

Improving Search Precision Using Google Desktop Search 1.0

Leveraging metadata to the max

LAWRENCE REEVE

Google Desktop Search 1.0 (GDS 1.0) is a locally executed service that lets users index and search the contents of their machines (<http://desktop.google.com/>). Google also offers the Google Desktop Search Software Development Kit (SDK) for extending GDS 1.0 with custom plug-ins to index content not natively supported by GDS 1.0 (<http://desktop.google.com/developer.html>). This extensibility offers the possibility of supporting not only new file types, but also extending the search facilities of existing applications. For example, the document searching function of digital libraries such as CiteSeer can be extended to allow author name searches with the goal of improving search precision. Higher precision searches are usually associated with the return of highly relevant documents. Local searching of these digital library resources has other potential advantages beyond increased search precision, including improved search times and reduced strain on server resources. I implemented *gdsCiteSeer*, a GDS 1.0 plug-in for locally indexing CiteSeer metadata, to determine how well search precision could be improved using Google Desktop Search. *gdsCiteSeer* is implemented using C# on the .NET platform, and the complete source code is available electronically (see "Resource Center," page 4).

CiteSeer is a publicly available digital library of scientific papers focused mostly

Lawrence is a full-time software developer and is pursuing a Ph.D. at Drexel University. He can be contacted at larry-reeve@comcast.net.

in the computer and information science areas (<http://citeseer.ist.psu.edu/>). CiteSeer is currently hosted by Penn State's School of Information Sciences and Technology. The accumulated metadata acquired through web crawling is made publicly available. The GDS 1.0 plug-in indexes this metadata to allow for searching by fields rather than by keyword in a general full-text search. The existing document search facility of CiteSeer does not have an advanced form for finding documents using specific fields, such as author name, author affiliation, document title, publication, or related fields. This prevents users from quickly finding documents when the values of these fields are known. For example, a search for documents by a particular author will likely return many non-relevant results, such as documents where an author has been cited within a returned document but is not one of the authors of the document.

CiteSeer Metadata

CiteSeer provides metadata, such as author name and publication date, for the documents it indexes. The metadata can be obtained via a web service protocol or file download. CiteSeer is compliant with the Open Archives Initiative Protocol for Metadata Harvesting (<http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm>), which allows for querying and browsing the available CiteSeer metadata using a web-based protocol.

The other way to acquire CiteSeer metadata is to download a metadata archive file directly from the CiteSeer web site (<http://citeseer.ist.psu.edu/oai.html>). The metadata archive file is available using the Dublin Core fields and Dublin Core fields along with additional metadata fields. For *gdsCiteSeer*, I used the second format because it contains expanded information (such as author affiliation) useful for indexing. One drawback of the metadata archive file is that it is not always synchronized with the latest documents available on the web site. Currently, there are 574,900 documents described in the meta-

data archive file, while the CiteSeer web site reports 723,152 documents available. The metadata archive file contains 5749 document description files in XML format. Each document description file describes 100 documents. Listing One is an XML metadata fragment about one document.

“GDS 1.0 provides a set of nonextensible schemas, called ‘Event Schemas,’ for indexing files”

The *gdsCiteSeer* class *CSFileReader* is responsible for parsing the XML fragments and extracting field values. The .NET *XmlDocument* class is used to load each document description file. A .NET *StreamReader* instance is used to read each file into memory as a single string. Because each document description file contains a list of `<record>` entries, *XmlDocument* fails because there is no parent entry. To get around this, the *XmlDocument* instance is passed a string of XML to load consisting of a `<records>` prefix, the document contents from *StreamReader.ReadToEnd*, and a `</records>` suffix; see Listing Two.

After the document has been loaded, and before it can be parsed, the XML namespaces *oai_citeseer* and *dc* have to be declared using the .NET *XmlNamespaceManager* class; see Listing Three. After the loading and setup steps are completed, it is straightforward to loop through all 100 document descriptions, extracting out the single field values useful for indexing. The author name field is unique because it is multivalued. Extracting a list of authors is done by first getting a list of author nodes using the .NET *XmlNodeList* class, then traversing each node in the list, extracting the author's full name and affiliation.

GDS 1.0 Schema

GDS 1.0 provides a set of nonextensible schemas, called "Event Schemas," for indexing files. The schema sets are organized hierarchically with child schemas inheriting from parent schemas. The base schema, an abstract schema called "Google.Desktop.Indexable," stores data common to all indexing types: document textual content, document length, and document format. Using this base schema, more specific schemas are defined by Google based on the document type (e-mail, instant messaging, and files), which in turn, are decomposed into four types (general file, web page, text file, and media file). Figure 1 shows the complete schema hierarchy. For indexing CiteSeer metadata, the Google.Desktop.WebPage schema is used. Table 1 lists the properties for the WebPage schema.

