27

GTP (General Text Parser) Software for Text Mining

Justin T. Giles, Ling Wo, Michael W. Berry
University of Tennessee, Knoxville, USA

CONTENTS

27.1 Introduction 457
27.2 Model Facilitated by GTP 458
27.3 GTP Usage and Files Generated 459
27.4 Overview of GTP Options 460
27.5 Query Processing with GTPQUERY 466
27.6 Example 466
27.7 Versions of GTP and GTPQUERY 471
27.8 Code Evolution 472
27.9 Future Work 472
Acknowledgements 472
References 472

Because of the seemingly transparent nature of search engine design and use, there is a tendency to forget the decisions and tradeoffs constantly made throughout the design process, which ultimately affect the performance of any information retrieval (IR) system. One of the major decisions is selecting and implementing any underlying computational model within a single (but integrated) software environment. We present the latest release an object-oriented (C++ and Java) software environment called GTP (or General Text Parser) which can be used by both novice and experts in information modeling to (i) parse text (single files or directories), (ii) construct sparse matrix data structures (with choices of different term weighting strategies), (iii) perform selected matrix decompositions for the representation of terms, documents, and queries in a reduced-rank vector space, and (iv) convert user-supplied natural language queries into appropriate query vectors for cosine-based matching against term and/or document vectors in that vector space.

27.1 Introduction

With the enormous growth in digital text-based information, the efficiency and accuracy of search engines is a growing concern. In order to improve performance,

0-8493-0052-5/00/$0.00+$.50

novices and experts in information modeling need ways to study and evaluate various information retrieval (IR) models. Limitations exist in large part due to the absence of publicly available software capable of facilitating such modeling. To address this void in modeling software, we present an object oriented software environment called General Text Parser (GTP). GTP is public domain software, which is freely available for anyone to download*.

GTP provides users with the ability to parse ASCII text as well as other forms of text (PostScript, PDF, etc.) via user-generated filters. For the more advanced user, GTP is equipped to construct sparse matrix data structures (based on different term weighting choices) and then perform matrix decompositions for the representation of terms, documents, and queries in a reduced-rank vector space. The underlying IR model facilitated by GTP is latent semantic indexing (or LSI) [3, 5], although with little effort by the user, the LSI-specific modules may be removed and replaced by an alternative model. Finally, GTP is capable of converting user-supplied natural language queries, through the GTPQUERY module, into appropriate query vectors for cosine-based matching against term and/or document vectors in the reduced-rank vector space generated.

This paper does not discuss in great detail the theoretical development of vector-space modeling and LSI (see [3, 4]). Section 27.2 provides a brief overview of LSI in order to facilitate understanding of the processes of GTP. The general flow of GTP is discussed in Section 27.3 with a follow-up section covering the details of the several command-line arguments of GTP. A new addition to the GTP package is GTPQUERY (Section 27.5), which is used for query processing. An example run of GTP and GTPQUERY is illustrated in Section 27.6, and Section 27.7 presents the various versions of GTP included the most recent Java release. Section 27.8 briefly discusses the code evolution from Fortran to Java. Finally, Section 27.9 presents future improvements to be made to GTP and GTPQUERY.

27.2 Model Facilitated by GTP

GTP is a software package that provides text parsing of small to large document collections and matrix decomposition for use in information retrieval applications. GTP has the ability to parse any document (using tag filters) and produce a list of keys (or words that have been extracted by the parser). Beyond being a simple text parser, GTP uses a vector-space approach to text modeling [6]. In using the vector-space approach, the documents and queries are represented as vectors of the terms parsed, basically creating a term-by-document matrix. The elements of this matrix are frequencies (or weighted frequencies) of terms (rows) with respect to their corresponding documents (columns).

*http://www.cs.utk.edu/∼lsi

The specific vector-space model exploited by GTP is latent semantic indexing (LSI). LSI expands on the general vector-space approach by exploiting a low-rank approximation of the term-by-document matrix [3, 6, 7]. LSI analyzes the global patterns of terms and word usage, thus allowing documents with dissimilar terms to have closely related vectors. To achieve this, LSI performs singular value decomposition (SVD) on the term-by-document matrix [3], whose nonzero elements may be weighted term frequencies. Term weighting is often used when performing the SVD. Using term weights, instead of raw frequencies, may increase or decrease the importance of terms (Section 27.4). Thus allowing the meaning of a document to rise to the surface, instead of having the raw frequencies determine content.

