CiteSeerX: AI in a Digital Library Search Engine

Jian Wu, Kyle William, Hung-Hsuan Chen, Madian Khabsa, Cornelia Caragea, Suppawong Tuarob, Alexander Ororbia, Douglas Jordan,Prasenjit Mitra, C. Lee Giles

■ *CiteSeerX is a digital library search engine that provides access to more than 5 million scholarly documents with nearly a million users and millions of hits per day. 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 show the usage status, payoff, development challenges, main design concepts, and deployment and maintenance requirements. We also present AI technologies, implemented in table and algorithm search, that are special search modes in CiteSeerX. While it is challenging to rebuild a system like CiteSeerX from scratch, many of these AI technologies are transferable to other digital libraries and search engines.*

CiteSeerX is a digital library search engine providing free access to more than 5 million scholarly 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. It was redesigned with a new architecture and new features, such as author and table search, and renamed CiteSeerX.

CiteSeerX is unique compared with other scholarly digital libraries and search engines. It is an open access digital library because all documents are harvested from the public web.

 ISSN 0738-4602

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 have only metadata (such as titles, authors, and abstracts) available, users have full-text access to all papers searchable in CiteSeerX. In addition, CiteSeerX keeps its own repository, which serves cached versions of papers even if their previous links are not alive any more. In additional to paper downloads, CiteSeerX provides automatically extracted metadata and citation context, which enables users to locate the relevant paragraphs and sentences. CiteSeerX provides all metadata through an OAI (Open Archive Initiative) service interface and on Amazon S3 (Amazon charges based on usage). Document metadata download service is not available from Google Scholar and only recently available from Microsoft Academic Search. Finally, CiteSeerX performs automatic extraction and indexing on paper entities such as tables and figures, a capability rarely seen in other scholarly search engines. For all features, CiteSeerX extracts information from the PDFs of scholarly documents, since this is their most common format.

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 services are also available online, 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

Figure 1 illustrates the top-level architecture of the CiteSeerX system. At the front end, all user requests are processed by the CiteSeerX web service, which is supported by three data servers. Searching requests are handled by the index server, cached full-text documents are provided by the repository server, and all metadata are retrieved from the database server. At the back end, the web crawler harvests PDF files across the World Wide Web. These files are passed to the data-extraction module where text contents are extracted and classified. Scholarly documents are then parsed and the metadata, such as titles, authors, and abstracts, are extracted. The ingestion module writes all metadata into the database. The PDF files are renamed with document IDs and saved to the repository server. Finally, the index data are updated. This architecture was recently migrated from a physical machine cluster to a private cloud using virtualization techniques (Wu et al. 2014). CiteSeerX extensively leverages open source software, which significantly reduces development effort. Red Hat Enterprise Linux (RHEL) 5 and 6 are the operating systems for all servers. heartbeat-idirectord provides virtual IP and load-balancing services. Tomcat 7 is used for web service deployment on web and indexing servers. MySQL is used as the database management system to store metadata. Apache Solr is used for the index, and the Spring framework is used in the web application.

Highlights of AI Technologies

In this section, we highlight four AI solutions that are leveraged by CiteSeerX and that tackle different challenges in metadata extraction and ingestion modules (tagged by *C, E, D,* and *A* in figure 1). Document classification (tag *C*) is used to separate academic papers from all PDF documents harvested by the web crawler; metadata extraction (tag *E*) is used to extract header and citation information automatically from academic papers; de-duplication (tag *D*) eliminates duplicate and near-duplicate academic papers based on their metadata and helps to construct the citation graph; author disambiguation (tag *A*) attempts to differentiate authors that share the same names or group name variants that belong to the same author. Author disambiguation enhances search accuracy by author names. It also enables better bibliometric analysis by allowing a more accurate counting and grouping of publications and citations. Overcoming these four challenges is critical to retrieving accurate and clean metadata used for searching and scientific studies. We also present AI technologies implemented in two specific search engines: table and algorithm search developed on top of CiteSeerX.

Document Classification

Textual contents extracted from crawled documents are examined by a binary filter, which judges whether they are scholarly or not. Here scholarly documents include all conference papers, journal and transaction articles, technical reports, books, and even slides and posters as long as they are freely accessible online. The current filter uses a rule-based model, which looks for the reference sections identified by key words such as *references, bibliography,* and their variants. While this appears simple, it performs surprisingly well with 80–90 percent precision and 70–80 percent recall evaluated based on manually labeled document sets. Here, the precision is defined as the number of correctly identified scholarly documents over all the documents examined by the filter; the recall is defined as the number of correctly identified scholarly documents over all scholarly documents in the labeled sample. The relatively low recall is caused by scholarly documents not containing any

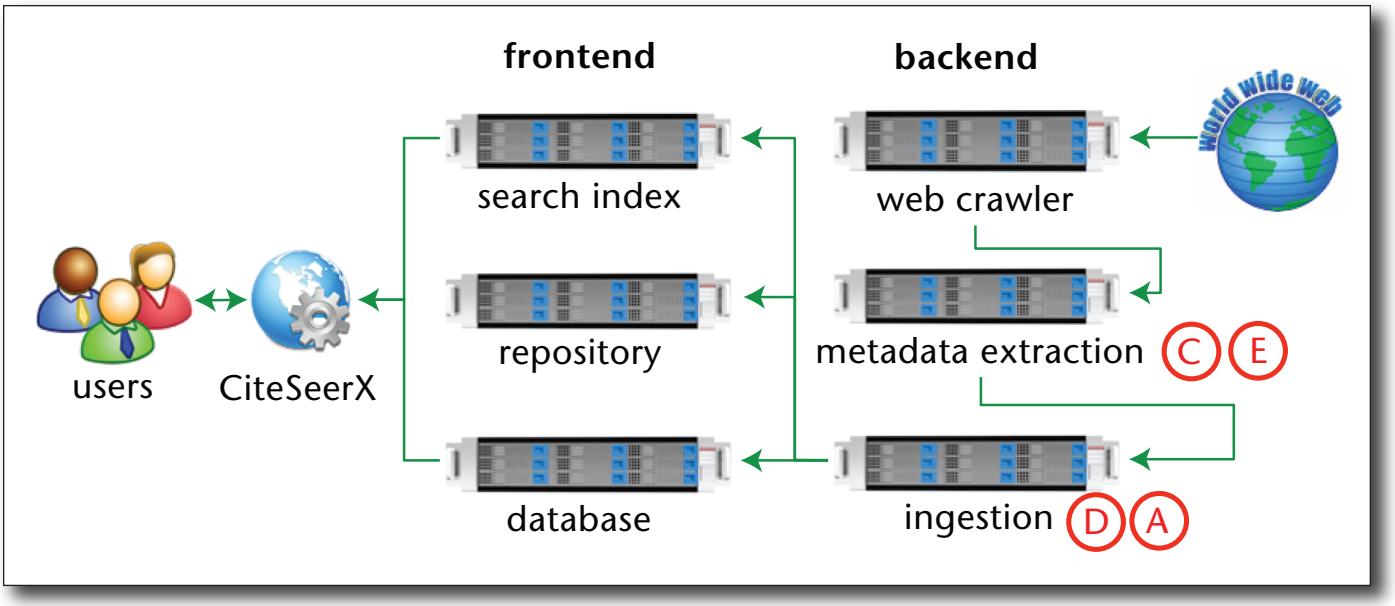


Figure 1. Top Level Architecture of CiteSeerX.