Indexing

The primary goal for gdsCiteSeer is to index enough data to allow document searches by author name and publication date. A subset of the XML document description fields is used for indexing: *dc:identifier*, *dc:date*, *dc:title*, and *oai_citepeer:author*. The *dc:date* is used for publication date and publication year. The *oai_citepeer:author* field stores author's full names, but for indexing, the full name is split into three fields: first name, last name, and full name. Other CiteSeer fields are also useful for indexing, but are not implemented at this time. For example, author affiliation allows searches for all authors at a particular institution, or for retrieving a particular author's work while they were affiliated with a specified institution. To use author affiliation, however, the field values must be normalized, as institution names are not consistently used throughout all of the collected metadata. In general, fields may need some preprocessing to make them useful for indexing.

The extracted metadata fields are mapped to GDS 1.0 schema property names; see Table 2. The GDS 1.0 *uri* property is set to the URL of the document on CiteSeer. This is done so that in the returned result, GDS 1.0 shows a link to the CiteSeer web page describing the document. None of the documents are stored locally, so a query result needs to allow a user to click on the hyperlink result and be redirected to the document page on the CiteSeer server. The GDS 1.0 *last_modified_time* property is set to the date of publication. GDS 1.0 uses this information in sorting results for display. The GDS 1.0 title property is set to the title of the document. The GDS 1.0 content property is set to an internally constructed text field. This property normally contains a document's full text, which is indexed by GDS 1.0. In gdsCiteSeer, the document's full text is not known, but the metadata is. The goal

is not to search by full text, but to search using CiteSeer metadata. `gdsCiteSeer` constructs its own document contents field, containing encoded metadata information for author name and publication date.

The GDS 1.0 schema currently cannot be extended except by Google. This restricts its usefulness as a general-purpose search facility. Other search libraries, such as Apache Lucene (<http://lucene.apache.org/>), allow for creating and indexing custom fields. Extending the GDS 1.0 schema would allow adding new properties to hold data such as publication month and publication year. To get around this limitation, a method inspired by the Swoogle system (<http://swoogle.umbc.edu/>) is used. Swoogle uses a technique called “swangling” to hash RDF triples into terms that can be indexed by a traditional information retrieval system. For `gdsCiteSeer`, a SHA1 hash is generated for author first name, last name, and full name, as well as publication date and year. The generated hash values are converted to hexadecimal strings. After each field has been hashed, they are joined together, space separated, and stored in the GDS 1.0 *content* property. GDS 1.0 then indexes the hash values as if they were standard terms.

The *FieldBuilder* class (Listing Four) is responsible for constructing fields for indexing and generating the hashes for each field. The *BuildField* method normalizes field values and adds a prefix indicating the field name. Author names are first normalized by removing leading and trailing spaces and lower-casing the value. The publication month and year are left padded with zeros for consistency. Each field is prefixed with its field name: “first:” for author first name, “last:” for author last name, “full:” for author full name, “pubyear:” for publication year, and “pubmonth:” for publication month. The *EncodeField* method computes a hash over the prefix/field value combination, and returns the hash value as a hexadecimal string. The current implementation uses the SHA1 hash, but other hash computations can be used, such as MD5.

GDS 1.0 plug-ins can index documents in two ways. The first is to wait for re-

quests to come in through the GDS 1.0 Crawler service, and the second is to explicitly tell the GDS 1.0 Indexer to index a document. `gdsCiteSeer` uses the second approach. The GDS 1.0 API is Windows COM-based and the GDS 1.0 SDK comes with several examples demonstrating its usage. Because `gdsCiteSeer` is C# based, the .NET COM Interop facility must be used. The C# project needs to set a COM reference to *GoogleDesktopSearchAPILib*.

The C# class *CiteSeerIndexer* is used to handle GDS 1.0 indexing. Before plug-ins can index a document, they must first be registered. This is done by creating an instance of *GoogleDesktopSearchRegisterClass* and calling *RegisterComponent*, passing it a title and a GUID unique to the plug-in. This causes a message box to appear, requesting authorization from users to install the plug-in. Because `gdsCiteSeer` indexes files and then quits, the registration is done after indexing by calling *UnregisterComponent* and passing it the plug-in GUID. Indexing occurs by first creating an instance of *GoogleDesktopSearchClass* and then calling *CreateEvent* with the name of the schema to create. In this case, the schema name is “Google.Desktop.WebPage.” Once an *IGoogleDesktopSearchEvent* is created using *CreateEvent*, the property values in Table 1 are set. Finally, the *Send* method is called to request GDS 1.0 indexing.

