustafa Kutlutürk
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Özgür Ulusoy
August, 2002

The Web has become a vast information resource in recent years. Millions of people use the Web on a regular basis and the number is increasing rapidly. The Web is the largest center in the world presenting almost all of the social, economical, educational, etc. activities and anyone from all over the word can visit this huge place even though he does not have to stand up from his sit. Due to its hugeness, finding desired data on the Web in a timely and cost effective way is a problem of wide interest. In the last several years, many search engines have been created to help Web users find desired information. However, most of these search engines employ topic-independent search methods that rely heavily on keyword-based approaches where the users are presented with a lot of unnecessary search results.

In this thesis, we present a data model using *topic maps* standards for Web-based information resources. In this model, *topics*, *topic associations* and *topic occurrences* (called as *topic metalinks* and *topic sources* in this study) are the fundamental concepts. In fact, the presented model is a *metadata* model that describes the content of the Web-based information resource and creates virtual knowledge maps over the modeled information resource. Thus, semantic indexing of the Web-based information resource is performed for allowing efficient search and querying the data on the resource.

Additionally, we employ full text indexing in the presented model by using a widely accepted method that is *inverted file index*. Due to the rapid increase of data, the dynamic update of the inverted file index during the addition of new documents is inevitable. We have implemented an efficient dynamic update scheme in the presented model for the employed inverted file index method.

The presented topic map data model provides combining the powers of both keyword-based search and topic-centric search methods. We also provide a prototype search engine verifying that our presented model contributes very much to the problem of efficient and effective search and querying of the Web-based information resources.

Keywords: Metadata, XML, topic maps, Web-based information resource, Web search, inverted file index, dynamic update, Web data modeling, semantic indexing.

ÖZET

WEB TABANLI BİLGİ KAYNAKLARI İÇİN BİR KAVRAM HARİTASI VERİ MODELİ GERÇEKLEŞTİRİMİ

Mustafa Kutlutürk
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Ağustos, 2002

Web son yıllarda yoğun bir bilgi kaynağı olmuştur. Milyonlarca insan düzenli olarak Web'i kullanmaktadır ve kullanıcı sayısı hızla artmaktadır. Web hemen hemen tüm sosyal, ekonomik, eğitimsel v.b. alanlardaki uğraşları sunan dünyadaki en geniş bilgi merkezidir ve dünyanın herhangibir yerinden bir kişi bu büyük merkezi yerinden kalkmak zorunda bile kalmadan ziyaret edebilir. Çok büyük olmasından dolayı, istenilen veriye Web'de zaman ve maliyet açısından verimli bir yolla erişebilmek, önemli bir araştırma konusudur. Web kullanıcılarına istedikleri bilgiye erişebilme konusunda yardımcı olmak için, son birkaç yılda bir çok arama motoru üretilmiştir. Bununla beraber, bu arama motorlarının birçoğu kavram bağımsız arama metodları kullanmaktadır ve kullanıcılara birçok gereksiz arama sonuçları sunan anahtar kelime tabanlı yaklaşımlara dayanmaktadır.

Bu tezde, Web tabanlı bilgi kaynakları için *kavram haritaları* standartlarını kullanan bir veri modeli sunulmaktadır. Bu modelde, *kavramlar*, *kavram ilişkileri* ve *kavram oluşumları* (bu çalışmada *kavram metalinkleri* ve *kavram kaynakları* olarak anılacaklar) temel unsurlardır. Aslında, sunulan model bir "*metaveri*" model olup Web tabanlı bilgi kaynağının içeriğini tanımlayarak modellenen bilgi kaynağı üzerinde "gerçek bilgi haritaları" üretmektedir. Böylece, verimli bir veri araması ve sorgulaması için Web tabanlı bilgi kaynağının kavramsal endeksi oluşturulur.

Ayrıca geniş kabul gören *ters çevrilmiş dosya endeksi* kullanılarak, sunulan modelde tam kelime endeksi de uygulanmıştır. Verinin hızlı artışına bağlı olarak, yeni dökümanlar eklenmesi ve ters çevrilmiş dosya endeksinin dinamik olarak güncellenmesi kaçınılmazdır. Sunulan modelde, kullanılan ters çevrilmiş dosya endeksi için verimli bir dinamik güncelleme metodu uygulanmıştır.

Sunulan kavram haritası veri modeli, anahtar kelime tabanlı arama ve kavram merkezli arama metodlarının güçlerini birleştirmektedir. Sunulan modelin Web tabanlı bilgi kaynaklarının verimli ve etkin bir şekilde aranması ve sorgulanması problemine büyük katkıda bulunduğunu gösteren bir prototip arama motoru da sunulmaktadır.

Anahtar sözcükler: Metadata, XML, Web tabanlı bilgi kaynağı, Web araması, ters çevrilmiş dosya endeksi, dinamik güncelleme, Web veri modellemesi, kavramsal endeksleme.

Eşime ve Kızıma.

ACKNOWLEDGEMENTS

I would like to express my special thanks and gratitude to Assoc. Prof. Dr. Özgür Ulusoy, from whom I have learned a lot, due to his supervision, suggestions, and support. I would like especially thank to him for his understanding and patience in the critical moments.

I am also indebted to Assist. Prof. Dr. İbrahim Körpeoğlu and Assist. Prof. Dr. Uğur Güdükbay for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to especially thank to my wife and my parents for their morale support and for many things.

I am grateful to all the honorable faculty members of the department, who actually played an important role in my life to reaching the place where I am here today.

I would like to individually thank all my colleagues and dear friends for their help and support especially to İsmail Sengör Altıngövde, Barla Cambazoğlu and Ayşe Selma Özel.

I would also like to thank all my commanders, especially Alb. Levent Altuncu, Bnb. Kenan Dinç and Bnb. Nuri Boyacı, Yzb. Can Güldüren, Yzb Ferhat Gündoğdu in YBS headquarters, for their motivation, advices and supports.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of information available on line has been doubling in size every six months for the last three years. Due to this enormous growth, the World Wide Web (WWW) has grown to encompass diverse information resources such as personal home pages, online digital libraries, products and service catalogues, and research publications, etc. On the other hand, the ability to search and retrieve information from the Web efficiently and effectively is an enabling technology for realizing its full potential. Unfortunately, the sheer volumes of data on these Web-based information resources are not in a fixed format whereas the textual data is riddle and the resources usually contain non-textual multimedia data. Thus, there is a lack of a strict schema characterizing data on the Web.

XML-the eXtensible Markup Language- [1], adopted as a standard by the Word Wide Web Consortium (W3C), is the ideal format for structuring the information and enabling reuse and application independence. Once it becomes pervasive, it is not hard to imagine that many information resources will structure their external view as a repository of XML data [43]. But when it comes to retrieving information, XML on its own can only provide part of the solution [16]. Users are not interested in receiving megabytes of raw information as the result of a query; they want fast access to selected information in a given context; they look for intelligent navigation in the information pool while exploring the subject of their interest.

However, most of the traditional Web search engines employ topic-independent search methods that rely heavily on matching terms in a user's natural language query with terms appearing in a document (i.e., data-centric) [44]. So, they do not work as expected and are insufficient to improve retrieval effectiveness. Different approaches are under study to overcome the poor efficiency of Web search engines; one of those is the adoption of a *metadata format* and inclusion of metadata on that format in Web documents to describe the content. This means that, *metadata* may be a candidate together with other sources of evidence, such as keyword extracted from document title, and physical and logical document structures, to index and search Web-based information resources [45].

The *topics maps* standard (ISO 13250) [46] is an effort to describe a metadata model for describing the content of the information resources. A Web-based information resource, using metadata extracted from it by some techniques, can be modeled in terms of *topics*, relationships among topics (*topic metalinks*), and topic occurrences (*topic sources*) within information resources. This emerging standard provides interchangeable hypertext navigation layer above diverse Web-based information resources, and enable us to create virtual knowledge maps for the Web, our intranets, or even print materials. Thus, topic maps allow Web users to benefit from semantic data modeling that may be employed in a variety of ways, one of which is to improve the performance of search engines.

In the last several years, many search engines have been created to help Web users find desired information. Basically, two types of search engines exist: *General-purpose search engines* and *special-purpose search engines* [25]. The former ones aim at providing the capability to search all pages on the Web whereas the latter ones focus on documents in confined domains such as documents in an organization or in a specific subject area. Whatever the type, each search engine has text database that is defined by the set of documents to be searched and an index for all documents in the database is created in advance.

An indexing structure used by many IR systems is the *inverted file* index. An inverted file index consists of an *inverted list* for each term that appears in the document collection. A term's inverted list stores a document identifier and a term weight for every document in which the term appears. As a rule of thumb, the size of inverted lists for a full text index is roughly the same size as the text document database itself [32]. When adding new documents, rebuilding the inverted file and indexing the entire collection from scratch is expensive in terms of time and disk space. Therefore, dynamic update of inverted file index should be handled so that the cost of the update can be proportional to the size of the new documents being added not to the size of the database.

DBLP (Digital Bibliography & Library project), as an example of Web-based information resource, is a WWW server with bibliographic information on major journals and proceedings on computer science [37]. In DBLP, the table of contents of journals and proceedings may be browsed like in a traditional library. Browsing is complemented by search engines for author names and titles. In fact, these search engines are of special-purpose type search engine. The search mechanism employed in these search engines is very poor in efficiency and effectiveness. It uses boolean search scheme for titles whereas the author search is based on a simple sub-string test.

In this thesis, our main aim is modeling a Web-based information resource with topic maps standard by providing automated topic and metalink extraction from that resource for querying the modeled information resource effectively. In this sense, we present a topic map data model for DBLP bibliographic information in which we define the titles, authors, journals and conferences as *topics* (e.g., "PaperName", "AuthorName", "JourConfOrg", etc.), the URL of the publications as *topic sources*, and the relation between these topics as *topic metalinks* (e.g., *AuthorOf*, *AuthoredBy*, *JourConfOf*, *JourConfPapers*, etc.). Thus, we build a Web-based topic map database of DBLP.

Since the DBLP bibliography collection is presented in a semi-structured format (as XML documents), extraction of topics, sources and metalinks from DBLP bibliography is easy and straightforward except *RelatedToPapers* and *PrerequisitePapers* metaliks. These two metalink types can be determined by using the cosine similarity quotient of the titles. So, full text indexing should be employed for the titles of the publications. We have implemented inverted file index for this purpose and employed a dynamic update mechanism for adding new bibliographic entries without having to re-index the entire collection. We do not claim that the employed dynamic update scheme is a new and very efficient one in the field of inverted file indexing. However, it yields good performance both in time and disk space requirements. Finally, we have developed a prototype search engine for querying the bibliographic entries of DBLP.

As a result, main contributions of this thesis to the solution of the problem of effective Web search and querying are as follows:

- The topic map data model provided for a Web-based information resource (i.e., DBLP) is a semantic data model describing the contents of the documents (i.e., DBLP bibliography collection) in terms of topics and topic associations (i.e., topic metalinks), and therefore it constitutes a metadata model.

- In order to allow keyword-based searching and extraction of some metalinks (i.e., *RelatedToPapers* and *PrerequisitePapers*) in a more efficient way, we have implemented inverted file index for the titles and authors, and we have employed a dynamic update scheme for indexing the new bibliographic entries without having to re-index the entire collection.

- Finally, we have developed a prototype search engine in which a user can search the publications with the highest similarity to some query terms. In addition, a complementary search is provided by using the specified topics. Thus, the presented topic map data model provides consuming the power of both traditional indexing and knowledge-based indexing together.

In the second chapter of this thesis, we first briefly summarize XML, RDF and topic maps standard. Then, we discuss the inverted file index and the earlier works in the literature that implement full text indexing by using inverted file index. Also, the dynamic update schemes for inverted file indexing in these works are presented. The outlines of an automatic citation indexing system, CiteSeer are given. Finally, the specific Web-based information resource exploited in this thesis, DBLP is discussed.

In chapter three, firstly the structure of DBLP data is given, and then the presented topic map data model is described in details. After that, the implemented inverted file index method and the employed dynamic update scheme are explained. Then, a complete example is presented for a better understanding.

In the fourth chapter, the details of implementation applied on initial collection and dynamic sets are presented one by one. Experimental results for both the employed dynamic update scheme and the presented data model are reported in chapter five. In the sixth chapter, the design issues and the search process of the search engine, developed as a prototype implementation, are given.

Finally, in the seventh chapter we conclude our work.

Chapter 2

Background and Related Work

2.1 Turning the Web into Database: XML

XML-the eXtensible Markup Language- [1] has recently emerged as a new standard for data representation and exchange on the Internet, recommended by the World Wide Web Consortium (W3C). The basic ideas underlying XML are very simple: tags on the data elements identify the meaning of the data, rather than, e.g., specifying how the data should be formatted (as in HTML), and relationships between data elements are provided via simple nesting and references [2]. Since it is designed for data representation, XML is simple, easily parsed, and self-describing. Web data sources have evolved from small collection of HTML pages into complex platforms for distributed data access and application development. In this sense, XML promises to impose itself as a more appropriate format for this new breed of Web sites [3]. Furthermore it brings the data on the Web closer to databases, thereby making it possible to pose SQL-like queries and get much better result than from today's Web search engines. In contrast to HTML tags that do not describe the content semantics of HTML documents, as it is stated in [4], XML allows Web document designers to specify the semantics of data in XML documents.

2.2 Metadata: Data about Data

Metadata, as stated in [5], is a recent *coinage* though not a recent concept. It can be defined as data about data: information that communicates the meaning of other information. The term metadata has come to appear in the context with the Web in early 1990's and can refer to either type: the tagging system that defines a set of fields and its contents, or the contents of certain fields that act as descriptors for other resources. Meta-information has two main functions [5]. The first one is to provide a means to discover that data set exists and how it might be obtained or accessed. The second function is to document the content, quality, and features of a data set, indicating its fitness for use. The former function targets resource discovery where as the latter one, exploited in this thesis, targets resource description.

Metadata was defined in [6] as *superimposed* information that is the data placed over the existing information resources to help organizing, accessing and reusing the information elements in these resources. It is stated that the need for metadata over the Web can be justified with three key observations: (i) the increasing amount of digital information on the Web, (ii) emerging mechanism allowing to address the information objects in a finer granularity, (iii) the increasing amount of inaccurate and/or worthless information over the Web.

Managing and exploiting information appearing on the Web was stated as a problem [7], and four major aspects of it were explained which are data quality, query quality, answer quality, and integration of the heterogeneous sources. The solution to this problem is the metadata as one essential and common ingredient for all of these aspects to be realized. The author proposed that there are three main kinds of metadata: schema, navigational and associative [7]. Schema metadata has formal logic relationship to data instances and important in ensuring data quality. Navigational metadata provides information on how to get to an information resource (e.g., URL (Uniform Resource Locator), URL + predicate (query)). Associative metadata, exploited in this thesis, provides additional information for application assistance. Furthermore three main kinds of associative metadata are: (i) descriptive: catalogue record (e.g., Dublin Core), (ii) restrictive: content rating (e.g., PICS), (iii) supportive: dictionaries, thesauri (e.g., PROTÉGÉ).

The *principles* of metadata are stated as modularity, extensibility, refinement, and multilingualism [8]. In fact the principles are those concepts judged to be common to all domains of metadata and which might inform the design of any metadata schema or application. Metadata modularity is a key organizing principle for environments characterized by vastly diverse sources of content, styles of content management, and approaches to resource description [8]. In a modular metadata world, data elements from different schemas as well as vocabularies can be combined in a syntactically and semantically interoperable way. Metadata systems must allow for extensions so that particular needs of a given application can be accommodated. The refinement principle encapsulates the specification of particular schemes or value sets that define the range of values for a given element, usage of controlled vocabularies, and addition of qualifiers that make the meaning of an element more specific [8]. It is essential to adopt metadata architecture that respect linguistic and cultural diversity. A basic starting point in promoting global metadata architecture is to translate relevant specification and standard documents into a variety of languages.

After a brief discussion of metadata, in the upcoming section we will describe the two main emerging standards: RDF and Topic Maps that facilitate the creation and exchange of metadata over the Web. Before that let us have a look at the concept that has motivated the idea behind RDF standard: The Semantic Web.

2.2.1 Semantic Web

The Web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help [9]. However one can easily recognize the fact that most of the information on the Web is currently for human consumption. Thus, there can be two basic approaches for browsing the Web; the first approach is to train the machines to behave like people and the second one is the Semantic Web approach. The former approach is worked in field of artificial intelligence and is not in the scope of this thesis. The latter approach, the Semantic Web [9], develops languages for expressing information in a machine processable form.

Actually, the word *semantic* comes from the Greek words for sign, signify, and significant, and today it used as *relating to meaning*. Semantic Web is an extension of the current Web in which information is given well defined meaning, better enabling computers and people to work in cooperation [10]. It is the idea of having data on the Web defined and linked in a way that it can be used for more effective discovery, automation, integration, and reuse across various applications.

Semantic Web can also be defined as a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. It can be thought as being an efficient way of representing data on the World Wide Web, or as a globally linked database. The vision of Semantic Web is stated in [11] as the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. In order to make this vision a reality for the Web, supporting standards and policies must be designed to enable machines to make more sense of the Web.

A comparison between the Semantic Web and the object-oriented systems is made in [12]. Consequently the main difference is that a relationship between two objects may be stored apart from other information about two objects in the Semantic Web approach whereas in the object-oriented system information about an object is stored in an object: the definition of the class of an object defines the storage implied for its properties. Furthermore Semantic Web, in contrast to relational databases, is not designed just as a new data model; it is also appropriate to the linking of data of many different models [12]. One of the great things it will allow is to add information relating different databases on the Web, to allow sophisticated operations to be performed across them.

For the Web to reach its full potential, it must evolve into a Semantic Web, providing a universally accessible platform that allows data to be shared and processed by automated tools as well as by people. In fact the Semantic Web is an initiative of the World Wide Web Consortium (W3C), with the goal of extending the current Web to facilitate Web automation, universally accessible content, and the 'Web of Trust' [10]. Meanwhile a particular priority of W3C is to use the Web to document the meaning of the metadata and their strong interest in metadata and Semantic Web has prompted development of the Resource Description Framework (RDF).

2.2.2 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [13] promises an architecture for the Web metadata and has been advanced as the primary enabling infrastructure of the Semantic Web activity in W3C. It can be viewed as an additional layer on top of XML that is intended to simplify the reuse of vocabulary terms across namespaces. RDF is a declarative language and provides a standard way for using XML to represent metadata in the form of statements about properties and relationships of items on the Web.

RDF is a foundation for metadata; it provides interoperability between applications that exchange machine-understandable information on the Web [13]. It emphasizes facilities to enable automated processing of Web resources. RDF can be used in a variety of application areas, for example: in resource discovery to provide better search engine capabilities, in cataloging for describing content and content relationships available in a particular Web site, page or digital library, in content rating, in describing collections of pages that represent a single logical document.

RDF includes two parts: the "Model and Syntax specification" [13] and the "Schema Specification" [14]. Model and Syntax Specification part introduces a model for representing RDF metadata as well as a syntax for encoding and transporting this metadata in a manner that maximizes the interoperability of independently developed Web servers and clients. Basic RDF model consists of three object types: *resources*, *properties*, and *statements*. *Resources* are the objects (not necessarily Web accessible) which are identified using Uniform Resource Identifier (URI). The attributes that are used to describe resources are called *properties*. RDF *statements* associate a property-value pair with a resource; they are thus triples composed of a *subject* (resource), a *predicate* (property), and an *object* (property value). A concrete syntax is also needed for creating and exchanging the metadata that is defined and used by RDF data model. Basic RDF syntax uses XML encoding and requires XML namespace facility for expressing RDF statements.

Actually RDF provides a framework in which independent communities can develop vocabularies that suit their specific needs and share vocabularies with other communities. The descriptions of these vocabulary sets are called RDF *Schemas* [14]. A schema defines the meaning, characteristics, and relationships of a set of properties. RDF data model, as described in the previous paragraph, defines a simple model for describing interrelationships among resources in terms of named properties and values. However RDF data model does not provide any mechanism for defining these properties and the relationships between these properties and other resources. That is the role of RDF Schema [14]. More succinctly, the RDF Schema mechanism provides a basic type system for use in RDF models. RDF Schemas might be contrasted with XML Document type Definitions (DTDs) and XML Schemas. Unlike an XML DTD or Schema, which gives specific constraint on the structure of XML document, an RDF Schema as it is stated in [14], provides information about the interpretation of the statements given in an RDF data model. Furthermore, while an XML Schema can be used to validate the syntax of an RDF/XML expression, since a syntactic schema would not be sufficient for RDF purposes, RDF Schemas may also specify constraints that should be followed by these data models.

2.2.3 Topic Maps

A topic map is a document conforming to a model used to improve information retrieval and navigation using topics as hubs in an information network [15]. Topic maps are created and used to help people find the information they need quickly and easily. The Topic Maps model, an international standard (ISO/IEC 13250:2000), is an effort to provide a metadata model for describing the underlying data in terms of *topics*, *topic associations*, and *topic occurrences*. In the following, definitions of the key concepts of this model are given as stated in [16], [17] and [18].

- *Topic*: A *topic* can be any "thing" whatsoever – a person, an entity, a concept, really anything – regardless of whether it exists or has any other specific characteristics about which anything may be asserted. The topic map standard defines subject as the term used for the real word "thing" and by the way the *topic* itself stands for it. As an example, in the context of Encyclopedia, the country Italy, or the city Rome are topics.
- *Topic type*: A topic has a *topic type* or perhaps multiple topic types. Thus, Italy would be a topic of type "country" whereas Rome would be of type "city". In other words, topic types represent a typical *class-instance* (or IS-A) relationship and they are themselves defined as topics by the standard.
- *Topic name*: Topics can have a number of characteristics. First of all they can have a *name* – or more than one. The standard provides an element form that consists of at least one *base name*, and optional *display* and *sort names*.
- *Topic occurrence*: As a second characteristic, a topic can have one or more *occurrences*. An *occurrence* of a topic is a link to an information

resource (or more than one) that is deemed to be somehow relevant to the subject that the topic represents. Occurrences may be of any number of different types (e.g., "article", "illustration" and "mention"). Such distinctions are supported in the standard by the concept of *occurrence role*. For example, topic "Italy" is *described* in an article, "Rome" is *mentioned* at a Web site of tourism interest. In this example the "article" and the "Web site" are topic *occurrences* whereas "describe" and "mention" are corresponding *occurrence roles*. In fact such occurrences are generally outside the topic map document itself and they are pointed at using whatever mechanisms the system supports, typically HyTime[19], Xpointers [20] or Xlink[21].