of the key words/key phrases above, for example, invited talks, slides, posters, and others. In some papers, such sections are called *literature cited* or *notes*. The 10–20 percent of false positives are due to nonscholarly documents such as government reports, resumes, newsletters, and some PDF files simply containing these key words/key phrases in their text bodies.

We have developed a sophisticated AI approach to increase the classification recall and precision (Caragea et al. 2014b). The new approach utilizes structural features 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, and 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 nonalphanumeric 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 nonalphanumeric characters. Section-specific features include whether the document has section headings such as abstract, introduction or motivation, conclusions, references or bibliography, and chapter. Containment features are self references such as *this paper, this book, this thesis,* and others.

To evaluate this new classification approach, we draw two random samples from the crawl repository (hereafter sample C) and from the CiteSeerX production repository (hereafter sample P). The crawl repository is populated by the web crawler, which actively crawls the web (Wu et al. 2012) and downloads PDF documents. The production repository contains academic papers selected from the crawl repository based on certain criteria along with associated metadata. Obviously, sample C has more diverse document types than sample P. The gold standard was generated by manually labeling documents in these two samples. The performance was evaluated with multiple classifiers, all of which are trained on the features above. The results on sample C indicate that the support vector machine (SVM) (Cortes and Vapnik 1995) achieves the highest precision (88.9 percent), followed by the logistic regression classifier (Bishop 2006), which achieves a slightly lower precision (88.0 percent). The naïve Bayes classifier achieves the highest recall (88.6 percent) at the sacrifice of a low precision (70.3 percent). Sample P yields similar results. In either sample, the classifiers based on the structural features significantly outperform the baseline in terms of precision, recall, and accuracy. We find the top three informative features of sample C are reference ratio, appearance of *reference* or *bibliography*, and document length; the top three for sample P are page count, number of words, and number of characters. We are working toward extending this binary classifier such that it is able to classify documents into multiple categories including papers, reports, books, slides, posters, and resumes. Documents in these categories can be aligned on topics and displayed on the same web page, which helps users to retrieve information more efficiently.

Metadata Extraction

Document metadata is extracted from textual content extracted from PDF files crawled from the web. This step is a prerequisite for documents to be indexed and clustered. The header metadata includes 15 fields: title, authors, affiliation, address, note, email, date, abstract, introduction, phone, key word, web, degree, publication number, and page information. The citation metadata basically contains the same fields as the header, but it has to be located and parsed by a different algorithm due to its special format.

Header Extraction

Header extraction is performed using SVMHeaderParse (Han et al. 2003), which is an SVM-based header extractor. The idea is to classify textual contents into multiple classes, each of which corresponds to a header metadata field. Although regular expressions and rule-based systems do not require any training models and are generally faster, they 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 dimensional data. In SVMHeaderParse, the traditional binary SVM is extended to a multiclass classifier. The whole process can be divided into three phases: feature extraction, line classification, and metadata extraction.

SVMHeaderParse first extracts word-specific and line-specific features from textual content. The word-specific extraction is performed using a rule-based, context-dependent word clustering method. The rules are extracted from various domain databases, including the standard Linux dictionary, Bob Baldwin's collection of 8441 first names and 19,613 last names, Chinese last names, U.S. state names and Canada province names, USA city names, country names, and month names. We also construct domain databases from training data: affiliation, address, degree, pubnum, note, abstract, introduction, and phone. Text orthographic properties are also extracted, for example, capitalization. The domain database's words and bigrams are then clustered based on their properties. For example, an author line *Chungki Lee James E. Burns* is represented as *Cap1NoneDictWord: :MayName: :Mayname: :SingleCap: :MayName:* after word clustering.

Line-specific features are also extracted such as the number of words a line contains, line number, the percentages of dictionary and nondictionary words, the percentage of date words, and the percentage of numbers in the line.

There are also features representing the percentages of the class-specific words in a line, for instance, the percentage of affiliation words, address words, date words, degree words, phone words, publication number words, note words, and page number words in a line.

The line classification algorithm includes two steps: independent line classification and contextual line classification. In step 1, 15 classifiers (corresponding to 15 header fields) 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 is then trained based on these labeled feature vectors with additional contextual information. Text lines are then reclassified 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.

The key to extract metadata from classified lines is identifying information chunk boundaries. Here, we focus on author name extraction. While it is relatively easy to identify chunks in punctuation-separated multiauthor lines, it is challenging to identify chunks in space-separated multiauthor lines. First, we generate all potential name sequences based on predefined name patterns; second, each name sequence is manually labeled; third, an SVM classifier is trained based on the labeled sample; finally, the potential name sequences are classified and the one with the highest score is predicted as the correct name sequence. An example is presented in figure 2. The core software we use for SVM training and testing is SVMlight (Joachims 1999).

To evaluate the extractor, we use a data set containing 935 labeled headers of computer science papers (Seymore, Mccallum, and Rosenfeld 1999 [hereafter S99]). The overall accuracy is 92.9 percent, which is better than the accuracy (90 percent) reported by S99. Specifically, the accuracies of author, affiliation, address, and publication number classes are improved by 7 percent, 9 percent, 15 percent, and 35 percent, 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, physics, and chemistry. We are planning a new extractor that autonomously chooses a domain-dependent model to extract header metadata.

Citation Extraction

CiteSeerX uses ParsCit (Councill, Giles, and Kan 2008) for citation extraction, which is an implemen-

score	label	possible name sequence
1.640	ACCEPT	Alan Fekete | David Gupta | Victor Luchangco | Nancy Lynch | Alex Shvartsman
0.900	REJECT	Alan Fekete | David Gupta | Victor Luchangco Nancy | Lynch Alex Shvartsman
0.006	REJECT	Alan Fekete | David Gupta Victor | Luchangco Nancy | Lynch | Alex Shvartsman

Figure 2. Example of Potential Name Sequence.

