CiteSeerX: AI in a Digital Library Search Engine

Jian Wu*, **Kyle Williams***, **Hung-Hsuan Chen***, **Madian Khabsa****, **Cornelia Caragea**†,
Alexander Ororbia*, **Douglas Jordan****, **& C. Lee Giles***,**
*Information Sciences and Technology
**Computer Science and Engineering
Pennsylvania State University, University Park, PA, 16802
†Computer Science and Engineering, University of North Texas, Denton, TX, 76203

Abstract

CiteSeerX is a digital library search engine that provides access to more than 4 million academic documents with nearly a million users and millions of hits per day. Artificial intelligence (AI) technologies are used in many components of CiteSeerX e.g. to accurately extract metadata, intelligently crawl the web, and ingest documents. We present key AI technologies used in the following components: document classification and deduplication, document and citation clustering, automatic metadata extraction and indexing, and author disambiguation. These AI technologies have been developed by CiteSeerX group members over the past 5–6 years. We also show the usage status, payoff, development challenges, main design concepts, and deployment and maintenance requirements. While it is challenging to rebuild a system like CiteSeerX from scratch, many of these AI technologies are transferable to other digital libraries and/or search engines.

Introduction

CiteSeerX is a digital library search engine that provides access to over 4 million academic documents. In 1997 its predecessor, CiteSeer, was developed at the NEC Research Institute, Princeton, NJ. The service transitioned to the College of Information Sciences and Technology at the Pennsylvania State University in 2003. Since then, the project has been directed by C. Lee Giles. CiteSeer was the first digital library search engine to provide autonomous citation indexing (Giles, Bollacker, and Lawrence 1998). After serving as a public search engine for nearly eight years, CiteSeer began to grow beyond the capabilities of its original architecture. With new features developed, such as author and table search, CiteSeer was redesigned with a new functional architecture[1] that had enhanced searching capabilities and a new name, CiteSeerXĊiteSeerX also provides academic paper metadata, which has been used for many research projects on data mining, machine learning, recommendation systems, social networks, etc.

CiteSeerX is in many ways unique compared to other academic digital libraries and search engines. It is an open access digital library because all documents are harvested from the public Web. This is different from arXiv, Harvard ADS and PubMed where papers are submitted by authors or pushed by publishers. Unlike Google Scholar and Microsoft Academic Search where a significant portion of documents only have metadata available, users have full text access to all papers searchable in CiteSeerX. Instead of simply listing papers, CiteSeerX provides automatically extracted paper metadata and citation context, which enables users to locate the relevant paragraphs and sentences. CiteSeerX provides all metadata through an OAI[2] service interface and on Amazon S3[3], which is not available from Google Scholar and only recently available from Microsoft Academic Search. Finally, CiteSeerX performs automatic extraction and indexing on paper components such as tables and figures, which is rarely seen in other academic search engines.

In addition to providing data services, CiteSeerX also provides a digital library search engine framework that can be deployed on similar sites. This framework, called SeerSuite (Teregowda et al. 2010), has been under active development and applied to other digital libraries. Some components are also available online via an API service, such as text extraction (Williams et al. 2014a).

AI techniques are used in many CiteSeerX components, including document classification, de-duplication, automatic metadata extraction, author disambiguation and more. Here, we describe the AI techniques used in these components and their performance. We also briefly discuss some AI techniques that are under active development.

CiteSeerX Overview

As shown in Figure 1, CiteSeerX can be divided into a frontend and a backend.

Inside the frontend, all query traffic enters through the load balancer and is processed by the web servers. The search index, paper metadata, and downloadable files are hosted by the index, database, and repository servers, respectively. Inside the backend, the web crawl cluster harvests PDF files across the Web. These files are passed to the extraction module where text contents are extracted and classified. Academic documents are then parsed and meta-

[1] Isaac Councill played a crucial role.

[2] Open Archive Initiative.

[3] Amazon charges based on usage

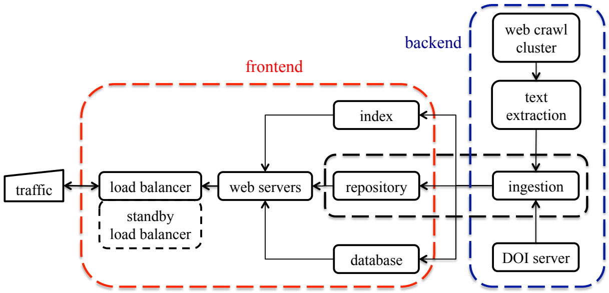

Figure 1: CiteSeerX architecture.

data are extracted, such as titles, and authors. The ingestion module writes all metadata into the database. The PDF files are renamed under a document ID (csxDOI) and saved to the production repository with XML metadata files. Finally, the index data are updated. This architecture was recently migrated from a physical cluster to a private cloud using virtualization techniques (Wu et al. 2014).