- *Topic association*: Up to now, the concepts of topic, topic type, name, occurrence and occurrence role allow us to organize our information resources according to topic, and create simple indexes, but not much more. The key to their true potential lies in their ability to model relationships between topics. For this purpose topic map standard provides a construct that is called *topic association*. A *topic association* is (formally) a link element that asserts a relationship between two or more topics. Just as the topics and the occurrences, the topic associations can also be grouped according to their type with the concept of *association types* that are also regarded as topics. In addition to this, each topic that participates in an association has a corresponding *association role* which states the role played by that topic in the association. The association roles are also topics. Thus, the assertion "Rome" *is-in* "Italy" is an *association* whereas the *is-in* is the *association type* and the *association roles* for the player topics "Rome" and "Italy" are "containee" and "container", respectively.

Scope, *identity (and public subjects)*, and *facets* are additional constructs that enrich the semantics of the model. When we talk about the topic "Paris", we can refer to the capital city of France, or the hero of Troy. In order to avoid ambiguities like this any assignment of characteristic (name, occurrence, or a role in association) to a topic must be valid within certain limits. The limit of validity of such an assignment is called its *scope*, scope is defined in terms of *themes*, and themes are also topics. In this example, topic “Paris” is of type “city” in the scope of (topic) “geography” and of type “hero” in the scope of “mythology”. Sometimes the same subject is represented by more than one topic link. This can be the case when two or more topic maps are merged. The concept that enables this is that of *public subject*, and the mechanism used is an attribute (the *identity* attribute) on the topic element. The final feature of the topic map standard to be considered in this thesis is the concept of the *facet*. Facets basically provide a mechanism for assigning property-value pairs to information resources and they are used for applying metadata which can then be used for filtering the information resources.

A topic map is an SGML (or XML) document that contains a topic map data model and organizes large set of information resources by building a structured network of semantic links over the resources [4]. An important aspect of this standard is that topic associations are completely independent of whatever information resources may or may not exist as occurrences of those topics. Since a topic map reveals the organization of knowledge rather than the actual occurrence of the topics, it allows a separation of information into two domains: the topic domain and the occurrence (document) domain. Because of this separation, different topic maps can be overlaid on information pools to provide different views to different users [4].

Indexes, glossaries and thesauri are all ways of mapping the knowledge structures that exist implicitly in books and other sources of information. In [16], it is explained how the topic maps can offer much more facilities than an ordinary index can do. For instance, any word in an index has one or several references to its locations in the information source and there is no distinction among these references. On the other hand topic maps can make references distinguished by the help of topic occurrence roles (i.e., one reference may point to the "description" of the word whereas another one may point to just a "mention" of it). Additionally a glossary can be implemented using just the bare bones of the topic map standard too. One advantage of applying the topic map model to thesauri is that it becomes possible to create hierarchies of association types that extend the thesaurus schema [16]. *Semantic networks* are very similar to that of the topics and associations found in indexes. As it is stated in [17], by adding the topic/occurrence axis to the topic/association model, topic maps provide a means of "bridging the gap" between knowledge representation and the field of information management. This is what the topic map standard achieves.

Another important issue with the topic maps is to determine how to cover internal representation of the model. Actually there are a number of approaches whereas the two main ones are object-based and relational [22]. The object-based approach requires that as structures in a Topic map instance are processed by the import mechanism, the objects relating to each construct can be created. The classes used to construct object model are Topic Map, Topic, Occurrence, Topic Association, Topic Association Role, Name, Facet and Facet Value.

The relational approach requires the creation of tables such as Topic, Topic Association and Topic Association Role, and construction of many join tables. One example of this approach, exploited in this thesis, is stated in [23]. A "*Web information space*" metadata model was proposed for Web information resources. In this model, information space is composed of three main parts: (i) *information resources* which are XML or HTML documents on the Web, (ii) *expert advice repositories* (specified using topics and relationships among topics called as metalinks) that contain domain expert-specified model of information resources, modeled as topic maps and stored as XTM documents, (iii) personalized information about users, captured as *user profiles* (XML documents), contain users' preferences and users' knowledge about the topics. Furthermore, a query language SQL-TC (Topic-Centric SQL) was proposed in [23] that is an integrated SQL-like topic-centric language for querying Web-based information resources, expert advice repositories and personalized user information. The language is general enough to be operational on any underlying expert data model, as long as the model supports metadata objects and their attributes. SQL-TC queries are expressed using topics, metalinks, and their sources and produce highly relevant and semantically related responses to user queries within short amounts of time.

Actually, Topic maps and RDF are similar in that they both attempt to alleviate the same problem of findability in the age of infoglut, define an abstract model, and an SGML/XML based interchange syntax [24]. However, there are some distinctions between these two standards. One of them, as stated in [24] maybe the key one, is that topic maps take a topic-centric view whereas RDF takes a resource-centric view. Topic maps start from topics and model a semantic network layer above the information resources. In contrary, RDF starts from resources and annotates them directly. Thus, RDF is said to be suitable for "resource-centric" applications whereas topic maps apply to "topic (knowledge)-centric" applications [24].

2.3 Indexing Documents on the Web

Finding desired data on the Web in a timely and cost-effective way is a problem of wide interest. In the last several years, many search engines have been created to help Web users find desired information. Each search engine has a text database that is defined by the set of documents that can be searched by the search engine [25]. Usually, an index for all documents in the database is created in advance. For each term that represents a content word or a combination of several content words, this index can identify the documents that contain the term quickly. The American Heritage Dictionary defines *index* as follows:

(in • dex) 1. Anything that serves to guide, point out or otherwise facilitate reference, as: **a.** An alphabetized listing of names, places, and subjects included in a printed work that gives for each item the page on which it may be found. **b.** A series of notches cut into the edge of a book for easy access to chapters or other divisions. **c.** Any table, file, or catalogue.

Although the term is used in the same spirit in the context of document retrieval and ranking, it has a specific meaning. Some definitions proposed by experts are: “The most important of the tools for information retrieval is the *index*-a collection of terms with pointers to places where information about documents can be found”, “*Indexing* is building a data structure that will allow quick searching of the text” and “An *index term* is (document) word whose semantics helps in remembering the document’s main theme”.

As it is stated in [25], there are basically two types of search engines. General-purpose search engines aim at providing the capability to search all pages on the Web. Google, AltaVista, and Excite are the most well known ones of this type. The other ones, special-purpose search engines, on the other hand, focus on documents in confined domains such as documents in an organization or in a specific subject area. ACM Digital Library, Citeseer and DBLP are of this type that focus on research papers in academic literature.

Actually, as stated in [26], four approaches to indexing documents on the Web are: (i) human or manual indexing; (ii) automatic indexing; (iii) intelligent or agent-based indexing; (iv) metadata, RDF, and annotation-based indexing. Manual indexing is currently used by several commercial, Web-based search engines, e.g., Galaxy, Infomine, and Yahoo. Since the volume of information on the Internet increases very rapidly, manual indexing is likely to become obsolete over the long term. Many search engines rely on automatically generated indices, either by themselves or in combination with other technologies (e.g., AltaVista, Excite, HotBot) [26]. In the third approach, intelligent agents are most commonly referred to as crawlers, but are also known as ants, automatic indexers, bots, spiders, Web robots, and worms. One of the promising new approaches is the use of metadata, i.e., summaries of Web page content or sites placed in the page for aiding automatic indexers. Dublin Core Metadata standard [27] and Warwick framework [28] are two well-publicized ones among the metadata standards for Web pages in the scope of fourth approach.

Different search engines may have different ways to determine what terms should be used to represent a given document [25]. For example, some may consider all terms in the document (i.e., *full-text indexing*) while others may use only a subset of the terms (i.e., *partial-text indexing*). Other examples of different indexing techniques involve whether or not to remove *stopwords* and whether or not to perform *stemming*. Furthermore, different stopword lists and stemming algorithms may be used by different search engines [25].

Thus, recognizing the concept of indexing documents on the Web, in the upcoming sections, we will discuss *inverted file* indexing and *citation* indexing methods that are two common ones in the literature. Before that, let us have a look at one of the most commonly used document weighting and similarity scheme; Vector-Space retrieval model as stated in [29].

2.3.1 Overview of Vector-Space Retrieval Model

Under the vector-space model, documents and queries are conceptually represented as vectors. If m distinct words are available for content identification, document d is represented as a normalized m-dimensional vector $D = \langle w_1, \ldots, w_m \rangle$, where w_j is the "weight" assigned to the j^{th} word t_j. If t_j is not represented in d, then w_j is 0. For example, the document with vector $D_1 = \langle 0.5, 0, 0.3, \ldots, \rangle$ contains the first word in the vocabulary (say, by alphabetical order) with weight 0.5, does not contain the second word, and so on.

The weight for a document word indicates how statistically important it is. One common way to compute D is to first obtain an un-normalized vector $D' = \langle w'_1, \ldots, w'_m \rangle$, where each w'_i is the product of a term frequency (*tf*) factor and an inverse document frequency (*idf*) factor. The *tf* factor is equal (or proportional) to the frequency of the i^{th} word within the document. The *idf* factor corresponds to the content discriminating power of the i^{th} word: a word that appears rarely in documents has a high *idf*, while a word that occurs in a large number of documents has a low idf. Typically, *idf* is computed by $\log(n/d_i)$, where n is the total number of documents in the collection, and d_i is the number of document having the i^{th} word. (If a word appears in every document, its discriminating power is 0. If a word appears in a single document, its discriminating power is as large as possible.) Once D' is computed, the normalized vector D is typically obtained by dividing each term by $\sqrt{\sum^{m}_{i=1}} (w'_i)^2$.

Queries in the vector-space model are also represented as normalized vectors over the word space, $Q = \langle q_1,\dots,q_m \rangle$, where each entry indicates importance of the word in the search. Here q_j is typically a function of the number of times word t_j appears in the query string times the *idf* factor for the word. The similarity between a query q and a document d, $sim(q,d)$, is defined as the inner product of the query vector Q and the document vector D. That is,

$$sim(q,d) = Q \cdot D = \sum^{m}_{j=1} q_j \cdot w_j$$

Notice that similarity values range between zero and one, inclusive, because Q and D are normalized.

2.3.2 Inverted File Index

An inverted file index has two main parts: a search structure or *vocabulary*, containing all of the distinct values being indexed; and for each value an *inverted list*, storing the identifiers of the records containing the value [29]. Queries are evaluated by fetching the inverted lists for the query terms, and then intersecting them for conjunctive queries and merging them for disjunctive queries. Once the inverted lists have been processed, the record identifiers must be mapped to physical record addresses. This is achieved with an *address table*, which can be stored in memory or on disk [29].

In [30], inverted index is defined as a data structure that maps a word, or atomic search item, to the set of documents, or set of indexed units, that contain that word – its *postings*. An individual posting may be a binary indication of the presence of that word in a document, or may contain additional information such as its frequency in that document and an offset for each occurrence. Since access to an inverted index is based on a single key (i.e., the word of interest) efficient access typically implies that the index, as exploited in this thesis, is either sorted, or organized as a hash table [30]. In this work, several space and time optimizations have been developed for maintaining an inverted text index using a B-Tree and a heap file. They have used a B-tree to store short postings list for each indexed word. When a posting list becomes too large for the B-Tree, portions of it are pulsed to a separate heap file. The heap is a binary memory file with contiguous chunks allocated as necessary for the overflow posting lists. For very long postings lists, heap chunks are linked together with pointers.

An inverted file indexing scheme based on compression was proposed in [31]. The only assumption made is that sufficient memory is available to support an in-memory vocabulary of the words used in the collection. If the search structure does not fit into memory, it is partitioned with an abridged vocabulary of common words held in memory and the remainder held on disc. It was declared that this would still be effective. Since many words only occur once or twice in the collection for large vocabularies, the cost of going to disc twice for rare words is offset by the fact that they have dramatically reduced the set of candidate records [31]. Three methods for indexing word sequences in association with an inverted file index are considered which are word sequence indexing, word-level indexing, and use of signature file for word pairs. One of the drawbacks in this scheme is that insertion of new records is complex and is best handled by batching, and database creation can be expensive. Also, there is some possibility of a bottleneck during inverted file entry decoding if long entries must be processed to obtain a small number of answers to some query.

2.3.3 Dynamic update of Inverted Indexes

An important issue with inverted file indexing is updating the index dynamically as new documents arrive. Traditional information retrieval systems, of the type used by libraries assume a relatively static body of documents [32]. Given a body of documents, these systems build inverted list index from scratch and stores each list sequentially and contiguously on disk (with no gaps). Periodically, e.g,, every weekend, new documents would be added to the database and a brand new index would be built. In many of today's environments, such full index reconstruction is not feasible. One reason is that text document databases are more dynamic. In place index update is inevitable for this type of systems. Since updating the index for each individual arriving document is inefficient, the goal is to batch together small number of documents for each in-place index update [32].

In [32], a new dynamic dual structure is proposed for inverted lists. In this structure, lists are initially stored in a "short list" data structure and migrated to a "long list" data structure as they grow. A family of disk allocation policies have been implemented for long lists whereas each policy dictates where to find space for a growing list, whether to try to grow a list in place or to migrate all or parts of it, how much free space to leave at the end of a list, and how to partition a list across disks. In this work, it is assumed that when a new documents arrives it is parsed and its words are inserted an in-memory inverted index.

One important fact that is taken into account in [32] is that some inverted lists (corresponding to frequently appearing words) will expand rapidly with the arrival of new documents while others (corresponding to infrequently appearing words) will expand slowly or not at all. In addition, new documents will contain previously unseen words. Short inverted lists (of infrequently appearing words) have been implemented in fixed size blocks where each block contains postings for multiple words. The idea is that every list starts off as a short list; when it gets "too big" it becomes a long list [32]. They have placed long inverted lists (of frequently appearing words) in variable length contiguous sequences of blocks on disk.

An inverted file index is defined in [33] such that it consists of a record, or inverted list, for each term that appears in the document collection. A term's inverted list, as exploited in this thesis, stores a document identifier and weight for every document in which the term appears. The inverted lists for a multi gigabyte document collection will range in size from a few bytes to millions of bytes, and they are typically laid out contiguously in a flat inverted file with no gaps between the lists [33]. Thus, adding to inverted lists stored in such a fashion requires expensive relocation of growing lists and careful management of free-space in the inverted file.

Actually, inverted index has been implemented on top of a generic persistent object management system in [33]. The INQUERY full-text information retrieval system and Mneme persistent object store schemes have been exploited in this work. In INQUERY, a term dictionary, built as a hash table, contains entries in which an entry contains collection statistics for the corresponding term and inverted lists are stored as Mneme objects where a single object of the exact size is allocated for each inverted list. The basic services provided by Mneme are storage and retrieval of objects and an object is a chunk of contiguous bytes that has been assigned a unique identifier.

The main extension made to old inverted file in [33] is that instead of allocating each inverted list in a single object of exact size, lists are allocated using a range of fixed size objects in which the sizes range from 16 to 8192 bytes by powers of 2 (i.e., 16, 32, 64, …, 8192). When a list created, an object of the smallest size large enough to contain the list is allocated. When it exceeds the object size, a new object of the next larger size is allocated, the contents of the old object are copied into new object, and the old object is freed. If a list exceeds the largest object size (8192 bytes) then a link list is started for these ones. Note that, the best performance was obtained with this model when documents are added in largest batches.

2.3.4 Citation Indexing

References contained in academic articles are used to give credit to previous work in the literature and provide a link between the “citing” and “cited” articles. A citation index [34] indexes these links between articles that researchers make when they cite other articles. As it is stated in [35], citation indexes can be used in many ways, e.g. (i) it can help to find other publications which may be of interest, (ii) the context of citations in citing publications may be helpful in judging the important contributions of a cited paper, (iii) it allows finding out where and how often a particular article is cited in the literature, thus providing an indication of the importance of the article, and (iv) a citation index can provide detailed analyses of research trends. The Institute for Scientific Information (ISI) [36] produces multidisciplinary citation indexes. One of them is the *Science Citation Index* (SCI) that is intended to be a practical, cost-effective tool for indexing the significant scientific journals.

An automatic citation indexing system (CiteSeer), which indexes academic literature in electronic format (e.g. postscript and pdf files on the Web), is presented in [35]. CiteSeer downloads papers that are made available on the Web, converts the papers to text, parses them to extract the citations and the context in which the citations are made in the body of the paper, and stores the information in a database. It provides most of the advantages of traditional (manually constructed) citation indexes, including: literature retrieval by following citation links (e.g. by providing a list of papers that cite a given paper), the evaluation and ranking of papers, authors, journals based on the number of citations, and identification of research trends [35]. Papers related to a given paper can be located using common citation information or word vector similarity. Compared to current commercial citation indexes, main advantage of CiteSeer is that index process is completely automatic (requiring no human effort) as soon as publications are available on the Web whereas the main disadvantage is that since many publications are not currently available on-line, CiteSeer is not able to provide as comprehensive an index as the traditional systems.

2.4 DBLP: Computer Science Bibliography

Digital libraries are a field of very active and diverse research. Many institutions are experimenting with on-line publications, electronic journals, interactive catalogues, search engines for technical reports, or other form of electronic publishing [37]. For example, ACM has developed an Electronic Publishing Plan [38]. The primary goals of BIBWEB project at the University of Trier, as stated in [37], are the followings:

- Bibliographic information on major CS journals and proceedings should be available on WWW for everybody (especially for students and researchers) without a fee.
- Databases often provide sophisticated search facilities, but most systems lack browsers that allow users to explore the database contents without knowing what to search for. A bibliographic information system should support both: searching and browsing.
- BIBTEX is a standard format to exchange bibliographies. The BIBWEB system will be compatible to BIBTEX (BIBTEX compatibility is currently restricted).
- The publication process and references between papers form a complex Web. Hypertext is an interesting tool to model some aspects of this Web.
- The World-Wide Web is used as the main interface to BIBWEB.

DBLP (Digital Bibliography & Library project) is the starting point of the BIBWEB project at the University of Trier [37]. The DBLP server, which is initially focused on Database systems and Logic Programming, now provides bibliographic information on major computer science journals and proceedings [39]. DBLP is file-system based and managed by some simple homemade tools to generate the authors' pages. There is no database management system behind DBLP; the information is stored in more than 125000 files [40]. The programs used to maintain DBLP are written in C, Perl and Java – they are glued together by shell scripts.

The initial DBLP server was a small collection of tables of contents (TOCs) of proceedings and journals from the field of database system research and logic programming [40]. The next idea was to generate “author pages” where an author page lists all publications (co)authored by a person. The generation of these pages works in two steps: In the first step all TOCs are parsed and then all bibliographic information is printed into a huge single text file “TOC_OUT”. After all parsing has been done, a second program (mkauthors) is started that reads TOC_OUT into a compact main memory data structure, produces a list of all author pages and the file AUTHORS which contains all author names. In the search process, the files AUTHORS and TOC_OUT are inputs to two CGI-programs “author” and “title” respectively, and a C written program performs “brute force” search (a sequential search) for each query [40]. Actually, bibliographic records of DBLP fit into the XML framework. In Figure 2.1, you will find a sample of bibliographic records.

```
<article key="GottlobSR96">
<author>Georg Gottlob</author>
<author>Michael Schrefl</author>
<author>Brigitte Röck</author>
<title>Extending Object-Oriented Systems with
Roles.</title>
<pages>268-296</pages>
<year>1996</year>
<volume>14</volume>
<journal>TOIS</journal>
<number>3</number>
<url>db/journals/tois/tois14.html#GottlobSR96</url>
</article>
```

Figure 2.1: A sample DBLP bibliographic record

Chapter 3

Topic Map Data Model

In this thesis, our main aim is to model a specific information resource on the web with topic map standards by employing metadata in the form of topics, topic associations, and topic sources. Information resources, dealt with in this work, are generally found on the Web as XML or HTML documents and must be modeled somehow for an efficient querying of them. The specific Web-based information resource that we have chosen to model is the DBLP (Digital Bibliography & Library Project) bibliography collection. We have modeled this source with topic map standards and maintained a topic map database. This chapter captures the structure of the presented data model and its details conceptually. Although we have given the overview of existing approach employed by DBLP in the previous section, let us first have a look at the structure of DBLP data in more details.

3.1 Structure of DBLP Data

Actually, DBLP bibliography data is a 90 megabyte sized XML document containing bibliographic entries for approximately 225,000 computer science publications (e.g., conference and journal papers, books, master and PhD theses, etc.). The full version of DTD for DBLP data is provided in Appendix A. You will find a part of this DTD in Figure 3.1.

```
<dblp>
<!ELEMENT
dblp(article|inproceedings|proceedings|book|
incollection|phdthesis|mastersthesis|www)*>
<!ENTITY % field
"author|editor|title|booktitle|pages|year|address|
journal|volume|number|month|url|ee|cdrom|cite|publis
her|note|crossref|isbn|series|school|chapter">
<!ELEMENT article       (%field;)*>
<!ATTLIST article  key CDATA #REQUIRED ...  >
<!ELEMENT inproceedings (%field;)*>
<!ATTLIST inproceedings key CDATA #REQUIRED>
<!ELEMENT proceedings   (%field;)*>
<!ATTLIST proceedings   key CDATA #REQUIRED>
...
<!ELEMENT author    (#PCDATA)>
<!ELEMENT title     (%titlecontents;)*>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT year      (#PCDATA)>
<!ELEMENT journal   (#PCDATA)>
<!ELEMENT url       (#PCDATA)>
...
</dblp>
```

Figure 3.1: Part of DTD for DBLP data

As one can easily understand form this DTD, there is a root element (dblp), delimited by <dblp> and </dblp> tags, and it contains a lot of elements such as article, inproceedings, proceedings, etc. Each of these elements has also sub-elements such as author, title, year, etc. In fact, these sub-elements are bibliographic entries for the element in which they are included, and can be visualized as metadata for that publication (article, inproceedings, proceedings, etc.).

