Schema-Independent Retrieval from Heterogeneous Structured Text

Charles L. A. Clarke G. V. Cormack F. J. Burkowski

Dept. of Computer Science*
University of Waterloo, Waterloo, Canada

Technical Report CS-94-39
November 23, 1994

Abstract

We present a query language for searching collections of structured text. Documents within the collection are not required to adhere to a global schema nor are individual documents required to be structured according to any defined schema at all. Nonetheless, queries may directly reference structure across differently formatted documents. We briefly discuss application of the language to multilingual collections, relational databases, text filtering and document analysis.

1 Introduction

Figure 1, a facsimile of a page from an edition of Shakespeare's *Tragedie of Macbeth*, demonstrates the complexity of structured text. This single page includes stage directions, speakers, speeches, the start of the play's first act, its entire first scene and the start of its second. Marginal notations indicate the act, scene and line numbers. Italics, small capitals, and two font sizes are used. The lineation of the spoken verse is consistent with its scansion. A page number appears at the bottom of the page.

Retrieval from structured text requires the indexing of structural elements and a query language able to directly reference these structural elements. In traditional commercial text-retrieval systems, the indexing of structure is usually limited to pre-defined elements such as words, sentences and paragraphs. In small systems or when all documents are simple and of a similar type, such as the archival store of articles for a newspaper or a collection of project documentation, this approach is adequate. In larger text collections or in specialized text collections, documents have more structure and more types of structure than can be captured by these simple elements. Various techniques may be used for specifying the organization or schema of such a text collection and queries may be based on this schema. In a very large text collection (hundreds of gigabytes or more), particularly when material from a variety of sources is being continually added to the collection, it may not be possible to define a schema for the text collection as a whole. Instead, the structure of each document should be indexed in the ways that are most appropriate for that particular document, with analogous structural elements in different documents being either indexed identically or easily related to one another.

This paper introduces GCL, a query language that permits schema-independent retrieval from structured text. GCL is unique in supporting a wide range of structured text retrieval using a single data type to express all operands and results. Even the indexing of words and markup is an instance of this data type. The discussion of GCL begins with our model for indexing document structure and continues with the language itself. Examples of the language's use are given. Following our exposition of GCL we discuss some of the ideas underlying the language and its relation to other work in the area. Finally, we look at some further applications of GCL, including multilingual collections and document analysis.

GCL is based on an algebra developed by the authors in an earlier paper [5]. That paper presents a

*Email concerning this technical report should be sent to claclark@plg.uwaterloo.ca.

Thunder and lightning. Enter three Witches. I.1

FIRST WITCH

When shall we three meet again?
In thunder, lightning, or in rain?

SECOND WITCH

When the hurly-burly's done,
When the battle's lost and won.

THIRD WITCH

That will be ere the set of sun.

FIRST WITCH

Where the place?

SECOND WITCH Upon the heath

THIRD WITCH

There to meet with Macbeth.

FIRST WITCH

I come Grey-Malkin.

SECOND WITCH Paddock calls!

THIRD WITCH Anon!

ALL

Fair is foul and foul is fair,
Hover through the fog and filthy air. *Exeunt*

Alarum within. I.2
Enter King Duncan, Malcom, Donalbain, Lennox,
with Attendants, meeting a bleeding Captain

KING

What bloody man is that? He can report,

53

Figure 1: Text Structure in *Macbeth*.

formal treatment of the theory behind GCL and includes an implementation framework under which the time required to evaluate a query is at worst proportional to the time required to solve the elementary terms in the query. This work is part of a larger project on Very Large Multi-User Multi-Server Text Databases being undertaken at the University of Waterloo. GCL has been implemented in the context of that project, and acts in the role of client/server interface language.

2 Document Markup

Partial markup for the opening of *Macbeth* is shown in figure 2. Start tags of the form "<*name*>" and end tags of the form "</*name*>" delineate stage directions, speakers, speeches, acts, lines and scenes. Act and scene numbers appear within similar delimiters at the start of each act and scene. Font changes, pagination and other markup associated with document presentation has been omitted. Note that a tag indicating the start of a line may occur in one speech with the corresponding tag indicating the end of the line occurring several speeches later.