27.3 GTP Usage and Files Generated

GTP is executed by the command *gtp* followed by a sequence of command line arguments. The basic command looks like the following

gtp filename –c common_words_file –t temp_dir

where *filename* is a file or directory of documents to be parsed, *common_words_file* is the file containing the common words to be ignored, and *temp_dir* is a directory where temporary working files are stored. When this command sequence is used, GTP traverses the directory structure(s) and inserts the files found into a red-black tree. GTP then maneuvers through the red-black tree and parses each document into keys. The only items generated by GTP at this point are the *keys* database, *RUN_SUMMARY* /*LAST_RUN* (Figure 27.1), and uncompressed *rawmatrix* (Figure 27.2) files (see Figure 27.3 for the general flow of GTP processing and Table 1 for a list of frequently generated files).

With the use of more options, GTP may produce several other files in varying formats. The *output* file, created by specifying the $-O$ option, is a binary file that contains all the vector (term and document) and singular value information produced by the SVD. The layout of the output file is as follows: header information consisting of the number of terms parsed, number of documents encountered, and the number of factors used; term vectors for all terms parsed; document vectors for all documents parsed; singular values computed. A Perl script called *readOutput* is provided with the software package, which displays the data in the output file in ASCII format.

The keys (*keys.pag* and *keys.dir* in C++; *keys* in Java) file is a database (dbm) of all the terms that were parsed in the GTP run. Stored with the terms is the term id that is a numeric value starting at 1 and incremented by 1 for each new term that is parsed. So, the first term parsed has an id of 1. The next new term encountered has an id of 2, and so on. The final information stored with the term and its id is the global weight of the term. The global weight is calculated based on options specified by the user, and takes into account all occurrences of the term in all the documents

File name	Type	Description (Section)
RUN_SUMMARY	ASCII	Summary of options used (27.3)
LAST_RUN	ASCII	Summary of options used on most recent GTP run (27.3,27.5)
keys.dir/keys.pag - C++ *keys - Java*	DBM	Database of keys generated (27.3, 27.5)
Output	Binary	Vector information generated by SVD (27.3,27.5,27.6)
rawmatrix/rawmatrix.Z	ASCII/ Compressed	Raw term-by-document matrix (27.3)
matrix.hb/matrix.hb.Z	ASCII/ Compressed	Contains the Harwell-Boeing compressed matrix. (27.3)
TITLES	ASCII	List of document titles. (Table 2)
lao2	ASCII	Summary of SVD calculation.
larv2	Binary	File of SVD vectors. Use the *readVect* script for viewing.
lalv2	Binary	File of SVD vectors. Use the *readVect* script for viewing.
nonz	ASCII	Total terms in the document collection, including those not parsed.

TABLE 27.1
Table 1: List of common files that GTP generates. A brief description of each file is given along with a pointer (reference) to the section describing the file.

being parsed, not just a single document. Under Solaris and Linux, the database that is used is Berkeley 4.3 BSD. Under Java, the World Wide Web Consortium (W3C) Jigsaw database package is used.

The *rawmatrix* or *rawmatrix.Z* file contains the term-by-document matrix in raw format (Figure 27.2). This file has the format of document number followed by term id/weight pairs for each term in the document. For large document collections, this file could be large, for this reason GTP compresses the *rawmatrix* by default. *matrix.hb* is a file that holds the sparse term-by-document matrix in Harwell-Boeing compressed format. The Harwell-Boeing format is also known as compressed column storage (CCS). CCS stores all nonzero elements (or frequencies) of a sparse term-by-document matrix by traversing each column. Stored with the frequencies are row and column pointers [3]. By default the *matrix.hb* file is compressed.