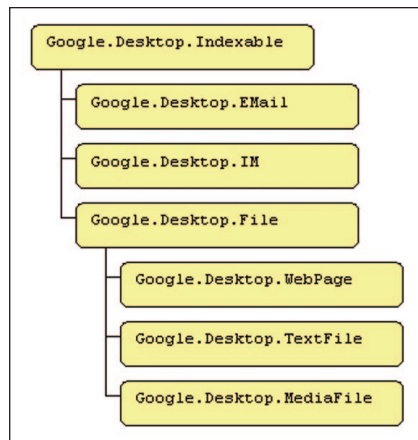


Figure 1: Google Desktop Search 1.0 event schema hierarchy.

Property Name	Description	Required?
content	Text content of page	Yes
format	MIME type of content	Yes
native_size	Size in bytes of the content	No
thumbnail	Small image of content	No
thumbnail_format	MIME type of thumbnail image image/gif, image/jpeg, or image/png	No
uri	The file's URI	Yes
last_modified_time	Date and time the file was last modified	Yes
title	Filename of file	No
author	File's author name	No
bookmarked	Indicates if file has been bookmarked	No
interaction_period	Amount of time user interacted with the web page	No

Table 1: *Google.Desktop.WebPage* schema properties (based on GDS 1.0).

A few issues were found when requesting indexing for all 574,900 documents. The GDS 1.0 Indexing service seems to queue up the indexing requests. The index requests are generally very fast to some point. GDS 1.0 Indexing occasionally stops accepting new requests for some period. These are usually indicated by two exceptions: *System.IO.EndOfStreamException* and

Google Desktop 2.0

Looking to the future, Google recently released its Google Desktop 2.0 beta, which extends indexing through five new schemas and adds (and changes) APIs. The new schemas are: Calendar, Contacts, Tasks, Notes, and Journal. User-defined schemas still are not permitted. Google has extended the API set to include indexing, querying, and displaying.

Some of the indexing API calls were renamed from the Google Desktop 1.0 API. Even though the class and method names may have changed, the parameters remain the same. When registering a new component for indexing documents, the 1.0 *GoogleDesktopSearchRegisterClass* is named “*GoogleDesktopIndexingComponentRegisterClass*” in 2.0. The *RegisterComponent* and *UnregisterComponent* methods of the *GoogleDesktopSearchRegisterClass* were renamed to “*RegisterIndexingComponent*” and “*UnregisterIndexingComponent*,” respectively.

Submitting a document for indexing is done using the same sequence of steps in 1.0 and 2.0. An indexing event is first created using *IGoogleDesktopSearchEvent* in 1.0. In 2.0, the event is called “*IGoogleDesktopEvent*.” Event properties are then set and the indexing event is submitted to Google Desktop. Setting properties and submitting events also remains the same in 1.0 and 2.0. Properties are set using the *Event.AddProperty* method, while indexing events are submitted using the *Event.Send* method.

The base schema *Google.Desktop.Indexable* has been extended with two additional properties: *cookie* and *cookie_raw*. Both properties are optional and are used for including additional information within an indexing event.

Finally, the COM reference library has changed names. The 1.0 library name is “*GoogleDesktopSearchAPILib*.” In 2.0, the library name is “*GoogleDesktopAPILib*.” Because Google Desktop 2.0 is in beta, these changes may not be finalized.

—L.R.

System.Runtime.InteropServices.COMException with the error code set to *S_INDEXING_PAUSED*. In both cases, the result is to wait and try the index request again. *System.IO.EndOfStreamException* generally required shorter waits than *System.Runtime.InteropServices.COMException*. GDS 1.0 Indexing also seems to continuously adjust its indexing priority, perhaps based on system usage. In some cases the indexing happened very rapidly and in other cases, indexing individual documents seems to take a long time. Indexing the approximately 575,000 documents took over 12 hours on a Pentium-4HT-3.2 GHz with 1 GB of memory.