We model text as a sequence of words. In the case of *Macbeth* we arbitrarily choose the sequence to include all stage directions, speakers, and speeches; we ignore case and punctuation; words that occur most frequently, such as "the" and "of", are treated as *stop words* and are represented in the sequence by the symbol "_":

```
thunder _ lightning enter three
witches first witch when shall we
three meet...
```

Stop words occupy positions in the sequence but will not be indexed in the database. Any position in the sequence not occupied by a word is assumed to be occupied by "_".

Markup indicating document structure is indexed at (rational-valued) locations between the words. This approach recognizes that markup describes the text but is not part of it. Conceptually, an infinite amount of description may be indexed between each pair of words.

Alternate descriptions of the same text may be independently indexed. Various editions of *Macbeth* use differing lineation. The eighth line of the first scene, in which the witches call to their familiars, is neither in trochaic tetrameter nor does it rhyme with the surrounding lines. Its lineation is sometimes

FIRST WITCH	I come Grey-Malkin.
SECOND WITCH	Paddock calls!
THIRD WITCH	Anon!

This alternate lineation could be indicated in the markup by tags such as `<alt-line>` and `</alt-line>`. If an edition of *Macbeth* was already part of a text collection our model allows the indexing of this alternate lineation to be added at any time.

Suppose we wish to search our text collection for a remembered quote, contained on single line, spoken by a single speaker: "Something wicked this way comes." A query must take into account the alternate lineation and the possibility that several speakers may speak portions of a single line of verse.

3 The Query Language

Each statement in the GCL query language either specifies a query to be solved or defines a macro. The simplest query consists of a doubly-quoted string. Each location in the database where the string occurs is a solution to the query. The query

```
"toil and trouble fire burn"
```

produces the five-word segments of the stored text that match the quote. The match will not necessarily be exact, since we ignored case and punctuation and treated "and" as a stop word.

The presence of markup in the text does not affect the match. If line start and end tags appear in the marked-up text

```
<speaker> All </speaker>
<line> Double, double, toil and trouble;
  </line>
<line> Fire burn and cauldron bubble.
  </line>
```

the "`</line>`" and "`<line>` tags are not considered when testing for a match. This behaviour is consistent with our general principle of keeping the markup independent of the text.

Markup may be included in a query, in which case it must be present in the text. The query

```
"trouble </line> fire"
```

```
<act> <act-number> 1 </act-number>
<scene> <scene-number> 1 </scene-number>
  <direction> Thunder and lightning. Enter three Witches.  </direction>
  <speaker> First Witch </speaker>
    <speech> <line> When shall we three meet again? </line>
     <line> In thunder, lightning, or in rain? </line> </speech>
  <speaker> Second Witch </speaker>
      <speech> <line> When the hurly-burly's done, </line>
      <line> When the battle's lost and won. </line> </speech>
  <speaker> Third witch </speaker>
      <speech> <line> That will be ere the set of sun. </line> </speech>
  <speaker> First Witch </speaker>
    <speech> <line> Where the place? </speech>
  <speaker> Second Witch </speaker>
      <speech> Upon the heath </line> </speech>
  <speaker> Third witch </speaker>
      <speech> <line> There to meet with Macbeth. </line> </speech>
  <speaker> First Witch </speaker>
    <speech> <line> I come Grey-Malkin. </speech>
  <speaker> Second Witch </speaker>
      <speech> Paddock calls! </speech>
  <speaker> Third witch </speaker>
      <speech> Anon! </line> </speech>
  <speaker> all </speaker>
    <speech> <line> Fair is foul and foul is fair, <line>
    <line> Hover through the fog and filthy air. </line>  </speech>
  <direction> Exeunt </direction>
</scene>
<scene> <scene-number> 2 </scene-number>
  <direction> Alarum within.  Enter King Duncan, Malcom,
  Donalbain, Lennox with Attendants, meeting a bleeding
  Captain </direction>
  <speaker> king </speaker>
    <speech> <line> What bloody man is that? He can report, </line>
    <line> As seemth by his plight, of the revolt ...
```

Figure 2: Markup for *Macbeth*.

would match in the fragment of text above.

