The BINGO! System for Information Portal Generation and Expert Web Search

Sergej Sizov, Michael Biwer, Jens Graupmann,
Stefan Siersdorfer, Martin Theobald, Gerhard Weikum, Patrick Zimmer

University of the Saarland
Department of Computer Science
Im Stadtwald, 66123 Saarbruecken
Germany

Abstract

This paper presents the BINGO! focused crawler, an advanced tool for information portal generation and expert Web search. In contrast to standard search engines such as Google which are solely based on precomputed index structures, a focused crawler interleaves crawling, automatic classification, link analysis and assessment, and text filtering. A crawl is started from a user-provided set of training data and aims to collect comprehensive results for the given topics.

The focused crawling paradigm has been around for a few years and many of our techniques are adopted from the information retrieval and machine learning literature. BINGO! is a system-oriented effort to integrate a suite of techniques into a comprehensive and versatile tool. The paper discusses its overall architecture and main components, important lessons from early experimentation and the resulting improvements on effectiveness and efficiency, and experimental results that demonstrate the usefulness of BINGO! as a next-generation tool for information organization and search.

1 Introduction

1.1 The Problem of Web and Intranet Information Search

Web search engines mostly build on the vector space model that views text documents (including HTML or XML documents) as vectors of term relevance scores [3, 19]. These terms, also known as features, represent word occurrence frequencies in documents after stemming and other normalizations. Queries are vectors too, so that similarity metrics between vectors, for example, the Euclidean distance or the cosine metric, can be used to produce a ranked list of search results, in descending order of (estimated) relevance. The quality of a search result is assessed a posteriori by the empirical metrics precision and recall: precision is the fraction of truly relevant documents among the top N matches in the result ranking (N typically being 10), and recall is the fraction of found documents out of the relevant documents that exist somewhere in the underlying corpus (e.g., the entire Web). More recently, the above basic model has been enhanced by analyzing the link structure between documents, viewing the Web as a graph, and defining the authority of Web sites or documents as an additional metric for search result ranking [5, 14]. These approaches have been very successful in improving the precision (i.e., "sorting out the junk" in more colloquial terms) for typical mass queries such as "Madonna tour" (i.e., everything or anything about the concert tour of pop star Madonna). However, link analysis techniques do not help much for expert queries where recall is the key problem (i.e., finding a few useful results at all).

Two important observations can be made about the above class of advanced information demands. First, the best results are often obtained from Yahoo-style portals that maintain a hierarchical directory of topics, also known as an ontology; the problem with this approach is, however, that it requires intellectual work for classifying new documents into the ontology and thus does not scale with the Web. Second, fully automated Web search engines such as Google, Altavista, etc. sometimes yield search results from which the user could possibly reach the actually desired information by following a small number of hyperlinks; here the problem is that exhaustively surfing the vicinity of a Web document may often take hours and is thus infeasible in practice. These two observations have motivated a novel approach known as focused crawling or thematic crawling [7], which can be viewed as an

Proceedings of the 2003 CIDR Conference

attempt to automate the above kinds of intellectual preprocessing and postprocessing.

1.2 The Potential of Focused Crawling

In contrast to a search engine's generic crawler (which serves to build and maintain the engine's index), a focused crawler is interested only in a specific, typically small, set of topics such as 19th century Russian literature, backcountry desert hiking and canyoneering, or programming with (the Web server scripting language) PHP. The topics of interest may be organized into a user- or community-specific hierarchy. The crawl is started from a given set of seed documents, typically taken from an intellectually built ontology, and aims to proceed along the most promising paths that stay "on topic" while also accepting some detours along digressing subjects with a certain "tunnelling" probability. Each of the visited documents is classified into the crawler's hierarchy of topics to test whether it is of interest at all and where it belongs in the ontology; this step must be automated using classification techniques from machine learning such as Naive Bayes, Maximum Entropy, Support Vector Machines (SVM), or other supervised learning methods [15, 17, 23]. The outcome of the focused crawl can be viewed as the index of a personalized information service or a thematically specialized search engine.

A focused crawler can be used for at least two major problems in information organization and search:

1. Starting with a reasonable set of seed documents that also serve as training data for the classifier, a focused crawl can populate a topic directory and thus serves as a largely automated *information portal generator*.
2. Starting with a set of keywords or an initial result set from a search engine (e.g., from a Google query), a focused crawl can improve the recall for an advanced *expert query*, a query that would take a human expert to identify matches and for which current Web search engines would typically return either no or only irrelevant documents (at least in the top ten ranks).

In either case the key challenge is to minimize the time that a human needs for setting up the crawl (e.g., provide training data, calibrate crawl parameters, etc.) and for interpreting or analyzing its results. For example, we would expect the human to spend a few minutes for carefully specifying her information demand and setting up an overnight crawl, and another few minutes for looking at the results the next morning. In addition, the focused crawler may get back to the user for feedback after some "learning" phase of say twenty minutes.

This mode of operation is in significant contrast to today's Web search engines which rely solely on precomputed results in their index structures and strictly limit the computer resource consumption per query in the interest of maximizing the throughput of "mass user" queries. With human cycles being much more expensive than computer and network cycles, the above kind of paradigm shift seems to be overdue for advanced information demands (e.g., of scientists).

1.3 Contribution and Outline of the Paper

This paper presents the BINGO! system that we have developed in the last two years at the University of the Saarland.[1]. Our approach has been inspired by and has adopted concepts from the seminal work of Chakrabarti et al. [7], but we believe it is fair to call our system a second-generation focused crawler. While most mathematical and algorithmic ingredients that we use in BINGO! (e.g., the classifier, cross-entropy-based feature selection, link analysis for prioritizing URLs in the crawl queue, etc.) are state-of-the-art, the overall system architecture is relatively unique (in the sense that most concepts have been around in the machine learning and information retrieval literature, but have not been considered in an integrated system context). The following are salient features of the BINGO! system:

- As human expert time is scarce and expensive, building the classifier on extensive, high-quality training data is a rare luxury. To overcome the potential deficiencies of the initial training documents, BINGO! uses a simple form of unsupervised, dynamic learning: during a crawl the system periodically identifies the most characteristic documents that have been automatically classified into a topic of interest and considers promoting these class "archetypes" to become new training data.
- The crawl is structured into two phases: a learning phase and a harvesting phase. The first phase performs a limited (mostly depth-first) crawl and uses a conservative tuning of the classifier in order to obtain a richer feature set (i.e., topic-specific terminology) and to find good candidates for archetypes. The second phase then switches to a much more aggressive breadth-first strategy with URL prioritization. Learning aims to calibrate the precision of the classifier, whereas harvesting aims at a high recall.
- BINGO! is designed as a comprehensive and flexible workbench for assisting a portal administrator or a human expert with certain information demands. To this end it includes a local search engine for querying the result documents of a crawl and various other data analysis techniques for postprocessing.

