Fashioning a Search Engine to Support Humanities Research

Frank Wm. Tompa
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
fwtompa@uwaterloo.ca

ABSTRACT

Scholarship in the humanities often requires the ability to search curated electronic corpora and to display search results in a variety of formats. Challenges that need to be addressed include transforming the texts into a suitable form, typically XML, and catering to the scholars’ search and display needs. We describe our experience in creating such a search and display facility.

1 BACKGROUND AND MOTIVATION

Many researchers in the humanities have turned to computers to help them to examine documents. Most often the goal is to understand the thoughts and reasoning of those documents’ authors, but it may also be to understand an author’s craft in creating the documents under investigation or to gain insight into the author’s life and times. Researchers who study document collections may come from a variety of backgrounds, leading the Society for Textual Scholarship to welcome “scholars from disciplines such as literature (in all languages), history, musicology, classical and biblical studies, philosophy, art history, legal history, history of science and technology, computer science, library science, lexicography, epigraphy, paleography, codicology, cinema studies, theatre, linguistics, and textual and literary theory.”1 Digital humanities research might require computationally advanced analyses [1, 2], but often the sheer volume of a corpus under study makes it necessary (or at least advantageous) to digitize documents of interest and to make them available through a search engine that is effective in finding passages of interest to the scholar throughout the research period [31].

Conventional search engines, however, are not designed to meet the needs of scholars from the humanities. For example, scholars usually want to find passages that exactly match their queries, rather than being satisfied with ranked lists of documents or document fragments that are judged to be relevant in response to a few keywords. When approximate matches are sought, scholars usually require that the form of approximation be carefully controlled to reflect normalization in spelling rather than semantic equivalence. Furthermore, a scholar’s corpus is often not freely available to the general public; it may include intellectual property that must be protected, and a document management system must therefore ensure that the documents are not unintentionally disseminated.

This paper describes the components of a search system developed for the Alberti Magni e-corpus, the texts digitized in a project undertaken to study the philosophical ideas discernable from the collected works of Albertus Magnus,2 a prolific and influential 13th-century philosopher and theologian [27]. Those works form an authoritative reference on Christian theology, philosophy, and natural science as it was understood in the Middle Ages. The Latin texts span a wide variety of subjects, including “logic, ethics, political philosophy, philosophy of nature, philosophical psychology, zoology, botany, mineralogy, geography, astronomy, meteorology, mathematics, metaphysics, systematic theology, interpretation of Scripture” [32]. The 60 texts comprising the Alberti Magni e-corpus range from short letters, such as Testamentum Domini Alberti, without any structure other than page boundaries, to long commentaries on other texts, including Super IV libros Sententiarum, a commentary on Peter Lombard’s theological tome, in which the extensive commentary follows the structure of Lombard’s work (a prologue followed by four books, each book—Liber—divided into sections, each designated as distinctio) and is interspersed with Lombard’s text, as described by the simplified grammar in Figure 1. Only the text written by Albertus Magnus should be searchable; the text being commented upon (i.e., the Lombard substructures in Figure 1) serves to anchor the commentary, but it is not to be searched and yet it is to be displayed when browsing.

The code base for the custom-developed search engine was adapted from that used to support the Electronic Campsey Project,3 which in turn was modelled on the software developed earlier to

1https://textualsociety.org/

2http://albertusmagnus.uwaterloo.ca/

3http://margot.uwaterloo.ca/campsey/cmphome_e.html
A scholar in the humanities, having chosen the texts to analyze, typically includes the recognition of layout structure (pages and interleaved by page and column breaks running text, typically divided into paragraphs and sentences), and embedded features (headings, italic text, Hebrew and Greek letters). OCR and many are hierarchically arranged in units designated as LIBER and TRACTATUS (tract), and CAPUT (chapter).

During the proof-reading stage, some initial markup to capture the structural units and features within a document are recognized by the search engine. This typically includes the recognition of layout structure (pages and columns), logical structure (chapters, sections, paragraphs, and sentences), and embedded features (headings, italic text, Hebrew text, and illustrations). All documents in the editions scanned for the Alberti Magni e-corpus are laid out on pages, but only some place the text into columns. All have sentences and paragraphs, and many are hierarchically arranged in units designated as LIBER (book), TRACTATUS (tract), and CAPUT (chapter).