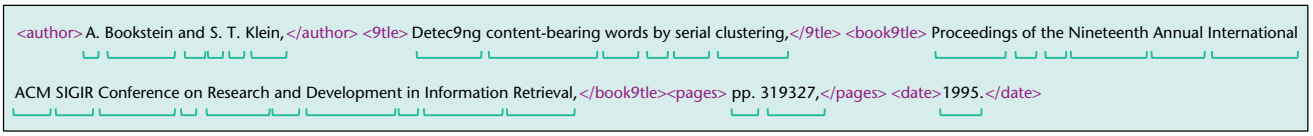


Figure 3. A Labeled Reference String.

Each green (gray) bracket represents a token.

tation of a reference string parsing package. The core of ParsCit is a conditional random field (CRF) (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 citation contexts. We summarize the learning model of ParsCit next.

In the reference section of a paper, each reference string can be viewed as a set of fields (for example, author, title, year) with punctuations or spaces as delimiters. We break down this string into a sequence of tokens, each of which is assigned a label from a set of classes (figure 3). To classify a token, we can make use of any information derived from the reference string including previous classification results. We encode separate features for each token using multiple types of features. Features include the orthographic cases, presence of punctuation, numeric properties, token location within the reference string, whether the token is a possible publisher name, a surname, a month name, and others.

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 key phrases. If it finds a label too early in the document, it seeks subsequent matches. 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. We do this by constructing a number of regular expressions matching common marker styles, for example, [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 next step is to tag individual reference strings automatically using 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. Citation 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 expression. For unmarked citation lists (such as *Wu and Giles [2012]*), the expression is constructed using author last names.

The evaluations were based on three data sets: Cora (S99), CiteSeerX, and the CS data set (Cortez et

	Precision	Recall
key-mapping	0.65	0.67
simhash	0.94	0.88

Table 1. Performance Comparison of ND Algorithms.

al. 2007). The results show that the performance of ParsCit is comparable to the original CRF-based system in Peng and Mccallum (2006). ParsCit outperforms FLUX-CiM (Cortez et al. 2007), which is an unsupervised reference string parsing system, on the key common fields of author and title. However, ParsCit makes some errors in not segmenting volume, number, and pages, as ParsCit currently does not further tokenize beyond white spaces (for example, 11(4):11-22 versus 11 (4) 11 - 22). It is desirable to incorporate some preprocessing heuristics for ParsCit to correct for such errors.

Document De-Duplication and Citation Graph

Duplication is rare in submission-based digital libraries, such as arXiv and PubMed. For a crawl-based digital library, such as CiteSeerX, document duplication is inevitable and should be handled intelligently. Two types of duplication are considered: bitwise and near duplication.

Bitwise duplicates are documents that are identical in all bits. Web crawlers may download the same document in two crawl batches, but the system should keep only the first copy. Bitwise duplicates have the same cryptographic hash values, such as SHA-1, so as long as the new document has exactly the same SHA-1 as an existing one in the database, it is dropped immediately.

Near duplicates (NDs) refer to documents with similar content but minor differences, for example, a preprint and a published version of the same paper. Intuitively, NDs can be detected if two papers have the same title and authors. However, matching title/author strings directly is inefficient because it is sensitive to the text extraction and parsing results, which are noisy in general.

NDs in CiteSeerX are detected using a key-mapping algorithm, applied after the metadata extraction module but before papers are ingested. Most papers we crawled do not contain complete publication information, that is, journal name, year, and page numbers, so we have to rely on title and author information, which appear in almost all papers. When a document is imported, a set of keys is generated by concatenating normalized author last names and normalized titles. For example, two keys are generated for the paper Focused Crawling Optimization by Jian Wu: *wu_focuscrawloptimize,* and *wu_crawloptimize*. The first key is matched against all keys in the key-map database table. If it is not found, a new document cluster is created with the above two keys, and the metadata extracted from this paper is imported as the metadata of the new cluster. If it matches an existing key in the table, this paper is merged into the cluster owning 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, that is, crawloptimize, in which the first word is removed from the original title. These provide alternative keys in paper matching.

An alternative to the key-mapping algorithm would be a full-text-based approach. This approach would not be susceptible to errors in metadata extraction; however, it could still suffer from other shortcomings.

We conducted an experiment to evaluate the performance of the key-mapping algorithm and compare it with the state-of-the-art simhash approach (Charikar 2002). In the experiment, 100,000 documents were randomly sampled from the CiteSeerX collection. Documents containing fewer than 15 tokens were filtered after standard information retrieval preprocessing such as stemming and removing stop words, resulting in a collection of 95,558 documents. Documents belonging to the same cluster were grouped. The precision was estimated by manually inspecting a random sample of 20 document pairs belonging to the same cluster. The recall was estimated based a set of 360 known ND pairs from a previous study (Williams and Giles 2013). We check each of these 360 pairs to determine if they had been assigned to the same cluster and recall was calculated as the proportion of the 360 pairs. Table 1 compares the performance of these two algorithms. The precision and recall of the simhash algorithm are quoted from Williams and Giles (2013).

As can be seen from table 1, the simhash method outperforms the key-mapping method. The main reason is that the key-mapping method is largely dependent on the quality of the author and title extraction since any errors could result in incorrect keys. The results are also only preliminary due to the relatively small sample size and more analysis is needed before strong conclusions can be drawn. Actually, there are cases where the key-mapping approach works better. For instance, given a pair consisting of a paper and an extended version of that paper, the full-text based methods may not consider them NDs due to there potentially being large differences in their text content. The key-mapping approach, however, may still group the papers together as they may have the same title and authors. Whether or not this is desirable will vary depending on the application.

Aside from full-text documents, a document cluster may also contain citations, which typically contain title, author, venue, and year information. When a citation is parsed from a paper, it is merged into a document cluster in the same way as a document. The concept of document cluster integrates papers and citations, making it more accurate to perform statistical calculations, ranking, and network analysis. The citation graph is naturally generated on top of document clusters. Each directional edge represents a citation relationship. Document clusters are modified when a user correction occurs. When a user corrects metadata of a paper, it is removed from its previous cluster and put into a new cluster based on the corrected metadata. The previous cluster is deleted if it becomes empty.