27.4 Overview of GTP Options

As GTP has grown into the sizable program (over 13,000 lines of code), it has increased in the number of command line arguments to an almost unbearable point. Even expert users have a difficult time keeping all the options straight. A wordy dialogue about each option would be as unbearable as the number of options available.

```
#
# Variables set when keys database created (i.e. constants):
#
extra_chars =
numbers = false
df = 1
gf = 1
local = tf
global = (null)
stop_file = ../../etc/common_words
nterms = 161
ndocs = 7
ncommon = 1000
minlen = 2
maxlen = 25
maxline = 10000
normal_doc = no normalization
delimiter = end of file only
log_file = none
temp_dir = /a/berry/bay/homes/giles/linuxgtp_orig/src3.1/run/tmp
keephb = true
keepkeys = false
decomposition= false
filter(s) =
../../filters/blfilter
../../filters/filter
#
# Variables that can be changed when SVD is run
#
run_name = Test Run 1 of gtp
file_dir = /a/berry/bay/homes/giles/linuxgtp_orig/src3.1/sample
nfact = 0
##
##Keys database created Sat Jul 13 16:19:12 EDT 2002
```

FIGURE 27.1
Example of the *RUN_SUMMARY* and *LAST_RUN* files. These files log all options the user requested at runtime. The *LAST_RUN* file only contains options relevant to the last run of GTP.

```
7 161
doc 1:
123 1.0000 80 3.0000 10 1.0000 30 1.0000 150 1.0000 144 1.0000 83 2.0000 89
1.0000 24 1.0000 53 1.0000 62 1.0000 9 1.0000 25 1.0000 60 2.0000 64 2.0000
51 1.0000 142 1.0000 110 1.0000 34 1.0000 140 1.0000 54 1.0000 159 1.0000
147 2.0000 66 1.0000 118 1.0000 155 1.0000 161 1.0000 129 1.0000 49 1.0000
153 1.0000 47 1.0000
doc2:
```

FIGURE 27.2
Example of the *rawmatrix* file, in which each record is composed of the document number followed by term id/weight pairs of terms that exist in that document.

With that in mind, a table has been created to explain each option and its dependencies on other options in the command line (Table 2).

A few of the options deserve more explanation than what the table provides. The *-d* and *-g* options can be a bit confusing to the new user. Both of these options determine the threshold of the local and global raw frequencies of terms. If the *-d* option is set at 2, this means that before a term is included in the keys database, it must occur more than twice in the document. Consequently, if the value is set at 0, then there is no minimum requirement for the frequency of a term in any particular document. The *-g* option acts in a similar fashion, but reflects the minimum threshold for the global frequency of a term. If the *-g* option is set at 2 then a term must occur more than twice in an entire document collection before it is included in the keys database. As with the *-d* option, if the value is set at 0, then there is no minimum frequency requirement. For small document collections, it is strongly advised to set both the local (*-d*) and global (*-g*) thresholds to 0 in order to capture all useful terms.

Another option that can be confusing to a new user is the *-w* option, which involves different term weighting schemes. Term weighting is especially important for vector space IR models (such as LSI). The importance of terms may increase or decrease depending on the type of weighting that is used. In determining which local weighting to use, the vocabulary and word usage patterns of the document need to be considered [3]. If the collection spans general topics (news feeds, magazine articles, etc.), using raw term frequency (tf) would suffice. If the collection were small in nature with few terms in the vocabulary, then binary frequencies would be the best to use.

In determining the global weight, the likelihood that the collection will change needs to be considered. If a collection is static, the inverse document frequency (idf) is a good choice [3]. In general, the document collection will work well with some weighting schemes and poorly with others. Any knowledge of the term distribution associated with a document collection before running GTP could be extremely helpful in determining which weighting scheme to use.

Arg	Additional Args.	Description	Dependencies
-help		Summarize options.	
-q		Suppress progress summary.	
-h		Create the Harwell-Boeing compressed matrix. Default is to not create it.	Required if using *-z* option.
-u		Keep the Harwell-Boeing compressed matrix in an uncompressed file (on output) if the matrix is created.	*-h*
-N		Include numbers as keys.	
-D		Do not create Unix compatible dbm key files (keys.dir/keys.pag in C++; keys in Java). Default is to generate them.	Do not use if you are to perform queries.
-K		Keep the keys file created in the temporary directory specified by the "-t temp_dir" argument.	
-T		Consider the first line of each document (up to 200 characters) to be the title of the document. Before this line is parsed, it will be written to the file *TITLES* in the current directory. Each title line in this file will exist on it's own line.	
-s		Normalize the document length. This ensures a unit length for columns in the term-by-document matrix.	
-m	*Int*	Set a new minimum key length of *int* for the parser.	
	Ex: *-m 5*	The default minimum length is 2.	
-M	*Int*	Set a new maximum key length of *int* for the parser.	
	Ex: *-M 35*	The default maximum length is 20.	