In our implementation, we have processed three main record types in DBLP data (i.e., `article, proceedings` and `inproceedings`). Each of these records has its own sub-elements such as `author, title, year, url, journal, booktitle,` etc. All of these sub-elements are descriptive metadata for the publication they belong to.

In each of these three types of elements, there is a `key` attribute that distinguishes it from the other elements. For any `article` record, `title` sub-element contains the name of that article whereas the `author` sub-element(s) contains the author(s) of it. The `year` sub-element states the publication date in the year format (e.g., 1985, 1998, 1999, etc.). The address of the Web page where anyone can find that publication is specified in the `url` sub-element. Almost all of the `article` elements have only `journal` sub-element (not `booktitle` element) that contains the name of the journal in which the corresponding article was published.

Every `inproceedings` element contains the same sub-elements as any `article` element does, except the `journal` sub-element. Instead of this sub-element, almost all of the `inproceedings` elements have the `booktitle` sub-element that specifies the name of the conference/symposium in which the corresponding paper is reported. On the other hand, any `proceedings` element contains the information about any conference/symposium and is followed by a group of `inproceedings` elements which participated in that conference/symposium respectively. So, beside `journal` or `booktitle` sub-element, it has `publisher` sub-element. Additionally, in any `proceedings` element, `editor` sub-element is substituted for `author` sub element.

A fragment DBLP data file containing the example of the related records is presented in Figure3.2.

```
<?xml version="1.0"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
<article key="journals/ai/KumarK83">
<author>Vipin Kumar</author>
<author>Laveen N. Kanal</author>
<title>A General Branch and Bound Formulation for Understanding
and Synthesizing And/Or Tree Search Procedures.</title>
<pages>179-198</pages>
<year>1983</year>
<volume>21</volume>
<journal>Artificial Intelligence</journal>
<number>1-2</number>
<url>db/journals/ai/ai21.html#KumarK83</url>
</article>
...
<proceedings key="conf/ssd/95">
<editor>Max J. Egenhofer</editor>
<editor>John R. Herring</editor>
<title>Advances in Spatial Databases, 4th International Symposium,
SSD'95, Portland, Maine, USA, August 6-9, 1995,
Proceedings</title>
<series href="db/journals/lncs.html">Lecture Notes in Computer
Science</series>
<volume>951</volume>
<publisher>Springer</publisher>
<year>1995</year>
<isbn>3-540-60159-7</isbn>
<url>db/conf/ssd/ssd95.html</url>
</proceedings>
<inproceedings key="conf/ssd/KuijpersPB95">
<ee>db/conf/ssd/KuijpersPB95.html</ee>
<author>Bart Kuijpers</author>
<author>Jan Paredaens</author>
<author>Jan Van den Bussche</author>
<title>Lossless Representation of Topological Spatial
Data.</title>
<pages>1-13</pages>
<cdrom>SSD/1995/P001.pdf</cdrom>
<booktitle>SSD</booktitle>
<year>1995</year>
<crossref>conf/ssd/95</crossref>
<url>db/conf/ssd/ssd95.html#KuijpersPB95</url>
</inproceedings>
</dblp>
```

Figure 3.2: A fragment of DBLP data file

3.2 The Presented Data Model

In fact, the presented topic map data model is a semantic data model describing the contents of the Web–based information resource (DBLP bibliographic collection in this work) in terms of *topics*, relationships among topics (called *metalinks*) and *topic sources*. Therefore, it constitutes a metadata model and allows much more efficient querying of the modeled information resource. In our implementation, we have used relational database techniques to organize topic-based information and maintained a Web-based topic map database. One advantage of this organization is that database is relatively stable. For example, if some changes occur in the information resource, our database will not change greatly and what we have to do is only change some columns of the related tuples. In the following, you will find the details of the three main entities employed in the model (i.e., *topics*, *topic sources*, and *metalinks*).

3.2.1 Topics

In the second chapter, the definition of topic in topic map standard was given in detail. We have assumed that a topic is an object with a certain amount of information and has the following attributes like the ones in [23] with some extensions.

Topics (Tid: integer, TName: string, TType: string, TVector: string, TAdvice: float)

- *T(opic)id* (of type integer) is a system defined id that uniquely identifies the corresponding topic. Since the topics are extracted one by one as the documents come, Tid is assigned in an incrementally manner, internally used for efficient implementation and not available to users.
- *T(opic)Name* (of type string) contains either a single word or multiple words and characterizes the data in the information resources. "I. S. Altıngövde", "DEXA", "2001" and "SQL-TC: A Topic Centric Query

Language for Web-Based Information Resources" are the examples of topic names.

- *T(opic)Type* (of type string) specifies the type of the topic. For example, the topics "I. S. Altıngövde", "DEXA", "2001" and "SQL-TC: A Topic Centric Query Language for Web-Based Information Resources" are of the type "AuthorName", "JourConfOrg", "PublicationDate" and "PaperName", respectively.

- *T(opic)Vector* (of type string) contains pairs of the *term-id* and *term weight* (in the alphabetical order) for each term existing in the TName field of the corresponding topic. Actually, the term weights are calculated according to TF/IDF weighting scheme and the chain of them is the vector of the topic in vector-space retrieval model. It is null for some topics except the topics of type "AuthorName" and "PaperName".
- *T(opic)Advice* (of type float) is the importance value of that topic. If there is more than one expert modeling the information resource, say n experts (i.e., E_i, $1 \leq i \leq n$), the expert E_i states his/her advice on topics as a Topic-Advice function TAdvice() that assigns an importance value to topics from one of [0,1] U {No, Don't-Care}.

The attributes (TName, TType) constitute a key for the topic entity, and the Tid attribute is also a key for topics. The main difference of our model from the one in [23] is that they have maintained the topic-based information as XML topic map (XTM) documents whereas we have maintained a topic map database. Another difference is the absence of the *T(opic)Domain* attribute. Since all of the topics extracted from DBLP collection have the same domain (i.e, Computer Science Publication), we have not specified the topic domain attribute for topic instances. They have also accepted more than one expert assigning a topic importance value in the range [0,1] and {No, Don't Care}. However, in this work, for the sake of simplicity, we have accepted that all topic instances have the same importance value, namely, one (1).

In our implementation, we have extracted five types of topics from DBLP data. These types are given in table3.1.

TName	TType
Title of the paper	PaperName
Name of the author(s) in the form of First Initial, Second Initial, Last Name specification.	AuthorName
Name of the journal or conference in which the paper is published.	JourConfOrg
The date of the journal or conference in the year format.	PublicationDate
Concatenation of the name and date of the journal or conference.	JourConf-and-Year

Table 3.1: Topic types for the DBLP bibliography data

The TName attribute of a topic of type "PaperName" is the string that is delimited by `<title>` and `</title>` tags in the `article` and `inproceedings` elements of DBLP bibliography data. Each `author` is also considered as a topic of type "AuthorName". The name of the journal/conference in which the paper is published is a topic of type "JourConfOrg". It can be found between `<journal>`…`</journal>` or `<booktitle>`…`</booktitle>` tags of the corresponding paper according to type of the entry (i.e., `journal` sub-element for `article` elements and `booktitle` sub-element for `inproceedings` elements). The topics of type "PublicationDate" are the dates in YYYY format and they are embedded between `<year>` and `</year>` tags. Finally, the topics of type "JourConf-and-Year" are obtained by concatenating the name and the date of the journal/conference in which the paper is published.

3.2.2 Topic Sources

Topic source entity corresponds to topic occurrence in the topic map standard and contains additional information about topic sources. In fact, it refers the information in the actual Web sites. A source entity has the following attributes.

Sources (Sid: integer, Web-address: string, Role: string, SAdvice: float)

- *S(ource)id* (of type integer) is a system-defined id that uniquely identifies a topic source. Since the topic sources are extracted one by one as the documents come (like topic extraction), Sid is assigned in an incrementally manner, internally used for efficient implementation and not available to users.
- *Web-address* (of type string) is the (URL) of the document that contains the topic. "db/journals/ai/ai21.html#KumarK83" is an example of Web-address of a topic source.
- *Role* (of type string) specifies the source type of topic source. For example, a topic of type "PaperName" can have more than one source. One of these sources may be the Web site that contains the full document (e.g, ps or pdf version), another may be any html page that mentions about that paper. Here, the "Website" and "Mentions" are examples of the role of a topic source.
- *S(ource)Advice* (of type float) is the importance value of that topic source. Like TAdvice() function, the expert E_i states his/her advice on topic sources as a Source-Advice function SAdvice() that assigns an importance value to sources from one of [0,1] U {No, Don't-Care}.

The attribute Sid is a key for sources. At this point, another difference between our model and the one in [23] comes out that they have added a *Source* attribute to the topic entity that contains the set of Sid(s) specifying the topic sources of corresponding topic. Instead, we have stored the additional information about topic sources (Web-address, Role, etc.) in sources entity as described above, whereas *Tsources* entity identifies the relation between a topic instance and a source instance. The Tsources entity has the following attributes.

Tsources (Tsid: integer, Tid:integer, Sid: integer)

- *T(opic)s(ource)id* (of type integer) is a system-defined id that uniquely identifies a topic source instance. Since the topic sources are extracted one by one as the documents come (like topic extraction), Tsid is assigned in an incrementally manner, internally used for efficient implementation and not available to users.
- *T(opic)id* (of type integer) is the id of the topic in topic entity.
- *S(ource)id* (of type integer) is the id of the topic source in source entity.

We have topic sources only for topics of type "PaperName". According to the DTD of the DBLP bibliography data, topic sources for topics of type "AuthorName" and "JourConfOrg" are not specified in DBLP dataset. If they had been specified, they might have been the main home page (URL) of the author and conference organization, respectively. Actually, topic sources for topics of type "AuthorName" and "JourConfOrg" can be obtained by searching corresponding topic on the Web. However, this task is beyond the scope of this work.

Each topic instance of type "PaperName" has at least one source. This source is the URL of the corresponding paper and contains the abstract, the title as well as a ps or a pdf copy of the paper. This URL is specified in the `<url>` subelement of the corresponding publication (i.e., `<article>` or `<inproceedings>` element), extracted and stored in the Web-address attribute of the source entity, and assigned an Sid value. Since all these URLs are the Web-addresses of the copy of the papers (ps or pdf version), all of the instances in Sources entity will have the same Role that is "Website". We have assumed that all topic sources have the same importance value, namely, one (1). So SAdvice of all the instances in the Sources entity will also be the same (i.e., 1) like the TAdvice of topic instances.

Finally, the Tid of the topic and the Sid of the source that belong to that topic will together build an instance of Tsources entity. Then, a Tsid value will be assigned to this instance in an incrementally manner.

3.2.3 Topic Metalinks

Topic Metalinks represent the relationships among the topics and correspond to topic associations in the topic map standard. Topic Metalinks allow Web designers to define metadata-based navigational pathways on the Web. They are stable in that, once defined, they will rarely change and be secondary to the primary link mechanism (i.e., hyperlinks) on the Web. In our model, the Metalinks entity has the following attributes.

Metalinks(Mid: integer, Mtype:string, Antecedentid: integer, Consequentid: integer, MAdvice: float)

- *M(etalink)id* (of type integer) is a system-defined id that uniquely identifies a metalink instance. Since the metalinks, like the topics, are extracted one by one as the documents come, Mid is assigned in an incrementally manner, internally used for efficient implementation and not available to users.
- *M(etalink)Type* (of type string) specifies the type of the relation between topics. For example, assume that a topic T1 of type "AuthorName" is the author of a topic T2 which is of type "PaperName". The metalinks representing the relation between T1 and T2 can be defined by the signatures "T2 → *AuthorOf* T1" and "T1 → *AuthoredBy* T2" respectively. In this example, *AuthorOf* and *AuthoredBy* are the examples of MType attribute. Note that the signatures are similar to ones defined in [23].

- *Antecedentid* (of type integer) is the Tid (assigned in Topics entity) of the topic that participates on the left side of the metalink instance. When the metalink instance is visualized as a statement, Antecedentid is the object of this statement.
- *Consequentid* (of type integer) is the Tid of the topic that participates on the right side of the metalink instance. When the metalink instance is visualized as a statement, Consequentid is the subject of this statement.
- *MAdvice (of type float)*, similar to TAdvice and SAdvice, is the importance value of that metalink instance. Just like the TAdvice and SAdvice, expert E$_i$ also states his/her advice on topic metalinks as a Metalink-Advice function MAdvice() that assigns an importance value to metalinks from one of [0,1] U {No, Don't-Care}.

For the DBLP dataset, we have defined the metalink types that are specified in Table 3.2. Note that the TTypes of the topics participating in the metalinks are shown in parentheses (e.g., PaperName, AuthorName, etc.).

M(etalink)Type	**Antecedentid**	**Consequentid**
AuthorOf	Tid (PaperName)	Tid (AuthorName)
AuthoredBy	Tid (AuthorName)	Tid (PaperName)
PublicationDateOf	Tid (PaperName)	Tid (PublicationDate)
InPublicationDate	Tid (PublicationDate)	Tid (PaperName)
JourConfOf	Tid (PaperName)	Tid (JourConf-and-Year)
JourConfPapers	Tid (JourConf-and-Year)	Tid (PaperName)
RelatedToPapers	Tid (PaperName)	Tid (PaperName)
PrerequisitePapers	Tid (PaperName)	Tid (PaperName)

Table 3.2: Metalink types for the DBLP bibliography data

AuthorOf and *AuthoredBy* metalink instances represent the relationship between a topic of type "PaperName" and a topic of type "AuthorName" and can be stated by the signatures "T2 → *AuthorOf* T1" and "T1 → *AuthoreBy* T2" respectively. In other words, it is the relation between a publication (article, inproceedings) and its author(s). In *AuthorOf* metalink instance, the Consequentid is the Tid of the "AuthorName" topic that is the author of the corresponding "PaperName" topic whereas Antecedentid is the Tid of that "PaperName" topic. *AuthoredBy* metalink defines the same relation in the opposite direction.

PublicationDateOf and *InPublicationDate* metalink instances define the relation between a paper and the year when the paper is published in. The signatures "T2 → *PublicationDateOf* T3" and "T3 → *InPublicationDate* T2" can state these two metalinks types. Here T2 and T3 are of types "PaperName" and "PublicationDate" respectively. In *PublicationDateOf* metalink instance, Antecedentid is the Tid of the "PaperName" topic whereas Consequentid is the Tid of "PublicationDate" topic that states when the paper is published. *InPublicationDate* metalink represents the same relation in the opposite direction.

JourConOf and *JourConfPapers* metalink instances simply define the relation between a paper and the journal/conference in which the paper is published. They can be stated by the signatures "T2 → *JourConfOf* T4" and "T4 → *JourConfPapers* T2", respectively. As defined in topics entity section, a "JourConf-andYear" topic is the concatenation of the name of the journal/conference and its date (e.g., DEXA 2001). Thus, in *JourConfOf* metalink instances, Consequentid is the Tid of "JourConf-andYear" topic whereas the Antecedentid is the Tid of the "PaperName" topic that is published in that journal/conference reported in a specific date. Similarly, *JourConfPapers* metalink defines the same relation in the opposite direction too.

Metalink types explained in the previous paragraphs are all easy to understand and the extraction of them is also straightforward. MAdvice of all these metalinks will also be the same (i.e., 1.0) like the TAdvice of topic instances and SAdvice of topic sources.

However, *RelatedToPapers* and *PrerequisitePapers* metalinks are not as easy and straightforward as the other ones. Both of these metalinks represent the relationships between two "PaperName" topics. Thus, each of Consequentid and Antecedentid is the Tid of the topic of type "PaperName".

Instances of *RelatedToPapers* metalink type represent related (similar) paper pairs and can be stated by the signature "T2 → *RelatedToPapers* T1". We can determine whether two papers are related by using the cosine similarity quotient of the TNames of the papers. Actually, the cosine quotient is calculated by sim() function employed in vector-space retrieval model which is explained in Section 2.3.1. If the value of the cosine quotient is very small (e.g., less than a predefined threshold Tsim1), we assume that the "PaperName" topics T1 and T2 are not related to each other. Otherwise (i.e., ≥Tsim1), T1 and T2 are related papers and the metalink instance T2 → *RelatedToPapers* T1 is inserted into Metalinks table that is Consequentid and Antecedentid will be assigned T1 and T2, respectively and MType will be *RelatedToPapers*. The value of the cosine quotient becomes the importance value of the metalink instance (i.e, MAdvice).

PrerequisitePapers metalink instance e.g, T2 → *PrerequisitePapers* T1, states that in order to understand the "PaperName" topic T2, the reader should first read "PaperName" topic T1. For locating *PrerequisitePapers* metalink instances, we have defined the following association rules: "PaperName" topic T1 is prerequisite of "PaperName" topic T2 (T2 → *PrerequisitePapers* T1) (a) if the two papers T1 and T2 have at least one common author, have the *RelatedToPapers* relationship between each other, and the publication date of the paper T1 is earlier than the publication date of T2, or (b) If the two papers have no common author, but have a very high similarity (e.g., greater than a predefined threshold Tsim2) and the publication date of T1 is earlier than the publication date of T2. Thus, if one of the rules holds the metalink instance is inserted into metalinks table, that is Consequentid and Antecedentid will be assigned T1 and T2, respectively and MType will be *PrerequisitePapers*. The importance value of the metalink instance will be the cosine similarity quotient of the topics and MAdvice will be assigned this value.

We have used some information retrieval techniques in order to calculate cosine similarity quotient of the “PaperName” topics. Once we construct the topic map database from the initial huge DBLP bibliographic collection, of course our work will not stop there. New researches on computer science will be conducted and reported in the journals or conferences, and the database will have to be updated with these new bibliographic information datasets (called as *dynamic sets* throughout this work) relatively much more smaller than the initial dataset. Thus, the employed IR techniques should not only be efficient and effective but also they should allow dynamic update mechanism. In our implementation, we have employed the inverted file index structure that is the most promising technique used by many IR systems. In the upcoming section, we explain the structure of the employed indexing mechanism in more details.

3.3 Inverted File Index

An inverted index is a set of document lists, one list for each term or concept in a document collection [41]. Each list identifies the documents that contain that term. The entries in the list are called postings, where a posting is a pair of document identifier and a term weight. An inverted index can drastically improve query response time because only those documents containing terms in common with the query need to be considered. Many approaches were proposed for implementing inverted indexes and we have explained some of them in Section 2.3.2. When the documents need to be added to a collection, re-indexing entire collection is not desirable. Because the cost of an update is proportional to the size of the database, not the size of the update. So, dynamic update of the inverted index is an important issue and should be handled in the index organization.

In our implementation, the topics of type “PaperName” and “AuthorName” are indexed distinctly. We have defined two distinct collections. One of them contains the topics of type “PaperName” (i.e., the titles of the papers) and named as *title collection* whereas the other one contains the topics of type “AuthorName” and named as *author collection*. Actually, the titles and the authors in the collections correspond to the documents in the definition of inverted index. For the sake of simplicity, we will describe only the indexing of the title collection. Note that, we have also implemented the indexing of the author collection.

The first assumption that we have made is that sufficient memory is available to support an in-memory vocabulary of the words used in the title collection. In our implementation, we have made two passes over the DBLP dataset. In the first pass, topics, sources and metalink instances (except the *RelatedToPapers* and the *PrerequisitePapers* metalink instances) are extracted and inserted into the topics, sources, tsources and metalinks tables, respectively. Actually, these are relational database tables and the structures of them were explained in details in the previous section.

During the insertion of the topics in the first pass, the TType of the inserted topic is controlled. If the inserted topic is “PaperName” topic (i.e., title of the paper) then it means that it belongs to the title collection. The TName field is tokenized and processed term by term. After removing stop-words, each term is stemmed and put into an in-memory wordlist with some additional information. At the end of the first pass, we have an in-memory wordlist of all the words that appear in all topic names of type “PaperName”. As well as the term itself, the wordlist also contains the rank of the term (number of topics containing that term), the start offset of the allocated block where the postings list for that term is written on the disk.

In the second pass, each inserted “PaperName” topic is processed one by one. For each term in the TName field of a “PaperName” topic, the weight of the term is calculated by employing TF/IDF term-weighting scheme. After calculating all term weights, unnormalized vector representation of the topic is obtained and then the term weights are normalized as described in Section 2.3.1. Now we have normalized vector at hand that contains term index and term weight pairs. After processing all “PaperName” topics, the postings lists for all terms are obtained. A posting in the list contains the Tid of the topic containing the term and the weight of the term in the normalized vector of that topic. Finally, the postings lists of all terms are written into a file on the disk in an ascending ordered manner with respect to Tid values.

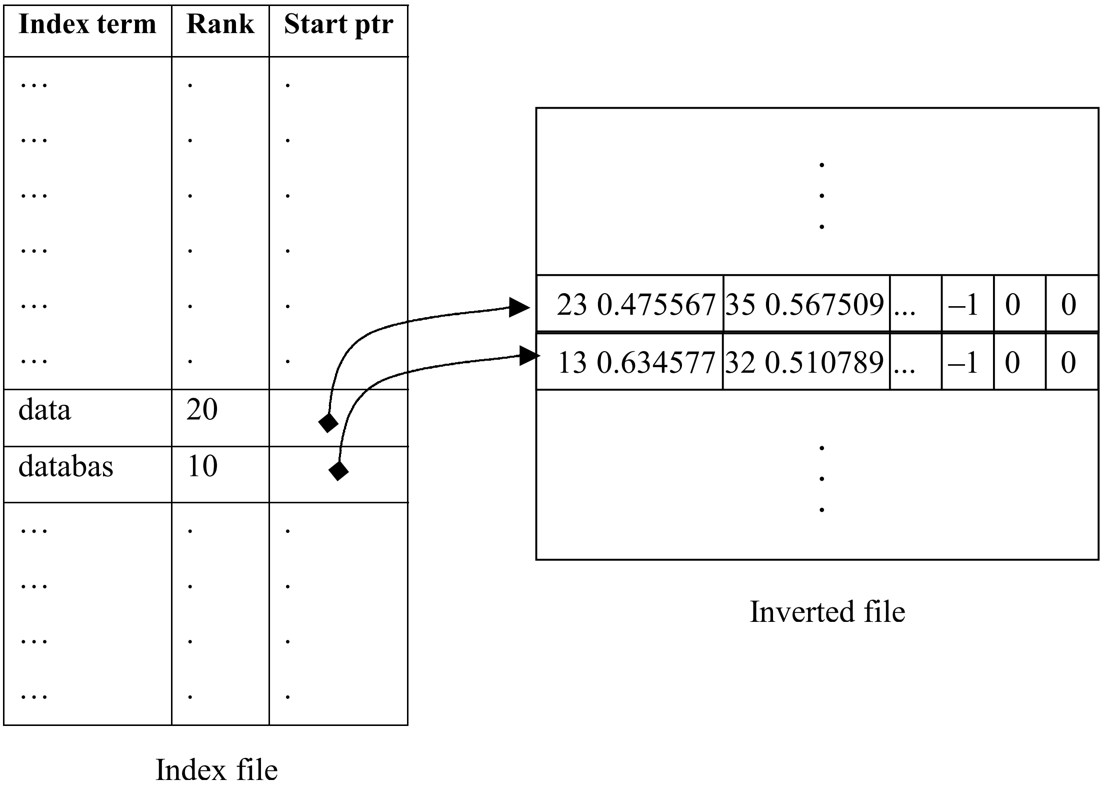
At the end of these two passes, two main files are created on the disk such that an *index file* and an *inverted file*. Index file is implemented as a text file that contains the words in the wordlist with additional information such as rank (i.e., no of topics containing that term) and start pointer (i.e., start offset of the postings list of that term in the inverted file). However, inverted file is created as a binary file so that a postings list for any term can be retrieved in one file access.