Author Disambiguation

In addition to document search, another important function 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 the 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 interpreted 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. It is infeasible to compare each pair of authors. 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 name pairs 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, simple 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 *A* and *J. Smith* of paper *B* belong to different affiliations then the two Smiths are less likely to be the same person. However, 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 of two different papers, including the similarity between two authors's first names, the similarity between their middle names, the authoring orders in two papers, the similarity between the emails of the first authors, the similarity between the affiliations of the first authors in two papers, the similarity between the author names of in two papers, and the similarity between the titles of two papers.

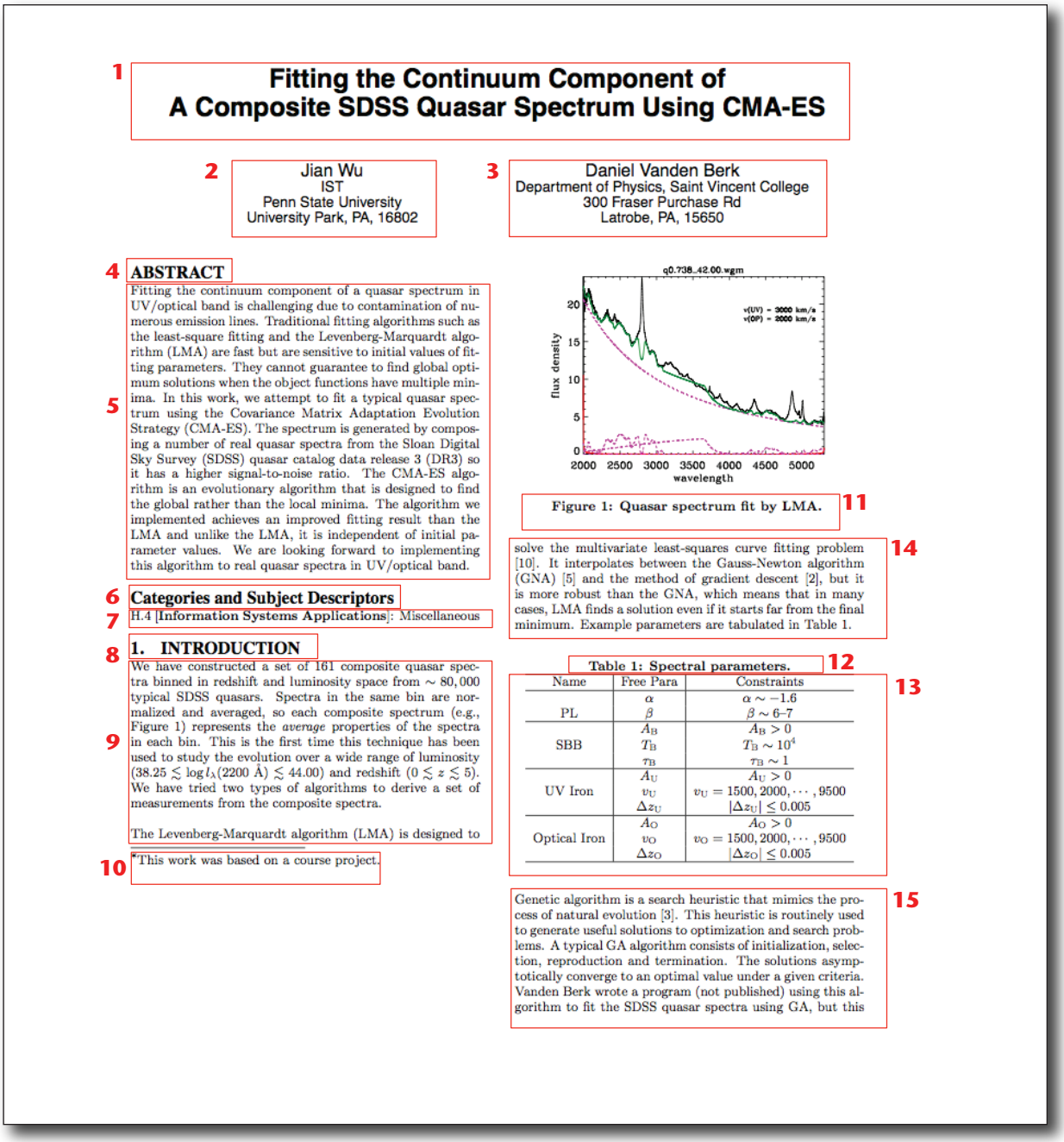
While we can classify authors by applying a supervised learning approach on the aforementioned features, such a classification may output results that violate transitivity principle. For example, even if author *A* and author *B* are classified as one person, and author *B* and author *C* are also classified as one person, author *A* and author *C* may be classified as two different persons. By applying density-based spacial clustering of application with noise (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 cluster boundaries. 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 Bayes, and decision trees. We found that the accuracy achieved by the Random Forest significantly outperforms the other learning methods (Treeratpituk and Giles 2009), and its model can be trained within a reasonable period. Thus, CiteSeerX applies Random Forest on the 12 features we listed for author disambiguation.

Table Extraction

CiteSeerX has partially incorporated a special mode (Liu et al. 2007) to search tables in scholarly papers. Tables are ubiquitous in digital libraries. They are widely used to present experimental results or statistical data in a condensed fashion and are sometimes the only source of that data. However, it can be very difficult to extract these tables from PDF files due to their complicated formats and schema specification and even more difficult to extract the data in the tables automatically. CiteSeerX uses the table metadata extractor developed by Liu et al. (2007), which comprises three major parts: a text information stripper, a table box detector, and a table metadata extractor.

The text information stripper extracts out the textual information from the original PDF files word by



1 **Fitting the Continuum Component of A Composite SDSS Quasar Spectrum Using CMA-ES**

2 Jian Wu
IST
Penn State University
University Park, PA, 16802

3 Daniel Vanden Berk
Department of Physics, Saint Vincent College
300 Fraser Purchase Rd
Latrobe, PA, 15650

4 **ABSTRACT**

5 Fitting the continuum component of a quasar spectrum in UV/optical band is challenging due to contamination of numerous emission lines. Traditional fitting algorithms such as the least-square fitting and the Levenberg-Marquardt algorithm (LMA) are fast but are sensitive to initial values of fitting parameters. They cannot guarantee to find global optimum solutions when the object functions have multiple minima. In this work, we attempt to fit a typical quasar spectrum using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). The spectrum is generated by composing a number of real quasar spectra from the Sloan Digital Sky Survey (SDSS) quasar catalog data release 3 (DR3) so it has a higher signal-to-noise ratio. The CMA-ES algorithm is an evolutionary algorithm that is designed to find the global rather than the local minima. The algorithm we implemented achieves an improved fitting result than the LMA and unlike the LMA, it is independent of initial parameter values. We are looking forward to implementing this algorithm to real quasar spectra in UV/optical band.