TABLE 27.2
Table 2: List of command-line options for GTP. Column 1 shows the argument. Column 2 shows additional options for the specified argument as well as examples. Column 3 gives a brief description of each argument. Column 4 shows which other arguments the current one depends on.

Table 2 continued...

-L	*Int* Ex: *-L 1000*	Specify a new maximum line length of *int*. If any record being parsed exceeds *int* characters, in length before a carriage return/line feed character, the user is informed of this and the portion of the record that caused the overrun is printed to the screen. The default maximum is 10,000.	
-S	*Int* Ex: *-S 2000*	Set the maximum number of common words to use to *int*. The default value is 1,000.	
-d	*Int* Ex1: *-d 0* Ex2: *-d 2*	Change the threshold for document frequency of any term to *int*. Default is 1.	
-g	*Int* Ex1: *-g 0* Ex2: *-g 4*	Change the threshold for global frequency of any term to *int*. Default is 1.	
-e	“Extra_characters” Ex: *“()$%”*	Specify a string of characters, each of which will be considered a valid character, in addition to all default characters, when tokenizing keys.	
-f	*Filter1 [opts] [filter2 [opts] ... filterN [opts]]* Ex1: *-f zcat html_filter* Ex2: *-f zcat “grep computer”*	Specify filters to pass each file through before the parser looks at it. If a filter has options, it needs to be surrounded by quotes.	At least one filter must be specified if *-f* is used.
-o	*Filename* Ex: *-o keystable*	Specify that the key, id# global frequency, document frequency, and weight of all keys are to be written to “filename”.	

-B	*New_delimiter* Ex: *-B /newdoc/*	Specify that a new document delimiter is needed. *New_delimiter* must be alone on a line in the file and must match exactly for GTP to recognize it. It can be up to 198 characters. Default is a blank line.	Cannot be used if *-x* is being used.
-x		Indicate that there is to be no delimiter other than the end of file.	Cannot be used if *-B* is being used.
-w	*Local global* Ex: *-w log entropy* or Ex: *-w entropy log*	Specify a custom weighting scheme. Local and global refer to local and global weighting formulas. Local can be tf (term frequency), log, or binary. Global can be normal, idf, idf2, or entropy. Default local is tr and global is not calculated.	
-R	*Run_name* Ex: *-R "Medical Docs"*	Specify a name for the current run of GTP.	
-z	*sdd* *rank inner_loop_criteria tolerance*	Performs semi-discrete decomposition. [3]	Cannot use if using *-z svd1 ...*; *-h*
-z	*svd1* *desc* *lanmax* *maxprs*	Performance singular value decomposition. [3, 5, 6]	Cannot use if using *-z sdd ...*; *-h*
-O		Specify that the *output* file is to be in one binary file for SVD. This is needed if you are going to use GTPQUERY.	*-z sdv1 ...*
-Z		Specify if parse procedure should be skipped so that an available matrix can be decomposed via SVD or SDD.	*-h* *-z svd1 ...* or *-z sdd ...*

27.5 Query Processing with GTPQUERY

GTPQUERY relies completely on files produced by a standard GTP run. Those files are *output*, *keys* (or *keys.pag* and *keys.dir* in C++; *keys* in Java), and LAST_RUN (Section 27.3). Query processing will fail if these files are not in the working directory.

Query processing is performed by finding a cosine similarity measure between a query vector and document vectors. Query vectors are generated by summing the term vectors of terms in the query (the term vectors are generated in GTP) then scaling each term vector dimension by the inverse of a corresponding singular value [3, 5]. In this way, the query vector can be considered a *pseudo*-document. In other words, the query vector mimics a document vector and may be projected into the term-document space [7]. At this point, if the user opted to scale the query vector it is done using the singular values produced by GTP. The scaling is done to emphasize the more dominant LSI factors (or dimensions). Cosine similarity is then calculated between the query vector and document vectors and the process is repeated if there are more queries. Options for GTPQUERY are provided to assist in query parsing and how the results are presented (Table 3).

The files that GTPQUERY produces are strictly results files. Each file has a prefix of *q_result.#*, where # is a number starting with 1. The number represents the corresponding number id of the query that was performed. The file contains a list of document ids and cosine similarity measures organized by most relevant to least relevant (Figure 27.4).

27.6 Example

To illustrate the process of running GTP and GTPQUERY, a small example is provided.