CiteSeerX extensively leverages open source software (OSS) which significantly reduces development effort. Red Hat Enterprise Linux (RHEL) 5 and 6 are the operation systems for all servers. *heartbeat-directord* provides virtual IP and load balancing services. Tomcat 7 is used for web service deployment and on indexing servers as a servlet container. MySQL is used as the database management system to store metadata. Apache Solr is used for the index and the Spring framework is extensively used by the CiteSeerX core codebase.

Highlights of AI Technologies

Document Classification

Textual contents extracted from crawled documents are checked by a binary filter which identifies whether they are academic or not. The current filter uses a rule-based model, which looks for keywords such as "references", "bibliography" and their variants. While this appears simple, it yields surprisingly well with 80–90% precision and 70–80% recall on a manually labeled document set. The relatively low recall is caused by academic documents not containing any of the keywords/key-phrases, e.g., invited talks, slides, posters etc. In some papers, such sections are called "literature cited" or "notes". The 10–20% of false positives are due to non-academic documents such as government reports, resumes, newsletters and simply some PDF files containing these keywords/key-phrases in their text body.

We are developing a sophisticated approach to increase the classification recall and precision (Caragea et al. 2014b). The approach utilizes structural features in documents to train a model which is then used to classify documents. We considered four types of structural features. *File specific features* include file size and page count. *Text specific features* include document length in terms of characters, words, lines, the average number of words/lines per page, the reference ratio (the number of references and reference mentions divided by the total number of tokens in a document), the space ratio (the percentage of space characters), the symbol ratio (the percentage of words that start with non-alphanumeric characters), the length ratio (the length of the shortest line divided by the longest line), the number of lines that start with uppercase letters and with non-alphanumeric characters. *Section specific features* include whether the document has section headings such as "abstract", "introduction" or "motivation", "conclusions", "acknowledgements", "references" or "bibliography" and "chapter". *Containment features* are self references such as "this paper", "this book", "this report", "this thesis", "this manual", etc.

An evaluation of the classifier was performed using two independently selected samples from the web crawl repository and from the CiteSeerX production repository. The first sample has more diverse document types than the second. The gold standard was generated by manually labeling documents in each of the two samples, The performance was evaluated with multiple classifiers. The results on the first sample indicate that the Support Vector Machine (Cortes and Vapnik 1995) achieves the highest precision (88.9%), F-measure (85.4%) and accuracy (88.11%), followed by the Logistic Regression classifier (Bishop 2006), which achieves a slightly lower precision (88.0%), F-measure (81.3%) and accuracy (87.39%). The Naïve Bayes (NB) classifier achieves the highest recall (88.6%) at the sacrifice of a low precision (70.3%). The second sample yields similar results. In either sample, the classifiers based on the structural features significantly outperform the baseline in terms of precision, recall and accuracy[4]. Using Information Gain (Mitchell 1997), we rank the most informative features. The top three for the first sample are reference ratio, appearance of "reference" or "bibliography", and document length; the top three for the second sample are page count, number of words, and number of characters. We are working towards extending this binary classifier such that it is able to classify academic documents in multiple categories such as papers, reports, books, slides, and posters. Documents in these categories can be aligned on topics and integrated on the same page, which helps users retrieve information more efficiently.

Document De-duplication and Citation Graph

In submission-based digital libraries, such as arXiv and DBLP, duplication is rare. For a crawl-based digital library, such as CiteSeerX, document duplication is inevitable and can be handled intelligently. Two types of duplication are considered: bitwise- and near-duplication. Bitwise-duplicates occur in web crawling and ingestion. They are detected by matching SHA-1[5] values of new documents against the existing ones in the database. Once detected, these documents are immediately removed.

Near-duplication (ND) refers to documents with similar content but minor differences. A common case is authors updating their downloadable PDF papers on their homepages

[4] An exception is the recall of the second sample. The "reference"/"bibliography" learner achieves a high recall of 94.2% simply because its parent sample was originally generated in that way.

[5] SHA-1 is a cryptographic hash function whose value is typically rendered as a 40 digits hexadecimal number.

with newer versions. In CiteSeerX, ND documents are not deleted but *merged* into a *document cluster*[6]. In most cases, the papers do not contain publication information, i.e., journal name, year, and page numbers, so we have to rely on title and author information, which are seen in almost all papers. Intuitively, ND documents can be detected if two papers have the same title and author names. However, matching title/author strings directly is inefficient because it is very sensitive to the parsing results, which are often noisy.