In processing initial DBLP dataset, the postings lists are written into the inverted file in a fixed-sized and contiguous manner. The reason for that is that we cannot know whether the word will appear in the coming dynamic sets or not. If the word appears, then a second storage is allocated from the end of inverted file, this time with blank spaces at the end. The size of this second storage is the fraction of the initial size of the first storage. By this way, we have optimized storage organization of the inverted file.

Three additional fields are added at the end of each storage containing the posting lists. The first two fields specify the start pointer and the size of the next storage, respectively whereas the third field specifies how much of the next storage is filled. This storage organization for postings lists in inverted file can be viewed as a linked list implementation of the lists but with at most two or three nodes for the most frequent terms. As a result, this organization allows dynamic update of the inverted file. The example given in the below paragraph explains realization of our index organization.

For example, assume that in the initial dataset there are 40 topics of type “PaperName” and number of topics that contain the term “data” is 20 whereas the term “database” participates in 10 topics. So the ranks of the terms “data” and “database” are 20 and 10, respectively. The integers preceding the float numbers in the postings lists of a term are the Tids of the topics containing that term. The float values are the normalized term weights of that term in the corresponding topic.

After processing initial dataset, index file and inverted file will have the view presented in Figure 3.3. Since there is no dynamic set yet, three additional fields at the end of each postings list will be initially –1, 0 and 0. Notice that the postings lists are written into the inverted file contiguously.

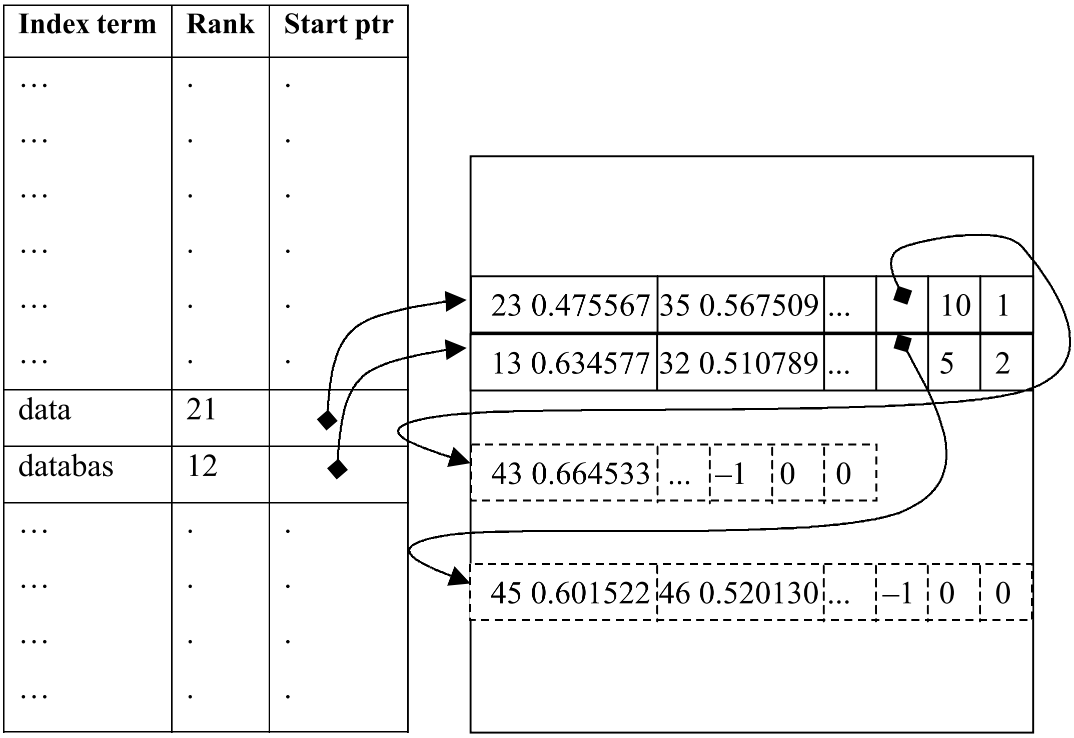


Index term	Rank	Start ptr
…	.	.
…	.	.
…	.	.
…	.	.
…	.	.
…	.	.
data	20	
databas	10	
…	.	.
…	.	.
…	.	.
…	.	.

23 0.475567	35 0.567509	...	–1	0	0
13 0.634577	32 0.510789	...	–1	0	0

Figure 3.3: Realization of index organization in initial dataset

Now assume that several dynamic sets come, and the topics are extracted and inserted into database. The term “data” in dynamic sets passes in one topic where as “database” passes in two topics, so the ranks of these terms are incremented by one and two, respectively. Since there is no blank space in the first storages, second storages are allocated in the inverted file in order to write the new postings lists. The size of the second storage of the term “data” is the half of the first storage (i.e., 10) and there is only one posting in the second storage. In Figure 3.4, the second storages for the postings lists of the term “data” and “database” are shown in dashed style. Now it is time to update three fields (initially –1, 0, 0) at the end of the first storage with the pointer of the second storage, 10 and 1, respectively. Similarly, the size of the second storage of the term “database” is the half of the first one that is 5, and three fields at the end of the first storage are updated with the corresponding values.



Index term	Rank	Start ptr
…	.	.
…	.	.
…	.	.
…	.	.
…	.	.
…	.	.
data	21	
databas	12	
…	.	.
…	.	.
…	.	.
…	.	.

23 0.475567	35 0.567509	...		10	1
13 0.634577	32 0.510789	...		5	2
43 0.664533	...	–1	0	0	
45 0.601522	46 0.520130	...	–1	0	0

Index file

Inverted file

Figure 3.4: Realization of index organization after dynamic update

3.4 A Complete Example

In this section, we present a part of instances of maintained topic map database for DBLP dataset. The content of input XML file containing the entries used in this example is presented in Figure 3.5.

```
<?xml version="1.0"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
...
<inproceedings key="conf/ssd/AbdelmotyWP93">
<ee>db/conf/ssd/AbdelmotyWP93.html</ee>
<author>Alia I. Abdelmoty</author>
<author>M. Howard Williams</author>
<author>Norman W. Paton</author>
<title>Deduction and Deductive Databases for Geographic
Data Handling.</title>
<pages>443-464</pages>
<cdrom>SSD/1993/P443.pdf</cdrom>
<year>1993</year>
<crossref>conf/ssd/93</crossref>
<booktitle>SSD</booktitle>
<url>db/conf/ssd/ssd93.html#AbdelmotyWP93</url>
</inproceedings>
...
<inproceedings key="conf/dexa/AbdelmotyPWFBD94">
<author>Alia I. Abdelmoty</author>
<author>Norman W. Paton</author>
<author>M. Howard Williams</author>
<author>Alvaro A. A. Fernandes</author>
<author>Maria L. Barja</author>
<author>Andrew Dinn</author>
<title>Geographic Data Handling in a Deductive Object-
Oriented Database.</title>
<pages>445-454</pages>
<year>1994</year>
<booktitle>DEXA</booktitle>
<url>db/conf/dexa/dexa94.html#AbdelmotyPWFBD94</url>
</inproceedings>
...
</dblp>
```

Figure 3.5: The XML file containing DBLP bibliographic entries

Given the dblp.dtd in Appendix A, the rules, applied for extracting topics, metalinks and sources, in the first pass over the input dataset can be specified as a *mapping M* in Figure 3.6.

```
M = {
//DTD set employed in the mapping
nms1: "http://dblp.uni-trier.de/dblp.dtd"

// Topic generation rules
t1: <nms1, TName = ValueOf (dblp.article.title) OR ValueOf
                    (dblp.inproceedings.title),
          TType = "PaperName", TAdvice = 1.0>,
t2: <nms1, TName = ValueOf (dblp.article.author) OR ValueOf
                    (dblp.inproceedings.author),
          TType = "AuthorName", TAdvice = 1.0>,
t3: <nms1, TName = ValueOf (dblp.article.journal) OR ValueOf
                    (dblp.inproceedings.booktitle),
          TType = "JourConfOrg", TAdvice = 1.0>,
t4: <nms1, TName = ValueOf (dblp.article.year) OR ValueOf
                    (dblp.inproceedings.year),
          TType = "PublicationDate", TAdvice = 1.0>,
t5: <nms1, TName = ValueOf (dblp.article.journal + dblp.article.year) OR
                    ValueOf (dblp.inproceedings.booktitle +
                    dblp.inproceedings.booktitle),
          TType = "JourConf-and-Year", TAdvice = 1.0>,

// Source generation rules
s1: <nms1, Web-address = ValueOf (dblp.article.url) OR ValueOf
                         (dblp.inproceedings.url),
          Role = "Website", SAdvice = 1.0>

// Metalink generation rules
m1: <AuthorOf: t1 → t2, Madvice = 1.0 >,
m2: <AuthoredBy: t2 → t1, Madvice = 1.0 >,
m3: <JourConfOf: t1 → t5, Madvice = 1.0 >,
m4: <JourConfPapersOf: t5 → t1, Madvice = 1.0 >,
m5: <PublicationDateOf: t1 → t4, Madvice = 1.0 >,
m6: <InPublicationDate: t4 → t1, Madvice = 1.0 >,
m7: <RelatedToPapers: t1 → t1 where Tid(t1) ≠ Tid(t1), Madvice = sim(t1,t1) >,
m8: <PrerequisitePapers: t1 → t1 where Tid(t1) ≠ Tid(t1), Madvice = sim(t1,t1)>
    }
```

Figure 3.6: Mapping *M*

In this mapping, the first line specifies that the element and attribute names provided would be in the namespaces of the DTD of DBLP. Note that the use of word “namespace” is different from the XML-namespaces at it is known in the literature. The value of a topic may not be the value of a single element or attribute, but even the concatenation of both (i.e., topic of type “JourConf-and-Year”). In metalink generation rules, the sim() function used in m7 and m8 is the text similarity of the two topics which is computed as explained in Section 2.3.1.

Tid	TName	TType	TAdvice	TVector
…	…	…	…	…
4	SSD	JourConfOrg	1.0	
…	…	…	…	…
86	1993	PublicationDate	1.0	
87	SSD1993	JourConf-and-Year	1.0	
…	…	…	…	…
152	Alia I. Abdelmoty	AuthorName	1.0	64 0.658733 348 0.658733 6950 0.363512
153	M. Howard Williams	AuthorName	1.0	6803 0.635206 9759 0.321646 17681 0.702180
154	Norman W. Paton	AuthorName	1.0	11759 0.597743 12359 0.717458 17316 0.357711
155	Deduction and Deductive Databases for Geographic Data Handling	PaperName	1.0	1627 0.228837 1628 0.191497 1737 0.551758 2773 0.576976 2924 0.523091
…	…	…	...	…
581	1994	PublicationDate	1.0	
…	…	…	…	…
10046	Alvaro A. A. Fernandes	AuthorName	1.0	27 0.320597 413 0.659966 4619 0.679457
10047	Maria L. Barja	AuthorName	1.0	1146 0.785896 8905 0.326343 10122 0.525231
…	…	…	…	…
10282	Andrew Dinn	AuthorName	1.0	548 0.516583 3720 0.856237
…	…	…	…	…
10553	DEXA	JourConfOrg	1.0	
…	…	…	…	…
11041	DEXA1994	JourConf-and-Year	1.0	
…	…	…	…	…
11114	Geographic Data Handling in a Deductive Object-Oriented Database.	PaperName	1.0	1627 0.224346 1628 0.187739 1737 0.415771 2773 0.565653 2924 0.512826 4729 0.260953 4871 0.300843
…	…	…	…	…

Table 3.3: Instances of topics

In the first pass, applying the rules given in the mapping *M* over DBLP dataset, we have extracted topics, metalinks and sources that are presented in Tables 3.3 to 3.6. We have only shown the extracted instances relating to the entries in Figure 3.5. Note that *RelatedToPapers* and *PrerequisitePapers* metalink instances were created once the second pass through the dataset ended and the vector representation of the "PaperName" topics were obtained.

Mid	**Mtype**	**Antecedentid**	**Consequentid**	**MAdvice**
…	…	…	…	…
403	AuthorOf	155	152	1.0
404	AuthoredBy	152	155	1.0
405	AuthorOf	155	153	1.0
406	AuthoredBy	153	155	1.0
407	AuthorOf	155	154	1.0
408	AuthoredBy	154	155	1.0
409	JourConfOf	155	87	1.0
410	JourConfPapers	87	155	1.0
411	PublicationDateOf	155	86	1.0
412	InPublicationDate	86	155	1.0
…	…	…	…	…
47081	AuthorOf	11114	152	1.0
47082	AuthoredBy	152	11114	1.0
47083	AuthorOf	11114	154	1.0
47084	AuthoredBy	154	11114	1.0
47085	AuthorOf	11114	153	1.0
47086	AuthoredBy	153	11114	1.0
47087	AuthorOf	11114	10046	1.0
47088	AuthoredBy	10046	11114	1.0
47089	AuthorOf	11114	10047	1.0
47090	AuthoredBy	10047	11114	1.0
47091	AuthorOf	11114	10282	1.0
47092	AuthoredBy	10282	11114	1.0
47093	JourConfOf	11114	11041	1.0
47094	JourConfPapers	11041	11114	1.0
47095	PublicationDateOf	11114	581	1.0
47096	InPublicationDate	581	11114	1.0
…	…	…	…	…
171635	RelatedToPapers	155	11114	0.911318
171636	PrerequisitePapers	11114	155	0.911318
…	…	…	…	…
171958	RelatedToPapers	11114	155	0.911318
…	…	…	…	…

Table 3.4: Instances of metalinks

Sid	**Web-address**	**Role**	**SAdvice**
…	…	…	…
46	db/conf/ssd/ssd93.html#AbdelmotyWP93	Website	1.0
…	…	…	…
5736	db/conf/dexa/dexa94.html#AbdelmotyPWFBD94	Website	1.0
…	…	…	…

Table 3.5: Instances of sources

Tsid	**Tid**	**Sid**
…	…	…
46	155	46
…	…	…
5736	11114	5736
…	…	…

Table 3.6: Instances of tsources

Chapter 4

Implementation Details

In the previous section, we have explained the outlines of the presented topic map data model and the employed inverted index scheme. In this section, we describe the details of our implementation. Since the DBLP dataset containing all the bibliographic entries was a huge and single XML file, it was not suitable for the implementation of dynamic update mechanism. Thus, we have rearranged the dataset so that it allows dynamic updates. For this reason, an initial collection has been created containing all the publications up to year 2000 by excluding the ones published in 2000 and 2001. The rest of the entries belonging to year 2000 and 2001 have been contained in dynamic sets. In the following, we first present the details of processing the initial collection and then go on with the processing of dynamic datasets.

4.1 Implementation platform

We have constructed topic map database of both initial collection and dynamic sets on the PC-Windows platform using MS SQL Server Database Management System, C and C++ programming languages. Firstly, we have created topics, metalinks, sources and tsources tables on a database server with MS SQL Server installed on it as DBMS.

In our implementation, two main programs have been used to construct the data model where the first program is for processing the initial collection and the second program is for processing the dynamic sets. The first program, written in C++ language, processing the initial collection, is run on another machine different from the database server, with 1 GB memory and Pentium III processor at 800 MHz. The second program, also written in C++ language, processing the dynamic sets one by one, is run on a client machine with 64 MB memory and Pentium II processor at 500 MHz.

4.2 Initial Collection

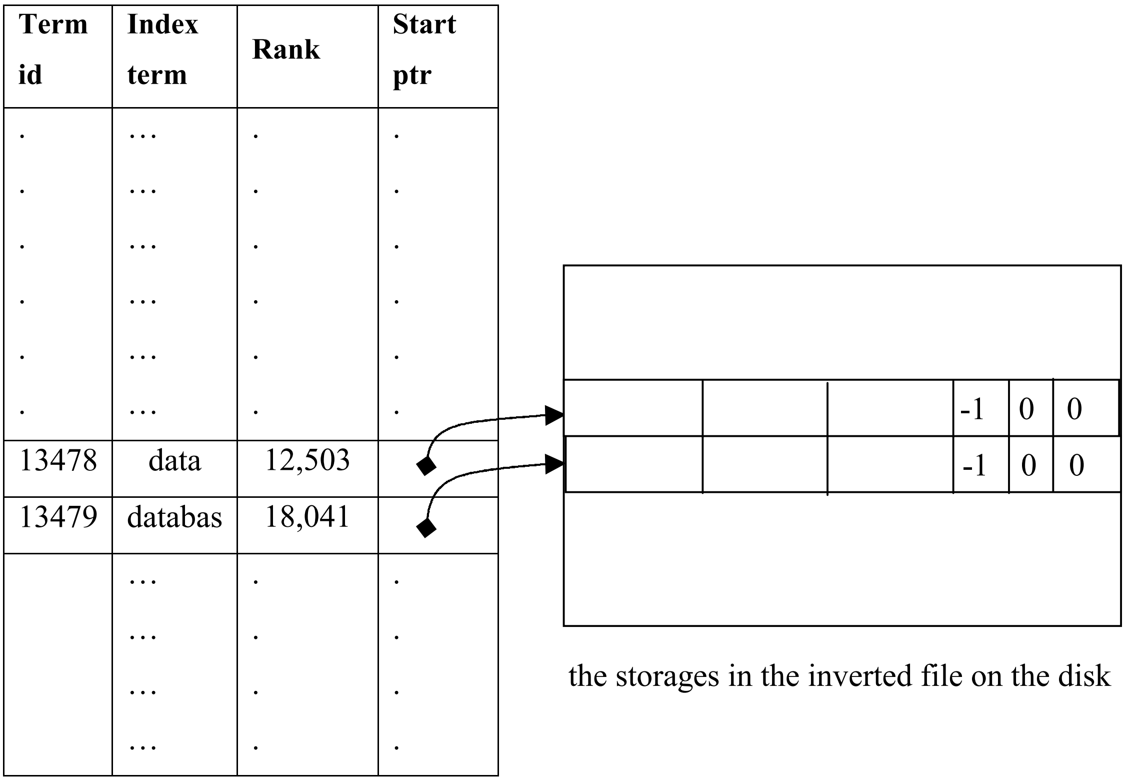
4.2.1 Construction of Inverted File Index

The first program processing the initial collection takes the initial DBLP XML file as an input and processes the bibliographic entries in that file one by one. The characteristics of the initial collection are presented in Table 4.1. Notice that the sum of articles, proceedings and inproceedings is not equal to total number of publications. This difference is due to the presence of other types of publications such as books, incollections and phdthesis, etc.

Number of articles	82,802
Number of proceedings	649
Number of inproceedings	113,276
Total number of publications	198,224
Text size (Mb)	80

Table 4.1: Initial DBLP dataset

As we have mentioned in the previous section, there are two passes over the input dataset in the first program. In the first pass, for each `article` and `inproceedings` element; topic, source and tsource instances are extracted and inserted into the corresponding tables. The metalink instances, except the *RelatedToPapers* and *PrerequisitePapers* metalinks, are also extracted and put into database in the first pass. Extraction of these instances has been done in a straightforward manner by tagging the sub-elements of the entries as defined in *mapping M* in the previous section.



Term id	Index term	Rank	Start ptr
.	…	.	.
.	…	.	.
.	…	.	.
.	…	.	.
.	…	.	.
.	…	.	.
13478	data	12,503	
13479	databas	18,041	
	…	.	.
	…	.	.
	…	.	.
	…	.	.

in-memory wordlist

			-1	0	0
			-1	0	0

the storages in the inverted file on the disk

Figure 4.1: Snapshot of the in-memory wordlist after the first pass.

At the end of this first pass, the topics, sources, tsources, and metalinks tables are filled with the corresponding extracted instances, and the in-memory wordlists of both "PaperName" and "AuthorName" topics are obtained. We have named all of the "PaperName" topics as *title collection* and all of the "AuthorName" topics as *author collection*. A sample snapshot of in-memory wordlist of "PaperName" topics is given in Figure 4.1. Notice that the terms in the wordlist are stemmed and the storages in which the postings will be written into are initially empty and the three fields (i.e., start pointer, size and the fullness of the next storage) at the end of the storages are initialized to –1,0,0 respectively.