6 **Categories and Subject Descriptors**

7 H.4 [**Information Systems Applications**]: Miscellaneous

8 **1. INTRODUCTION**

9 We have constructed a set of 161 composite quasar spectra binned in redshift and luminosity space from $\sim 80,000$ typical SDSS quasars. Spectra in the same bin are normalized and averaged, so each composite spectrum (e.g., Figure 1) represents the *average* properties of the spectra in each bin. This is the first time this technique has been used to study the evolution over a wide range of luminosity ($38.25 \lesssim \log l_\lambda(2200\ \text{Å}) \lesssim 44.00$) and redshift ($0 \lesssim z \lesssim 5$). We have tried two types of algorithms to derive a set of measurements from the composite spectra.

The Levenberg-Marquardt algorithm (LMA) is designed to

10 *This work was based on a course project.

11 **Figure 1: Quasar spectrum fit by LMA.**

solve the multivariate least-squares curve fitting problem [10]. It interpolates between the Gauss-Newton algorithm (GNA) [5] and the method of gradient descent [2], but it is more robust than the GNA, which means that in many cases, LMA finds a solution even if it starts far from the final minimum. Example parameters are tabulated in Table 1.

12 **Table 1: Spectral parameters.**

13

Name	Free Para	Constraints
PL	α	$\alpha \sim -1.6$
	β	$\beta \sim 6$–7
SBB	$A_{\rm B}$	$A_{\rm B} > 0$
	$T_{\rm B}$	$T_{\rm B} \sim 10^4$
	$\tau_{\rm B}$	$\tau_{\rm B} \sim 1$
UV Iron	$A_{\rm U}$	$A_{\rm U} > 0$
	$v_{\rm U}$	$v_{\rm U} = 1500, 2000, \cdots, 9500$
	$\Delta z_{\rm U}$	$\lvert\Delta z_{\rm U}\rvert \le 0.005$
Optical Iron	$A_{\rm O}$	$A_{\rm O} > 0$
	$v_{\rm O}$	$v_{\rm O} = 1500, 2000, \cdots, 9500$
	$\Delta z_{\rm O}$	$\lvert\Delta z_{\rm O}\rvert \le 0.005$

Genetic algorithm is a search heuristic that mimics the process of natural evolution [3]. This heuristic is routinely used to generate useful solutions to optimization and search problems. A typical GA algorithm consists of initialization, selection, reproduction and termination. The solutions asymptotically converge to an optimal value under a given criteria. Vanden Berk wrote a program (not published) using this algorithm to fit the SDSS quasar spectra using GA, but this

14 15

Figure 4. An Example of the Box-Cutting Result.

word by analyzing the output of a general text extractor such as PDFBox or PDFLib TET. These words are then reconstructed with their position information and written into a document content file, which specifies the position, line width, and fonts of each line. An example line in the file is

[X,Y]=[31,88] Width=[98] font=[14] Text=[This paper]

Based on the document content file, the tables are identified using a box-cutting method, which attempts to divide all literal components in a page into *boxes*. Each box is defined as a rectangular region containing adjacent text lines with a uniform font size. The spacing between two text lines in each box must be less than the maximal line spacing between two adjacent lines in the same paragraph. Figure 4 illustrates the results of the box-cutting method when applied on an article page. These boxes are then classified into three categories: small-font, regular-font, and large-font, depending on how their font sizes compare to the font size of the document body text. Of course, other methods can be considered.

The next step is to find tables and their metadata in these boxes (Liu et al. 2006). First, the algorithm generates table candidates by matching small-font boxes against a list of key words, such as *Table* and *TABLE*. If it detects the tabular structure from white space information, it confirms this box as a real table. If the algorithm does not find any table candidates in the first iteration, it starts the second iteration on regular-font boxes and then large-font boxes. Most tables are in fact detected among small-font boxes. Figure 4 shows that Box 12 is detected to contain the key word Table. Its neighbor — Box 13 — is confirmed as the table body.

Work in this area is still of interest. The ICDAR conference has held table competitions for several years with the goal of improving extraction precision and recall.

Algorithm Extraction and Indexing

Recently AlgorithmSeer, a prototype of the algorithm search mode of CiteSeerX (Tuarob, Mitra, and Giles 2014) has been designed. In this section, we focus on AI technologies used in algorithm extraction and indexing.

Algorithms are ubiquitous in computer science and related fields. They offer stepwise instructions for solving computational problems. With numerous new algorithms developed and reported every year, it would be useful to build systems that automatically identify, extract, index, and search this ever increasing collection of algorithms. Such systems could be useful to researchers and software developers looking for cutting-edge solutions to their problems.

A majority of algorithms in computer science documents are represented as pseudocodes. We develop three methods for detecting pseudocodes in scholarly documents including a rule-based, a machine-learning based, and a combined method. All methods process textual content extracted from PDF documents. The rule-based method detects the presence of pseudocode captions using a set of regular expressions that captures captions containing at least one algorithm key word or its variants, for example, *algorithm, pseudocode,* and others. This method yields high detection precision (87 percent), but low recall (45 percent), because a majority of pseudocode (roughly 26 percent) does not have associated captions.

The machine-learning method directly detects the presence of pseudocode content. This method relies on the assumption that pseudocodes are written in a sparse, programminglike manner, which can be visually spotted as sparse regions in documents. Our experiments show that a sparse region contains at least four consecutive lines, whose nonspace characters are less than 80 percent of the average number of characters per line. We use 47 features to characterize each sparse region, including a combination of font-style based, context based, content based, and structure based features. The machine-learning method can capture most noncaptioned pseudocodes, but some still remain undetected. Such pseudocodes are either written in a descriptive manner or are presented as figures.

The combined method combines the benefits of both rule-based and machine-learning methods through a set of heuristics in two steps, (1) for a given document, run the rule-based and the machine-learning methods; (2) for each pseudocode box detected by the machine-learning method, if a pseudocode caption detected by the rule-based method is in proximity, the pseudocode box and the caption are combined. The combined method achieves a precision of 87 percent and a recall of 67 percent (Tuarob et al. 2013).

Indexable metadata of an algorithm includes caption, reference sentences, and synopsis. The former two are directly extracted from the original text. The synopsis is a summary of an algorithm, which is generated using a supervised learning approach to rank and retrieve top relevant sentences in the document (Bhatia et al. 2011). The synopsis generation method employs a naïve Bayes classifier to learn features from the captions and references. The features include content-based and context-based features (Tuarob et al. 2013). Again, algorithm search is still work in progress with much left to do at this time.