The paper describes the BINGO! architecture and its components, and it demonstrates the system's effectiveness by two kinds of experiments for information portal generation and expert search. The rest

[1]BINGO! stands for <u>B</u>ookmark-<u>In</u>duced <u>G</u>athering <u>o</u>f <u>I</u>nformation

of the paper is organized as follows. Section 2 gives an overview of the system's main concepts, the corresponding software components, and their interplay. When we had built the first prototype based on these concepts and started experimenting, we realized a number of shortcomings regarding the search effectiveness, i.e., the quality of the crawl results, and also efficiency, i.e., speed and resource consumption. These observations led to substantial improvements that are discussed in Sections 3 and 4 on effectiveness and efficiency. Section 5 presents our recent experimental results. We conclude with an outlook on ongoing and future work.

2 Overview of the BINGO! System

The BINGO! crawling toolkit consists of six main components that are depicted in Figure 1: the focused crawler itself, an HTML document analyzer that produces a feature vector for each document, the SVM classifier with its training data, the feature selection as a "noise-reduction" filter for the classifier, the link analysis module as a distiller for topic-specific authorities and hubs, and the training module for the classifier that is invoked for periodic re-training.

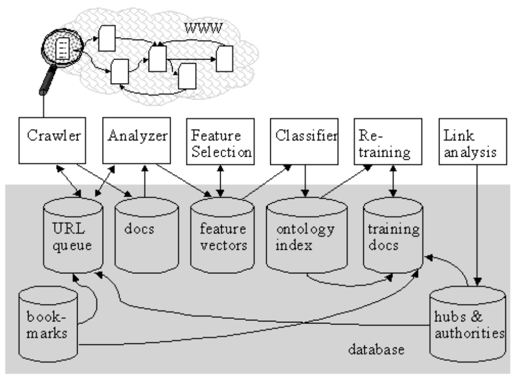

Figure 1: The BINGO! architecture and its main components

The crawler starts from a user's bookmark file or some other form of personalized or community-specific topic directory. These intellectually classified documents serve two purposes: 1) they provide the initial seeds for the crawl (i.e., documents whose outgoing hyperlinks are traversed by the crawler), and 2) they provide the initial contents for the user's topic tree and the initial training data for the classifier. Figure 2 shows an example of such a tree. Note that a single-node tree is a special case for generating an information portal with a single topic and no subclass structure or for answering a specific expert query. In the latter case the training data is a virtual document derived from the user query, and this training basis can be extended by prompting the user for relevance feedback after a short initial crawl and adding appropriate documents.

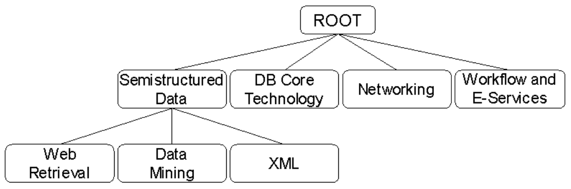

Figure 2: Example of a topic tree

All crawled documents, including the initial data, are stored in an Oracle9i database which serves as a cache. The role of the database system as a storage manager to BINGO! is further discussed in Section 4.

2.1 Crawler

The crawler processes the links in the URL queue using multiple threads. For each retrieved document the crawler initiates some analysis steps that depend on the document's MIME type (e.g., HTML, PDF, etc.) and then invokes the classifier on the resulting feature vector. Once a crawled document has been successfully classified, BINGO! extracts all hyperlinks from the document and adds them to the URL queue for further crawling; the links are ordered by their priority, i.e., SVM confidence in our case.

2.2 Document Analyzer

BINGO! computes document vectors according to the standard bag-of-words model, using stopword elimination, Porter stemming, and $tf * idf$ based term weighting [3, 16]. This is a standard IR approach where term weights capture the term frequency (tf) of the corresponding word stems in the document and the, logarithmically dampened, inverse document frequency (idf) which is the reciprocal of the number of documents in the entire corpus that contain the term. We consider our local document database as an approximation of the corpus for idf computation and recompute it lazily upon each retraining.

The document analyzer can handle a wide range of content handlers for different document formats (in particular, PDF, MS Word, MS PowerPoint etc.) as well as common archive files (zip, gz) and converts the recognized contents into HTML. So these formats can be processed by BINGO! like usual web pages. Many useful kinds of documents (like scientific publications, whitepapers, or commercial product specifications) are published as PDF; incorporating this material improves the crawling recall and the quality of the classifier's training set by a substantial margin.

2.3 Feature Selection

The feature selection algorithm provides the BINGO! engine with the most characteristic features for a given topic; these are the features that are used by the classifier for testing new documents. A good feature for this purpose discriminates competing topics from each other, i.e., those topics that are at the same level of the topic tree. Therefore, feature selection has to be topic-specific; it is invoked for every topic in the tree individually. As an example, consider a directory with topics mathematics, agriculture, and arts, where mathematics has subclasses algebra and stochastics. Obviously, the term *theorem* is very characteristic for math documents and thus an excellent discriminator between mathematics, agriculture, and arts. However, it is of no use at all to discriminate algebra versus stochastics. A term such as *field*, on the other hand, is a good indicator for the topic algebra when the only competing topic is stochastics; however, it is useless for a classifier that tests mathematics versus agriculture.