2 INITIAL TEXT MARKUP

A scholar in the humanities, having chosen the texts to analyze, often begins by mechanically scanning myriads of printed document pages, applying optical character recognition (OCR), and laboriously correcting errors (by comparing the output against the original images, perhaps aided by a language-specific spelling checker). This digitization process results in a stream of text, with some typography and layout preserved.

Successful text search and display requires that structural units and features within a document are recognized by the search engine. This typically includes the recognition of layout structure (pages and columns), logical structure (chapters, sections, paragraphs, and sentences), and embedded features (headings, italic text, Hebrew text, and illustrations). All documents in the editions scanned for the Alberti Magni e-corpus are laid out on pages, but only some place the text into columns. All have sentences and paragraphs, and many are hierarchically arranged in units designated as LIBER (book), TRACTATUS (tract), and CAPUT (chapter).

During the proof-reading stage, some initial markup to capture the structural units and features can also be added to the text. OCR software captures not only the sequence of letters that comprise a text, but also its spacing and typography, and thus some of this markup may be derivable from a parallel analysis of the physical layout of the scanned pages [23]. Beyond what the OCR system provides, some structural markup is simple to insert manually. Unfortunately, there is a cost to adding markup by hand: the more markup that is required, the greater the additional cognitive burden that is placed on the proof reader and the greater the time to insert markup by hand [18]. For the Alberti Magni e-corpus, the following markup conventions were designed to balance completeness and simplicity:

- enclose all page and column numbers within slashes (e.g., /24/ for the start of page 24, /135B/ for the start of the second column on page 135);
- insert $err#corr$ to indicate that the erroneous text in the original manuscript err has been corrected to corr;
- surround transliterations of Greek letters with [GREEK: and ]; and similarly surround transliterations of Hebrew letters with [HEBREW: and ];
- indicate the presence of graphics by the notation [ILLUSTRATION];
- identify source text upon which Albertus Magnus is commenting by inserting [@author:] at the start and [@] at the end, where author is the name of the source text’s author;
- surround tables with |- at the start of each table’s first line and |- at the end of each table’s last line, and insert | between table columns on each line;
- where non-standard section headings are used, supplement them with synthetic headings in a standardized form; for example, to indicate that the heading Cuius primus tractatus est de homine indicates the start of the first “tractatus”, which is entitled “De homine”, the scholar inserts the mark up: +TRACTATUS IXDE HOMINE+Cuius primus tractatus est de homine

The resulting documents have sufficient structure explicitly and unambiguously marked that it can be converted into well-formed XML that includes tags marking all textual elements of interest to the researcher. However, the ability to transform a document collection into XML in theory does not imply that it is straightforward to perform the transduction in practice.

3 RECOGNIZING TEXT STRUCTURE

Before new texts are added to the e-corpus, they need to be converted to XML, often built upon the TEI tag set developed by the Text Encoding Initiative [7, 29]. The Alberti Magni e-corpus requires tags that mark pages, columns, titles, structural sections, paragraphs, sentences, hierarchical locations that serve as targets for citations into the text, and various other features present in the corpus. Rather than building individual taggers for each document, a single conversion pipeline was designed to be applied to all of the works in the e-corpus; in this way, as conversion errors are found, the tagging pipeline can be corrected and reapplied to all documents to ensure that similar, but undetected, errors in documents already processed are also corrected.

For the Alberti Magni e-corpus, the input text is stored in rich text format (RTF), using predetermined markup conventions as described in the previous section. The texts are first converted to HTML using rtf2html. Thereafter the text passes through a series of perl scripts, the first of which removes superfluous white space and HTML encoding (but preserving italics and bold tags). Next, simple atomic elements are tagged: Greek, Hebrew, and illustration indicators are replaced by tags; table delimiters are replaced by corresponding HTML table, row, and cell tags; page and column markers are replaced by empty “milestone” tags with attributes encoding the page and column numbers; and editorial corrections marked as $err#corr$ are changed to <corr w="err">corr</corr>.

![Figure 1: Nesting structure of Albertus Magnus’s Super IV libros Sententiarum; all undefined non-terminals representing running text, typically divided into paragraphs and sentences and interleaved by page and column breaks](https://sourceforge.net/projects/rtf2html/)
These changes appear to be straightforward, and it is fairly simple to implement a converter that handles the vast majority of the text correctly. However, complications arise when the various markup conventions overlap in the text or occur in unanticipated contexts. The remainder of this section illustrates some of the complications encountered when recognizing the structure found in the Alberti Magni e-corpus.