In CiteSeerX, the ND is detected using a key mapping algorithm. A set of keys are generated for a new document when it is ingested. These document keys are built by concatenating *normalized* author last names and *normalized* titles. For example, for a paper with the title *Focused Crawling Optimization* by *Jian Wu*, two keys are generated: `wu_focuscrawloptimize`, and `wu_crawloptimize`. The first key is matched against all keys in the `keymap` table. If it is not found, a new cluster is created with the above two keys, and the metadata of this paper is used as the metadata of the new cluster. If it matches an existing key in the table, this paper is merged into the cluster that owns that key. In either case, the two keys above are written into the `keymap` table as cluster keys. If a paper contains multiple authors, additional keys are generated using the second author's last name. Note that an *offset title* is used in key generation, i.e., `crawloptimize`, in which the first word is removed. These keys are used when an error in text or header extraction causes the first word in a title to be missed. When a citation is parsed from a paper, it is merged into a cluster in a similar way as a document. Thus, a cluster may contain documents, citations, or both.

An alternative to key based clustering would be a full-text approach. In an experiment, we compared the performance of our clustering method to a state of the art full-text based method (Charikar 2002) that has been shown to work well for academic documents (Williams and Giles 2013). We randomly selected 12,928 clusters having two or more full text papers. We found that only 6,766 ($\sim$ 52%) of these clusters passed the full-text based similarity filter. In many of the failed clusters, the papers are similar but have different lengths. On the other hand, the key mapping algorithm relies on title and author strings. It fails when the first few words in titles and authors are changed. Thus, full-text based methods for near duplicate detection of documents with different lengths seem to be required.

Document clusters are modified when a user correction occurs. When a user corrects paper metadata, it is removed from its existing cluster and assigned to a new cluster based on the new metadata. Should the cluster it previously belonged to become empty, then that cluster is deleted. The citation graph is generated when papers are ingested. The nodes are document clusters and each directional edge represents a citation relationship. The concept of document cluster integrates papers and citations, making it more convenient to perform statistical calculations, ranking and network analysis.

Metadata Extraction

Document metadata is extracted from textual content after it is extracted from the PDF files. This metadata is then automatically indexed for searching, clustering and other purposes. The metadata include three types: header, body and citation. The header includes 15 fields: titles, authors, affiliation, address, note, email, date, abstract, introduction, phone, keyword, web, degree, publication number and page information. The citation basically contains the same fields as the header, but it has to be located and parsed by a different algorithm due to its different format.

Header Extraction Header extraction is performed using SVMHeaderParse (Han et al. 2003), which is an SVM-based header extractor. Although regular expressions and rule-based systems do not require any training and are in general faster, they are largely depend on the application domain and a set of rules or regular expressions that can only be set by domain experts. SVMs are well known for their generalization performance in handling high dimentional data. In SVMHeaderParse, the traditional binary SVM is extended to a multi-class classifier in the "One class versus all others" approach. We summarize the approach below.

SVMHeaderParse first extracts features from a text file extracted from a PDF document. The extraction is performed using a rule-based, context-dependent word clustering method for word-specific feature generation, with the rules extracted from various domain databases and text orthographic properties of words, e.g., capitalization. The domain databases include the standard Linux dictionary, Bob Baldwin's collection of 8441 first names and 19613 last names, Chinese last names, US state names and Canada province names, USA city names, country names and month names. Additional domain databases are generated from training data. Words and bigrams are then clustered based on their domain database properties. Line-specific features are also extracted including the number of words the line contains (see Table 1).

The line classification algorithm includes two steps: independent line classification and contextual line classification. In the Step 1, 15 classifiers (corresponding to 15 header fields above) are trained on 15 labeled feature vector sets, each of which is generated by collecting all feature vectors labeled as a certain class. The goal is to classify text lines into a single class or multiple classes. In the second step, the classification is improved by taking advantage of the context around a line. Specifically, the class labels of the N lines before and after the current line L are encoded as binary features and concatenated to the feature vector of line L formed in Step 1. A line classifier for each metatag is then trained based on these labeled feature vectors with additional contextual information. Test lines are then re-classified by the contextual classifiers, which is repeated such that, in each iteration, the feature vector of each line is extended by incorporating the neighborhood label information predicted in the previous iteration, and converges when the percentage of lines with new class labels is lower than a threshold

We extract metadata from the classified lines. The key is to identify information chunk boundaries. Here, we fo-

[6] This is different from the clustering concept in data mining.

Table 1: Word- and line-specific features.

Word-specific	
email, introduction, single cap letter, postal code abstract, keywords, URL, month name, preposition, degree	
Line-specific	
word#, line position, dictionary words%, date words% non-dictionary words%, page number words%	
Pairs	
Word-specific	Line-specific
publication number	publication number%
note	note words%
affiliation	affiliation words%
address	address words%
first cap dict. words	first cap dict. words%
non dict. words	first cap non dict. words%
three digit number	numbers%

% are percentages of certain type of words in a line.

cus on author name extraction. While identifying chunks in punctuation-separated multi-author lines is relatively easy, it is challenging to identify chunks in space-separated multi-author lines. We first generates all potential name sequences based on some pre-defined name patterns; second, each name sequence is then manually labeled; third, an SVM name classifier is trained based on the labeled sample; finally, the test potential name sequences are classified and the one with the highest score is predicted as the correct name sequence.