We use the Mutual Information (MI) measure for topic-specific feature. This technique, which is a specialized case of the notions of cross-entropy or Kullback-Leibler divergence [16], is known as one of the most effective methods [24]. The MI weight of the term X_i in the topic V_j is defined as:

$$MI(X_i, V_j) = P[X_i \wedge V_j] \, log \frac{P[X_i \wedge V_j]}{P[X_i]P[V_j]} \qquad (1)$$

Mutual information can be interpreted as measure of how much the joint distribution of features X_i and topics V_j deviate from a hypothetical distribution in which features and topics are independent of each other (hence the remark about MI being a special case of the Kullback-Leibler divergence which measures the differences between multivariate probability distributions in general).

The result of feature selection for a given topic is a ranking of the features, with the most discriminative features listed first. Our experiments achieved good results with the top 2000 features for each topic as the input to the classifier (compared to tens of thousands of different terms in the original documents). For efficiency BINGO! pre-selects candidates for the best features based on tf values and evaluates MI weights only for the 5000 most frequently occurring terms within each topic. As an example consider the class "Data Mining" in the topic tree of Figure 2 with home pages and DBLP pages of 5 to 10 leading researchers for each topic. Our feature selection finds the following word stems with the highest MI values: *mine, knowledg, olap, frame, pattern, genet, discov, cluster, dataset.*

2.4 Classifier

Document classification consists of a training phase for building a mathematical decision model based on intellectually pre-classified documents, and a decision phase for classifying new, previously unseen documents fetched by the crawler. For training BINGO! builds a topic-specific classifier for each node of the topic tree.

New documents are classified against all topics of the ontology tree in a top-down manner. Starting with the root, which corresponds to the union of the user's topics of interest, we feed the document's features into each of the node-specific decision models (including node-specific feature selection) and invoke the binary classifiers for all topics with the same parent. We refer to these as "competing" topics as the document will eventually be placed in at most one of them. Each of the topic-specific classifiers returns a yes-or-no decision and also a measure of confidence for this decision (see below). We assign the document to the tree node with the highest confidence in a positive decision. Then the classification proceeds with the children of this node, until eventually a leaf node is reached. If none of the topics with the same parent returns yes, the document is assigned to an artificial node labeled 'OTHERS' under the same parent.

Our engine uses support vector machines (SVM) [6, 23] as topic-specific classifiers. We use the linear form of SVM where training amounts to finding a hyperplane in the m-dimensional feature vector space that separates a set of positive training examples (document collection D_i^+ of the topic V_i) from a set of negative examples (document collection D_i^- of all competing topics V with the same parent as V_i) with maximum margin. The hyperplane can be described in the form $\vec{w} \cdot \vec{x} + b = 0$, as illustrated in Figure 3. Computing the hyperplane is equivalent to solving a quadratic optimization problem [23]. The current BINGO! version uses an existing open-source SVM implementation [1].

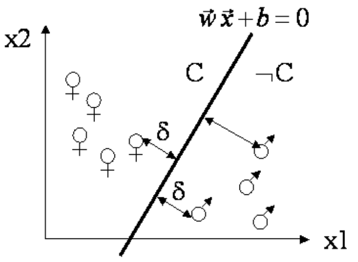

Figure 3: The separating hyperplane of the linear SVM classifier

Note that in the decision phase the SVM classifier is very efficient. For a new, previously unseen, document in the m-dimensional feature space $\vec{d} \in \mathbf{R}^m$ it merely needs to test whether the document lies on the "left" side or the "right" side of the separating hyperplane. The decision simply requires computing an

m-dimensional scalar product of two vectors.

We interpret the distance of a newly classified document from the separating hyperplane as a measure of the classifier's confidence. This computation is an inexpensive byproduct of the classifier. We use this kind of SVM confidence for identifying the most characteristic *"archetypes"* of a given topic. Note that training documents have a confidence score associated with them, too, by simply running them through the classifier's decision model after completed training. Further note that the initial training data does not necessarily yield the best archetypes, in particular, for expert Web search where the initial training data is merely a representation of the user's query terms. Therefore, BINGO! periodically considers promoting the best archetypes to become training documents for retraining the classifier. To estimate the precision of the new classifier, we use the computationally efficient $\xi\alpha$-method [13]. This estimator has approximately the same variance as leave-one-out estimation and slightly underestimates the true precision of the classifier (i.e., is a bit pessimistic). The prediction of the classifier's performance during a crawl is valuable for tuning the feature space construction; we will further discuss this issue in Section 3.

2.5 Link Analysis

The link structure between documents in each topic is an additional source of information about how well they capture the topic [4, 5, 7, 14]. Upon each retraining, we apply the method of [4], a variation of Kleinberg's HITS algorithm, to each topic of the directory. This method aims to identify a set S_A of authorities, which should be Web pages with most significant and/or comprehensive information on the topic, and a set S_H of hubs, which should be the best link collections with pointer to good authorities. The algorithm considers a small part of the hyperlink-induced Web graph $G = (S, E)$ with a node set S in the order of a few hundred or a few thousand documents and a set of edges E with an edge from node p to node q if the document that corresponds to p contains a hyperlink that points to document q. The node set S is constructed in two steps: 1) We include all documents that have been positively classified into the topic under consideration, which form the "base set" in Kleinberg's terminology. 2) We add all successors of these documents (i.e., documents that can be reached along one outgoing edge) and a reasonably sized subset of predecessors (i.e., documents that have a direct hyperlink to a document in the base set). The predecessors can be determined by querying a large unfocused Web database that internally maintains a large fraction of the full Web graph.

The actual computation of hub and authority scores is essentially an iterative approximation of the principal Eigenvectors for two matrices derived from the adjacency matrix of the graph G. Its outcome are two vectors with authority scores and hub scores. We are interested in the top ranked authorities and hubs. The former are perceived as topic-specific archetypes and considered for promotion to training data, and the latter are the best candidates for being crawled next and therefore added to the high-priority end of the crawler's URL queue. These steps are performed with each retraining.

2.6 Learning Phase vs. Harvesting Phase

Building a reasonably precise classifier from a very small set of training data is a very challenging task. Effective learning algorithms for highly heterogeneous environments like the Web would require a much larger training basis, yet human users would rarely be willing to invest hours of intellectual work for putting together a rich document collection that is truly representative of their interest profiles. To address this problem, we distinguish two basic crawl strategies:

- The *learning phase* serves to identify new archetypes and expand the classifier's knowledge base.
- The *harvesting phase* serves to effectively process the user's information demands with improved crawling precision and recall.