The result of a query in GCL is a sequence of *extents* or ranges in the database. Each extent represents the occurrence of a solution to the query. Extents may overlap, but they may not nest.

There are eight GCL query operators, which may be broken into three categories: one *ordering* operator, three *combination* operators, and four *containment* operators. The ordering operator is used to link textual elements, the combination operators are used to group textual elements, and the containment operators are used to express structural relationships.

Macros allow symbolic names to be assigned to frequently-used queries and allow queries to be expressed in terms of general concepts — lines, verses, pages or paragraphs — rather than in terms of the physical characteristics of the document markup. A particular macro, say for matching lines, might incorporate a variety of different physical markup techniques. Depending on the environment, the person generating a query will often not be aware of the details of a macro being used. Macros may be parameterized; when a parameterized macro is used each parameter will have a query substituted in its place.

The following subsections cover the details of macros and the various categories of query operators. For the convenience of the reader, figure 3 summarizes the syntax of GCL in Backus-Naur Form.

3.1 The Ordering Operator

The *ordering operator* "..." links textual elements. The form of a query using the ordering operator is

query ... *query*

For example, the query

```
"Macbeth" ... "Thane of Fife"
```

returns all extents such as "Macbeth! Beware Macduff; Beware the Thane of Fife" that begin with a solution to the query on the left-hand-side of the operator and are followed-by and end-with a solution to the query on the right-hand-side of the operator.

An important and common usage of the ordering operator is to connect the start and end tags of structural elements, producing extents that correspond to those structural elements. The result of connecting the start and end tags for lines with the ordering operator

```
"<line>" ... "</line>
```

is all the lines in the database.

3.2 Combination Operators

All three combination operators have the basic form

quantity `of` (*list of queries*)

Each solution to a combination operation covers the solutions to a specific number of the queries in the associated list. The quantity associated with the combination operation determines the number of included solutions. The most general of the combination operators expresses the quantity as an integer (which must be greater than zero, and less than or equal to the length of the query list). Solutions to

```
2 of ("Macbeth", "Birnam", "Dunsinane")
```

include "Macbeth shall never vanquish'd be until Great Birnam", "Birnam wood to high Dunsinane", and "Birnam rise, and our high-plac'd Macbeth". Each solution begins and ends with a solution to one of the queries in the list.

For convenience, two special forms of the combination operator exist: A query of the form

`one of` (*list of queries*)

merges the solutions to the queries in the list into a single sequence. A query of the form

`all of` (*list of queries*)

combines solutions to the queries into extents that contain a solution for each of the queries.

3.3 Containment Operators

The four containment operators are central to referencing textual structure:

query `containing` *query*
query `contained in` *query*
query `not containing` *query*
query `not contained in` *query*

A solution to a containment operation is also a solution to the query on the left-hand side of the containment operation. A containment operation acts as filter, eliminating solutions to the query on the left-hand side.

The quote 'Macbeth! Beware Macduff; Beware the Thane of Fife" is not a solution to the query

statement ::=
    macro-definition
  | *query*

macro-definition ::=
    identifier `=` *query*
  | *identifier* `(` *parameters* `)` `=` *query*

query ::=
    query `containing` *query*
  | *query* `contained in` *query*
  | *query* `not containing` *query*
  | *query* `not contained in` *query*
  | `one of` `(` *queries* `)`
  | *quantity* `of` `(` *queries* `)`
  | `all of` `(` *queries* `)`
  | *query* `...` *query*
  | `(` *query* `)`
  | *quantity* `words`
  | *identifier* `(` *queries* `)`
  | *identifier*
  | *quoted-string*

queries ::= *query* | *query* `,` *queries*

parameters ::= *identifier* | *identifier* `,` *parameters*

quantity ::= *positive-integer*

Figure 3: Syntax of GCL.

```
("Macbeth" ... "Thane of Fife")
  contained in
("<line>" ... "</line>")
```

since the quote stretches over two lines.