Page layout conventions are such that page and column boundaries are usually placed where there is white space in a text. However, they can also occur in the middle of words. In order to recognize a complete word that crosses a page or column boundary and still record exactly where the word is split, an element is introduced to mark the starting fragment of a broken word, with an attribute encoding the whole word, and a second element is introduced to mark the second fragment of the word. Thus, for example, the tags in $<\text{frag} w="sicut">$sic</fragment>$ are used where the word sicut breaks between the bottom of the first column on page 16 and the top of the next. To complicate things further, a page or column break can also occur in the middle of a word: $<\text{frag}>$con/488A/trarius</fragment>$, $<\text{frag}>$con/488A/trarium</fragment>$; thus conversion code must be carefully written to accommodate this variant. Furthermore, if a page or column break is surrounded by italic text, the slashes around the page or column number may well also be in italics: these four variants are indistinguishable by a proof-reader of the text in printed form. Again, conversion code must be able to handle all these variants.

Inserting tags that reflect the collected works’ structures requires significantly more attention. The starts of sections are often clearly indicated by key words or phrases at the start of input lines (especially if supplemented by synthetic section headers as described in the previous section), and these markers are often followed by section numbers, but there is sufficient variety to make this task non-trivial. There are 23 markers signifying the start of various types of numbered sections (see Figure 2), and a variety of numbering conventions are used. For example, some structures use Roman numerals (e.g., CAPUT IV.) and other use Arabic numerals (e.g., Sermo 39); the work Sermones parisienses is divided into numbered sections without preceding key words (e.g., I, II), and Determinatio Magistri Alberti de novo spiritu is similarly divided, but using Arabic numerals (e.g., I, 2), further complicated by the presence of a section numbered 1a between the section numbered 1 and the section numbered 2; and some sections containing Biblical commentary are numbered by ranges of the verses referenced (e.g., Versus 43-44). In addition, recognizing the starts of structures is further complicated by the possible occurrence of corrections for the structure name, the section number, or both ("ARTICULUS#ARTICULUS XV. ") vs. "ARTICULUS XIX. # IX. §" vs. "$ARTICULUS#ARTICULUS III. §$".5

Having recognized a section heading, any text following it without an intervening blank line is tagged as a title for that section. Text following a blank line is then interpreted as the body of the section, consisting of paragraphs with nested sentences. Because paragraphs are separated by blank lines in the input, marking paragraph boundaries is straightforward. A naïve approach to marking sentence boundaries was adopted, based simply on punctuation followed by a capitalized word, but more sophisticated methods could be applied if it were found to be necessary [19].

The end of a section is implicitly indicated by the start of some other section that cannot be nested within it. For example, consider the works with sections marked LIBER and nested sections marked TRACTATUS, which in turn have nested sections marked CAPUT: each LIBER section ends when the next section marked LIBER is encountered; similarly, a section marked TRACTATUS ends when a section marked either LIBER or TRACTATUS begins. However, the structure is not necessarily regular: LIBER I in the work Topica starts with two CAPUT sections before the first TRACTATUS with its nested CAPUT sections begins; several works have TRACTATUS and CAPUT sections, without first being divided into LIBER sections; all the LIBER sections in the work Politica are divided into CAPUT sections without any divisions marked TRACTATUS. Many other works use radically different structures and substructures, as indicated in Section 1 above, making it important to characterize the constraints on the nesting of structures. Figure 2 lists all the structural divisions in the e-corpus that correspond to (usually) numbered sections, together with the possible text sequences that occur before the section number appears. The third column indicates constraints on the nesting of those divisions, as follows:

- leaf no nested substructures
- >struc disallowed in strict
- -struc no nested substructures and disallowed in strict

There are 23 additional rules for those substructures (such as prologues, sections marked EXPOSITIO TEXTUS, etc.) that are never numbered. Using these tables, the ends of structures are recognized much like a shift-reduce parser processes context free languages [9].

---

5All examples in this section are present in the input texts and were discovered while developing the conversion pipeline.
The combined search and display code is invoked from a web page that uses a front-end handler to read parameters for the search, calls a back-end search engine (here, sgrep), formats the results (using DRAGOMAN), and presents the results on a subsequent web page that provides a form for specifying the next query. The code depends on several configuration files that are referenced from a single “directory file” read at compile time by the front-end handler. This directory file specifies the location of the metadata table (which stores the names and locations of files to be searched), the commands required to invoke the search engine, the location of the style sheets for controlling how to display search results, the location of the list of searchable units (e.g., paragraphs, pages, sections), and the location of a file containing messages to be displayed by the front end. Because the metadata table, list of style sheets, list of searchable units, and list of messages are all referenced indirectly, their contents can be changed as desired without recompilation.