Depending on the phase, different focusing rules come into play to tell the crawler when to accept or reject Web pages for addition to the URL queue (see Section 4.2).

In the learning phase we are exclusively interested in gaining a broad knowledge base for the classifier by identifying archetypes for each topic. In many cases such documents can be obtained from the direct neighborhood of the initial training data, assuming that these have been chosen carefully. For example, suppose the user provides us with home pages of researchers from her bookmarks on a specific topic, say data mining; then chances are good that we find a rich source of topic-specific terminology in the vicinity of these home pages, say a conference paper on some data mining issue. i.e., a scientists homepage with links to her topic-specific publications. Following this rationale, BINGO! uses a depth-first crawl strategy during the learning phase, and initially restricts itself to Web pages from the domains that the initial training documents come from.

BINGO! repeatedly initiates re-training of the classifier, when a certain number of documents have been crawled and successfully classified with confidence above a certain threshold. At such points, a new set of training documents is determined for each node of the topic tree. For this purpose, the most characteristic documents of a topic, coined *archetypes*, are determined in two, complementary, ways. First, the link analysis is initiated with the current documents of a topic as its base set. The best authorities of a tree node are regarded as potential archetypes of the node. The second source of topic-specific archetypes builds on the confidence of the classifier's yes-or-no decision for a given node of the ontology tree. Among the automatically classified documents of a topic those documents whose yes decision had the highest confidence measure are selected as potential archetypes. The union of both top authorities and documents with high SVM confidence form a new set of candidates for promotion to training data.

After successfully extending the training basis with additional archetypes, BINGO! retrains all topic-specific classifiers and switches to the harvesting phase now putting emphasis on recall (i.e., collecting as many documents as possible). The crawler is resumed with the best hubs from the link analysis, using a breadth-first strategy that aims to visit as many different sites as possible that are related to the crawl's topics. When the learning phase cannot find sufficient archetypes or when the user wants to confirm archetypes before initiating a long and resource-intensive harvesting crawl, BINGO! includes a user feedback step between learning and harvesting. Here the user can intellectually identify archetypes among the documents found so far and may even trim individual HTML pages to remove irrelevant and potentially dilluting parts (e.g., when a senior researcher's home page is heterogeneous in the sense that it reflects different research topics and only some of them are within the intended focus of the crawl).

3 Making BINGO! Effective

The BINGO! system described so far is a complete focused crawler with a suite of flexible options. When we started experimenting with the system, we observed fairly mixed success, however. In particular, some of the crawls lost their focus and were led astray by inappropriate training data or a bad choice of automatically added archetypes. Based on these lessons we improved the system in a number of ways that are described in this section.

3.1 Classifier Training on Negative Examples

An SVM classifier needs both positive and negative training examples for computing a separating hyperplane. As negative examples we used the positive training data from a topic's competing classes, which are the topic's siblings in the topic tree. For topics without proper siblings, e.g., for a single-topic crawl, we added a virtual child "OTHERS" to all tree nodes which was populated with some arbitrarily chosen documents that were "semantically far away" from all topics of the directory. This approach worked, but in some situations it was not sufficient to cope with the extreme diversity of Web data. In some sense, saying what the crawl should not return is as important as specifying what kind of information we are interested in.

As a consequence of this observation we now populate the virtual "OTHERS" class in a much more systematic manner. As the positive training examples for the various topics all contain ample common-sense vocabulary and not just the specific terms that we are interested in, we included training documents in the "OTHERS" classes that capture as much of the common-sense terminology as possible. In most of our experiments we use about 50 documents from the top-level categories of Yahoo (i.e., sports, entertainment, etc.) for this purpose. Since our focused crawls were mostly interested in scientific topics, this choice of negative examples turned out to be a proper complement to improve the classifier's learning.

3.2 Archetype Selection

The addition of inappropriate archetypes for retraining the classifier was a source of potential diffusion. To avoid the "topic drift" phenomenon, where a few out-of-focus training documents may lead the entire crawl into a wrong thematic direction, we now require that the classification confidence of an archetype must be higher than the mean confidence of the previous training documents. So each iteration effectively adds x new archetypes ($0 \leq x \leq min\{N_{auth}; N_{conf}\}$ where N_{auth} is the number of high-authority candidates from the link analysis and N_{conf} is the number of candidates with top ranks regarding SVM confidence), and it may also remove documents from the training data as the mean confidence of the training data changes. Once the up to $min\{N_{auth}; N_{conf}\}$ archetypes of a topic have been selected, the classifier is re-trained. This step in turn requires invoking the feature selection first. So the effect of re-training is twofold: 1) if the archetypes capture the terminology of the topic better than the original training data (which is our basic premise) then the feature selection procedure can extract better, more discriminative, features for driving the classifier, and 2) the accuracy of the classifiers test whether a new, previously unseen, document belongs to a topic or not is improved using richer (e.g, longer but concise) and more characteristic training documents for building its decision model. In the case of an SVM classifier, the first point means transforming all documents into a clearer feature space, and the second point can be interpreted as constructing a "sharper" (i.e., less blurred) separating hyperplane in the feature space (with more slack on either side of the hyperplane to the accepted or rejected documents).

3.3 Focus Adjustment and Tunnelling

Learning Phase with Sharp Focus

During the learning phase BINGO! runs with a very strict focusing rule. As the system starts only with a relatively small set of seeds, we can expect only low classification confidence with this initial classifier. Therefore, our top priority in this phase is to find new archetypes to augment the training basis. The crawler accepts only documents that are reachable via hyperlinks from the original seeds and are classified into the same topic as the corresponding seeds. We call this strategy *sharp focusing*: for all documents $p, q \in E$ and links $(p, q) \in V$ accept only those links where $class(p) = class(q)$.