Development and Deployment

Although CiteSeerX utilizes many open source software packages, many of the core components are designed and coded by CiteSeerX contributors. The current CiteSeerX codebase inherited little from its predecessor's (CiteSeer) codebase. The core part of the main web apps were written by Isaac Councill and Juan Pablo Fernández Ramírez. Other components were developed by graduate students, postdocs, and software engineers, over at least three to four years.

Usage and Payoff

CiteSeer started running in 1998 and its successor CiteSeerX has been running since 2008. Since then, the document collection has been steadily growing (see figure 5). The goal of CiteSeerX is to improve the dissemination of and access to scholarly and scientific literature. Currently, CiteSeerX has registered users from around the world and is hit more than 2 million times a day. The download rate is about 3–10 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 metadata for all CiteSeerX papers. We are receiving about 5000 OAI requests each month on average. Researchers are

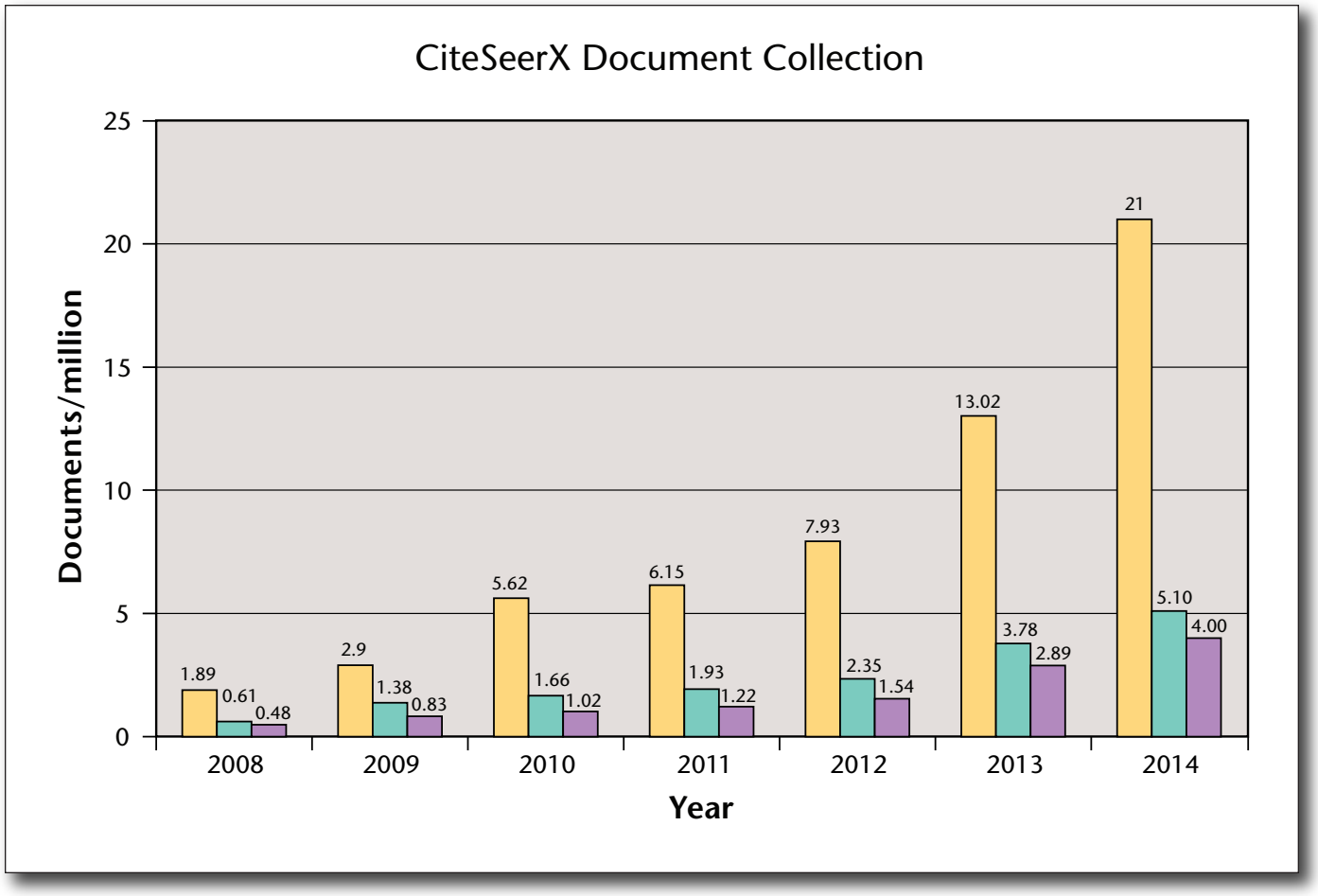


Figure 5. Changes in the CiteSeerX Document Collection Since 2008.

The indexed counts reflect unique papers without ND.

interested in more than just our metadata but also real documents. CiteSeerX usually receives a dozen or so data requests per month through 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 data sets for experiments. For these requests, dumps of our database are available on Amazon S3. This alleviates our cost for 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, Giles, and Kan (2008) used CiteSeerX data along with another two data sets to evaluate the performance of ParsCit. 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, Klamma, and Jarke (2011) studied the knowledge network created at the journal/conference level using citation linkage to identify the development of subdisciplines based on the combination of DBLP and CiteSeerX data sets. Recently, Caragea et al. (2014a) generated a large cleaned data set by matching and merging CiteSeerX and DBLP metadata. This data set 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 papers were from computer and information sciences. Recently, a large number of papers have been crawled and ingested from mathematics, physics, and medical science by incorporating papers from open-access repositories, such as PubMed (subset), and crawling URLs released by Microsoft Academic Search. It is challenging to extract metadata from noncomputer science papers due to their different header and citation formats. As a result, the current extraction tools need to be revised to be more general and efficient for papers in new domains. We are working to implement a conditional random field (CRF)–based meta-

data extraction tool to replace SVMHeaderParse, and rebuild the entire production database and repository. We expect this to encourage users from multiple disciplines to search and download scholarly papers from CiteSeerX, and to be useful for studying cross-discipline citation and social networks.

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

Besides data services, CiteSeerX has released the digital library search engine framework, SeerSuite (Teregowda et al. 2010), which can be used for building personalized digital library search engines. As far as we know, at least five other SeerSuite instances are running in the world.