Evaluations are performed using the dataset (S99) provided by Seymore et al. 1999, which contains 935 labeled headers of computer science papers. The SVM parameters are chosen using ten-fold cross-validation on 500 training headers and 435 test headers. The overall accuracy is 92.9%, which is better than the 90% reported by S99. The SVM-based approach achieves better performances in most of the classes compared to the Hidden Markov Model. Specifically, the accuracies of author, affiliation, address, and publication number classes are improved by 7%, 9%, 15%, and 35%, respectively. Note that the evaluation above was based on a set of high-quality extracted text files. In real situations, the extracted metadata may be noisy if the input text files are poorly extracted from the original PDF files. In addition, the design of the current classifier is optimized for computer science papers. It does not necessarily perform equally well for other subject domains such as medical science. A general classifier or a domain dependent multi-classifier model is desired.

Citation Extraction CiteSeerX uses ParsCit (Councill, Giles, and Kan 2008) for citation extraction, which is an implementation of a reference string parsing package. The core of ParsCit is a conditional random field (CRF[7]; Lafferty et al. 2001) model used to label the token sequences in the reference strings. This core was wrapped by a heuristic model with added functionality to identify reference strings from plain text files and to retrieve the citation contexts. We summarize the learning model of ParsCit below.

In the reference section of a paper, each reference string R can be viewed as a set of fields (e.g., author, title, year) with punctuations or spaces as delimiters. We break down R into a sequence of tokens $r_1, \cdots, r_n$, each of which is assigned a label from a set of classes $c_1, \cdots, c_m$. To classify a token r_i, we can make use of any information derived from R including previous classification results for $r_1, \cdots, r_{i-1}$. Example training features are listed in Table 2.

Table 2: Examples of features in ParsCit training.

Token identity:	lowercased, lowercased without punctuation
N-gram prefix/suffix:	first 1–4 characters, last 1–4 characters last character (upper/lower/numeric)
Orthographic case:	*InitialCaps*, *MixedCaps*, *ALLCAPS*
Punctuation:	*leadingQuotes*, *endingQuotes*, *multipleHyphens*, *continuingPunctuation*, *stopPunctuation*, *pairedBraces*, *possibleVolume*
Number:	*year*, *possiblePageRange*, *possibleVolume*, *ordinal*, *hasDigit*, *noDigits* etc.
Dictionary:	publisher, surnames, female/male names, months
Token Location:	Token position within the reference string
Possible editor:	Whether a token such as “eds.” is present

ParsCit attempts to find the reference section before parsing the reference string. For generality, ParsCit is designed to be independent of specific formatting and finds reference strings using a set of heuristics given a plain UTF-8 text file. It first searches for sections labeled “references”, “bibliography” or common variations of these keyphrases. If a label is found too early in the document, subsequent matches are sought. The final match is considered the starting point of the reference section. The ending point is found by searching for subsequent section labels such as appendices, acknowledgments or the document end.

The next phase is to segment individual reference strings. This is done by constructing a number of regular expressions matching common marker styles, e.g.,, “[1]” or “1”, then counting the number of matches to each expression in the reference string text. The regular expression with the greatest number of matches is indicated. If no reference string markers are found, several heuristics are used to decide where individual reference starts and ends based on the length of previous lines, strings that appear to be author name lists, and ending punctuation. The CRF model above is applied to the list of individual reference strings. After the CRF modeling, each tagged field is normalized into a standard representation. This makes the structure of metadata more uniform and easy for future analysis.

ParsCit also extracts citation context based on the reference marker discovered above by scanning the body text and locating citations matching a particular reference string. Ci-

[7] https://code.google.com/p/crfpp/

tation context allows a user to quickly and easily see what other researchers say about an article of interest. For marked citation lists, the expression can be a square bracket or other parenthetical expressions. For unmarked citation lists (such as APA style), the expression is constructed using the author last names.

The evaluations were performed on three datasets: Cora (S99), CiteSeerX, and FLUX-CiM (Cortez et al. 2007). The results show that the core module of ParsCit that performs reference string segmentation performs satisfactorily and is comparable to the original CRF based system in Peng & McCallum (2004). ParsCit outperforms FLUX-CiM, which is an unsupervised reference string parsing system, on the key common fields of "author" and "title". However, it makes some errors in not segmenting "volume", "number" and "pages", as ParsCit currently does not further tokenize beyond white spaces (e.g., "11(4):11-22" versus "11 (4) 11 - 22"). It is desirable to incorporate some preprocessing heuristics to ParsCit to correct for such errors.