The above strategy requires that at least some of the crawled documents are successfully classified into the topic hierarchy; otherwise, the crawler would quickly run out of links to be visited. This negative situation did indeed occur in some of our early experiments when the training data contained no useful links to related Web sources. Therefore, BINGO! also considers links from rejected documents (i.e., documents that do not pass the classification test for a given topic) for further crawling. However, we restrict the depth of traversing links from such documents to a threshold value, typically set to one or two. The rationale behind this threshold is that one often has to "tunnel" through topic-unspecific "welcome" or "table-of-contents" pages before again reaching a thematically relevant document.

Harvesting Phase with Soft Focus

Once the training set has reached $min\{N_{auth}; N_{conf}\}$ documents per topic, BINGO! performs retraining and the harvesting phase is started. The now improved crawling precision allows us to relax the hard focusing rule and to accept all documents that can successfully be classified into anyone of the topics of interest, regardless of whether this is the same class as that of its hyperlink predecessor. We call this strategy *soft focusing*: for all documents $p, q \in E$ and links $(p, q) \in V$ accept all links where $class(p) \neq ROOT/OTHERS$. The harvesting usually has tunneling activated.

3.4 Feature Space Construction

Single terms alone and the resulting $tf * idf$-based document vectors are a very crude characterization of document contents. In addition to this traditional IR approach we are also investigating various richer feature spaces:

- *Term pairs:* The co-occurrence of certain terms in the same document adds to the content characterization and may sometimes even contribute to disambiguating polysems (i.e., words with multiple meanings). The extraction of all possible term pairs in a document is computationally expensive. We use a sliding window technique and determine only pairs within a limited word distance.
- *Neighbor documents:* Sometimes a document's neighborhood, i.e., its predecessors and successors in the hyperlink graph, can help identifying the topic of the document. We consider constructing feature vectors that contain both the current document's terms and the most significant terms of its neighbor documents. This approach is somewhat risky as it may as well dillute the feature space (as reported in [8]); so it is crucial to combine it with conservative (MI based) feature selection.
- *Anchor texts:* The short texts in hyperlink tags of the HTML pages that point to the current document may provide concise descriptions of the target document. However, it is very crucial to use an extended form of stopword elimination on anchor texts (to remove standard phrases such as "click here").

The way we are using the above feature options in BINGO! is by constructing combined feature spaces or by creating multiple alternative classifiers (see next subsection). For example, BINGO! can construct feature vectors that have single-term frequencies, term-pair frequencies, and anchor terms of predecessors as components. For all components feature selection is applied beforehand to capture only the most significant of these features. The classifier can handle the various options that BINGO! supports in a uniform manner: it does not have to know how feature vectors are constructed and what they actually mean. Vectors with up to several thousand components can be handled with acceptable performance.

3.5 Meta Classification

BINGO! can construct a separate classifier (i.e., trained decision model) for each of the various feature space options outlined in the previous section (including combination spaces). Right after training it uses the $\xi\alpha$ estimator [13] for predicting the quality (i.e., classification precision) of each alternative and then selects the one that has the best estimated "generalization performance" for classifying new, previously unseen, documents. The same estimation technique can be used, with some extra computational effort, for choosing an appropriate value for the number of most significant terms or other features that are used to construct the classifier's input vectors after feature selection.

In addition, BINGO! can combine multiple classifiers at run-time using a meta classifier approach. Consider the set $V = \{v_1, \ldots, v_h\}$ of classifiers. Let $res(v_i, D, K) \in \{-1, 1\}$ be the decision of the i-th method for the classification of document D into class C, $w(v_i) \in \mathbf{R}$ be weights and $t_1, t_2 \in \mathbf{R}$ be thresholds. Then we can define a meta decision function as follows:

$$Meta(V, D, C) =$$
$$\begin{cases} +1 \text{ when } \sum_{i=1}^{h} w_i \cdot res(v_i) > t_1 \\ -1 \text{ when } \sum_{i=1}^{h} w_i \cdot res(v_i) < t_2 \\ 0, \text{ otherwise} \end{cases} \quad (2)$$

The zero decision means that the meta classifier is unable to make a clear decision and thus abstains.

Three special instances of the above meta classifier are of particular importance (one of them using the $\xi\alpha$ estimators [13]):

1. *unanimous decision*: for definitively positive classification the results of all classifiers must be equal: as follows:
   $w(v_i) = 1$ for all $v_i \in V$, $\quad t_1 = h - 0.5 = -t_2$
2. *majority decision*: the meta result is the result of the majority of the classifiers:
   $w(v_i) = 1$ for all $v_i \in V$, $\quad t_1 = t_2 = 0$.
3. *weighted average* according to the $\xi\alpha$ estimators:
   $w(v_i) = precision_{\xi\alpha}(v_i)$ for all $v_i \in V, t_1 = t_2 = 0$

Such *model combination and averaging* techniques are well known in the machine learning literature [17]. They typically make learning-based decision functions more robust and can indeed improve the overall classification precision. This observation was also made in some of our experiments where unanimous and weighted average decisions improved precision from values around 80 percent to values above 90 percent. By default, BINGO! uses multiple alternative classifiers in parallel and applies the unanimous-decision meta function in the crawl's learning phase and the weighted average in the harvesting phase. Each of these parallel classifiers requires computing a scalar product between vectors with a few thousand components for each visited Web page that needs to be classified. When the crawler's run-time is critical, we therefore switch to a single feature space and a single classifier, namely, the one with the best $\xi\alpha$ estimator for its precision. This still requires training multiple classifiers, but in this run-time-critical case this is done only once before the harvesting phase is started. For the learning phase we always use the meta classifier.

3.6 Result Postprocessing

The result of a BINGO! crawl may be a database with several million documents. Obviously, the human user needs additional assistance for filtering and analyzing such result sets in order to find the best answers to her information demands. To this end BINGO! includes a local search engine that employs IR and data mining techniques for this kind of postprocessing.

The search engine supports both exact and vague filtering at user-selectable classes of the topic hierarchy, with relevance ranking based on the usual IR metrics such as cosine similarity [3] of term-based document vectors. In addition, it can rank filtered document sets based on the classifier's confidence in the assignment to the corresponding classes, and it can perform the HITS link analysis [14] to compute authority scores and produce a ranking according to these scores. Different ranking schemes can be combined into a linear sum with appropriate weights; this provides flexibility for trial-and-error experimentation by a human expert.