3.4 Macros

A macro definition has the form

identifier (*parameters*) = *query*

where the parameter list is optional. The statements

```
LINE =
  one of (
    "<line>"..."</line>",
    "<alt-line>"..."</alt-line>"
  )

SPEECH = "<speech>"..."</speech>"
```

define `LINE` as a macro representing the lines in the database (including alternate lineation) and `SPEECH` as a macro for the speeches in the database. The statement

```
QLINE (quote) =
  (quote contained in LINE)
             contained in SPEECH
```

defines `QLINE` as a parameterized macro that searches for a quote contained on a single line and spoken by a single speaker. The query

```
QLINE ("Something wicked this way comes")
```

uses this macro to search for the specified quote.

The environment presented by a text retrieval system will usually include pre-defined macros to assist query building. A collection including the plays of Shakespeare might predefine macros for lines, scenes, acts, speakers, and speeches. These predefined macros can take into account variations in structure that might not be apparent to a casual user. If plays by other authors are added to the collection the macros may be changed to reflect the structure of these plays without effecting the user's understanding of the system.

3.5 Proximity Constraints

A query of the form

quantity `words`

may be used in combination with the containment operators to place size and distance restrictions on solutions to queries. A solution to the query

```
SPEECH contained in 5 words
```

is a speech of five words length or less.

4 Discussion

4.1 Generalized Concordance Lists

The query operators of GCL operate on elements of a single data type, *generalized concordance lists* or *GC-lists*, and return elements of this data type as their result. A GC-list is an ordered list of extents or ranges from the sequence of words that make up a collection of text. Extents in a GC-list may overlap, but they may not nest. This seemingly inconsequential property is the soul of GCL. We use two examples to illustrate this point.

Example 1 (nesting)
Consider the line

Fair is foul, and foul is fair,

This line contains four possible solutions to the query

```
all of ("fair", "foul")
```

specifically

Fair is foul
Fair is foul, and foul
foul, and foul is fair
foul is fair

Indeed, every extent between every "foul" and every "fair" in the entire collection is a possible solution to the query. But because extents in a GC-list cannot nest, most of these possible solutions are not returned by GCL. Only solutions such as

Fair is foul
foul is fair

that contain no embedded "foul" or "fair" are returned. Non-nesting of extents also guarantees that the solution to the query

```
"<line>" ... "</line>
```

contains all individual lines and not groups of lines.

Example 2 (overlap)
Consider the line

> All hail, Macbeth! Hail to thee, Thane of Cawdor

This line contains two possible solutions to the query

```
all of ("hail", "Macbeth")
```

specifically

> hail, Macbeth
> Macbeth! Hail

If we do not allow extents to overlap, which do we choose?

The GCL "`all of`" operator is the equivalent of the boolean "AND" operator found in most traditional commercial text retrieval systems. Usually the "AND" operator identifies documents that contain solutions to both its operands. Unlike the "AND" operator, the "`all of`" operator does not have any implied containment associated with it. It is only by allowing overlapping extents that independence from implied containment constraints can be achieved.

4.2 Schema Independence

Numerous markup conventions exist for specifying document structure. These conventions are generally concerned with specifying the structure necessary for formatting and display, or for document transmission from one computer system to another. Modern markup conventions concentrate on the specification of the logical structure of documents and avoid physical characteristics — fonts, layout, and pagination — that vary as the formatting style and the display medium change. The most flexible of the widely-recognized conventions is the Standard Generalized Markup Language (SGML) [3, 9]. The markup style that we have used throughout the paper is similar to that of SGML.

An SGML document contains a description of its structure, its *document type definition* or DTD, in addition to its text. Although a large group of documents will generally share a common document structure through the use of a publically declared DTD, the structure of an individual document could be unique. This facility for explicitly describing a document's schema or meta-structure (the structure of its structure) is central to SGML. Conventions other than SGML typically mix the specification of logical and physical structure and are restricted in the variety of logical structure that can be directly expressed. The actual structure of these documents often cannot be inferred from the markup alone.