Author Disambiguation

In addition to document search, another important functionality of CiteSeerX is author search, which enables users to find an author's basic information and previous publications. Author search is also the foundation of several other services we provide, such as collaborator search (Chen et al. 2011) and expert search (Chen et al. 2013).

To search for an author, a typical query string is the author's name. However, processing a name-based query is complex. First, different authors may share the same name. It is not straightforward to decide if "John Smith" of one paper is the same as the one in another paper. Second, one author may have several name variations. For instance, Dr. W. Bruce Croft could be recorded as "W. B. Croft" or "Bruce Croft". The ambiguity of names is a common issue for most digital libraries.

To disambiguate authors, one could define a distance function between two authors and identify the similarity between a pair of authors based on this distance function. However, a digital library usually contains millions of papers and tens of millions of un-disambiguated authors. Comparing every pair of authors requires $O(n^2)$ time complexity, which is intractable for a large n. To reduce the number of comparisons, CiteSeerX groups names into small blocks and claims that an author can only have different name variations within the same block. Thus, we only need to check every pair of names within the same block. CiteSeerX groups two names into one block if the last names are the same and the first initials are the same.

In many cases, name information alone is insufficient for author disambiguation. Other information related to authors is used including, but not limited to, their affiliations, emails, collaborators (coauthors), and contextual information, such as the key phrases and topics of their published papers. CiteSeerX relies on this additional information for author disambiguation. For example, if we found "John Smith" of paper 1 and "J. Smith" of paper 2 belong to different affiliations then the two Smiths are less likely to be the same person. On the other hand, if both papers discuss similar topics and have a similar set of authors, the two names are very likely to refer to the same person. Specifically, CiteSeerX selects 12 features for two target authors (a and b) of two different papers (p and q), including the similarity between a and b's first names, the similarity between their middle names, the relationship between the authoring order of a in p and the order of b in q, the similarity between the emails of the first author in p and the first author in q, the similarity between the terms in the affiliations of the first author in p and the first author in q, the similarity between the names of the authors of p and the authors of q, the similarity between the title of p and the title of q, and few other features.

While we can classify authors by applying a supervised learning approach on the above features, such a classification may output results that violate transitivity principle. For example, even if authors a and b are classified as one person, and b and c are also classified as one person, authors a and c may be classified as two different persons. By applying DBSCAN, a clustering algorithm based on the density reachability of data points, CiteSeerX resolves most of these inconsistent cases (Huang, Ertekin, and Giles 2006). The remaining small portion of ambiguous cases are those located at the boundaries of clusters. These authors are difficult to disambiguate even manually due to insufficient or incorrectly parsed author information.

We have compared the accuracy of the author disambiguation problem using several supervised learning approaches, including Random Forest, Support Vector Machine, Logistic Regression, Naïve Bayesian, and decision trees. We found that the accuracy achieved by the Random Forest significantly outperforms the other learning methods, based on the analysis of variance (ANOVA) (Treeratpituk and Giles 2009). In addition, the Random Forest model can be trained within a reasonable period. Thus, CiteSeerX applies Random Forest on the 12 features for author disambiguation.

Development and Deployment

Although CiteSeerX utilizes many open source software packages, many of the core components are not directly available from open source repositories and require extensive programming and testing. The current CiteSeerX codebase inherited little from its predecessor's (CiteSeer) codebase due to stability and consistency considerations. The core part of the main web apps were written by Dr. Isaac Councill and Juan Pablo Fernández Ramírez and many components were developed by other graduate students, postdocs and software engineers, some which took at least 3-4 years.

Usage and Payoff

CiteSeer started running in 1998 and its successor CiteSeerX has been running since 2008. Since then, the document collection has steadily increased (see Table 3). The goal of CiteSeerX is to improve the dissemination of and access to academic and scientific literature. Currently, CiteSeerX has registered users from around the world and is hit more than 2 million times a day according to web server logs. The download rate is on average 10 PDF files per second (Teregowda, Urgaonkar, and Giles 2010). Besides the web search, CiteSeerX also provides an OAI Protocol for metadata harvesting in order to facilitate content dissemination. By programmatically accessing the CiteSeerX OAI harvest URL, it is possible to download the metadata for all papers that exist in CiteSeerX with an average of about 5,000 requests per month. Researchers are interested in more than just CiteSeerX metadata. For example, CiteSeerX usually receives a dozen or so data requests per month via the contact form on the CiteSeerX website (Williams et al. 2014b). Those requests include graduate students seeking project data sets and researchers that are looking for large datasets for experiments. For these requests, dumps of our database are available on Amazon S3. This alleviates our cost distributing the data since users pay for downloads.