Querying

To query the hashed field values, a plain-text query itself must first be hashed. This is done using a Windows-forms-based custom query builder that converts plain-text field values into hash values, then calls the Google Desktop web interface with the query. Figure 2 shows the query builder interface with an example query. The user

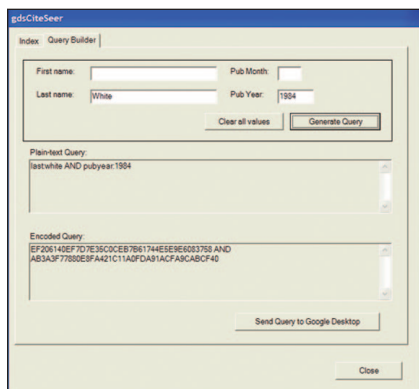


Figure 2: gdsCiteSeer query builder.

CiteSeer Field Name	GDS Property Name
hashed values; see Indexing section for description	content
MIME type 'text/plain'	format
<not used>	native_size
<not used>	thumbnail
<not used>	thumbnail_format
dc:identifier	uri
dc:date	last_modified_time
dc:title	title
<not used>	author
<not used>	bookmarked
<not used>	interaction_period

Table 2: CiteSeer-to-GDS 1.0 data mapping.

CiteSeer/GDS	Query 1	Query 2	Query 3	Avg Precision
n=20	13/20	18/20	10/20	0.68/1.00
n=40	27/40	36/40	17/40	0.67/1.00
n=60	39/60	51/60	26/60	0.64/1.00

Table 3: Precision performance of CiteSeer versus GDS 1.0 (n = number of documents retrieved).

Listing One

```
<record>
  <header>
    <identifier>oai:CiteSeerPSU:1/</identifier>
    <datestamp>1993-08-11</datestamp>
    <setSpec>CiteSeerPSUset</setSpec>
```

```
</header>
<metadata>
  <oai_citeseer:oai_citeseer
    xmlns:oai_citeseer="http://copper.ist.psu.edu/oai/oai_citeseer/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

enters values into the available search fields using plain text, then clicks on the Generate Query button. The plain text in each of the fields is then composed into a plain-text conjunctive query and then an encoded (hashed) form of the query is generated. When users click on the Send Query to Google Desktop button, the encoded query is sent to the local Google Desktop web server. The URL to submit the search is stored in the registry value *HKEY_CURRENT_USER\Software\Google\Google Desktop\APT\search_url*. The encoded query is appended to this URL value, and called using the *Start* method of the *.NET Process* class.

Evaluation

To determine how effective the hashing of field values to simulate schema properties is, the precision of several searches is used. In information retrieval, the common measures of performance are precision and recall used together.

- Recall is defined as Retrieved & Relevant documents/All Relevant documents.
- Precision is defined as Retrieved & Relevant documents/All Retrieved documents.

Recall and Precision are evaluated on a scale of 0 to 1, and each generally shares an inverse relationship with the other. Recall is generally hard to determine with a large document set because all relevant documents must be identified. In this case, gdsCiteSeer is designed to improve precision, so only precision is measured. The precision-at-n-documents method is used, which measures the precision at each set of documents returned, where *n* defines the size of a set of documents. The com-

parison looks at the average precision of a set of queries run against CiteSeer document search and the average precision of the same set of queries using GDS 1.0. Three queries are used:

- All papers by author Peter Lee, an author who appears frequently in CiteSeer.
- All papers by author Jeffrey Ullman, whose name is not common.
- All papers by author John Smith, whose first and last names are both common names.

To increase precision using CiteSeer, the *w* operator is used to specify that terms must co-occur. For example, *peter w/2 lee* is used for the first query. In gdsCiteSeer, the field values are specified in the appropriate text box and the query is generated using the full name field. For example, *full:peter lee*.

Table 3 shows the results of measuring average precision at each page of query results. CiteSeer returns precision around the mid- to high-0.60 range. Using gdsCiteSeer, the precision is 1.00. In general, achieving such high precision values is cause for suspicion. However, gdsCiteSeer is using fielded search, similar to a database query, where the precision is expected to be very high. This is in contrast to CiteSeer, which is using a full-text search and trying to find papers by a particular author.

Conclusion

gdsCiteSeer shows that by acquiring metadata from external sources, and indexing it locally, higher precision searching can result. The main impediment to achieving this, using GDS 1.0, is the lack of an extensible schema. However, alternative methods, such as value hashing, can be used to generate indexable terms not only for GDS 1.0 but other desktop search engines that do not have a fielded query capability. There are a few drawbacks to using the hashing approach. Data must be normalized. For example, names need to be decomposed into first and last fields. This was done using a split function based on whitespace. This is not always accurate. Also, only exact matches are allowed, so a full-name query for "Bob Smith" misses names indexed as "Bob I. Smith." Even with these limitations, gdsCiteSeer shows that methods such as value hashing can improve search precision until extensible schemas are added to Google Desktop Search.

DDJ