1. The document collection to be parsed is one file with each document separated by a blank line.

```
Numerical Libraries and The Grid

The Semantic Conference
Organizer Software

GTP: Software for Text Mining
```

Arg	Add. Args.	Description
-help		Summarize options.
-S		Scale the query vector by the singular values before calculating cosine similarity.
-n	*Int* Ex: *-n 15*	Set the number of factors to use. Default is the value of *nfact* found in in *LAST_RUN* file generated by GTP.
-u	*Float* Ex: *-u 0.75*	Set the upper threshold value for query results returned. If the upper threshold is set to 0.75, then all query results returned will be equal to or less than 0.75 (default is 1).
-l	*Float* Ex: *-l 0.25*	Set the lower threshold value for query results returned. If the upper threshold is set to 0.25, then all query results returned will be greater than or equal to 0.25 (default is -1).
-k	*Int* Ex: *-k 20*	Set the number of results returned to *int* (default is all).
-f	*Filter1 [opts] [filter2 [opts] ... filterN [opts]]* Ex1: *-f zcat html_filter* Ex2: *-f zcat "grep computer"*	Specify filters to pass each file through before the parser looks at it. If a filter has options, it needs to be surrounded by quotes.
-B	*New_delimiter* Ex: *-B /newquery/*	Specify that a new query delimiter is needed. *New_delimiter* must be alone on a line in the file and must match exactly for the query processing to recognize it. It can be up to 198 characters, and the default delimiter is a single blank line. Cannot be used in conjunction with the *-x* option.
-x		Indicate that there is to be no delimiter other than the end of file. This cannot be used in conjunction with the *-B* option.

TABLE 27.3

Table 3: List of command-line options for GTPQUERY. Column 1 shows the arguments. Column 2 includes any additional arguments to be added to the root argument. Column 3 displays a brief description of each argument. There are no dependencies for each argument.

2. The GTP command is invoked using the command below. GTP provides the user with messages regarding what it is currently processing.

```
gtp ../sample -c ./common_words -t ./tmp -h -z svd1 sample -d 0 -g 0 -O -w log
entropy

Creating keys — Mon Jun 24 16:45:51 EDT 2002
Calculating term weights — Mon Jun 24 16:45:51 EDT 2002
Creating raw matrix — Mon Jun 24 16:45:51 EDT 2002
Creating Harwell-Boeing matrix — Mon Jun 24 16:45:51 EDT 2002
matrix size: 3 x 10 nelem: 11
Decompose Harwell-Boeing matrix — Mon Jun 24 16:45:51 EDT 2002
Using svd method – las2 Mon Jun 24 16:45:51 EDT 2002
time for gtp is 0.000000
time for decomposition is 0.000000
Total time is 0.000000
Writing summary to RUN_SUMMARY — Mon Jun 24 16:45:51 EDT 2002
```

3 Once GTP has terminated, the *keys* and binary *output* files will have been created. Those generated by this example may be seen below: **Keys file:**

Key	ID	Global Weight
Conference	1	1.000000
Grid	2	1.000000
Gtp	3	1.000000
Libraries	4	1.000000
Mining	5	1.000000
Numerical	6	1.000000
Organizer	7	1.000000
Semantic	8	1.000000
Software	9	0.369070
Text	10	1.000000

Output file (ASCII equivalent created using *readOutput* on binary file produced):

```
Header
terms = 10
docs = 3
factors = 3
commentSize = 20
updatedTerms = 0 * not activated *
updatedDocs = 0 * not activated *
comment = Creating output file
```

```
Output file continued ...
Term Vectors
Term 0        (conference)
  0   0.3908561766
  1   -0.3573666215
  2   -0.1973721683

Term 9        (text)
  0   0.3908561766
  1   0.3573666215
  2   0.1973721683
```

```
Document Vectors
Document 0 (Numerical Libraries and the Grid)
  0   0.0000000000
  1   -0.4834610820
  2   0.8753658533
  ...
Document 2 (GTP: Software for Text Mining)
  0   0.7071067691
  1   0.6189771295
  2   0.3418586254

Singular Values
  0   1.2537220716
  1   1.2003111839
  2   1.2003111839
```

4 A query may now be performed using the above *keys* and *output* files. The queries used for this example are two queries in a single file, *queryfile*, delimited by a single blank line.