The CiteSeerX (and CiteSeer) data have been used in much novel research work. For example, the CiteSeer data was used to predict the ranking of computer scientists (Feitelson and Yovel 2004). In terms of algorithm design, Councill et al. used CiteSeerX data along with another two datasets to test the performance of the ParsCit software. Madadhain et al. (2005) used CiteSeerX as a motivating example for *JUNG*, which is a language to manipulate, analyze, and visualize data that can be represented as a graph or network. Pham et al. (2011) studied the knowledge network created at the journal/conference level using citation linkage to identify the development of sub-disciplines based on the combination of DBLP and CiteSeerX datasets. Recently, Caragea et al. (2014) generated a large cleaned dataset by matching and merging CiteSeerX and DBLP metadata. This dataset contains cleaned metadata of papers in both CiteSeerX and DBLP including citation context, which is useful for studying ranking and recommendation systems (Chen et al. 2011). Other data sets have also been created (Bhatia et al. 2012).

Previously, the majority of CiteSeerX indexed documents were from computer and information sciences. Recently, a large number of papers have been crawled and ingested from mathematics, physics and medical science. We are increasing our document collection by actively crawling the web using new policies and seeds in order to include new domains. We expect this to encourage users from multiple disciplines to search and download academic papers from CiteSeerX and to be useful for studying cross discipline citation and social networks.

In addition to increasing the collection volume, CiteSeerX also strives to increase the quality of metadata. For example, we are using multiple data cleaning techniques to sanitize and correct wrong metadata. In addition, we are developing new algorithms to improve the quality of text and metadata extraction. Users will soon see cleaner and accurate descriptions and more reliable statistics.

Besides data services, CiteSeerX has also made the digital library search engine framework, *SeerSuite* (Teregowda et al. 2010), available. This framework allows research institutions or individuals to build personalized digital library search engines using their own collection of PDF documents. Most of time, the CiteSeerX group can provide free technical support. As far as we know, at least five other SeerSuite instances are running across the world.

Table 3: Six year document collection growth.

Year	2008	2009	2010	2011	2012	2013
Crawled	1.89	2.90	5.62	6.15	7.93	13.02
Ingested	0.61	1.38	1.66	1.93	2.35	3.80
Index	0.48	0.83	1.02	1.22	1.54	2.89

Numbers are in millions and are counted as *all* documents by the end of each calendar year. Only cluster metadata are indexed, so the number reflect the unique documents after near-duplication filtering.

Main Design Concepts

While the development environment is mostly Linux, components are developed with portability in mind with Java as the main development language. The web application makes use of a model view controller (MVC) architecture implemented within the Spring framework, which allows development to concentrate on application design rather than designing access method connecting applications. The application presentation uses a mix of Java server pages and JavaScript to generate the user interface. The design of the user interface allows the appearance to be modified by switching Cascading Style Sheets. The use of JavaServer Page Standard Tag Library (JSTL) supports template based construction. The web pages allow further interaction through the use of the JavaScript framework.

The web application is composed of servlets corresponding to user queries. These servlets interact with the index and database for keyword search and for the database and index to generate document summaries and the repository to serve cached files. Servlets use Data Access Objects (DAO) to interact with databases and the repository. The Web application is deployed through a web application archive file (.war).

The database design partitions the entire data across three main databases, which contain document metadata and are focused on transactions and version tracking. The second database stores the citation graphs. The third database stores user information, queries, and document portfolios. The databases are deployed using MySQL and are driven by InnoDB for its ACID compliant and hence fully transactional features with *rollback* and *commit*.

Metadata extraction methods are built using Perl. The main module needs to connect multiple components such as copying data from the crawler, text extraction, document classification, header and citation extraction. It works in a batch mode that usually runs on a separate machine to achieve optimal performance. The ingestion system is written in Java. It contains methods to interact with the extraction server, the database, the Solr index and the repository servers. The crawl document importer middleware is written in Python and uses the Django framework. The name disambiguation module is written in Ruby and uses the *Ruby on Rails* framework.

The current CiteSeerX codebase has been designed to be modular and portable. Adding new functionality, such as other databases or repository types is as easy as implementing various DAO classes. Adding other functionality to the front end is easy as well. One example is implementing Solr auto-complete, which is very simple as CiteSeerX uses JSP for its front end functionality. In addition, the use of Java beans enables CiteSeerX to dynamically load objects in memory. One can specify which implementation of the class to use allowing for more flexibility at runtime.

Development Cost

CiteSeerX is not just a digital library that just allows users to search and download documents from a large database and repository. It encapsulates and integrates AI technologies designed to optimize and enhance document acquisition, extraction, and searching processes. We argue that the implementation of these AI technologies makes CiteSeerX unique and more valuable. This makes it difficult to estimate the cost of rebuilding CiteSeerX. An estimation using the SLOCcount software suggests that the total cost to rewrite just the core web app Java code from scratch might be as high as \$1.5 million. By default, SLOCcount uses the basic COCOMO model, which makes the estimate from the number of lines of code assuming a water-fall software development model. Table 4 lists the estimation of effort, time and cost in the default configuration.