Given a query, the sgrep system (like Pat and Wumpus) returns matching regions rather than searching for documents that contain similar words, where a region can be any contiguous sequence of one or more characters within the document being searched. The system finds all text regions that match a query exactly, returning either the number of such regions or the extents themselves (as pairs of offsets for the start and end of each region or as strings of characters found between those offset pairs). The sgrep query language is a “region algebra” [5] that treats string queries as searches for regions whose contents exactly match the query string; includes operators for testing for region inclusion, adjacency, and proximity; and includes operators for combining lists of regions through conjunction, disjunction, and negation.

The reliance on matching regions leads to some surprising results for users familiar with the word-based retrieval engines commonly used to search the web. For example, a search for the string et in Albertus’s Mineralia matches 4788 two-character regions that contain the string et, only 3026 of which are instances of the word et. A search for “et” (the word surrounded by blanks) identifies 2906 instances of the word, but other instances are abutted by punctuation marks or XML tags. If the goal is to find all instances of the word “et,” the solution is to provide an option to find all regions that correspond to maximal contiguous sequences of letters and
retrieving those that coincide with the regions matching the string et after case-folding. As a second surprising example, a search for regions matching in ipsum returns 4 matches of the phrase in Albertus’s *Mineralia*, but it misses the instance on page 10 where the two words are separated by a column break. A somewhat acceptable solution to searching for the phrase consisting of these two words is to find all sentences that contain the string in followed a blank line (eventually) by the string ipsum within that same sentence; unfortunately this returns 39 spurious matches that include other intervening words as well. As a consequence, the results returned when searching for phrases must be carefully examined.

Importantly, sgrep also supports the formation of regions that extend from a region in one list until the nearest region in another list. For example,

\[
\text{define(UNIT\_STAG,("<d: \ldots \d: >\))}
\]

first creates a region list \( L_1 \) of all regions (of length 3) matching the string "\(<d:"\) and a second list \( L_2 \) of all single-character regions that contain "\( \d:\)"; then forms regions from the \( \text{start} \) of each region \( x \in L_1 \) to the \( \text{end} \) of each region \( y \in L_2 \) such that there does not exist \( z \in L_1 \cup L_2 \) such that \( z \neq x, z \neq y, \) and \( z \) falls within the region extending from the \( \text{start} \) of \( x \) to the \( \text{end} \) of \( y \). That is, the list of regions defined by \text{UNIT\_STAG} above includes each region defined by the \( \text{closest pairs} \) of "\(<d:"\) and "\( \d: >\)" inclusive of both ends. Instead of "\( \ldots \)" the notation "\( -_-\)" includes the start regions but excludes the end regions, "\( _-\)" excludes the start regions and includes the end regions, and "\( _-_-\)" excludes both the start and end regions from the returned regions list (i.e., the returned regions extend from the \( \text{ends} \) of the first regions to the \( \text{starts} \) of the second paired regions).

To use sgrep, it is necessary to describe the encoding of structures to be searched and returned (e.g., pages, paragraphs, titles). Paragraphs are straightforward, since they are encoded as XML elements having the generic identifier \( p \):

\[
\text{define(PARA,(ELEMENTS("p")))}
\]

Titles are slightly more complicated: some titles are synthesized for the e-corpus, and so both they and their original print forms are tagged differently than section titles that are maintained without alterations, and the names of works in the corpus also use distinctive tags. As a result, for this corpus, titles are encoded as XML elements with any of four generic identifiers:

\[
\text{define(TITLE,( (ELEMENTS("title") or ELEMENTS("synTitle") or ELEMENTS("printTitle") or ELEMENTS("origTitle"))))}
\]

Finally, because pages and columns are indicated by milestones only, their extents must be described as extending from one milestone (inclusive) to the next (exclusive):

\[
\text{define(PG,(ELEMENTS("page")))}
\]

\[
\text{define(PAGE,( (start \_ PG) or (PG \_ end)) or (PG .. end)))}
\]

\[
\text{define(COLUMN,( (start \_ COL) or (COL \_ COLEND) or (COL .. end)))}
\]

Similarly to identifying pages and columns based on such extents, the location text to be displayed for a match (as in Figure 3) can be computed from the one found in the extent that is defined from the preceding location attribute, regardless of tag (inclusive) to the location attribute that follows (exclusive).