```

xsi:schemaLocation="http://copper.ist.psu.edu/oai/oai_citeseer/
http://copper.ist.psu.edu/oai/oai_citeseer.xsd
<dc:title>36 Problems for Semantic InterpretatiOn</dc:title>
<oai_citeseer:author name="Gabriele Scheler">
  <address>80290 Munchen I Germany</address>
  <affiliation>Institut fur Informatik;
    Technische Universitat Munchen</affiliation>
</oai_citeseer:author>
<dc:subject>Gabriele Scheler 36 Problems
  For Semantic Interpretation</dc:subject>
<dc:description>This paper presents...</dc:description>
<dc:contributor>The Pennsylvania State University CiteSeer
  Archives</dc:contributor>
<dc:publisher>unknown</dc:publisher>
<dc:date>1993-08-11</dc:date>
<dc:format>ps</dc:format>
<dc:identifier>http://citeseer.ist.psu.edu/1.html</dc:identifier>
<dc:source>ftp://flop.informatik.tu-muenchen.de/
  pub/fki/fki-179-93.ps.gz</dc:source>
<dc:language>en</dc:language> /
<dc:rights>unrestricted</dc:rights>
</oai_citeseer:oai_citeseer>
</metadata>
</record>

```

Listing Two

```

StreamReader xmlStream = new StreamReader(filename);
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.LoadXml("<records>" + xmlStream.ReadToEnd() + "</records>");

```

Listing Three

```

XmlNamespaceManager nsmgr = new XmlNamespaceManager(xmlDoc.NameTable);
nsmgr.AddNamespace("oai_citeseer", "http://copper.ist.psu.edu/
  oai/oai_citeseer/");
nsmgr.AddNamespace("dc", "http://purl.org/dc/elements/1.1/");

```

Listing Four

```

using System;
using System.Security.Cryptography;
using System.Text;

namespace gdsCiteSeer
{
  public class FieldBuilder
  {
    public enum EFieldNames
    {
      FirstName,
      LastName,
      FullName,
      PubMonth,
      PubYear
    }
    public static string EncodeField(string field)
    {
      if (field == null || field.Length == 0)
        return String.Empty;
      StringBuilder encodedField = new StringBuilder();
      byte[] dataToEncode = Encoding.UTF8.GetBytes(field);
      SHA1 hasher = new SHA1CryptoServiceProvider();
      //MD5 hasher = new MD5CryptoServiceProvider();
      byte [] hashResult = hasher.ComputeHash(dataToEncode);

      // Note: BitConverter.ToString separates hex values with dashes
      for (int idx=0; idx < hashResult.Length; idx++)
        encodedField.Append(hashResult[idx].ToString("X2"));
      return encodedField.ToString();
    }
    public static string BuildField(
      EFieldNames fieldName,
      string fieldValue)
    {
      string fieldValueReturned = null;
      switch(fieldName)
      {
        case EFieldNames.FirstName:
          fieldValueReturned = fieldValue.Trim().ToLower();
          return "first:" + fieldValueReturned;
        case EFieldNames.LastName:
          fieldValueReturned = fieldValue.Trim().ToLower();
          return "last:" + fieldValueReturned;
        case EFieldNames.FullName:
          fieldValueReturned = fieldValue.Trim().ToLower();
          return "full:" + fieldValueReturned;
        case EFieldNames.PubMonth:
          fieldValueReturned = fieldValue.Trim();
          if (fieldValueReturned.Length > 2)
            fieldValueReturned =
              fieldValueReturned.Substring(0, 2);
          else if (fieldValueReturned.Length < 2)
            fieldValueReturned =
              fieldValueReturned.PadLeft(2, '0');
          return "pubmonth:" + fieldValueReturned;
        case EFieldNames.PubYear:
          fieldValueReturned = fieldValue.Trim();
          if (fieldValueReturned.Length > 4)
            fieldValueReturned =
              fieldValueReturned.Substring(0, 4);
          else if (fieldValueReturned.Length < 4)
            fieldValueReturned =
              fieldValueReturned.PadLeft(4, '0');
          return "pubyear:" + fieldValueReturned;
      }
      return String.Empty;
    }
  }
}

```

DDJ