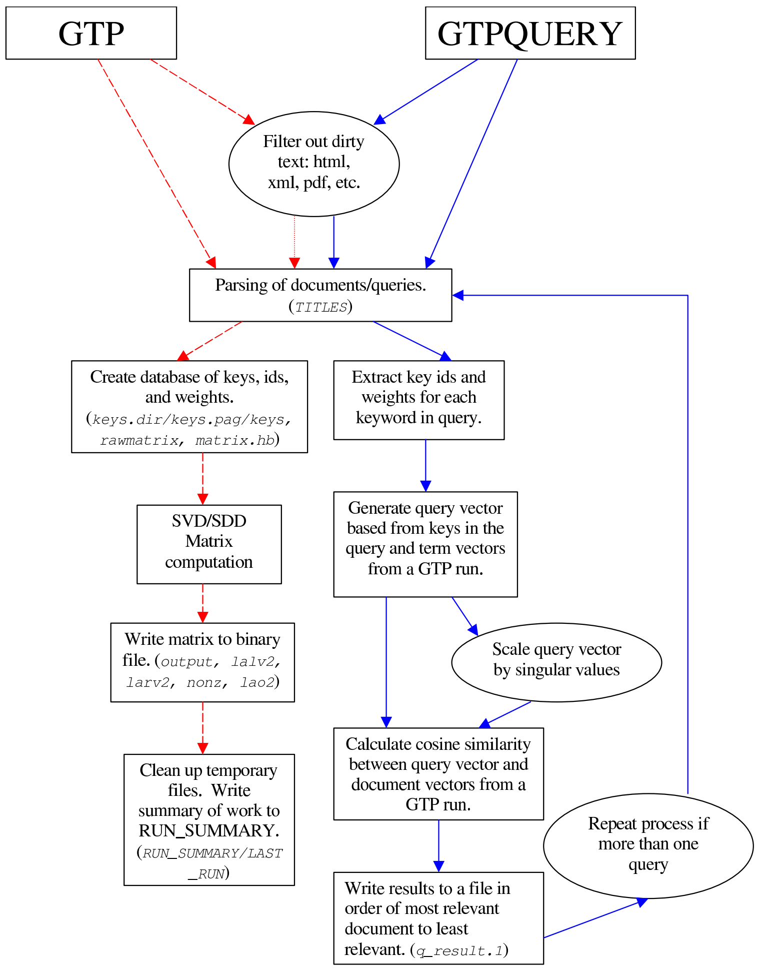
```
Numerical Software

Text Mining Software
```

5 The command used for the query processing exploits scaling by the singular values as the only optional command line argument.

```
gtpquery queryfile -c ./common_words -S

Query 1 done.
Query 2 done.
```



GTP
GTPQUERY
Filter out dirty text: html, xml, pdf, etc.
Parsing of documents/queries. (TITLES)
Create database of keys, ids, and weights. (keys.dir/keys.pag/keys, rawmatrix, matrix.hb)
Extract key ids and weights for each keyword in query.
SVD/SDD Matrix computation
Generate query vector based from keys in the query and term vectors from a GTP run.
Write matrix to binary file. (output, lalv2, larv2, nonz, lao2)
Scale query vector by singular values
Calculate cosine similarity between query vector and document vectors from a GTP run.
Clean up temporary files. Write summary of work to RUN_SUMMARY. (RUN_SUMMARY/LAST_RUN)
Repeat process if more than one query
Write results to a file in order of most relevant document to least relevant. (q_result.1)

FIGURE 27.3
Flowchart of GTP and GTPQUERY. Files produced are listed in parentheses.

Doc ID	Cos. Similarity
50	0.975631
2	0.756432
14	0.500123
...	...

FIGURE 27.4
Query results file example.

6 Once the query processing is done, a file for each query processed will have been generated. In this case, the files would be named *q_result.1* and *q_result.2*. Each file is shown below (only numerical values would be written, items in parentheses are not included in the results files).

q_result.1 (Numerical Software)			**q_result.2** (Text Mining Software)		
1	0.755929	(Numerical Libraries and the Grid)	3	1	(GTP: Software for Text Mining)
3	0.654654	(GTP: Software for Text Mining)	1	5.04984e-17	(Numerical Libraries and the Grid)
2	5.27521e-09	(The Semantic Conference Organizer Software)	2	-1.40715e-16	(The Semantic Conference Organizer Software)

This example illustrates using GTP as a means to parse documents and generate factors using SVD. After running GTP, the example performed a query using files generated by GTP. As stated before, GTP may be used in many other ways besides parsing and performing decomposition. One may strictly use GTP as a means to parse documents and generate a database of keys and frequencies. Titles of documents (first line of the document) may also be extracted.