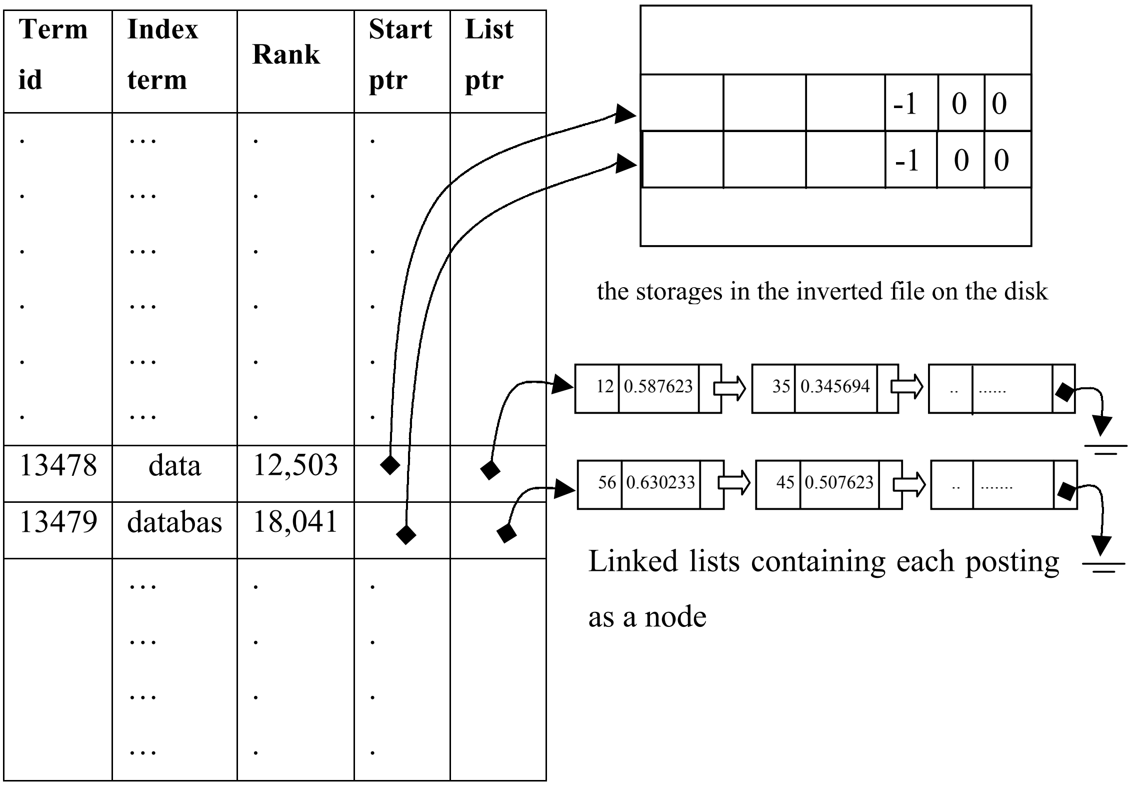
Actually, the second pass is required for obtaining the normalized vector representations of the topics in the *title* and *author collections* where the vector representations of *title collection* are also used for computing the text similarity of any two topics in the *title collection*. The similarity value is compared with the threshold values Tsim1 and Tsim2 so that it can be decided if those two topics have *RelatedToPapers* or *PrerequisitePapers* or both relationships between each other.

In the second pass, each topic in both *title* and *author collection* is processed one by one in order to obtain vector representations. For each topic, the weights of the terms contained in that topic are calculated by employing a widely used term-weighting scheme that is TF/IDF scheme [29]. In this scheme, the weight of a term is computed using the formula v_t = (log(TF$_{v,\ t}$) + 1).log(IDF$_t$), where v_t denotes the vector v element for term t, TF$_{v,\ t}$ (term frequency) is the number of occurrences of term t in the topic represented by v, and IDF$_t$ is the inverse document frequency that is defined as the ratio of the number of all topics in the collection to the number of topics including the term t. Once the weight of all the terms in a topic is computed, then each term weight is divided by

$$\sqrt{\sum_{t=1}^{m}(v_t)^2}$$

where v_t is the weight of term t and m is the number of distinct terms in the topic. Finally, the normalized vector of a topic is typically obtained.

Once the normalized vector of a topic is obtained, each term weight together with the Tid of the topic construct a posting in the postings list of that term. The postings lists are implemented as a linked list organization in the memory. For each term, the start pointer of the linked list is allocated dynamically and stored in the *list ptr* field of in-memory wordlist. In the linked list, each node contains a posting and a pointer to the next posting. Each time a topic is processed and a posting is obtained for a term, a node is appended to the end of the linked list of the corresponding term. In Figure 4.2, the snapshot of in-memory wordlist at a time when some postings added to the linked lists is presented. The pointers in the last nodes of the linked lists show null values.



Term id	Index term	Rank	Start ptr	List ptr
.	…	.	.	
.	…	.	.	
.	…	.	.	
.	…	.	.	
.	…	.	.	
.	…	.	.	
13478	data	12,503	◆	◆
13479	databas	18,041	◆	◆
	…	.	.	
	…	.	.	
	…	.	.	
	…	.	.	

-1 0 0
-1 0 0
the storages in the inverted file on the disk
12 0.587623
35 0.345694
.. ..
56 0.630233
45 0.507623
.. ..
Linked lists containing each posting as a node

in-memory wordlist

Figure 4.2: The view of in-memory wordlist and its pointers

Since the memory will not be sufficient for all postings lists of all terms, the lists are written into the *inverted file* (on the disk) each time when they have reached a predefined threshold (*Tlist*) size. For example, when the amount of nodes in the linked list for the term “data” reaches *Tlist* = 25, the postings in the nodes are written into the corresponding storage in the *inverted file*. After writing the list, the program frees the nodes of the linked list of the term “data” and initializes the *list ptr* field to null value. By this way, we have optimized memory consumption in obtaining the inverted lists.

One advantage of implementing the *inverted file* as a binary file is that the write operation of an inverted list occurs just in one file access. We have assumed that a file access consists of both finding the start address of the disk storage (in the *inverted file*) where the inverted list will be written and writing the inverted list into that storage. That is, firstly, an *fseek()* command moves the file pointer to the start address of the place in the *inverted file* where the list will be written, and then an *fwrite()* command writes the postings in the link list of the term.

At the end of the second pass, we have two files at hand for each collection (i.e., *title* and *author collection*). One of them is the *index file* that consists of the in-memory wordlist with its additional fields such as index term, rank, start ptr. This file is handled as a text file and when a dynamic set comes, it is read into the memory with its last updated values. Because the wordlist is obtained from a collection containing 196,078 "PaperName" topics, the terms participating in the *index file*, that is the terms obtained from the initial huge collection can be thought as the frequent words.

The other file, i*nverted file* contains the postings lists of all the terms in a contiguous and fix-sized manner. It is implemented as a binary file so that the file access operations can be handled in an efficient manner in both writing the list to the file and retrieving the list from the file.

Finally, the normalized vectors of the topics are also stored on the disk as a text file and named as *vector file*. For each "PaperName" topic, four entries (i.e., Tid of the topic, the normalized topic vector, Tid(s) of the author(s) of the publication and the publication date) are contained in this *vector file* to be utilized during the *RelatedToPapers* and *PrerequisitePapers* metalinks extraction.

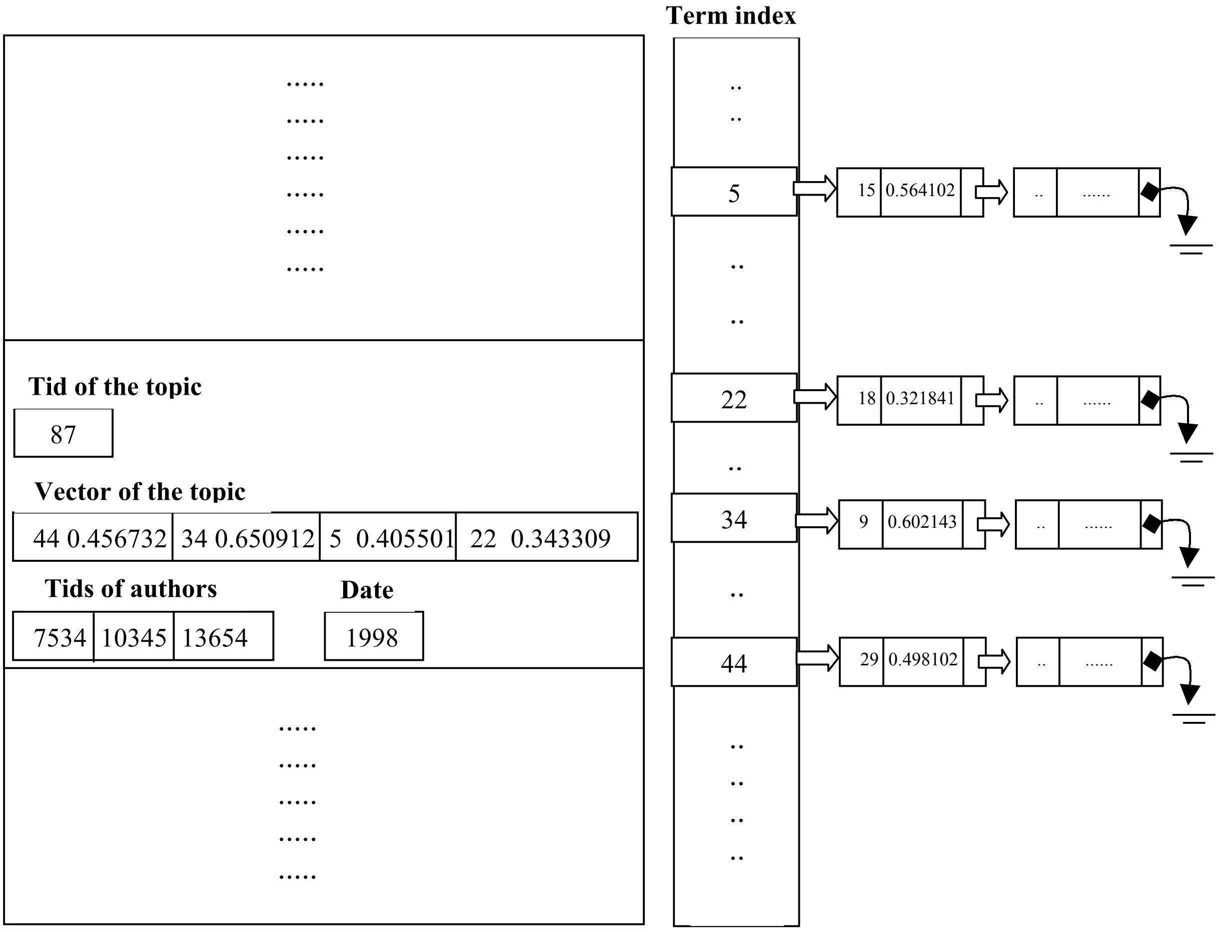
4.2.2 RelatedToPapers and PrerequisitePapers Metalinks

The last step in the first program is to obtain *RelatedToPapers* and *PrerequisitePapers* metalink instances. For this purpose, in the first program, we have used the NLoop$_{\text{Sim-SVT}}$ algorithm presented in [42] with some important extensions. This algorithm takes two relations R and S as inputs and outputs the joined tuples according to a join condition. The pseudo code of the NLoop$_{\text{Sim-SVT}}$ algorithm is provided in Appendix B.

In our implementation, the *title collection* can be thought as a relation, and the vector representations stored in the *vector file*, can be thought as the tuples of this relation. Two copies of the *vector file* are sent as input to the algorithm where the first copy is sent as *collectionR* and the second copy is sent as *collectionS*. So, we can say that a self-join operation is applied on the *title collection*.

Two data structures *BufR* and *BufS*, implemented as array of structs, are used to hold the four entries (i.e., Tid of the topic, normalized topic vector, Tid(s) of the author(s) and publication date) of each topic in memory. The entries in *collectionR* are read into *BufR* starting from the first topic until the buffer is full. Then, the entries in *collectionS* are read into *BufS* again starting from the first topic until the buffer is full. An in-memory inverted list of *BufR* is obtained by tokenizing the vectors with respect to the term indices. Another data structure, *InvertedListR*, is used to hold this inverted list.

Each topic in *BufS* is processed one by one to find the cosine similarity with the topics in *BufR*. For each topic in *BufS*, the topic vector is processed posting by posting. For each term in a posting, the postings list of the term is accessed in the *InvertedListR*. Remember that, the postings list retrieved from *InvertedListR* contains pairs of Tid and weight values. The visualization of the data structures *BufR* and *InvertedListR* are presented in Figure 4.3.



BufR implemented as array of structs

InvertedListR implemented as array of pointers to the linked lists

Figure 4.3: Visualization of *BufR* and *InvertedListR* at any time.

The term weight in a topic vector in *BufR* is multiplied by each weight value at each posting in the linked list of that term in *InvertedListR*. Another data structure, *Candidates*, is implemented as an array that contains *similarity* values computed by summing up the product values. At any time, the number of candidate topics to be similar to a topic in *BufS* can be at most the number of existing topics in *BufR*. The product values in Tidth node of *Candidates* list are summed up incrementally each time a product is computed for that Tid and finally stored as similarity value. The same steps are applied for the next term in the topic vector until all the terms are exploited.

Now, we have the *Candidates* list of a topic with the similarity values at hand. Note that the idea of creating the in-memory inverted list of all the topics in *BufR* is one of the important extensions to the $\text{NLoop}_{\text{Sim-SVT}}$ algorithm. By this way, determining the *Candidates* list of a topic in *BufS* is obtained in a very efficient pruned manner. The similarity values in the *Candidates* list are compared one by one with the threshold values Tsim1 and Tsim2. Finally, the *RelatedToPapers* and *PrerequisitePapers* metalink instances for a topic in *BufS* are obtained according to the rules defined in Section 3.2.3. Note that the other three entries (i.e., Tid, Tid(s) of the author(s) and publication date) of a topic in *BufR* are used during the application of these rules. Then the same steps are applied for all the topics existing in *BufS*. Once all the topics in *BufS* are exploited then next topic vectors in *collectionS* are read into the *BufS* and the same steps are applied to these topics. When the end of the *collectionS* is reached, all the topics will have been compared with the topics in the *BufR* in the sense of text similarity.

Now, it is time to read next topic vectors in *collectionR* into *BufR* and to read the vectors in *collectionS* into *BufS* starting from the first topic vector again. The same steps explained in the previous paragraphs are applied until the topic vectors in *collectionS* are exploited. The program goes on like this until exploiting all the topic vectors in *collectionR*. Note that this part of the first program runs in a nested-loop manner where the *collectionR* is the outer relation and the *collectionS* is the inner relation.

Notice that we have not used the postings lists in the *inverted file* for determining the *Candidates* list of a topic. The reason is that, since a topic will be compared against all of the topics containing the terms in that topic, there will be so many file accesses for computing the similarity of all the topics. Instead, we have implemented an in-memory inverted list held in *InvertedListR* and exploited all the topic vectors in a block nested loop manner. Thus, we have optimized this high file access cost by just processing the topics in a nested loop manner (i.e., reading the topic vectors into *BufR* and *BufS*) and creating an in-memory inverted list of the outer relation (i.e., *InvertedListR*). The results of the first program processing the initial collection are presented in Table 4.2 and Table 4.3.

Entity	Type	Number of instances	Total
Topics	PaperName	196,078	332,786
	AuthorName	130,501	
	JourConfOrg	1,199	
	PublicationDate	47	
	JourConf-and-Year	4,960	
Sources	-	196,078	196,078
Tsources	-	196,078	196,078
Metalinks	AuthoredBy	405,691	4,061,473
	AuthorOf	405,691	
	PublicationDateOf	196,078	
	InPublicationDate	196,078	
	JourConfOf	196,078	
	JourConfPapers	196,078	
	RelatedToPapers	2,303,718	
	PrerequisitePapers	162,062	

Table 4.2: Extracted topics, sources and metalinks from the initial collection.

In this program, the first pass capturing the extraction and the insertion of the topic, source, tsource and metalink instances takes about 120 minutes totally. In fact, extraction process takes at most 2 or 3 minutes. Because of the network latency and the indices employed on the database tables, the insertion of the instances into the corresponding tables takes too much time. In the second pass, creation of *inverted* and *vector file* takes about 10 minutes whereas determining the *RelatedToPapers* and *PrerequisitePapers* metalink instances takes about 60 minutes. Because of the same reasons presented in the first pass, the insertion of these two metalink instances takes too much time that is about 130 minutes in the second pass.

Collection	Topics	Terms	Postings	Inverted file (kb)
Title collection	196,078	39,244	1,166,110	20,060
Author collection	130,501	77,098	315,228	8,540

Table 4.3: Characteristics of the *inverted* and *index files*

At the end of processing initial collection, the characteristics of created *index file* and *inverted file* are presented in Table 4.3. Number of distinct words participating in the title collection (i.e., “PaperName” topics) is 39,224. Some of these terms will be surely more frequent than the others (e.g., rank of the term “system” is 23,450). However, since these terms are obtained by processing an initial huge collection containing 196,078 topics, we can say that all of these terms can be accepted as frequent with respect to the new terms participating in the dynamic sets.

On the other hand, the size of the *inverted file* containing 1,166,110 postings is not so large. The reason for this is that there is no blank space reserved for the postings that will come out in dynamic sets. In the upcoming section, we explain the processing of dynamic sets.

4.3 Dynamic Sets

As we have mentioned in the previous section, dynamic sets captures the bibliographic entries for the publications that are published in 2000 and 2001. In the real life implementation, each time a list of new publications arrives the bibliographic entries for these publications will be prepared in the same format as the initial collection.

4.3.1 Dynamic Update of Inverted File

We have created about 14 dynamic sets by extracting the publications reported in 2000 and 2001 from the original DBLP dataset. The details of dynamic sets are presented in Table 4.4. Although the *author collections* are also processed in our implementation, we have not presented the characteristics and the results of the *author collections* for the sake of simplicity.

Dynamic set	**Text size (kb)**	**Publications**	**Postings**	**Terms**
1	850	2,027	11,454	2,663
2	832	2,107	12,798	2,965
3	817	2,006	12,747	3,262
4	980	2,052	12,582	2,532
5	839	2,032	13,038	2,852
6	816	1,991	12,239	2,636
7	806	1,977	12,583	2,857
8	848	1,995	13,195	2,857
9	834	2,003	13,004	2,583
10	823	1,873	12,380	2,761
11	801	2,022	11,637	2,997
12	894	1,998	12,167	2,779
13	940	1,974	13,071	2,669
14	893	1,956	13,226	2,739
Total	11,973	28,013	176,121	39,152

Table 4.4: Properties of title collections in Dynamic Sets

The number of entries in each dynamic set varies from 1,956 to 2,052. We have created each dynamic set such that no publications published in the same journal/conference are contained in different dynamic sets, and each dynamic set has about 1% of the previously indexed collection. All the publications in dynamic sets are of type article, proceeding or inproceedings. The total number of publications in dynamic sets corresponds to about 15% of the initial collection. However, each time a dynamic set is processed and the dynamic update of the inverted index is constructed, we index about 1% of the previously indexed collection in an incrementally and dynamic manner.

The second program processing dynamic sets differs from the first program in determining the *RelatedToPapers* and *PrerequisitePapers* metalink instances. Instead of using the extended NLoop$_{\text{Sim-SVT}}$ algorithm and creating an in memory inverted list as in the first program, we have used *inverted files* in the second program. The reason for this is that the number of entries in each dynamic set is not so large to cause very high file access costs during the retrieval of postings lists.

In the second program, there are two main passes over the input dataset, too. This time, we have an *index file* and an *inverted file* at hand obtained from the initial collection before starting to process dynamic sets. Now, let us name these files as *old_index* and *old_inverted*, respectively. In the first pass, similar to the first program, all the topic, metalink and source instances are extracted except *RelatedToPapers* and *PrerequisitePapers* metalink instances. Then, these extracted instances are inserted into the corresponding tables on the database server.

Actually, the program firstly reads the *old_index file* into the memory in the first pass. The data structure *old_wordlist* is used to hold the terms read from the *old_index file* in the memory. The structure of the *old_wordlist* is the same as the in-memory wordlist implemented in the first program. Before inserting an extracted “PaperName” topic of the input dynamic set, it is processed term by term. Each term is first searched in the *old_wordlist*. Since the *old_wordlist* has been sorted in alphabetical order after processing initial collection, searching in *old_wordlist* is implemented as binary search. Then, the rank field of the term is incremented by one as each occurrence comes upon. However, if the term is new to the initial collection, in other words if the term is not in the *old_wordlist*, then it is searched in another in-memory wordlist that is *new_wordlist*. The data structure *new_wordlist* is used to hold these new terms and has the same structure with the *old_wordlist*. That is, each time processing a term, the term is first searched in the *old_wordlist*. If the term is not there then it is searched in the *new_wordlist*. If the term is not in the *new_wordlist* either, then the term is added to the *new_wordlist*.

Since the *old_wordlist* is obtained by processing a huge initial collection, the size of the *new_wordlist* will not be so large as well as the ranks of the terms in it. One exception to this assumption may be the ranks of the terms in the *new_wordlist*. That is, the terms involving in new concepts occurring in computer science may have more rank than some of the terms in the *old_wordlist*. The *new_wordlist* is sorted in alphabetical order each time a new word is added to the list. By this way, the search of a word in *new_wordlist* is also implemented as binary search. The cost of sorting the *new_wordlist* is tolerable because the size of it will be very small with respect to the size of the *old_wordlist*.

After having processed the extracted “PaperName” and “AuthorName” topics term by term and updating the *old_wordlist* and the *new_wordlist*, the second pass starts. In the second pass, the normalized vectors of the topics and linked lists containing the postings of the corresponding terms are created. The computations to obtain these vectors and linked lists are the same as in the first program. Another difference from the first program comes out at this point that since the postings lists of all the topics extracted in a dynamic set will not be so large in size to fit into memory, the linked lists containing the postings are held in memory until finishing to process all the “PaperName” topics in that dynamic set.

Since we have two wordlists (i.e., *old_wordlist* and *new_wordlist*) at hand, we have created another binary file for storing the postings of the terms belonging to *new_wordlist*. This file is named as *new_inverted file* and it has the same structure with the *old_inverted file*. Thus, the postings list of a term in the dynamic sets may be written into one of these two inverted files according to term. For this purpose, the postings lists are processed one by one, too. One can think that why a posting of a term is not written at the time it is obtained. This manner will increase the disk access cost very much. Because, there will be as many file accesses as the number of total postings in a dynamic set for writing these postings into the corresponding *inverted_file* (e.g., there are 11454 postings in the first dynamic set). By processing the postings lists one by one, we have decreased this cost to minimum level. Because, this time there will be as many file accesses as the number of distinct words in a dynamic set (e.g., 2663 terms in the first dynamic set). Note that, as in the initial collection, we have assumed that a file access consists of both finding the start address of the disk storage (in the *inverted file*) where the inverted list will be written and writing the inverted list into that storage.