Main Design Concepts

While RHEL is the major development environment, the application uses Java as the main programming language for portability. The architecture of the web application is implemented within the Spring framework, which allows developers to concentrate on application design rather than access methods connecting to applications, for example, databases. The application presentation uses a mixture of Java server pages and JavaScript to generate the user interface. The web application is composed of servlets, which are Java classes used to extend the capabilities of web servers (that is, Tomcat) by means of responding to HTTP requests. These servlets interact with the indexing and database servers for key word search and generating document summary pages. They also interact with the repository server to serve cached files.

We partition the entire data across three databases. The main database contains document metadata and is focused on transactions and version tracking. The second database stores the citation graphs, and the third stores user information, queries, and document portfolios. The databases are deployed using MySQL and are driven by InnoDB for its fully transactional features with rollback and commit.

Metadata extraction methods are built on top of a Perl script, with a multithread wrapper in Java. The wrapper is responsible for acquiring documents from the crawl database and repository, as well as controlling jobs. The Perl script assembles multiple components such as text extraction, document classification, and header and citation extraction. It works in an automatic mode, in which each thread keeps processing new batches until it receives a stop command. It is fast enough to extract one document per second on average. The ingestion system is written in Java, which 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.

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*	$1,462,297

Table 1. Performance Comparison of ND Algorithms.

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 more classes. 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 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. The implementation of these AI technologies makes CiteSeerX unique and more valuable, but meanwhile makes it difficult to estimate the cost of rebuilding itself. 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. Table 2 lists the estimation of effort, time, and cost in the default configuration.

Table 2 implies it could take lots of effort to build a CiteSeerlike system from scratch. 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 hundreds of thousands of documents per day. For example, the document classifier should not be too slow compared to 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 composed of 18 physical machines into a private cloud platform (Wu et al. 2014) for scalability, stability, and maintainability. Our cost analysis indicates that, based on the current market, moving to a private cloud is more cost-effective compared with 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 modules are compatible with the new environment.

Maintenance

CiteSeerX has been maintained by graduate students, postdocs, and department information-technology (IT) support since it moved to Pennsylvania State University. 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 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 percent–50 percent are eventually identified as scholarly and ingested into the database.

Conclusion

We described CiteSeerX, an open access digital library search engine, focusing on AI technologies used in multiple components of the system. CiteSeerX eliminates bitwise duplicates using hash functions; it uses a key-mapping algorithm to cluster documents and citations. The metadata is automatically extracted using an SVM-based header extractor, and the citations are parsed by ParsCit, a reference string parsing package. Author names are disambiguated to facilitate the author search. CiteSeerX has modules to extract tables and index them for search.

Algorithms are also extracted from PDF papers using combinations of rule-based and machine-learning methods. These AI technologies make CiteSeerX unique and add value to its services. CiteSeerX has a large worldwide user base, with a downloading rate of 3–10 documents per second. Its data is updated daily and is utilized in many novel research projects. CiteSeerX takes advantages of existing open source software. It contains more than 40,000 lines of code 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. Migrating to the cloud environment is a milestone to scale up the whole system. New features that could be incorporated into CiteSeerX are 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 Citeseerx Digital Library. *D-Lib Magazine* 18(7/8).

Bhatia, S.; Tuarob, S.; Mitra, P.; and Giles, C. L. 2011. An Algorithm Search Engine for Software Developers. In *Proceedings of the 3rd International Workshop On Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, 13–16. New York: Association for Computing Machinery.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning* (Information Science and Statistics). Secaucus, NJ: 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. In *Proceedings of the 36th European Confernence on Information Retrieval,* 311–322. Berlin: Springer.

Caragea, C.; Wu, J.; Williams, K.; Gollapalli, S. D.; Khabsa, M.; and Giles, C. L. 2014b. Automatic Identification of Research Articles From Crawled Documents. Paper presented at the 2014 WSDM Workshop on Web-Scale Classification: Classifying Big Data From the Web. 28 February 2014, New York, NY.

Charikar, M. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing (STOC '02),* 380–388. New York: Association for Computing Machinery.

Chen, H.-H.; Gou, L.; Zhang, X.; and Giles, C. L. 2011. Collabseer: A Search Engine for Collaboration Discovery. In *Proceedings of the 2011 ACM/IEEE Joint Conference on Digital Libraries (JCDL '11),* 231–240. New York: Association for Computing Machinery.

Chen, H.-H.; Treeratpituk, P.; Mitra, P.; and Giles, C. L. 2013. Csseer: An Expert Recommendation System Based on Citeseerx. In *Proceedings of the 13th ACM/IEEE Joint Conference on Digital Libraries (JCDL '13)*, 381–382. New York: Association for Computing Machinery.

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. In *Proceedings of the 13th ACM/IEEE Joint Conference on Digital Libraries (JCDL '13),* 369–370. New York: Association for Computing Machinery.

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. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL '07),* 215–224. New York: Association for Computing Machinery.

Councill, I.; Giles, C. L.; and Kan, M.-Y. 2008. Parscit: An Open-Source CRF Reference String Parsing Package. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation* (LREC'08). Paris: European Language Resources Association (ELRA).

Feitelson, D. G., and Yovel, U. 2004. Predictive Ranking of Computer Scientists Using Citeseer Data. *Journal of Documentation* 60(1): 44–61.

Giles, C. L., and Councill, I. 2004. Who Gets Acknowledged: Measuring Scientific Contributions Through Automatic Acknowledgement Indexing. *Proceedings of the National Academy of Sciences of the United States of America* 101(51): 17599–17604.

Giles, C. L.; Bollacker, K.; and Lawrence, S. 1998. Citeseer: An Automatic Citation Indexing System. In *Proceedings of the IEEE International Forumn on Research and Technology Advances in Digital Libraries* (ADL '98), 89–98. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Han, H.; Giles, C. L.; Manavoglu, E.; Zha, H.; Zhang, Z.; and Fox, E. A. 2003. Automatic Document Metadata Extraction Using Support Vector Machines. In *Proceedings of the 2003 ACM/IEEE Joint Conference on Digital Libraries* (JCDL '03), 37–48. New York: Association for Computing Machinery.

Huang, J.; Ertekin, S.; and Giles, C. L. 2006. Efficient Name Disambiguation for Large-Scale Databases. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD '06). 536–544. Berlin: Springer.

Joachims, T. 1999. Making Large-Scale SVM Learning Practical. In *Advances in Kernel Methods,* 169–184. Cambridge, MA: The MIT Press.

Khabsa, M.; Treeratpituk, P.; and Giles, C. L. 2012. ACKseer: A Repository and Search Engine for Automatically Extracted Acknowledgments from Digital Libraries. In *Proceedings of the 2012 ACM/IEEE Joint Conference on Digital Libraries* (JCDL '12), 185–194. New York: Association for Computing Machinery.