Table 4: Development effort and cost estimation using the SLOCcount package.

Description	Estimation
Total physical source lines of code	44,756
Development effort (Person-Years)	10.82
Schedule estimate (Years)	1.32
Estimated average number of developers	8.18
Total estimated development cost[1]	\$1,462,297

Estimates are from the basic COCOMO model in which software development effort and cost are based on program size.

[1] Assumes an average salary of \$56,286/year, 2.40 overhead.

While building a prototype for a CiteSeer-like system can be accomplished with some effort, doing so is simple compared to building a running system. Looking at these challenges in the context of AI, one would face numerous obstacles. To begin with, all the machine learning algorithms need to be scalable to support processing tens to hundreds of thousands of documents per day. For example, the document classifier should not be slower than the crawling rate. Similarly, the information extraction part must not be a bottleneck in the ingestion pipeline. However, having fast machine learning algorithms should not compromise accuracy. This also extends to citation matching and clustering, which compares millions of candidate citations. Therefore, obtaining the best balance between accuracy and scalability is a major challenge that is addressed by relying on heuristic based models for certain problems, while relying on algorithms that need to optimize intractable solutions for others.

Recently, CiteSeerX was migrated from a cluster comprised of 18 physical machines into a private cloud (Wu et al. 2014). Six months were required to finally complete this migration. Our cost analysis indicates that, for now, moving to a private cloud is more cost-effective compared to a public cloud solution such as Amazon EC2. The major challenges include lack of documentation, resource allocation, system compatibility, a complete and seamless migration plan, redundancy/data backup, configuration, security and backward availability. Most of the AI technologies that we used are compatible with the new environments. This migration is a milestone for CiteSeerX because the cloud (virtual) environment will benefit the entire system in terms of scalability, stability, and maintainability.

Maintenance

CiteSeerX was developed by graduate students, postdocs and software engineers in a higher education setting. With limited funding and human resources, it is challenging to maintain such a system and add new features. CiteSeerX is currently maintained by one postdoc, two graduate students, an IT technician and one undergraduate student. However, this changes as students leave. Only the postdoc is full time. The maintenance work includes, but is not limited to, periodically checking system health, testing and implementing new features, fixing bugs and upgrading software, and adding new documents.

CiteSeerX data is updated regularly. The crawling rate varies from 50,000 to 100,000 PDF documents per day. Of the crawled documents, about 40% are eventually identified as being academic and ingested into the database. Assuming a document takes 2MB of disk space, the total data growth rate is about 20GB per day. By analyzing the access logs, we found that about 10 PDF papers are downloaded per second on average. Assuming an average size of a PDF paper is 1MB, this is an outbound traffic of 25TB per month. Currently, this is one of the major costs for implementing CiteSeerX on a service such as Amazon EC2.

Conclusion

We described CiteSeerX, which is an open access digital library search engine, focusing on the AI technologies used in multiple components of the system. CiteSeerX eliminates bitwise duplications using hash functions; it uses a key matching algorithm to merge documents and citations. The metadata is automatically extracted using an SVM-based header extractor, and the citations are parsed by ParsCit, which is built on top of a CRF model. Author names are disambiguated to facilitate the author search. CiteSeerX also has modules to extract tables and index them for search. These AI technologies make CiteSeerX unique and add value to its services. CiteSeerX has a large worldwide user base, with a mean downloading rate of 10 documents per second. Its data is updated daily and is utilized in much novel research. CiteSeerX takes advantages of existing open source software. It contains more than 40,000 lines of code just in its Java codebase. It could take up to 10 person-years to rebuild this codebase from scratch. Despite limited funding and human resources, CiteSeerX has been maintained regularly with many features planned for the near future. Cloud migration is preparation for growth in documents. New features that could be incorporated into CiteSeerXare algorithm search (Tuarob et al. 2013), figure search (Choudhury et al. 2013) and acknowledgment search (Giles and Councill 2004; Khabsa, Treeratpituk, and Giles 2012) .

Acknowledgments

We acknowledge partial support from the National Science Foundation and suggestions from Robert Neches.

References

Bhatia, S.; Caragea, C.; Chen, H.-H.; Wu, J.; Treeratpituk, P.; Wu, Z.; Khabsa, M.; Mitra, P.; and Giles, C. L. 2012. Specialized research datasets in the citeseer^X digital library. *D-Lib Magazine* 18(7/8).

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Caragea, C.; Wu, J.; Ciobanu, A.; Williams, K.; Fernandez-Ramirez, J.; Chen, H.-H.; Wu, Z.; and Giles, C. L. 2014a. Citeseerx: A scholarly big dataset. ECIR '14, 311–322.