If the postings list is of the term belonging to the *old_wordlist* then a second storage is allocated at the end of the *old_inverted file* such that the size of the second storage is a fraction of the size of the first storage. This time, the second storage will have blank spaces reserved for the postings of that term that can be occur in the upcoming dynamic sets. In our implementation, this fraction is the half of the first storage. After writing the postings list of the term into the *old_inverted file*, the three fields at the end of the first storage are updated with the start pointer of the second storage, the size of the second storage (i.e., half of the first storage) and how much space of the second storage is full (i.e., number of postings written in the dynamic sets up to that time), respectively. Note that three fields at the end of this allocated second storage are also initialized to –1, 0, 0, respectively. If the second storage is full then a next storage is allocated in a similar fashion with the second storage. The postings list is always written starting from the first blank space in the available storage that is not full yet.

If the postings list is of the term belonging to the *new_wordlist* and the term has the first occurrence in the dynamic sets, then a storage is allocated at the end of *new_inverted file* in a fixed sized manner, and the postings list is written into that storage. For other possible occurrences of a term belonging to the *new_wordlist*, the same steps are applied as in the *old_wordlist* and *old_inverted file* (i.e., contiguous and fixed sized first storages, second storages with blank spaces etc.). Note that the only difference is that the postings lists of the terms belonging to *new_wordlist* are written into another file that is *new_inverted file*. The structure of the employed data structures and the created files of both the old and new wordlists are all the same.

4.3.2 RelatedToPapers and PrerequisitePapers Metalinks

After updating the *old_wordlist* and the *new_wordlist*, and writing the postings lists of the terms into the corresponding *inverted file*, now it is time to execute the last step in the second program processing the dynamic sets. In the last step of the second program, the *RelatedToPapers* and *PrerequisitePapers* metalink instances are determined by comparing the cosine similarity of the extracted topics in the dynamic set with the topics inserted into *topics* table.

At any point, the topics table contains all the extracted "PaperName" topics up to that time including the ones extracted in that dynamic set. Once a metalink instance is determined it is inserted into the metalinks tables. One of the main contributions in processing a dynamic set is that in order to find these two metalinks (i.e., *RelatetToPapers* and *PrerequisitePapers*) for a "PaperName" topic extracted in a dynamic set, we have only compare it with the "PaperName" topics in the database that have the terms in common with the topic at hand instead of comparing it with all of the topics in the database.

Each "PaperName" topic vector, obtained from a dynamic set is processed one by one. Each topic vector is also processed posting by posting. If the term t_j in a topic vector v_i belongs to the *old_wordlist* then the postings lists of the term t_j are retrieved from the *old_inverted file* by using the *start ptr* field of the term t_j in the *old_wordlist* data structure. However, if the term t_j is not in the *old_wordlist*, then it means that it is in the *new_wordlist* and the postings lists are retrieved from the *new_inverted file*. Remember that, since each topic vector contains term index and term weight pairs, the wordlist which the term t_j belongs to is determined by just comparing the term index of t_j with the index of the last term in *old_wordlist*. The retrieved postings lists of the term t_j are stored in a memory buffer. After this step, the used data structures and the applied computations to obtain the *Candidates* list of the topic T_i are all the same as in the first program.

The similarity values in the *Candidates* list of the topic T_i are compared one by one with the threshold values Tsim1 and Tsim2. Finally, the *RelatedToPapers* and *PrerequisitePapers* metalink instances for topic T_i are obtained according to the rules defined in Section 3.2.3. For example, if a topic T_j in the *Candidates* list of the topic T_i has a similarity value above Tsim2 (means *RelatedToPapers* metalink already exists between T_i and T_j), then there will be no need to look for a common author between these two topics and the *PrerequisitePapers* metalink instance is directly inserted into the metalinks tables where the Consequentid is the Tid of the topic with an earlier publication date of the two topics. Since the topic T_i is obtained in that dynamic set, the publication date of this topic is already at hand. In order to determine the publication date of the topic T_j, firstly, an SQL select statement is executed which returns the *InPublicationDate* metalink instance where the Consequentid is the Tid of T_j. Secondly, another SQL select statement is executed which returns the TName in topics table where the Tid is the Antecedentid returned in the previous SQL statement.

However, if T_j has a similarity value below Tsim2 but above Tsim1 (i.e., *RelatedToPapers* metalink again exists between T_i and T_j), the author(s) of the T_j is obtained by an SQL select statement returning the *AuthoredBy* metalink instance(s) where the Consequentid is the Tid of T_j and the Antecedentid(s) is the Tid of the authors of the topic T_j. Similar to the publication date, the author(s) of the topic T_i are already at hand. If both topics T_i and T_j have at least one common author, then there exists a *PrerequisitePapers* metalink between these two topics where the Consequentid will be again the Tid of the topic with an earlier publication date.

In order to retrieve the publication date and the author(s) of a candidate topic in an efficient manner, the indexes on the topics and metalinks tables should be managed carefully. We have put a non-clustered index on both the Mtype and Consequentid columns of the metalinks table. We have also put a non-clustered index on the TType column of the topics table.

In our implementation, Tsim1 and Tsim2 are assigned 0.5 and 0.7, respectively. If the two topics T_i and T_j are published in the same year, there will be no *PrerequisitePapers* metalink between them. Finally, for the topic T_i, all of the topics in *Candidates* list of T_i are exploited in the same manner and the extraction of the *RelatedToPapers* and the *PrerequisitePapers* metalink instances are finished for the topic T_i. All of the same steps are applied for all the "PaperName" topics extracted in the dynamic set.

By the way, processing a dynamic set by the second program is ended. The main difference between the first and the second programs in evaluating the *Candidates* list of a topic is that the postings lists of the terms of a topic are retrieved from an in-memory inverted list (i.e., *InvertedListR*) in the former one, whereas the postings lists are retrieved from the disk (i.e., *old_inverted file* or *new_inverted file*) in the latter one. One can think that this disk access cost decreases the efficiency of the second program processing dynamic sets. However, since the number of "PaperName" topics to be processed in a dynamic set is about 2000 (i.e., 1 % of the initial collection) and an in memory inverted list of all the topics must not be constructed as in the first approach, the disk access cost is tolerable. In addition to this, only the postings lists of the terms contained in the extracted topics are retrieved from the corresponding *inverted file*.

Following the description of the implementation details of processing both the initial collection and dynamic sets, in the upcoming section, we discuss the experimental results that are obtained after processing dynamic sets.

Chapter 5

Experimental Results

The details of the second program processing the dynamic sets are given in the previous section. Remember that there are two main passes over the input data set in the second program where the updates on the *old_wordlist* and the *new_wordlist* are performed in the first pass and the vector representation of the topics are obtained in the second pass. Finally, *RelatedToPapers* and *PrerequisitePapers* metalinks are determined in the last step of the second program. This second program is executed 14 times to process all of the dynamic sets. The characteristics of *title collections* obtained from these dynamic sets have been given in Table 4.4 in the previous section.

5.1 Employed Dynamic Update Scheme

The first assumption we made is that sufficient memory is available to support an in-memory vocabulary of the words used in the *title collections*. We have observed that both of the *old_wordlist* containing 39,244 terms and the *new_wordlist* containing at most 3,387 terms have no problem to fit into memory with the additional fields (e.g., index, rank, start_ptr, etc) even if the size of the memory is 64 MB. Another assumption is that the size of the *new_wordlist* containing the new terms is very small with respect to the size of the *old_wordlist*. In Figure 5.1, the size of the *new_wordlist* after each dynamic set is shown.

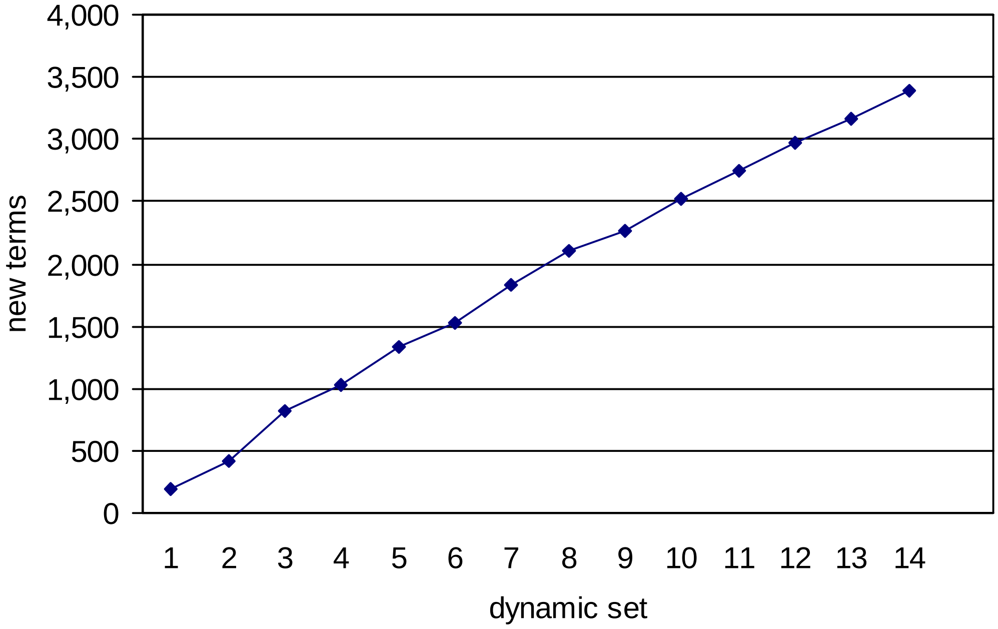


Figure 5.1: Cumulative number of new terms after each dynamic set.

Although the number of new terms added to the *new_wordlist* in a dynamic set depends on the content of that dynamic set, we have observed that it increases linearly not exponentially. The important thing is that the *new_wordlist* has 3,387 terms after all of the dynamic sets are processed. In other words, the size of the *new_wordlist* never exceeds 8.6% of the size of the *old_wordlist* even during the processing of the last dynamic set. So, each time a new term is encountered, instead of adding the term to the *old_wordlist* and sorting this huge list, we have added the new terms to the *new_wordlist* and sorted this relatively very small list. By this way, the cost of sorting is decreased to minimum. In addition to this, since each search process of a term involves in binary search, the cost of searching is decreased to minimum, too.

As we have mentioned in the previous section, once the first occurrence of a term belonging to either *old_wordlist* or *new_wordlist* is encountered, the postings list of the term is written into a storage (i.e., first storage) allocated at the end of the corresponding *inverted file* in a fixed size and contiguous manner. If the term is encountered in the dynamic sets again, the postings lists obtained after this occurrence are written into a second storage allocated with blank spaces. In our implementation, the size of this second storage is the half of the size of the first storage. When the second storage is full, another storage is allocated where the size of this third storage will be half of the size of the second storage. The storage allocation strategy will go on like this.

This storage allocation strategy allows us to use the disk space in an effectively manner. There will be no blank spaces in the storage of a term that is not encountered again throughout the processing of the initial collection and the dynamic sets. However, the blank spaces in the second storages will be filled with the new postings with the following occurrences of the terms. Therefore, the sizes of the both *old_inverted file* and *new_inverted file* will increase much in the first dynamic set and little in the following dynamic sets. In Figure 5.2, we show the increase in the size of the *old_inverted file* after each dynamic set.

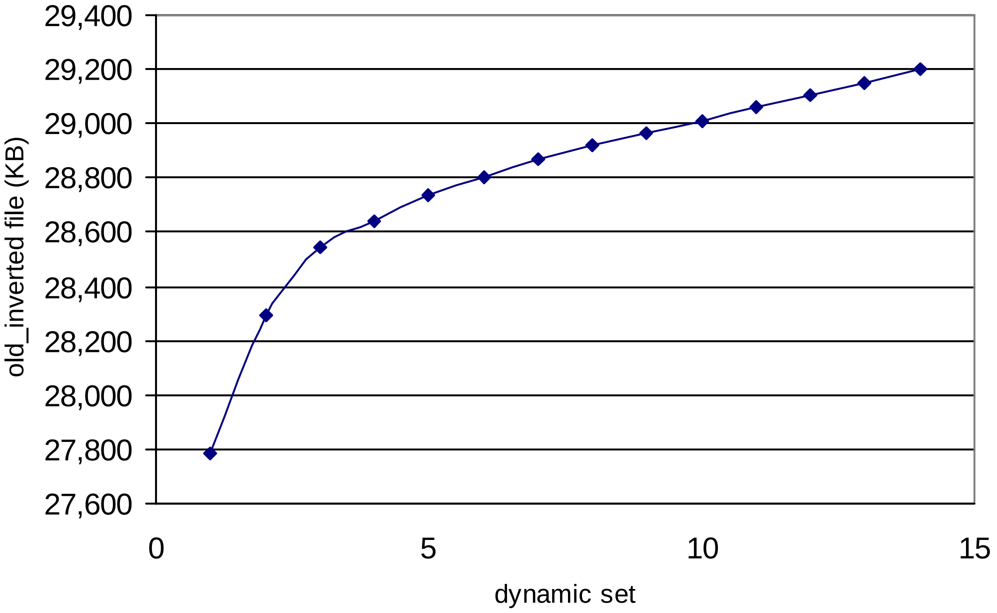


Figure 5.2: The size of the *old_inverted file* after each dynamic set.

Although average number of postings written into the *old_inverted file* is about 12,000 in each dynamic set, the increase in the size of this file is not the same in each dynamic set. The reason is that the second storages for most of the frequent terms are allocated with the blank spaces in the first dynamic set. So, the file size performs a huge jump and increases from 20,060 KB (obtained from initial collection) to 27,784 KB at the end of the first dynamic set. When the new postings of these terms are obtained in the following dynamic sets, these postings are written into these blank spaces in the second storages.

On the other hand, the fraction of the number of terms with more than *two-storages* in the total number of terms should not be increased in a highly manner and most of the frequent terms should not have more than *two-storages*. If it happens, retrieving of the inverted lists of these terms from the corresponding *inverted file* during the determination of the *Candidates* list of a topic requires more than two file accesses (i.e., traversing the storages). We have categorized each term in one of three types: a term with *one-storage*, a term with *two-storages*, a term with *more-storages*. The fraction of the terms belonging to each category is presented in Figure 5.3.

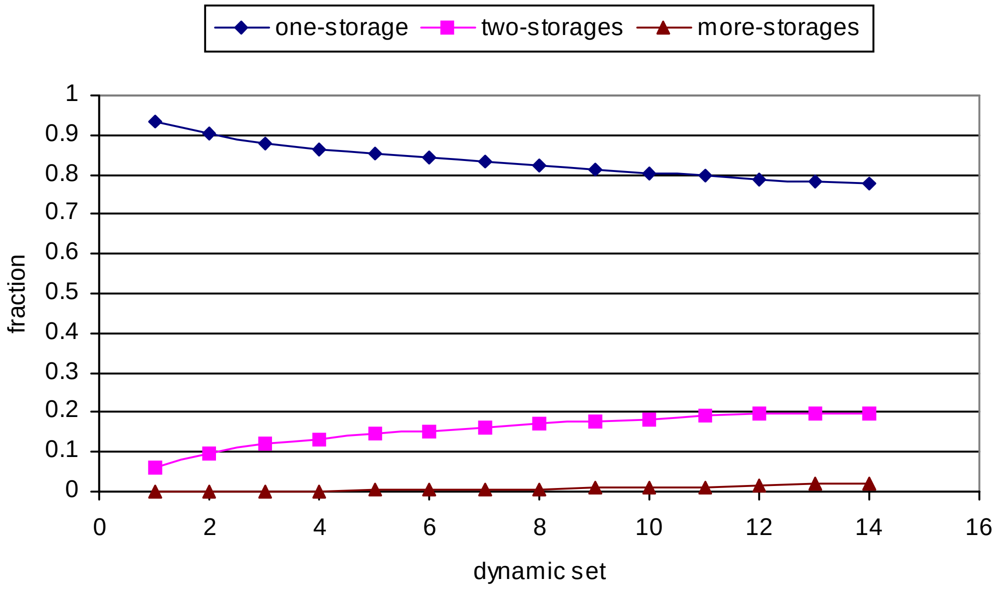


Figure 5.3: The fraction of terms in each category per dynamic set.

We have observed that the terms dropping into *more-storages* category are either the ones which have very small sized first storages (i.e., infrequent terms) or the ones which involve in the new concepts coming out in computer science. For example, the infrequent terms such as ‘b2b’, ‘bluetooth’ and ‘fipa’ have six storages whereas the terms involving in new concepts such as ‘e-commerce’, ‘javacard’ and ‘xml’ have six storages, too. In addition to this, most of the frequent terms such as ‘data’, ‘database’ and ‘system’ have at most *two-storages* as we have assumed. Finally, in 42,631 terms (total number of terms in both of the *old_wordlist* and the *new_wordlist*) obtained at the end of processing dynamic sets, 33,291 words (i.e., 78% of the total number of words) are in *one-storage* category, 8,496 words (i.e., 20% of the total number of words) are in *two-storages* category and 844 words (i.e., 2% of the total number of words) are in *more-storages* category. Note that since the 15% of the initial collection is dynamically updated, these fraction values can be accepted as in the suitable ranges.

Now, let us look at the time spent for dynamic updating of the *inverted files* for each dynamic set. Figure 5.4 shows the cumulative time required to incrementally index the postings obtained in each dynamic set and cumulatively add them to the existing index.

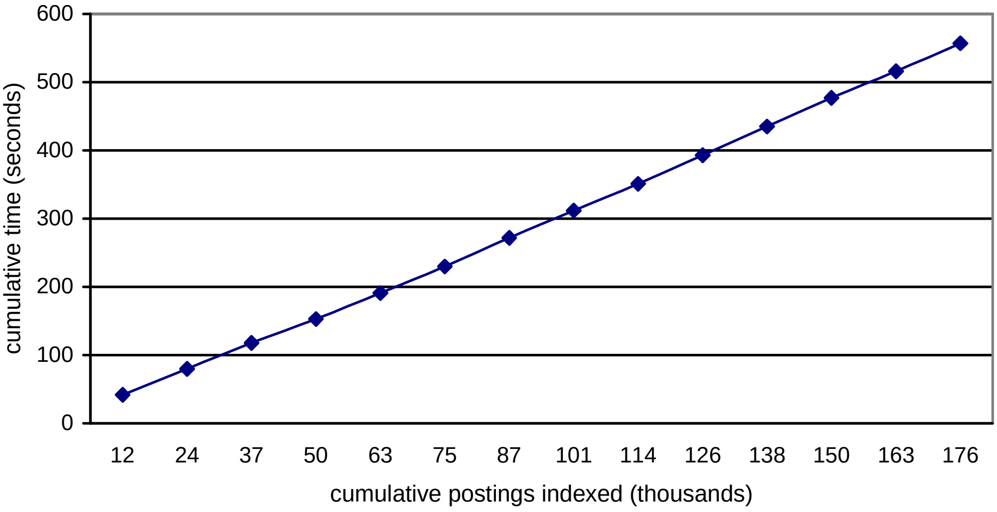


Figure 5.4: The cumulative time needed to build final index.

As it is shown in Figure 5.4, the cumulative time to update the postings increases linearly as the cumulative number of postings increases. Note that incrementally indexing the entire dynamic sets (15% of the initial collection) takes about 557 seconds totally.

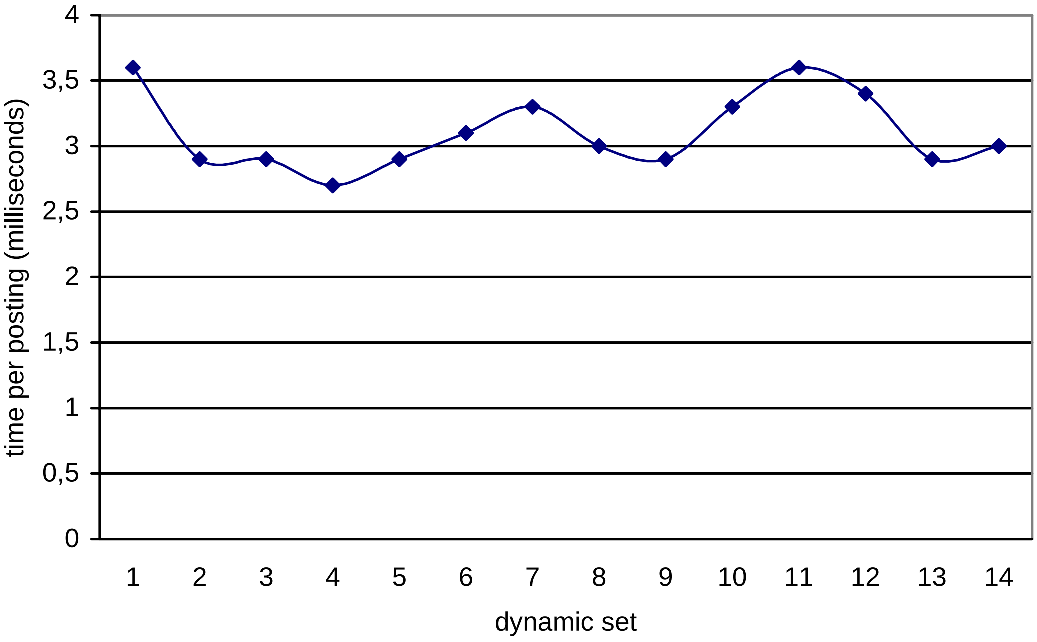


Figure 5.5: Update time per posting in each dynamic set.

We have also computed the average time per posting in each update of dynamic sets as shown in Figure 5.5. We have observed that, although the size of the *inverted files* and the number of terms with *two-storages* increases as shown in Figure 5.2 and Figure 5.3, respectively, the time per posting for each dynamic set update changes in a small range (i.e., from 2.7 milliseconds to 3.6 milliseconds). In other words, it is almost stable throughout all the dynamic sets and there is no outstanding increase as the number of dynamically updated sets increase.

In our implementation, dynamic update of inverted files in a dynamic set process captures; parsing the "PaperName" topics, updating the fields of both *old_wordlist* and *new_wordlist*, computing the weights of the terms, obtaining the vector representations of the topics and the postings lists of the terms, sorting the postings lists in ascending Tid order, and finally finding the first blank spaces in the suitable storages in the *inverted files* by traversing them (first storage, second storage, etc.) and writing the postings lists into that storages.