Filtering and ranking alone cannot guarantee that the user finds the requested information. Therefore, when BINGO! is used for expert Web search, our local search engine supports additional interactive feedback. In particular, the user may select additional training documents among the top ranked results that he sees and possibly drops previous training data; then the filtered documents are classified again under the retrained model to improve precision. For information portal generation, a typical problem is that the results in a given class are heterogeneous in the sense that they actually cover multiple topics that are not necessarily closely related. This may result from the diversity and insufficient quality of the original training data.

To help the portal administrator for better organizing the data, BINGO! can perform a cluster analysis on the results of one class and suggest creating new subclasses with tentative labels automatically drawn from the most characteristic terms of these subclasses. The user can experiment with different numbers of clusters, or BINGO! can choose the number of clusters such that an entropy-based cluster impurity measure is minimized [9]. Our current implementation uses the simple $K - means$ algorithm [16, 17] for clustering, but we plan to add more sophisticated algorithms.

4 Making BINGO! Efficient

Our main attention in building BINGO! was on search result quality and the effectiveness of the crawler. When we started with larger-scale experimentation, we realized that we had underestimated the importance of performance and that effectiveness and efficiency are intertwined: the recall of our crawls was severely limited by the poor speed of the crawler. In the last months we focused our efforts on performance improvement and reimplemented the most performance-critical function components.

BINGO! is implemented completely in Java and uses Oracle9i as a storage engine. The database-related components (document analysis, feature selection, etc.) are implemented as stored procedures, the crawler itself runs as a multi-threaded application under the Java VM. As crawl results are stored in the database, we implemented our local search engine as a set of servlets under Apache and the Jserv engine. Our rationale for Java was easy portability, in particular, our student's desire to be independent of the "religious wars" about Windows vs. Linux as the underlying platform.

This section discusses some of the Java- and database-related performance problems and also some of the key techniques for accelerating our crawler. We adopted some useful tips on crawl performance problems from the literature [10, 11, 20] and also developed various additional enhancements.

4.1 Lessons on Database Design and Usage

The initial version of BINGO! used object-relational features of Oracle9i (actually Oracle8i when we started), in particular, *nested tables* for hierarchically organized data. This seemed to be the perfect match for storing documents, as the top-level table, and the corresponding sets of terms and associated statistics as a subordinate table (document texts were stored in a LOB attribute of the top-level table). It turned out, however, that the query optimizer had to compute Cartesian products between the top-level and the subordinate table for certain kinds of queries with selections and projections on both tables. Although this may be a problem of only a specific version of the database system, we decide to drastically simplify the database design and now have a schema with 24 flat relations, and also simplified the SQL queries accordingly.

Crawler threads use separate database connections associated with dedicated database server processes. Each thread batches the storing of new documents and avoids SQL insert commands by first collecting a certain number of documents in workspaces and then invoking the database system's bulk loader for moving the documents into the database. This way the crawler can sustain a throughput of up to ten thousand documents per minute.

4.2 Lessons on Crawl Management

Networking aspects

A key point for an efficient crawler in Java is control over blocking I/O operations. Java provides the convenient `HTTPUrlConnection` class, but the underlying socket connection is hidden from the programmer. Unfortunately, it is impossible to change the default timeout setting; thus, a successfully established but very slow connection cannot be cancelled. The recommended way to overcome this limitation of the Java core libraries is to control the blocking connection using a parallel "watcher thread". To avoid this overhead, BINGO! implements its own socket-based HTTP connections following RFC 822 [18].

The Java core class `InetAddress`, used for the representation of network addresses and resolving of host names, is another potential bottleneck for the crawler [11]. It was observed that the caching algorithm of `InetAddress` is not sufficiently fast for thousands of DNS lookups per minute. To speed up name resolution, we implemented our own asynchronous DNS resolver. This resolver can operate with multiple DNS servers in parallel and resends requests to alternative servers upon timeouts. To reduce the number of DNS server requests, the resolver caches all obtained information (hostnames, IP addresses, and additional hostname aliases) using a limited amount of memory with LRU replacement and TTL-based invalidation.

Since a document may be accessed through different path aliases on the same host (this holds especially for well referenced authorities for compatibility with outdated user bookmarks), the crawler uses several fingerprints to recognize duplicates. The initial step consists of simple URL matching (however, URLs have an average length of more than 50 bytes [2]; our implementation merely compares the hashcode representation of the visited URL, with a small risk of falsely dismissing a new document). In the next step, the crawler checks the combination of returned IP address and path of the resource. Finally, the crawler starts the download and controls the size of the incoming data. We assume that the filesize is a unique value within the same host and consider candidates with previously seen IP/filesize combinations as duplicates. A similar procedure is applied to handle redirects. The redirection information is stored in the database for use in the link analysis (see Section 2.5). We allow multiple redirects up to a pre-defined depth (set to 25 by default).

Document type management

To avoid common crawler traps and incorrect server responses, the maximum length of hostnames is restricted to 255 (RFC 1738 [22] standard), the maximum URL length is restricted to 1000. This reflects the common distribution of URL lengths on the Web [2], disregarding URLs that have GET parameters encoded in them.

To recognize and reject data types that the crawler cannot handle (e.g., video and sound files), the BINGO! engine checks all incoming documents against a list of MIME types [12]. For each MIME type we specify a maximum size allowed by the crawler; these sizes are based on large-scale Google evaluations [2]. The crawler controls both the HTTP response and the real size of the retrieved data and aborts the connection when the size limit is exceeded.

Crawl queue management

The proper URL ordering on the crawl frontier is a key point for a focused crawler. Since the absolute priorities may vary for different topics of interest, the queue manager maintains several queues, one (large) incoming and one (small) outgoing queue for each topic, implemented as Red-Black trees.

The engine controls the sizes of queues and starts the asynchronous DNS resolution for a small number of the best incoming links when the outgoing queue is not sufficiently filled. So expensive DNS lookups are initiated only for promising crawl candidates. Incoming URL queues are limited to 25.000 links, outgoing URL queues to 1000 links, to avoid uncontrolled memory usage.