Caragea, C.; Wu, J.; Williams, K.; G., S. D.; Khabsa, M.; and Giles, C. L. 2014b. Automatic Identification of Research Articles from Crawled Documents. WSDM-WSCBD '14.

Charikar, M. 2002. Similarity estimation techniques from rounding algorithms. STOC '02, 380–388.

Chen, H.-H.; Gou, L.; Zhang, X.; and Giles, C. L. 2011. CollabSeer: a search engine for collaboration discovery. JCDL '11, 231–240.

Chen, H.-H.; Treeratpituk, P.; Mitra, P.; and Giles, C. L. 2013. CSSeer: an expert recommendation system based on CiteseerX. JCDL '14, 381–382.

Choudhury, S. R.; Tuarob, S.; Mitra, P.; Rokach, L.; Kirk, A.; Szep, S.; Pellegrino, D.; Jones, S.; and Giles, C. L. 2013. A figure search engine architecture for a chemistry digital library. JCDL '13, 369–370.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

Cortez, E.; da Silva, A. S.; Gonçalves, M. A.; Mesquita, F.; and de Moura, E. S. 2007. Flux-cim: Flexible unsupervised extraction of citation metadata. JCDL '07, 215–224.

Councill, I. G.; Giles, C. L.; and Kan, M.-Y. 2008. Parscit: an open-source crf reference string parsing package. LREC '08.

Feitelson, D. G., and Yovel, U. 2004. Predictive ranking of computer scientists using Citeseer data. *Journal of Documentation* 60:44–61.

Giles, C. L., and Councill, I. 2004. Who gets acknowledged: Measuring scientific contributions through automatic acknowledgement indexing. *PNAS* 101(51):17599–17604.

Giles, C. L.; Bollacker, K.; and Lawrence, S. 1998. CiteSeer: An automatic citation indexing system. JCDL '98, 89–98.

Han, H.; Giles, C. L.; Manavoglu, E.; Zha, H.; Zhang, Z.; and Fox, E. A. 2003. Automatic document metadata extraction using support vector machines. JCDL '03, 37–48.

Huang, J.; Ertekin, S.; and Giles, C. 2006. Efficient name disambiguation for large-scale databases. PKDD '06. 536–544.

Khabsa, M.; Treeratpituk, P.; and Giles, C. L. 2012. AckSeer: a repository and search engine for automatically extracted acknowledgments from digital libraries. JCDL '12, 185–194.

Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. ICML '01, 282–289.

Madadhain, J.; Fisher, D.; Smyth, P.; White, S.; and Boey, Y. 2005. Analysis and visualization of network data using jung. *Journal of Statistical Software* 10:1–35.

Mitchell, T. M. 1997. *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., first edition.

Peng, F., and McCallum, A. 2006. Information extraction from research papers using conditional random fields. *Inf. Process. Manage.* 42(4):963–979.

Pham, M.; Klamma, R.; and Jarke, M. 2011. Development of computer science disciplines: a social network analysis approach. *Social Network Analysis and Mining* 1(4):321–340.

Seymore, K.; Mccallum, A.; and Rosenfeld, R. 1999. Learning Hidden Markov Model Structure for Information Extraction. AAAI '99 Workshop on Machine Learning for Information Extraction.

Teregowda, P. B.; Councill, I. G.; Fernández, R. J. P.; Khabsa, M.; Zheng, S.; and Giles, C. L. 2010. Seersuite: developing a scalable and reliable application framework for building digital libraries by crawling the web. WebApps'10, 14–14.

Teregowda, P.; Urgaonkar, B.; and Giles, C. 2010. Cloud computing: A digital libraries perspective. CLOUD '10, 115–122.

Treeratpituk, P., and Giles, C. L. 2009. Disambiguating authors in academic publications using random forests. JCDL '09, 39–48.

Tuarob, S.; Bhatia, S.; Mitra, P.; and Giles, C. L. 2013. Automatic detection of pseudocodes in scholarly documents using machine learning. ICDAR, 738–742.

Williams, K., and Giles, C. L. 2013. Near duplicate detection in an academic digital library. DocEng '13, 91–94.

Williams, K.; Li, L.; Khabsa, M.; Wu, J.; Shih, P.; and Giles, C. L. 2014a. A web service for scholarly big data information extraction. ICWS '14.

Williams, K.; Wu, J.; Choudhury, S. R.; Khabsa, M.; and Giles, C. L. 2014b. Scholarly Big Data Information Extraction and Integration in the CiteSeerX Digital Library. IIWeb '14.

Wu, J.; Teregowda, P.; Williams, K.; Khabsa, M.; Jordan, D.; Treece, E.; Wu, Z.; and Giles, C. L. 2014. Migrating a digital library into a private cloud. IC2E '14.