Of course we do not claim that the employed dynamic update mechanism is a new and very efficient one in the field of inverted file indexing. However, the results we have presented above show that the employed dynamic update scheme in our implementation yields good performance in the sense of disk storage usage and incremental indexing time.

5.2 Updates on the Topic Map Database

The details of the extracted topic, metalink and source instances from the initial collection were given in Section 4.1. In this section, we give the details of the updates applied on the topic map database in each dynamic set process. In each dynamic set, we have processed about 2,000 publications to extract the topic, metalink and source instances.

In the first pass of the second program, once a topic, except the "PaperName" topic, is extracted by tagging the elements in the input XML file (i.e., a dynamic set) it is controlled if it is already in the topics table. Since a "PaperName" topic cannot be published twice, the topics of type "PaperName" are inserted without such a control as it is extracted. Actually, this control provides us to eliminate duplicated topics especially of types "AuthorName", "PublicationDate", "JourConfOrg" and "JourConf-and-Year". If the topic is not in the database then it is inserted into the topics table with its attribute values (e.g., TType, Tid, etc.). Then, the global variable *topic_id* is incremented by one. However, if it is in the database then the Tid of the topic is retrieved to use in the extraction of the metalink instances of type *AuthoredBy*, *AuthorOf*, *InPublicationDate*, etc.

Once all of the sub-elements of a publication are processed, all the topic, metalink (except the *RelatedToPapers* and *PrerequisitePapers* metalinks), source and tsource instances related to that publication will have been inserted into corresponding tables. This process takes about 10-12 minutes for each dynamic set containing about 2,000 publications. As we have mentioned before, the insertion of instances takes the dominant time because of the network latency and the indices put on the database tables. The extraction of the instances from the input file takes very little time with respect to the insertion of the instances.

Dynamic Set	**Topics**			**Metalinks**	**Sources**	**Time (minutes)**
	"PaperName"	"AuthorName"	Total			
1	2,027	702	2,729	17,326	2,027	9
2	2,107	1,028	3,135	17,790	2,107	11
3	2,006	1,105	3,111	16,726	2,006	11
4	2,052	1,172	3,224	16,034	2,052	10
5	2,032	1,453	3,485	18,368	2,032	11
6	1,991	1,196	3,187	17,706	1,991	10
7	1,977	1,420	3,397	18,050	1,977	11
8	1,995	1,786	3,781	19,068	1,995	11
9	2,003	1,807	3,810	18,272	2,003	10
10	1,873	2,001	3,874	17,164	1,873	11
11	2,022	794	2,816	16,722	2,022	9
12	1,998	1,001	2,999	17,968	1,998	12
13	1,974	1,774	3,748	18,480	1,974	10
14	1,956	1,892	3,848	17,794	1,956	9
Total	28,013	19,131	47,144	247,468	28,013	145

Table 5.1: Characteristics of the extracted instances from each dynamic set.

In Table 5.1, total number of instances of all types extracted from each dynamic set excluding *RelatedToPapers* and *PrerequisitePapers* metalink types and the time required to extract and insert these instances are given. At the end of the last dynamic set, 28,013 "PaperName" topics are extracted and inserted into the topics table whereas 196,078 "PaperName" topics were extracted from the initial collection. As we have mentioned before, the size of *title collection* built from the dynamic sets is about 15% of the size of the *title collection* built from initial collection in the sense of number of topics, too.

As stated in the previous section, the last step of the second program processing the dynamic sets consists of both determining the *RelatedToPapers* and *PrerequisitePapers* metalink instances and inserting them into the metalinks table. Table 5.2 shows the details of last step in the second program for each dynamic set in the sense of number of extracted metalinks, number of file accesses to retrieve the postings lists, etc.

Dynamic Set	**Metalinks**			**File accesses**	**Comparison (millions)**	**Time (minutes)**
	RelatedToPapers	*PrerequisitePapers*	Total			
1	67,121	5,689	72,810	22,713	27	46
2	59,521	6,637	66,158	25,368	28	45
3	60,727	8,317	69,044	25,066	27	45
4	35,813	1,547	37,360	24,949	31	32
5	28,815	1,022	29,837	25,774	31	26
6	31,164	1,173	32,337	24,266	30	28
7	26,623	881	27,504	24,849	31	25
8	24,307	1,030	25,337	26,120	34	22
9	28,069	734	28,803	25,832	34	24
10	25,589	998	26,587	24,505	32	23
11	66,305	6,269	72,574	23,034	28	45
12	46,129	4,394	50,523	24,110	30	36
13	24,627	812	25,439	25,975	35	22
14	25,744	662	26,406	26,181	34	24
Total	550,554	40,165	590,719	348,742	432	443

Table 5.2: The details of the extracted metalink instances from each dynamic set.

In Table 5.2, *file accesses* column specifies the number of disk accesses to retrieve the postings lists of the terms in the extracted topics from the *inverted files*. Remember that in order to obtain the *Candidates* list of a topic T_i in the sense of text similarity, all the terms were processed one by one and the postings list of a term t_j was retrieved from the corresponding *inverted file* in a memory buffer. On the other hand, the *comparisons* column specifies the total number of comparisons of the similarity values in the *Candidates* lists of all the extracted topics in a dynamic set with Tsim1 and Tsim2 values.

Thus, the values in the tables verifies our expectation in the sense that the dominant portion of the total time required to extract all the *RelatedToPapers* and *PrerequisitePapers* metalinks in a dynamic set is spent to insert these metalinks not to determine them. For example, although the number of file accesses and comparisons performed to determine the metalinks in dynamic set 13 (i.e., about 26,000 file accesses and 35 million comparisons) are much more than those of dynamic set 1 (i.e., about 22,700 file accesses and 27 million comparisons), the total amount of time spent in dynamic set 13 (i.e., 22 minutes) is less than the half of the time spent in dynamic set 1 (i.e., 46 minutes).

As a result, all of the observations presented in the previous and this section verify that our employed inverted index mechanism and the dynamic update scheme for incrementally updating of this index yields very good performance.

Chapter 6

A Prototype Search Engine

Actually, the basic aim of our work is presenting a more efficient data model than the one currently in use for DBLP organization to be utilized in the Web site management of this organization. The details of the model and the architecture of the search engine that are currently in use by this organization have been explained in detail in Section 2.4 and Section 3.1. We have also presented the details of our data model in Section 3.2. The final step of our work is development of a domain specific Web search engine to be used with the presented topic map data model for DBLP Web site. For this purpose, we have developed a console application that provides a visual query interface for querying the bibliographic information of the publications modelled with the presented topic map data model.

6.1 Outlines of Visual Interface

When a user is connected to DBLP site and wants to search some publications in the field of Computer Science with some specific search criteria, the interface will be presented to the user to allow him or her to access the bibliographic information about the searched publications in a very efficient way. It will be efficient because the presented topic map model will index the bibliographic information semantically and provide alternative navigational pathways through the topic metalinks to the user. In addition to this, the model will provide more meaningful results to users' queries than simple keyword search based Web search engines. Actually there are three main pages in this prototype implementation.

In the *search page* of the interface, there is a *title* box in which the user will enter some terms. These terms are the ones such that at least one of them must be contained in the tiles of the publications. There is a *similarity above* box next to the title box. The user should enter a similarity threshold into this box so that he can search the publications having similarity (with the query) above a threshold. There will be also an *author(s)* box in which the user will enter the names of the author(s) of the searched publications. The terms entered to this box may be the first or the second name or both of them.

Note that a boolean search for the terms in the *title* box and a sub-string search for the terms in the *author* box are employed in the current search scheme of the DBLP Web site. The *year* box will be a list box and allows the user to select a publication date for searched publication(s) in the year format. Two options, such that *in* or *after* the selected date, are also provided.

The user may also want to search some publications reported in a specific journal/conference organization published in a specific date. For this purpose, there is a *journal/conference* box also a list box where the user can select the journal or the conference in which the searched publication(s) are reported. Finally, there is a *top matches* box that will facilitate the limitation on the number of returned results. This box is also a list box.

After entering all the criteria, the search process is started by just clicking on a *search* button in the search page. Then, another page, *result page*, is presented to the user in which the list of publications according to the entered criteria is provided. The user can navigate the list and learn the titles of the publications he searched for. When he selects a publication, related or prerequisite publications to a publication in the returned list can also be obtained by just hitting on the *related publications* or *prerequsite publications* button on the result page, respectively.

The *Related publications* button allows the user to see the list of related publications with the selected publication in the result set whereas the *prerequisite publications* button allows to see the list of publications which are proposed to be read for easily understanding of the selected publication in the returned set. In fact, related and prerequisite publications to a selected publication in the returned set are obtained by using the *RelateToPapers* and *PrerequisitePapers* metalink instances of that publication in the metalinks table. The list of related or prerequisite publications is presented in the third page of the interface.

6.2 Search Process with an Illustration

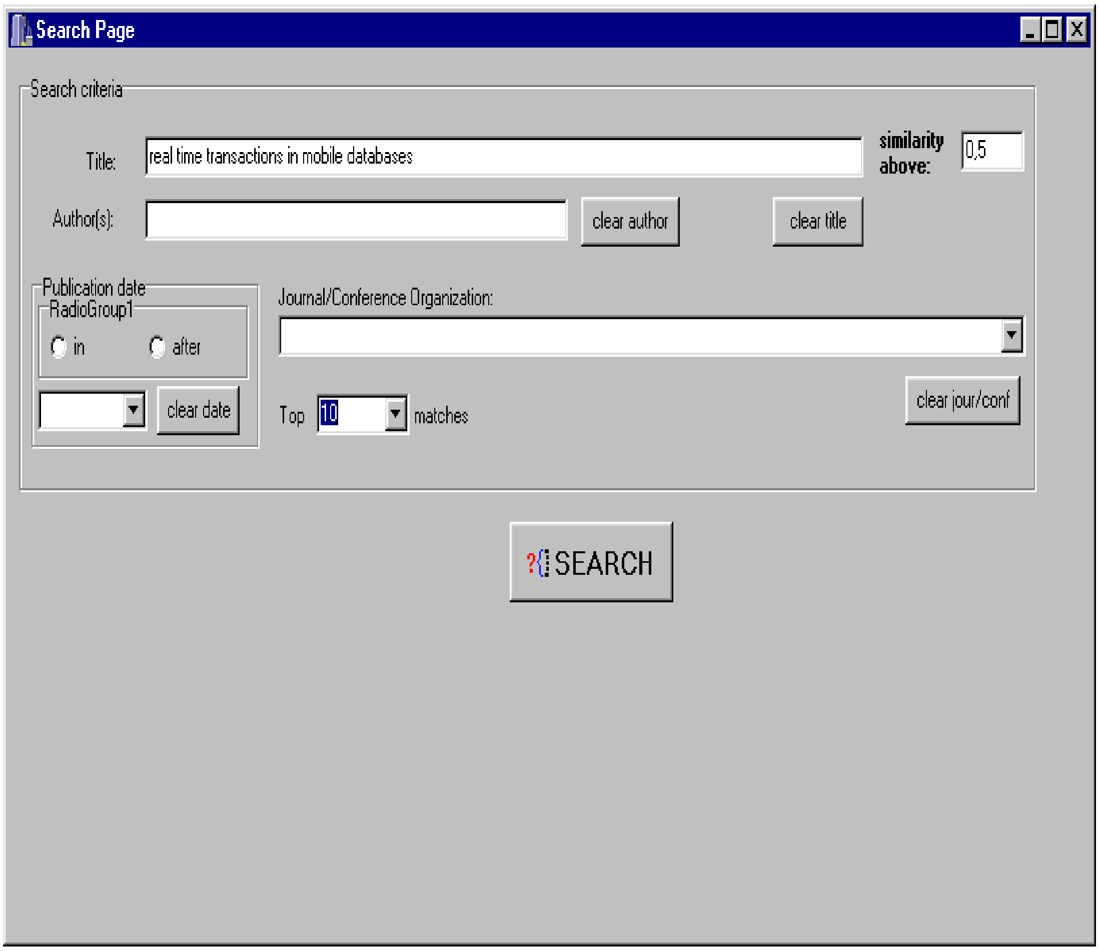
Once the search button on the search page is pressed, a dynamic query string is constructed in which all the criteria are contained as parameters. The metadata-based criteria are straightforward to process. On the other hand, finding the documents having a text similarity with the query above a threshold is almost the same as in extracting a *RelatedToPapers* metalink instance in the processing dynamic sets.

Firstly, the *old_wordlist* and the *new_wordlist* are read into the memory from the *old_index file* and *new_index file*, respectively. Remember that both of these index files are implemented as simple text files. The terms entered into the *title* box (i.e., query terms) are processed one by one. After removing the stop words and stemming the terms with the same tools used in the first and second program, the normalized vector representation of the query is obtained. The calculation of the weights of the terms and the normalization process are all the same as in the first and second program.

Once the normalized vector representation of the query in *title* box is obtained, it is processed as a vector of the "PaperName" topic extracted in a dynamic set. In order to build the *Candidates* list of the query, the same steps are applied as in the last step of the second program that extracts the *RelatedToPapers* and *PrerequisitePapers* metalink instances. The index of a term in vector representation of the query is simply compared with the last index in the *old_wordlist* to decide if the term is in the *old_wordlist* or *new_wordlist*. If the index of the term is bigger that the last index then it means that the term is in the *new_wordlist* and the postings lists of it will be retrieved from the *new_inverted file*. Otherwise, the posting lists of it are retrieved from *old_inverted file*. Once the *Candidates* list is obtained, it is sorted in descending order according to the similarity values stored for each Tid in the list.

Now we have the *Candidates* list of the topics at hand in which the topics are sorted in descending similarity values. This list is limited to the number entered in the *top matches* box and added to the dynamically created query string. Then, an SQL statement is executed which returns the “PaperName” topic(s) with the Tid(s) in the *Candidates* list and matching to the other criteria. All the topics in the returned set are presented to the user in the result page.

In order to illustrate this search process let us look at an example. Assume that a user who is a graduate student in Computer Science performs a research on the issue of ‘Real-time transactions in mobile databases’ for his thesis work. So, he wants to know the publications published on this subject. In this example, we have also assumed that the user has no restriction about the publication date and the author(s) of the publications, and also he does not care about the journal/conference in which the publications are reported. In addition, he wants to know the publications similar to his query above threshold 0.5 and limit the size of the returned set to 10 publications (i.e., top matches). The snapshot of the search page for this example is presented in Figure 6.1.



Title: real time transactions in mobile databases

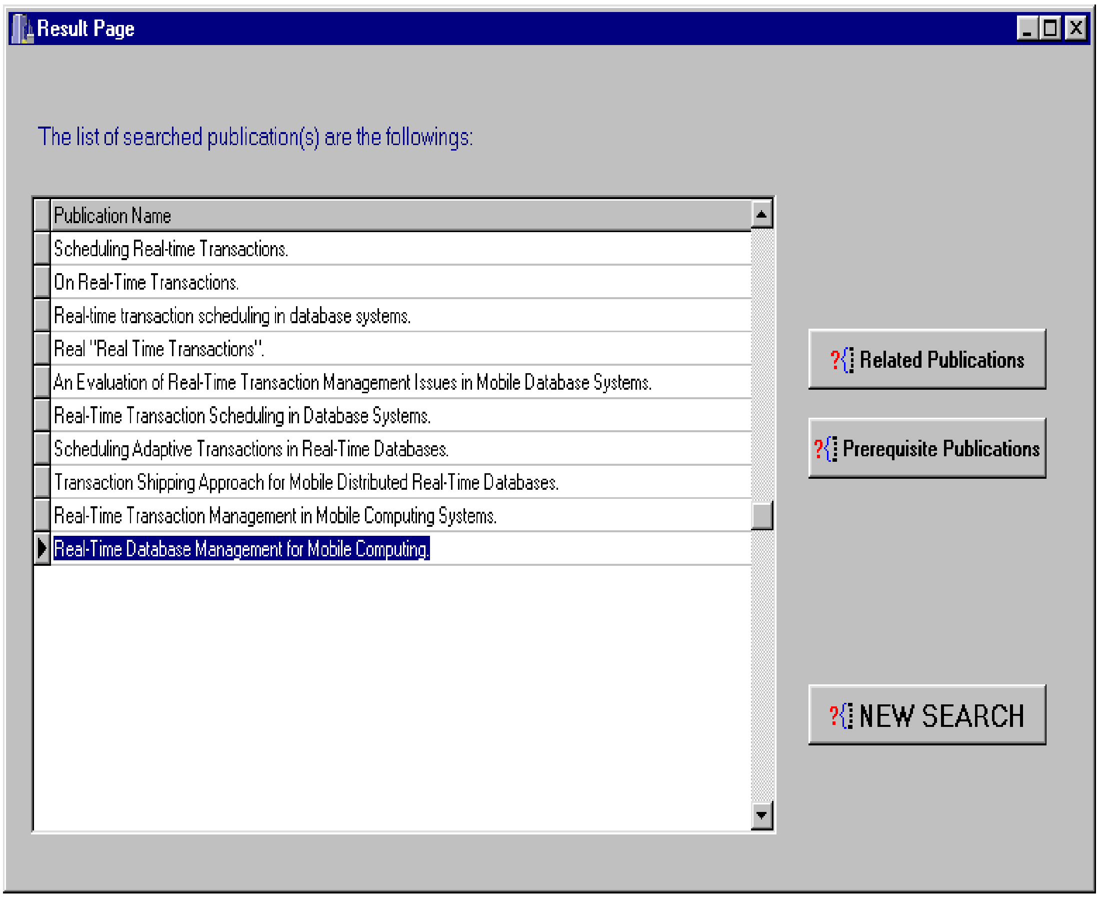
Author(s):

similarity above: 0,5

Top 10 matches

Figure 6.1: The snapshot of the *search page* for the example.

When the user clicks the *search* button, the result page is presented to the user with the publications matching to the criteria. The snapshot of the result page with the returned publications is presented in Figure 6.2. Note that the user can also restrict the returned set by providing other criteria such as author name, journal/conference organization and publication date. For instance, if he wants to see the publications written by some author(s), then the returned set will be pruned. Actually, the top matches box only prunes the set containing the publications that have similarity above 0.5 with the query. However, we have assumed that the user does not care about these criteria for this example.



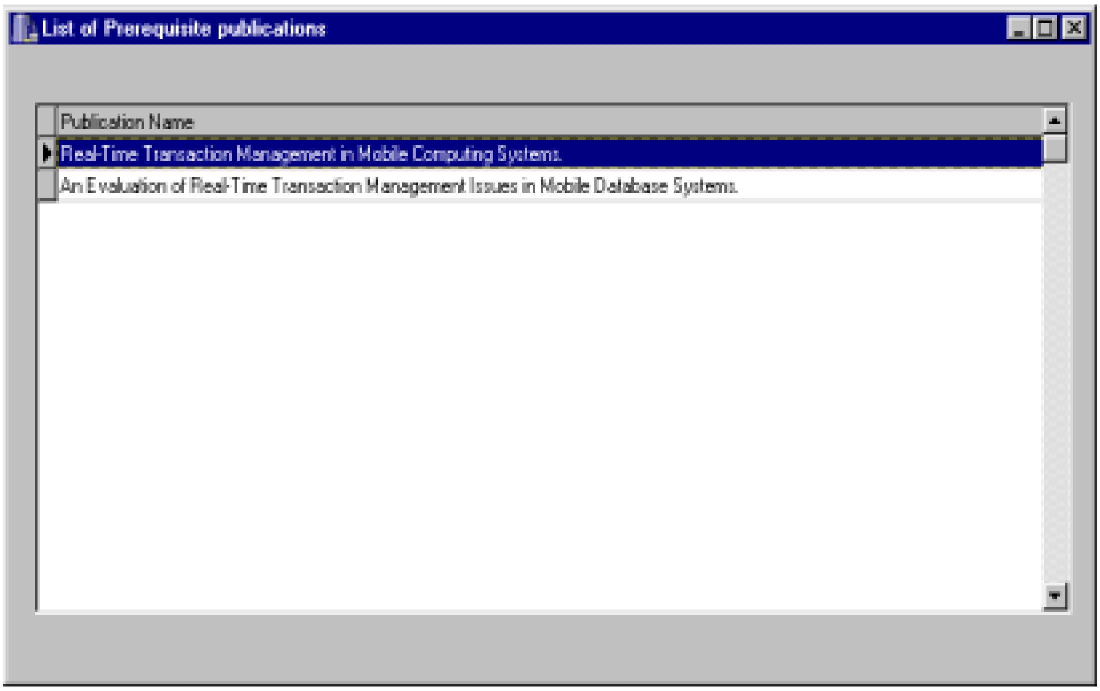
The list of searched publication(s) are the followings:

Publication Name
Scheduling Real-time Transactions.
On Real-Time Transactions.
Real-time transaction scheduling in database systems.
Real "Real Time Transactions".
An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems.
Real-Time Transaction Scheduling in Database Systems.
Scheduling Adaptive Transactions in Real-Time Databases.
Transaction Shipping Approach for Mobile Distributed Real-Time Databases.
Real-Time Transaction Management in Mobile Computing Systems.
Real-Time Database Management for Mobile Computing.

Figure 6.2: The snapshot of the *result page* for the example.

Now, the user gets the list of publications according to entered criteria. Assume that he certainly wants to read the last publication in the returned set (i.e., ‘Real-Time Database Management for Mobile Computing’). So, in order to understand that publication more meaningfully he should see the prerequisite publications to that publication. For this purpose, he selects that publication in the returned set and then clicks the prerequisite publications button.

The list of prerequisite publications is presented in another page to the user. The snapshot of this page is presented in Figure 6.3. According to defined rules for *PrerequsitePapers* metalink instance, there are two publications in the database. As a result, he can learn about the publications that will be helpful in his research area.



Publication Name
Real-Time Transaction Management in Mobile Computing Systems.
An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems.

Figure 6.3: The snapshot of the last page for the example.

Actually, the visual interface presented in this section is a prototype version. Of course it requires some corrections and arrangements to be more user friendly and functional. On the other hand, we have planned to develop a full version which works on the internet and can be used by a browser. The ASP (active server pages) tool may be used to implement this full version. All these issues are planned as future works. However, by presenting a prototype implementation, we have wanted to emphasize the efficiency and the effectiveness of the presented topic map data model in the sense of modeling and querying a specific Web-based information resource (i.e., DBLP).