Lafferty, J. D.; Mccallum, A.; and Pereira, F. C. N. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning* (ICML 2001), 282–289. San Francisco: Morgan Kaufmann.

Liu, Y.; Bai, K.; Mitra, P.; and Giles, C. L. 2007. Tableseer: Automatic Table Metadata Extraction and Searching in Digital Libraries. In *Proceedings of the 7th ACM/IEEE-Computer Society Joint Conference on Digital Libraries* (JCDL '07), 91–100. New York: Association for Computing Machinery.

Liu, Y.; Mitra, P.; Giles, C. L.; and Bai, K. 2006. Automatic Extraction of Table Metadata from Digital Documents. In *Proceedings of the 6th ACM/IEEE-Computer Society Joint Conference on Digital Libraries* (JCDL '06), 339–340. New York: Association for Computing Machinery.

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): 1–35.

Peng, F., and Mccallum, A. 2006. Information Extraction from Research Papers Using Conditional Random Fields. *Information Processing Management* 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. In Machine Learning for Information Extraction: Papers from the AAAI Workshop, ed. M. E. Califf. Technical Report WS-99-11. Menlo Park, CA: AAAI Press.

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. Paper presented at the UNSENIX Conference on Web Application Development (WebApps '10), 23–24 June, Boston, MA.

Teregowda, P.; Urgaonkar, B.; and Giles, C. L. 2010. Cloud Computing: A Digital Libraries Perspective. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing* (Cloud '10), 115–122. Piscataway, NJ: Institute for Electrical and Electronics Engineers.

Treeratpituk, P., and Giles, C. L. 2009. Disambiguating Authors in Academic Publications Using Random Forests. In *Proceedings of the 2009 ACM/IEEE Joint Conference on Digital Libraries* (JCDL '09), 39–48. New York: Association for Computing Machinery.

Tuarob, S.; Bhatia, S.; Mitra, P.; and Giles, C. L. 2013. Automatic Detection of Pseudocodes in Scholarly Documents Using Machine Learning. Paper presented at the Twelfth International Conference on Document Analysis and Recognition (ICDAR 2013), 25–28 August, Washington, D.C.

Tuarob, S.; Mitra, P.; and Giles, C. L. 2014. Building a Search Engine for Algorithms. *ACM SIGWEB Newsletter* (Winter): Article 5.

Williams, K., and Giles, C. L. 2013. Near Duplicate Detection in an Academic Digital Library. In *Proceedings of the 14th ACM Conference on Document Engineering* (Doceng '13), 91–94. New York: Association for Computing Machinery.

Williams, K.; Li, L.; Khabsa, M.; Wu, J.; Shih, P.; and Giles, C. L. 2014a. A Web Service for Scholarly Big Data Information Extraction. In *Proceedings of the 21st IEEE International Conference on Web Services* (ICWS 2014). Piscataway, NJ: Institute for Electrical and Electronics Engineers.

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. In *Proceedings of the 2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW).* Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Wu, J.; Teregowda, P.; Ramírez, J. P. F.; Mitra, P.; Zheng, S.; and Giles, C. L. 2012. The Evolution of a Crawling Strategy for an Academic Document Search Engine: Whitelists and Blacklists. In *Proceedings of the 3rd Annual ACM Web Science Conference,* 340–343. New York: Association for Computing Machinery.

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. In P*roceedings of the IEEE International Conference on Cloud Computing (IC2E 2014).* Piscataway, N.J.: Institute for Electrical and Electronics Engineers.

Jian Wu is a postdoctoral scholar in College of Information Sciences and Technology at Pennsylvania State University. He received his Ph.D. in astronomy and astrophysics in 2011. He is currently the technical director of CiteSeerX. His research interests include information extraction, web crawling, cloud computing, and big data.

Kyle Williams is a Ph.D. candidate in information sciences and technology at Pennsylvania State University. His research interests include information retrieval, machine learning and digital libraries. His work involves the use of search, machine learning and text inspection for managing document collections.

Hung-Hsuan Chen is a researcher at the Computational Intelligence Technology Center at the Industrial Technology Research Institute. He received his Ph.D. in 2014 at Pennsylvania State University. He is interested in information extraction, information retrieval, data mining, natural language processing, and graph analysis. He developed CSSeer and CollabSeer.

Madian Khabsa is a Ph.D. candidate in computer science and engineering at Pennsylvania State University. His research interests are big data, information retrieval and extraction, applied machine learning, and data mining. He is also interested in building and contributing to large-scale systems.

Cornelia Caragea is an assistant professor at University of North Texas, where she is part of the interdisciplinary Knowledge Discovery from Digital Information (KDDI) research cluster, with a dual appointment in computer science and engineering and library and information sciences. She obtained her Ph.D. from Iowa State University in 2009. Her research interests are machine learning, text mining, information retrieval, social and information networks, and natural language processing, with applications to scholarly big data and digital libraries.

Suppawong Tuarob is a Ph.D. candidate in computer science and engineering at Pennsylvania State University. He earned his BSE and MSE in computer science and engineering from the University of Michigan–Ann Arbor. His research involves data mining in large-scale scholarly data, social media, and health-care domains.

Alexander Ororbia is a Ph.D. graduate student in information sciences and technology at Pennsylvania State University. He received his BS in computer science and engineering at Bucknell University, with minors in mathematics and philosophy. His research focuses on developing connectionist architectures capable of deep learning.

Douglas Jordan is a senior in the College of Engineering at Pennsylvania State University majoring in computer science and mathematics. He is in charge of transferring CiteSeerX data.

Prasenjit Mitra is a professor in the College of Information Sciences and Technology at Pennsylvania State University. He serves on the graduate faculty of the Department of Computer Sciences and Engineering and is an affiliate faculty member of the Department of Industrial and Manufacturing Engineering at Pennsylvania State University. He received his Ph.D. from Stanford University in 2004. His research contributions have been primarily in the fields of information extraction, information integration, and information visualization.

C. Lee Giles is the David Reese Professor at the College of Information Sciences and Technology at Pennsylvania State University, University Park, PA, with appointments in computer science and engineering and supply chain and information systems. He is a Fellow of the ACM, IEEE, and INNS (Gabor prize). He is probably best known for his work on estimating the size of the web and on the search engine and digital library, CiteSeer, which he cocreated, developed, and maintains. He has published more than 400 refereed articles with nearly 26,000 citations with an h-index of 77 according to Google Scholar.