In specifying document structure for retrieval, any dependence on document meta-structure is extremely inhibiting. In a collection of heterogeneously-formatted documents a query must be formulated with consideration to the meta-structure of every document that might potentially be of interest. In a very large collection, merely grasping the variations in meta-structure is a difficult task.

The straightforward approach to indexing structure is to index markup tags. This approach has limitations if a markup tag is treated as a word like any other. From the viewpoint of a user, a tag is not part of the text itself but a part of the presentation of the text. A user incorporating a remembered quote into a query cannot anticipate the markup that might occur within the quote in documents where the quote appears.

It must be possible to incrementally index text structure. If a particular word processor format does not explicitly tag section headers, but at a later time this information is obtained from another source — perhaps inferred from font sizes or page layout — it should be possible to add this information to the index.

It must be possible to index as a group equivalent structural elements from different document formats. Across a variety of document formats, a particular structural element — paragraph boundaries, for example — might be indexed using both markup specific to the document format and markup generic to the collection as a whole.

Our views on structured text retrieval may be summerized by three *schema independence principles*:

- *Meta-structure Independence.* Adherence to a meta-structure should not be required of documents. Queries referencing document structure should not be expressed in terms of document meta-structure.

- *Markup Independence.* Markup describes text but is not part of it. It should be possible to index the structure of a document in a variety of differing ways.
- *Query Element Independence.* Analogous structural elements from different document formats should either be indexed as a group or a mechanism should be provided for easily relating one to another.

Our markup indexing approach provides both markup independence and query element independence by indexing words at integral positions and indexing markup at arbitrary rational positions between the words. The query operators of GCL provide meta-structure independence; the macro facility in combination with predefined environments provides query element independence.

4.3 Related Work

Work in the area of structured text retrieval generally either extends the relational algebra with support for structured text, uses a context-free grammar to describe text structure and bases queries on this grammar, or attempts both [1, 2, 6, 7, 8]. This heavy dependence on relations and/or grammars to describe text structure and formulate queries is in direct contrast to our work.

GCL owes some of its intellectual and cultural heritage to two earlier structured text retrieval languages developed at the University of Waterloo. The Pat text searching system [10] was developed for use with the New Oxford English Dictionary. Pat provides meta-structure independence, and limited query element independence through the use of region definitions, which resemble our macros. Query operators in Pat are similar to those in GCL, but no equivalents to our combination operators are provided. Pat does not use a uniform data type for query operands and results, causing significant semantic problems. Queries expressible in Pat (with a few superficial exceptions) are a subset of of those expressible in GCL.

The language of Burkowski [4] is the direct ancestor of GCL. That language provides markup independence but relies heavily on document meta-structure for query formulation. The language's query operators are similar to those of both Pat and GCL; no equivalents to our combination operators or ordering operator are provided. The language uses a single data type, the *concordance list*, for query operands and results. Our GC-list is a generalization of this data type. Extents in a concordance list are not premitted to overlap, causing some semantic inconsistencies in the language.

5 Further Applications

We conclude the paper by briefly reviewing a number of points concerning the application of GCL.

5.1 SQL (with Full-Text Extensions)

GCL is usable for browsing data in a wide variety of formats. A great deal of on-line data is currently in the form of SQL relations. A simple representation of a SQL relation as a marked-up sequence is sufficient to permit searching of an SQL database using GCL. This is particularly useful if inclusion of full-text data in relational databases becomes more prevalent [1, 2].

We treat the data in a table as if loaded sequentially in row-major order. The extent of each table is considered to be tagged with "`<table>`" and "`</table>`" tags. The name of the table is indexed immediately following the table start tag and is delineated with "`<table-name>`" and "`</table-name>`" tags. Each tuple in the relation is delineated with "`<tuple>`" and "`</tuple>`" tags. Each tuple consists of a sequence of attribute name/value pairs marked-up as follows:

<name> *name* <value> *value* </value>

with the end of the name being implied by the "`<value>`" tag.