Chapter 7

Conclusion and Future Work

In this thesis, a topic map data model is presented for a Web-based information resource (i.e., DBLP) to allow efficient querying and searching of the information resource. The presented model is a semantic data model describing the contents of the documents in terms of topics, metalinks and sources. DBLP (Digital Bibliography & Library project) is a WWW server with bibliographic information on major journals and proceedings on computer science.

In our implementation, topic, source and metalink (except *RelatedToPapers* and *PrerequistePapers* metalink types) extractions are easy and straightforward. On the other hand, in order to determine if the two “PaperName” topics are related we compare the text similarity (i.e., cosine quotient) of both of the topics. For this purpose, the weights of the terms are computed as in the vector space model, and the vector representations of the topics are obtained. Also, an inverted file index is implemented to index the titles and authors in the bibliographic entries. This index is used in both finding the text similarity and allowing a keyword-based searching.

Instead of re-indexing the entire collection at each time new bibliographic entries added, a dynamic update scheme is developed. We do not declare that the employed dynamic update scheme is a new and very efficient one. But, according to the results obtained from the experiments presented in the fifth chapter, its performance is good both in time and space requirements.

In real life, for DBLP, adding a dynamic set containing new bibliographic entries to the previous collection per month takes about almost 40 minutes including all topics, sources and metalinks extraction, insertion of them into the database and dynamic update of the inverted file index. Finally, we develop a prototype search engine for querying DBLP in which both keyword-based search and metadata-based search are provided together for an efficient querying of DBLP.

As a result, the presented topic map data model for DBLP adds new semantics and metadata information to bibliographic entries contained in it. The main advantage of our proposal is that we employ metadata in the form of topic maps for querying DBLP bibliography collection to return more meaningful results in an efficient and effective way. Another advantage is that we combine the powers of two basic indexing methods: Inverted file index for keyword-based searching and semantic indexing of the content of the information resource for navigational purposes by using the topic maps standard.

The future work will include the following issues:

- Adding an "IndexTerm" topic type into the topics, *IndexTermOf* and *IndexedBy* metalink types into the metalinks, and author sources into the sources relations, if the keywords in the abstract field of the publications and the Web addresses of the authors' home pages (URLs) can be gathered somehow from the Web,
- Incorporating citation indexing with the existing indexes, if the citation data is available for all the bibliographic entries,

- Developing a full version search engine that employs a more sophisticated GUI working on the Web so that it can be accessible by a browser and allows the use of SQL-TC by naive Web surfers.

Bibliography

[1] XML-data: World Wide Web Consortium (W3C) Note, January 1998. Available at http://www.w3.org/TR/1998/NOTE-XML-data-0105/.

[2] J. Widom. Data Management for XML: Research Directions. In *IEEE Data Engineering Bulletin,* vol. 22, no 3, pp. 44-52, 1999.

[3] G. Mecca, P. Merialdo, P. Atzeni. Araneous in the Era of XML. In *IEEE Data Engineering Bulletin,* vol. 22, no 3, pp. 19-26, 1999.

[4] G. Özsoyoğlu, M. Anderson, and Z. M. Özsoyoğlu. Web Search with Metadata Links and Multimedia Presentations. In *Sixth Int. Workshop on Multimedia Information Systems,* Chicago, October 2000. Available at http://art.cwru.edu/TOpapers/MIS2000.pdf.

[5] M. Dillon. Metadata for Web Resources: How Metadata Works on the Web. In *Library of Congress*, January 23, 2001. Available at http://www.loc.gov/catdir/bibcontrol/dillon_paper.html.

[6] D. Maier and L. Delcambre. Superimposed Information for the Internet. In *(Informal) Proceedings of WebDB,* pp 1-9, 1999.

[7] K. G. Jeffery. Metadata: An Overview and Some Issues. In *ERCIM News Online Edition*, News No.35. Available at http://www.ercim.org/publication/Ercim_News/enw35/intro.html

[8] E. Duval, W. Hodgins, S. Sutton, S. L. Weibel. Metadata Principles and Practicalities. In *D-Lib Magazine*, vol. 8, No. 4, April 2002. Available at http://www.dlib.org/dlib/april02/weibel/04weibel.html.

[9] T. Berners-Lee, Semantic Web Road Map. W3C draft, available at http:// www.w3.org.designissues/semantic.html, January 2000.

[10] E. Miller. Semantic Web Activity Statement. Avaliable at http://www.w3.org/2001/sw/Activity.

[11] M. S. Lancher and S. Decker. On the Integration of Topic Maps and RDF Data. In *1st International Semantic Web Working Symposium (SWWS `01)*, Stanford University, CA, pp. 467-478, July 29-Aug 1, 2001.

[12] T. Berners-Lee. What the Semantic Web Can Represent. In *W3C Note,* pp. 1-6, September 1998. Available at http://www.w3.org/DesignIssues/RDFnot.html.

[13] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. Available at http://www.w3.org/TR/REC-rdf-syntax.

[14] D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification, W3C proposed recommendation, October 2002. Avaliable at http://www.w3.org/TR/PR-rdf-schema.

[15] S. R. Newcomb and M. Biezunski. Topic Maps Go XML. In *Proceedings of the XML Europe 2000*, No. 2, Paris, France. Available at http://www.gca.org/papers/xmleurope2000/pdf/s11-02.pdf

[16] H. H. Rath and S. Pepper. Topic Maps at Work. *Chapter 1 in: XML Handbook.* Prentice Hall, 2nd edition, 1999.

[17] S. Pepper. The TAO of Topic Maps: Finding the Way in the Age of Infoglut. In *Proceedings of the XML Europe 2000*, No. 1, Paris, France. Available at http://www.gca.org/papers/xmleurope2000/pdf/s11-01.pdf

[18] İ. S. Altıngövde. Topic-Centric Querying of Web Resources. *Master of Science Thesis, Bilkent University*, *Dept. of Computer Eng*., September 2001.

[19] Hypermedia/time-based Structuring Language (HyTime) Users' Group Home Page, http://www.hytime.org.

[20] XML Pointer Language (Xpointer) version 1.0, http://www.w3.org/tr/xptr.

[21] XML Linking Language (XLink) version 1.0, http://www.w3.org.tr/xlink.

[22] G. Moore. Topic Map Technology – the Sate of Art. In *Proceedings of the XML Europe 2000,* No. 4 Paris, France. Available at http://www.gca.org/papers/xmleurope2000/pdf/s22-04.pdf

[23] İ. S. Altıngövde, S. A. Özel, Ö. Ulusoy, G. Özsoyoğlu and Z. M. Özsoyoğlu. SQL-TC: A Topic-Centric Query Language for Web-Based Information Resources. In *12th International Conference on Database and Expert Systems Applications (DEXA) 2001*, Munich, Germany, 2001.

[24] S. Pepper. Topic Maps and RDF: A First Cut. Available at http://www.ontopia.net/topicmaps/learn_more.html.

[25] W. Meng, C. Yu and K. L. Liu. Building Efficient and Effective Metasearch Engines. In *Proceedings of ACM Computing Surveys*, Vol. 34, No. 1, March 2002.

[26] M. Kobayashi and K. Takeda. Information Retrieval on the Web. In *Proceedings of ACM Computing Surveys*, Vol. 32, No. 2, June 2000.

[27] Dublin Core home page, http://www.dublincore.org

[28] C. Lagoze. The Warwick Framework. In *D-Lib Magazine*, No. 7, July/August 1996. Available at http://www.dlib.org/dlib/july96/lagoze/07lagoze.html.

[29] L. Gravano, H. Garcia-Molina and A. Tomasic. GIOSS: Text-Source Discovery over the Internet. In *Proceedings of ACM Transactions on Database Systems*, Vol. 24, Issue 2, pp. 229-264, 1999.

[30] D. Cutting and J. Pedersen. Optimizations for Dynamic Inverted Index Maintenance. In *Proceedings of the 13th International Conference on research and Development in Information Retrieval*, pp.405-411, September 1990.

[31] J. Zobel, A. Moffat and R. Sack-Davis. An Efficient Indexing Technique for Full-Text Database Systems. In *Proceedings of 18'th Conference on Very Large Databases,* pp. 352-362, Canada 1992.

[32] A. Tomasic, H. Garcia-Molina and K. Shoens. Incremental Updates of Inverted Lists for Text Document Retrieval. In *Proceedings of the ACM SIGMOD Inter. Conf. On Management of Data,* pp 289-300, Minneapolis, MN, May 1994.

[33] E. W. Brown, J. P. Callan and W. B. Croft. Fast Incremental Indexing for Full-Text Information Retrieval. In *Proceedings of the 20th VLDB Conference*, pp 192-202, Santiago, Chile, 1994.

[34] E. Garfield. Citation Indexing: Its Theory and Application in Science, Technology, and Humanities. Wiley, New York, 1979.

[35] C. L. Giles, K. D. Bolacker and S. Lawrence. CiteSeer: An Automatic Citation Indexing System. In *Digital Libraries 98 – The Third ACM Conference on Digital Libraries*, pp. 89-98, June 1998.

[36] Institute for Scientific Information (ISI) home page http://www.isinet.com.

[37] M. Ley. DB&LP: A WWW Bibliography on Databases and Logic Programming. Available at http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/l/Ley:Michael.html.

[38] Digital Libraries. *Special Issue* of *Communications of the ACM 38*, No. 4, April 1995.

[39] DBLP home page, http://dblp.uni-trier.de/.

[40] DBLP FAQ: Which software is behind DBLP? Available at http:// dblp. unitrier.de/db/about/faq.html.

[41] C. L. Viles and J. C. F. Pedersen. On the Update of Term Weights in Dynamic Information Retrieval Systems. In *Proceedings of the CIKM*, Baltimore, MD, November 1995.

[42] İ. S. Altıngövde, S. A. Özel, Ö. Ulusoy, G. Özsoyoğlu, Z. M. Özsoyoğlu and Al-Hamdani. Sideway Algebra for Object-Oriented Database Applications. In *Proceedings of International Conference on Very Large Databases (VLDB'02)*, August 2002.

[43] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier and D. Suciu. Querying XML Data. In *IEEE Data Engineering Bulletin,* vol. 22, no 3, pp. 10-18, 1999.

[44] W. Cohen, A. McCallum and D. Quass. Learning to Understand the Web. In *IEEE Data Engineering Bulletin,* vol. 23, no 3, pp. 17-24, 2000.

[45] M. Agosti, F. Crivellari and M. Melucci. The Effectiveness of Metadata and other Content Descriptive Data in Web Information Retrieval. In *Proceedings of the Third IEEE Meta-Data Conference (META-DATA '99)*, Bethesda MD, April 1999**.**

[46] M. Biezunski. Understanding Topic Maps. Available at http://www.infoloom.com/ whitepaper.htm.

Appendix A

DTD for DBLP Data

<!--

DBLP XML Records are available from ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz

To use this simple DTD you have to concat the records and to enclose them into

```
<?xml version="1.0"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>

   ... records ...

</dblp>
```

Copyright 2001 by Michael Ley (ley@uni-trier.de)

Copying of the "DBLP" bibliography collection is permitted provided that the copies are not made or distributed for direct commercial advantage and credit for the source is given. To copy or republish otherwise requires specific permission. ACM, IEEE Computer Society, and The VLDB Endowment have the permission to publish DBLP on CDROM/DVD and on their Web servers.

-->

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                        phdthesis|mastersthesis|www)*>
```

```
<!ENTITY%field
"author|editor|title|booktitle|pages|year|address|journal|volume|number|month|url|
ee|cdrom|cite|publisher|note|crossref|isbn|series|school|chapter">

<!ELEMENT article      (%field;)*>
<!ATTLIST article
                key CDATA #REQUIRED
                reviewid CDATA #IMPLIED
                rating CDATA #IMPLIED
>

<!ELEMENT inproceedings (%field;)*>
<!ATTLIST inproceedings key CDATA #REQUIRED>

<!ELEMENT proceedings   (%field;)*>
<!ATTLIST proceedings   key CDATA #REQUIRED>

<!ELEMENT book          (%field;)*>
<!ATTLIST book          key CDATA #REQUIRED>

<!ELEMENT incollection  (%field;)*>
<!ATTLIST incollection  key CDATA #REQUIRED>

<!ELEMENT phdthesis     (%field;)*>
<!ATTLIST phdthesis     key CDATA #REQUIRED>

<!ELEMENT mastersthesis (%field;)*>
<!ATTLIST mastersthesis key CDATA #REQUIRED>

<!ELEMENT www           (%field;)*>
<!ATTLIST www           key CDATA #REQUIRED>

<!ELEMENT author    (#PCDATA)>
<!ELEMENT editor    (#PCDATA)>
<!ELEMENT address   (#PCDATA)>

<!ENTITY % titlecontents "#PCDATA|sub|sup|i|tt|ref">
<!ELEMENT title     (%titlecontents;)*>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT pages     (#PCDATA)>
<!ELEMENT year      (#PCDATA)>
<!ELEMENT journal   (#PCDATA)>
<!ELEMENT volume    (#PCDATA)>
<!ELEMENT number    (#PCDATA)>
<!ELEMENT month     (#PCDATA)>
<!ELEMENT url       (#PCDATA)>
```

```
<!ELEMENT ee        (#PCDATA)>
<!ELEMENT cdrom     (#PCDATA)>
<!ELEMENT cite      (#PCDATA)>
<!ELEMENT school    (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ATTLIST publisher
              href CDATA #IMPLIED
>
<!ELEMENT note      (#PCDATA)>
<!ATTLIST cite
              label CDATA #IMPLIED
>
<!ELEMENT crossref  (#PCDATA)>
<!ELEMENT isbn      (#PCDATA)>
<!ELEMENT chapter   (#PCDATA)>
<!ELEMENT series    (#PCDATA)>
<!ATTLIST series
              href CDATA #IMPLIED
>

<!ELEMENT ref (#PCDATA)>
<!ATTLIST ref href CDATA #REQUIRED>
<!ELEMENT sup (%titlecontents;)*>
<!ELEMENT sub (%titlecontents;)*>
<!ELEMENT i   (%titlecontents;)*>
<!ELEMENT tt  (%titlecontents;)*>

<!ENTITY quot  "&#034;">
<!ENTITY reg   "&#174;">
<!ENTITY micro "&#181;">
<!ENTITY times "&#215;">

<!-- (C) International Organization for Standardization 1986 Permission to copy
in any form is granted for use with conforming SGML systems and applications
as defined in ISO 8879, provided this notice is included in all copies.
-->
<!-- Character entity set. Typical invocation: <!ENTITY % HTMLlat1 PUBLIC
"ISO 8879-1986//ENTITIES Added Latin 1//EN//XML">
-->
<!-- This version of the entity set can be used with any SGML document which
uses ISO 8859-1 or ISO 10646 as its document character set. This includes XML
documents and ISO HTML documents.
-->

<!ENTITY Agrave  "&#192;" ><!-- capital A, grave accent -->
<!ENTITY Aacute  "&#193;" ><!-- capital A, acute accent -->
```

```
<!ENTITY Acirc   "&#194;" ><!-- capital A, circumflex accent -->
<!ENTITY Atilde  "&#195;" ><!-- capital A, tilde -->
<!ENTITY Auml    "&#196;" ><!-- capital A, dieresis or umlaut mark -->
<!ENTITY Aring   "&#197;" ><!-- capital A, ring -->
<!ENTITY AElig   "&#198;" ><!-- capital AE diphthong (ligature) -->
<!ENTITY Ccedil  "&#199;" ><!-- capital C, cedilla -->
<!ENTITY Egrave  "&#200;" ><!-- capital E, grave accent -->
<!ENTITY Eacute  "&#201;" ><!-- capital E, acute accent -->
<!ENTITY Ecirc   "&#202;" ><!-- capital E, circumflex accent -->
<!ENTITY Euml    "&#203;" ><!-- capital E, dieresis or umlaut mark -->
<!ENTITY Igrave  "&#204;" ><!-- capital I, grave accent -->
<!ENTITY Iacute  "&#205;" ><!-- capital I, acute accent -->
<!ENTITY Icirc   "&#206;" ><!-- capital I, circumflex accent -->
<!ENTITY Iuml    "&#207;" ><!-- capital I, dieresis or umlaut mark -->
<!ENTITY ETH     "&#208;" ><!-- capital Eth, Icelandic -->
<!ENTITY Ntilde  "&#209;" ><!-- capital N, tilde -->
<!ENTITY Ograve  "&#210;" ><!-- capital O, grave accent -->
<!ENTITY Oacute  "&#211;" ><!-- capital O, acute accent -->
<!ENTITY Ocirc   "&#212;" ><!-- capital O, circumflex accent -->
<!ENTITY Otilde  "&#213;" ><!-- capital O, tilde -->
<!ENTITY Ouml    "&#214;" ><!-- capital O, dieresis or umlaut mark -->
<!ENTITY Oslash  "&#216;" ><!-- capital O, slash -->
<!ENTITY Ugrave  "&#217;" ><!-- capital U, grave accent -->
<!ENTITY Uacute  "&#218;" ><!-- capital U, acute accent -->
<!ENTITY Ucirc   "&#219;" ><!-- capital U, circumflex accent -->
<!ENTITY Uuml    "&#220;" ><!-- capital U, dieresis or umlaut mark -->
<!ENTITY Yacute  "&#221;" ><!-- capital Y, acute accent -->
<!ENTITY THORN   "&#222;" ><!-- capital THORN, Icelandic -->
<!ENTITY szlig   "&#223;" ><!-- small sharp s, German (sz ligature) -->
<!ENTITY agrave  "&#224;" ><!-- small a, grave accent -->
<!ENTITY aacute  "&#225;" ><!-- small a, acute accent -->
<!ENTITY acirc   "&#226;" ><!-- small a, circumflex accent -->
<!ENTITY atilde  "&#227;" ><!-- small a, tilde -->
<!ENTITY auml    "&#228;" ><!-- small a, dieresis or umlaut mark -->
<!ENTITY aring   "&#229;" ><!-- small a, ring -->
<!ENTITY aelig   "&#230;" ><!-- small ae diphthong (ligature) -->
<!ENTITY ccedil  "&#231;" ><!-- small c, cedilla -->
<!ENTITY egrave  "&#232;" ><!-- small e, grave accent -->
<!ENTITY eacute  "&#233;" ><!-- small e, acute accent -->
<!ENTITY ecirc   "&#234;" ><!-- small e, circumflex accent -->
<!ENTITY euml    "&#235;" ><!-- small e, dieresis or umlaut mark -->
<!ENTITY igrave  "&#236;" ><!-- small i, grave accent -->
<!ENTITY iacute  "&#237;" ><!-- small i, acute accent -->
<!ENTITY icirc   "&#238;" ><!-- small i, circumflex accent -->
<!ENTITY iuml    "&#239;" ><!-- small i, dieresis or umlaut mark -->
<!ENTITY eth     "&#240;" ><!-- small eth, Icelandic -->
```

```
<!ENTITY ntilde  "&#241;" ><!-- small n, tilde -->
<!ENTITY ograve  "&#242;" ><!-- small o, grave accent -->
<!ENTITY oacute  "&#243;" ><!-- small o, acute accent -->
<!ENTITY ocirc   "&#244;" ><!-- small o, circumflex accent -->
<!ENTITY otilde  "&#245;" ><!-- small o, tilde -->
<!ENTITY ouml    "&#246;" ><!-- small o, dieresis or umlaut mark -->
<!ENTITY oslash  "&#248;" ><!-- small o, slash -->
<!ENTITY ugrave  "&#249;" ><!-- small u, grave accent -->
<!ENTITY uacute  "&#250;" ><!-- small u, acute accent -->
<!ENTITY ucirc   "&#251;" ><!-- small u, circumflex accent -->
<!ENTITY uuml    "&#252;" ><!-- small u, dieresis or umlaut mark -->
<!ENTITY yacute  "&#253;" ><!-- small y, acute accent -->
<!ENTITY thorn   "&#254;" ><!-- small thorn, Icelandic -->
<!ENTITY yuml    "&#255;" ><!-- small y, dieresis or umlaut mark -->
```

Appendix B

NLoop$_{\text{Sim-SVT}}$ Algorithm

```
Algorithm NLoop$_{Sim-SVT}$
Input : Relations R and  S;
            Text-valued join attributes r.A and s.B;
            Buffers $B_S$ and $B_R$, each of size BUFSIZE;
            Enumerated variables BufR and BufS with two Enumerated values
            {NotFull, QuitLoop};
            Similarity function sim()=Cosine();
            Similarity threshold $t_{sim}$
Output: {r.s | $r \in R$ and $s \in S$ and  $f_{out}(r, s) \geq V_t$ and Cosine($\mathbf{u_r}$, $\mathbf{u_S}$) > $t_{sim}$ }
{
      Sort R by $sv_r * $ Cosine($\mathbf{u_r}$, $\mathbf{f_S}$);  Sort S by  $sv_s$;
      BufR = NotFull, R-tupleCount = 0;
      while ( R-tupleCount < |R|  and  BufR=NotFull ){
         i = 0 ;
         while ( R-tupleCount < |R| and i < BUFSIZE and BufR=NotFull){
            read $r_i$ into $B_R$;
            if ($sv_{ri} * sv_{s1} *$ Cosine($\mathbf{u_{ri}}$, $\mathbf{f_S}$) < $V_t$)
               then BufR = QuitLoop;
            i++;
            R-tupleCount++;
         } // read a block of R
         BufS = NotFull ;  S-tupleCount = 0;
         while(S-tupleCount<|S| and BufS=NotEmpty and i>0){
            j = 0;
            while (S-tupleCount < |S| and j< BUFSIZE and BufS = NotFull){
            read $s_j$ into $B_S$;
            if  ($sv_{r1} * sv_{sj} *$ Cosine($\mathbf{u_{r1}}$, $\mathbf{f_S}$)< $V_t$)
               then BufS = QuitLoop;
            j++;
            S-tupleCount++;
            } // read a block of S
            // compare tuples in the blocks
            for each $r \in B_R$
                 for each $s \in B_S$
                     if ($sv_r * sv_s *$ Cosine ($\mathbf{u_r}$, $\mathbf{u_s}$)$\geq V_t$ and Cosine ($\mathbf{u_r}$, $\mathbf{u_s}$)$\geq t_{sim}$ )
                        then add r.s into the output;
         } // finish reading S
      } // finish reading R
}
```