27.7 Versions of GTP and GTPQUERY

There are several different versions of GTP that are supported. For Solaris and Linux based machines there is a C++ version. Also provided for Solaris is a parallel version based on MPI (Message Passing Interface) [9], which performs only the SVD computations in parallel. A Java version has recently been released and has been successfully executed on Solaris, Linux, and MacOS X platforms. The Java version has not been tested on Windows as of yet. There has, however, been a successful port of the Solaris C++ version to work with Borland Builder on the Windows platform. The Windows version is not supported at this time. All versions include the query-processing module except the parallel and Windows version.

There are no differences between the Solaris and Linux C++ versions. There are, however, a few differences between the C++ and Java versions. The most obvious is that Java is slower. Since Java uses a virtual machine, calculations are performed at a much slower rate. This is most noticeable when running GTP on a large document collection. As an example, GTP was run using the option to perform the SVD. The document collection used was about 48MB in size, consisting of almost 5,000 documents and over 42,000 unique terms parsed. Time for completion using the C++ version was about 560 seconds. Time for completion using the Java version was about 1.5 hours. Possible solutions to fix this slowdown are discussed later (Section 27.9). Secondly, the Java version is not capable of accepting user-generated filters for the parsing process. The Java version does, however, provide an internal HTML filter. The user can create his/her own filters in Java and incorporate them into GTP.

27.8 Code Evolution

The original design of the GTP parser was based on C code and shell scripts originally distributed by Telcordia Technologies, Inc. (formerly Bellcore). The SVD code was originally written in Fortran [2], converted to C [4], finally converted to C++ for GTP. Hence, the GTP code to date is basically C code wrapped in C++. Since GTP does not use genuine C++ code, true object-oriented techniques (OO) and features were not incorporated. The Java version, which was converted from the C++ version, solves some of the OO mistakes due to the numerous built-in Java classes and the ease of use of these classes.

27.9 Future Work

What does the future hold for GTP? Currently in development is a graphical user interface (GUI) front-end that is written in Java. This GUI will be compatible with all versions of GTP (C++ and Java). The GUI has been a long awaited addition to GTP and will solve many of the issues surrounding the multitude of command-line options that users are plagued with currently. The GUI is currently slated to control GTP runs (parsing and matrix decomposition) as well as controlling queries and displaying results using the current query module [8].

Also being developed is the ability to remotely store files generated by a GTP run as well as the ability to perform a query using the remotely stored files. This is being done using IBP and Exnode technologies [1]. By using the remote storage locations, or IBP depots, a user will have the ability to run GTP on large document collections and not need to worry about local hard drive space restraints. These IBP depots have the ability to allow users to store gigabytes of information for a limited time.

Acknowledgements

This research was supported in part by National Science Foundation under grant no. NSF- EIA-99-75015.

References

[1] M. Beck, T. Moore, and J. Plank. An End-to-End Approach to Globally Scalable Network Storage. In *Proceedings of the ACM SIGCOMM 2002 Conference*, Pittsburgh, PA, August 19-23, 2002.

[2] M.W. Berry. Multiprocessor Sparse SVD Algorithms and Applications. PhD Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, October 1990.

[3] M. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, Philadelphia, PA, 1999.

[4] M.W. Berry, T. Do, G.O'Brien, V. Krishna, and S. Varadhan. SVDPACKC (Version 1.0) User's Guide. Technical Report No. CS-93-194, Department of Computer Science, University of Tennessee, 1993.

[5] M.W. Berry, Z. Drmač, and E.R. Jessup. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review* 41(2):335-362, 1999.

[6] M.K. Hughey. Improved Query Matching Using kd-Trees: A Latent Semantic Indexing Enhancement. *Information Ret*rieval (2):287-302, 2000. *Information Retrieval* (2):287-302, 2000.

[7] T.A. Letsche. Large-Scale Information Retrieval with Latent Semantic Indexing. *Information Sciences* (100):105-137, 1997.

[8] P.A. Lynn. Evolving the General Text Parser (GTP) Utility into a Usable Application via Interface Design. MS Thesis, Department of Computer Science, University of Tennessee, Knoxville, December 2002.

[9] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, 1995.