Assume we have a table named "`Reports`" that stores technical papers with a column for each of the tile, authors, abstract and body. A query for papers written by "Clarke, Cormack and Burkowski" (a select operation) would be

```
CCB = all of ("Clarke", "Cormack", "Burkowski")
AUTHORS = "<name> Authors <value>"..."</value>"
CCB-AUTHORS = CCB contained in AUTHORS
TABLE = "<table>"..."</table>"
REPORTS = "<table-name> Reports </table-name>"
REPTABLE = TABLE containing REPORTS
TUPLES = "<tuple>"..."</tuple>"
REPTUPLES = TUPLES contained in TECHTABLE
REPTUPLES containing CCB-AUTHORS
```

5.2 Text Filtering

Throughout the paper we have been assuming the use of GCL as a query language for an indexed text database. GCL may also be used to directly search un-indexed text in a linear fashion in the manner of the UNIX `grep` utility. If our database permits the on-line addition of text to the collection we may use this approach to filter incoming text against a set of stored queries that reflect the ongoing interests of the database's users.

5.3 Document Analysis

Document analysis techniques that infer document structure can take advantage of schema independence. If an analysis technique can infer only partial or inconsistent structure this information may still be stored in the database and searched using the query language. The language may be used as a tool to enforce consistency. For example, if all speakers must be followed by a speech the query

```
("</speaker>"..."<speaker>")
            not containing SPEECH
```

must have no solutions.

5.4 Multilingual Collections

Schema independence assists in the creation of collections that include documents written in different languages and individual documents written in multiple languages provided that the text can be mapped into a sequence of words (or ideograms) with markup indexed at and between the words. Markup independence allows tags specific to a particular document's format, which would usually be mnemonic in the language of that document, to be related to analogous structural elements in documents written in other languages.

Acknowledgements

The Multi-User Multi-Server Very Large Text Database project is funded by the Government of the Province of Ontario through its Information Technology Research Centre. The Natural Sciences and Engineering Research Council of Canada and the University of Waterloo provided additional financial support.

References

[1] Air Transport Association. *Advanced Retrieval Standard — Structured Fulltext Query Language (SFQL).* ATA 89-9C.SFQL.

[2] G. E. Blake, M. P. Consens, P. Kilpeläinen, P.-Å. Larson, T. Snider, and F. W. Tompa. Text/relational database management systems — harmonizing SQL and SGML. In *Proc. Applications of Databases*, pages 267–280, Vadstena, Sweden, June 1994.

[3] M. Bryan. *SGML — An Author's Guide to the Standard Generalized Markup Language.* Addison-Wesley, New York, 1988.

[4] Forbes J. Burkowski. An algebra for hierarchically organized text-dominated databases. *Information Processing and Management*, 28(3):333–348, 1992.

[5] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. Technical Report CS-94-30, University of Waterloo Computer Science Department, Waterloo, Ontario, Canada, N2L 3G1, September 1994. Available by anonymous ftp in `/cs-archive/CS-94-30` on `cs-archive.uwaterloo.ca`.

[6] Gaston H. Gonnet and Frank Wm. Tompa. Mind your grammar — a new approach to modelling text. In *Proc. 13th VLDB Conference*, pages 339–346, Brighton, England, 1987.

[7] Ralf Hartmut Güting, Roberto Zicari, and David M. Choy. An algebra for structured office documents. *ACM Transactions on Office Information Systems*, 7(4):123–157, April 1989.

[8] Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. A grammar-based approach towards unifying hierarchical data models. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 263–272, Portland, Oregon, 1989.

[9] International Standards Organization. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, October 1986. ISO 8879.

[10] Airi Salminen and Frank Wm. Tompa. Pat expressions — an algebra for text search. Technical Report OED-92-02, UW Centre for the New Oxford English Dictionary, Waterloo, Ontario, CANADA, N2L 3G1, July 1992.