In all queues, URLs are prioritized based on their SVM confidence scores (see Section 2). The priority of tunnelled links (see 3.3) is reduced by a constant factor for each tunnelling step (i.e., with exponential decay), set to 0.5 in our experiments.

We also learned that a good focused crawler needs to handle crawl *failures*. If the DNS resolution or page download causes a timeout or error, we tag the corresponding host as "slow". For slow hosts the number of retrials is restricted to 3; if the third attempt fails the host is tagged as "bad" and excluded for the rest of the current crawl.

5 Experiments

5.1 Testbed

In the experiments presented here, BINGO! was running on a dual Intel 2GHz server with 4 GB main memory under Win2k, connected to an Oracle9i database server on the same computer. The number of crawler threads was initially restricted to 15; the number of parallel accesses per host was set to 2 and per recognized domain to 5. The engine used 5 DNS servers located on different nodes of our local domain. The maximum number of retrials after timeouts was set to 3. The maximum allowed tunneling distance was set to 2. The allowed size of the URL queues for the crawl frontier was set to 30,000 for each class. To eliminate "meta search capabilities", the domains of major Web search engines (e.g., Google) were explicitly locked for crawling. The feature selection, using the MI criterion, selected the best 2000 features for each topic.

In the following subsections we present two kinds of experiments: 1) the generation of an information portal from a small seed of training documents, and 2) an expert query that does not yield satisfactory results on any of the popular standard search engines such as Google.

5.2 Portal Generation for a Single Topic

To challenge the learning capabilities of our focused crawler, we aimed to gather a large collection of Web pages about *database research*. This single-topic directory was initially populated with only *two* authoritative sources, the home pages of David DeWitt and Jim Gray (actually 3 pages as Gray's page has two frames, which are handled by our crawler as separate documents).

The initial SVM classification model was built using these 2 positive and about 400 negative examples randomly chosen from Yahoo top-level categories such as sports and entertainment (see Section 3). In the learning phase, BINGO! explored the vicinity of the initial seeds and added newly found archetypes to the topic. To this end the maximum crawl depth was set to 4 and the maximum tunnelling distance to 2, and we restricted the crawl of this phase to the domains of the training data (i.e., the CS department of the University of Wisconsin and Microsoft Research, and also additional Yahoo categories for further negative examples). Since we started with extremely small training data, we did not enforce the thresholding scheme (3.2) (requirement that the SVM confidence for new archetypes would have to be higher than the average confidence of the initial seeds). Instead, we rather admitted all positively classified documents (including the ones that were positively classified into the complementary class "OTHERS", i.e., the Yahoo documents). Altogether we obtained 1002 archetypes, many of them being papers (in Word or PDF), talk slides (in Powerpoint or PDF), or project overview pages of the two researchers, and then retrained the classifier with this basis.

Property	**90 minutes**	**12 hours**
Visited URLs	100,209	3,001,982
Stored pages	38,176	992,663
Extracted links	1,029,553	38,393,351
Positively classified	21,432	518,191
Visited hosts	3,857	34,647
Max crawling depth	22	236

Table 1: Crawl summary data

The harvesting phase then performed prioritized breadth-first search with the above training basis and seed URLs, now without any domain limitations (other than excluding popular Web search engines). We paused the crawl after 90 minutes to assess the intermediate results at this point, and then resumed it for a total crawl time of 12 hours. Table 5.2 shows some summary data for this crawl.

To assess the quality of our results we used the DBLP portal (*http://dblp.uni-trier.de/*) as a comparison yardstick. The idea was that we could automatically construct a crude approximation of DBLP's collection of pointers to database researcher homepages. DBLP contains 31,582 authors with explicit homepage URLs (discounting those that have only a URL suggested by an automatic homepage finder). We sorted these authors in descending order of their number of publications (ranging from 258 to 2), and were particularly interested in finding a good fraction of the top ranked authors with BINGO!. To prevent giving BINGO! any conceivably unfair advantage, we locked the DBLP domain and the domains of its 7 official mirrors for our crawler. In evaluating the results, we considered a homepage as "found" if the crawl result contained a Web page "underneath" the home page, i.e., whose URL had the homepage path as a prefix; these were typically publication lists, papers, or CVs. The rationale for this success measure was that it would now be trivial and fast for a human user to navigate upwards to the actual homepage.

We evaluated the recall, i.e., the total number of found DBLP authors, and the precision of the crawl re-

Best crawl results	Top 1000 DBLP	All authors
1,000	27	91
5,000	79	498
all (21,432)	218	1,396

Table 2: BINGO! precision (90 minutes)

Best crawl results	Top 1000 DBLP	All authors
1,000	267	342
5,000	401	1,325
all (518,191)	712	7,101

Table 3: BINGO! precision (12 hours)

sult. For the latter we considered the number of pages found out of the 1000 DBLP-top-ranked researchers, i.e., the ones with the most publications, namely, between 258 and 45 papers. The crawl result was sorted by descending classification confidence for the class "database research", and we compared the top 1000 results to the top 1000 DBLP authors.

Tables 2 and 3 show the most important measures on crawl result quality. Most noteworthy is the good recall: we found 712 of the top 1000 DBLP authors (without ever going through any DBLP page). The precision is not yet as good as we wished it to be: 267 of these top-ranked authors can be found in the 1000 documents with highest classification confidence. So a human user would have to use the local search engine and other data analysis tools to further explore the crawl result, but given that the goal was to automatically build a rich information portal we consider the overall results as very encouraging. Note that our crawler is not intended to be a homepage finder and thus does not use specific heuristics for recognizing homepages (e.g., URL pattern matching, typical HTML annotations in homepages, etc.). This could be easily added for postprocessing the crawl result and would most probably improve precision.

5.3 Expert Web Search

To investigate the abilities of the focused crawler for expert Web search, we studied an example of a "needle-in-a-haystack" type search problem. We used BINGO! to search for *public domain open source implementations of the ARIES recovery algorithm.*

A direct search for "public domain open source ARIES recovery" on a large-scale Web search engine (e.g., Google) or a portal for open source software (e.g., sourceforge.net) does not return anything useful in the top 10 ranks; it would be a nightmare to manually navigate through the numerous links that are contained in these poor matches for further surfing. As an anecdotic remark, the open source software portal even returned lots of results about bin*aries* and libr*aries*.

Our procedure for finding better results was as follows. In a first step, we issued a Google query for "aries recovery method" and "aries recovery algorithm" to retrieve useful and starting points for a focused crawl. The top 10 matches from Google were intellectually inspected by us, and we selected 7 reasonable documents for training; these are listed in Figure 4.

1 http://www.bell-labs.com/topic/books/db-book/slide-dir/Aries.pdf
2 http://www-2.cs.cmu.edu/afs/cs/academic/class/15721-f01/www/lectures/recovery_with_aries.pdf
3 http://icg.harvard.edu/ cs265/lectures/readings/mohan-1992.pdf
4 http://www.cs.brandeis.edu/ liuba/abstracts/mohan.html
5 http://www.almaden.ibm.com/u/mohan/ARIES_Impact.html
6 http://www-db.stanford.edu/dbseminar/Archive/FallY99/mohan-1203.html
7 http://www.vldb.org/conf/1989/P337.PDF

Figure 4: Initial training documents

Note that Mohan's ARIES page (the 5th URL in Figure 4) does not provide an easy answer to the query; of course, it contains many references to ARIES-related papers, systems, and teaching material, but it would take hours to manually surf and inspect a large fraction of them in order to get to the source code of a public domain implementation.

These pages were used to build the initial SVM classification model. As negative examples we again used a set of randomly chosen pages from Yahoo top-level categories such as "sports". The focused crawler was then run for a short period of 10 minutes. It visited about 17,000 URLs with crawling depth between 1 and 7; 2,167 documents were positively classified into the topic "ARIES".

Finally, we used the result postprocessing component (see 3.6) and performed a keyword search filtering with relevance ranking based on cosine similarity. The top-10 result set for the query "source code release" contains links to open-source projects Shore and Minibase, which implement ARIES media recovery algorithm (Figure 5). Additionally, the third open source system, Exodus, is directly referenced by the Shore homepage. A MiniBase page (further up in the directory) was also among the top 10 crawl results according to SVM classification confidence; so even without further filtering the immediate result of the focused crawl would provide a human user with a very good reference.

We emphasize that the expert Web search supported by our focused crawler required merely a minimum amount of human supervision. The human expert had to evaluate only 30 to 40 links (20 for training set selection, and 10 to 20 for result postprocessing), collected into prepared lists with content previews. Including crawling time and evaluation, the overall search cost was about 14 minutes. This overhead is significantly lower than the typical time for manual surfing in the hyperlink vicinity of some initial authorities (such as IBM Almaden).

0.025	**http://www.cs.wisc.edu/shore/doc/** overview/node5.html
0.023	http://www.almaden.ibm.com/cs/ jcentral_press.html
0.022	http://www.almaden.ibm.com/cs/garlic.html
0.021	http://www.cs.brandeis.edu/l̃iuba/ abstracts/greenlaw.html
0.020	http://www.db.fmi.uni-passau.de/k̃ossmann/ papers/garlic.html
0.018	http://www.tivoli.com/products/index/ storage-mgr/platforms.html
0.015	**http://www.cs.wisc.edu/shore/doc/** overview/footnode.html
0.014	http://www.almaden.ibm.com/cs/clio/
0.011	**http://www.cs.wisc.edu/coral/minibase/** logmgr/report/node22.html
0.011	http://www.ceid.upatras.gr/courses/minibase/ minibase-1.0/documentation/html/minibase/ logmgr/report/node22.html

Figure 5: Top 10 results for query "*source code release*"

6 Conclusion

In this paper we have presented the BINGO! system for focused crawling and its applications to information portal generation and expert Web search. Many concepts in BINGO! have been adopted from prior work on Web IR and statistical learning, but we believe that the integration of these techniques into a comprehensive and versatile system like BINGO! is a major step towards a new generation of advanced Web search and information mining tools. The experiments that we presented in this paper have shown the great potential of the focused crawling paradigm but also some remaining difficulties of properly calibrating crawl setups for good recall and high precision.

Our future work aims to integrate BINGO! engine with a Web-service-based portal explorer and a semantically richer set of ontology services. On the other hand, we plan to pursue approaches to generating "semantically" tagged XML documents from the HTML pages that BINGO! crawls and investigate ways of incorporating ranked retrieval of XML data [21] in the result postprocessing or even as a structure- and context-aware filter during a focused crawl.

References

[1] The open-source biojava project. *http://www.biojava.org.*

[2] Google research project. *WebmasterWorld Pub Conference*, 2002.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[4] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. *ACM SIGIR Conference*, 1998.

[5] S. Brin and L. Page. The anatomy of a large scale hyper-textual Web search engine. *7th WWW Conference*, 1998.

[6] C.J.C. Burges. A tutorial on Support Vector Machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *8th WWW Conference*, 1999.

[8] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. *SIGMOD Conference*, 1998.

[9] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* Wiley, 2000.

[10] A. Heydon and M. Najork. Mercator: A scalable, extensible Web crawler. *WWW Conference*, 1999.

[11] A. Heydon and M. Najork. Performance limitations of the Java core libraries. *ACM Java Grande Conference*, 1999.

[12] The Internet Assigned Numbers Authority (IANA). http://www.iana.org.

[13] T. Joachims. Estimating the generalization performance of an SVM efficiently. *European Conference on Machine Learning (ECML)*, 2000.

[14] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5), 1999.

[15] D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. *European Conference on Machine Learning (ECML)*, 1998.

[16] C.D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[17] T. Mitchell. *Machine Learning.* McGraw Hill, 1996.

[18] Hypertext Transfer Protocol. http://www.w3.org/protocols/http/.

[19] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill, 1983.

[20] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed Web crawler. *International Conference on Data Engineering (ICDE)*, 2002.

[21] A. Theobald and G. Weikum. Adding relevance to XML. *3rd International Workshop on the Web and Databases (WebDB)*, 2000.

[22] RFC 1738: Uniform Resource Locators (URL). http://www.w3.org/addressing/rfc1738.txt.

[23] V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

[24] Y. Yang and O. Pedersen. A comparative study on feature selection in text categorization. *International Conference on Machine Learning (ICML)*, 1997.