In all, 26 sgrep rules are defined to support boolean search within any of six structures (paragraphs, titles, sentences, sections, pages, or columns) present in the Alberti Magni e-corpus and to support the searches needed to highlight matches and display the locations of the matched results.

With these rules defined, to find all paragraphs that contain the string "elicit", the search engine first identifies all paragraph regions \( P \) that contain regions (of length 6) exactly matching the search string. It next finds all regions \( M \) that exactly match the search string within regions in \( P \). If it then finds all regions \( L \) that start at one location attribute and end with another and contain the start of the regions exactly matching the search string (i.e., the "e" of "elicit"). The search engine outputs the number of matching regions in \( P \) and the number of matching regions in \( M \). After that, for each region in \( P \), it constructs the location identifier for the first location attribute in \( L \) that falls within that region and prints the location followed by the contents of the matching paragraph from \( P \), surrounding each word containing a match in \( M \) with <match> and </match> to provide highlighting. Figure 4 shows the two paragraphs in the document "De sex principiis" that contain the three instances of words that include the string "elicite", each identified by the locations of those matches. Similar processing is applied when searching for phrases
These offsets are then used to locate the corresponding instances of (i.e., strings that include blanks, such as "quod equus et homo").6 ordered words (e.g. "prima*secundum"), or boolean combinations of these (e.g., "dicit Aristoteles" + (bestia | animal | brut)).

Variations in Latin spelling, especially when the sources are varied, add a complication to search. For example, the letters j and i are often used interchangeably, as are the letter pairs ci and ti; the letter h is sometimes omitted and apostrophes are used inconsistently. To simplify search, therefore, the search engine includes an option for searching with normalized spelling, where the following rules are used for orthogonal normalization: remove diacritics, ignore capitalization, ignore h and apostrophe (‘), use the equivalences in Table 1, and ignore repeated letters. Thus, for example, philosophiae is normalized as filosofie, Tyrrhenum is normalized as tirenum, and proemio is normalized as prenio.

A complication arises from region-based search that is not present in word-based search. With exact match, a search for the string philosophia matches all instances of the prefix of philosophiae. However, philosophiae is normalized as philosofie, so a search for its normalized form does not match the normalized form of instances of philosophiae. As a result, users may be surprised to learn that search under normalization can produce fewer results than search for the same string without normalization. To adjust for this anomaly, a normalized search for any string ending in a is similarly interpreted as searching also for that string ending in ae; a normalized search for any string ending in o is interpreted as searching also for that string ending in oe; a normalized search for any string ending in c is interpreted as searching also for that string ending in ci; and a normalized search for any string ending in p is interpreted as searching also for that string ending in ph. This approach guarantees that no matches are missed, although some false positives might be returned, such as instances of the string ille when searching for strings matching illa (normalized).

Search engines based on inverted indexes (postings lists) typically support normalized search by incorporating orthographic normalization as part of text tokenization prior to indexing. However, because sgrep does not use an index, an alternative method is needed to search for normalized text. The solution is to build a "shadow" text X for each document D in which each word w_i is replaced by its normalized form n_i, but the offset of n_i in X remains unchanged from the offset of w_i in D. Subsequently a search for normalized query terms in X returns a list of matching offsets. These offsets are then used to locate the corresponding instances of the unnormalized words in D. Figure 5 shows a partial result from a search with normalized spelling.

The indexing cost of this approach is the time to normalize all documents and an amount of space equal to the sum of the sizes of the original documents. For a corpus supporting research in the humanities, even one as large as the Alberti Magni e-corpus, this is quite manageable: it takes only a couple of seconds7 to create the 130MB normalized version of the Alberti Magni e-corpus. Because of the need to merge lists, the engine’s response time for searching all sixty documents increases with normalization from 6 seconds (unnormalized) to 9 seconds (normalized) for a rarely occurring string (such as convertibilia), from 10 to 12 seconds for a frequently occurring string (such as illa), and from 13 seconds to 19 seconds for an extremely common string (such as a).

5 DISPLAYING MATCHED TEXT

After a search, the matching portions of a document need to be displayed, as illustrated in Figures 4 and 5. To this end, the DRAGOMAN text display subsystem processes text sequentially, replacing simple XML opening and closing tags, attributes, and entity references by the text specified in a user-selected style sheet and chosen by the designer to reflect the structure of the text through typography. The result is then passed to an HTML renderer (i.e., a web browser) for display.

The style sheet dictates how to interpret tags, attributes, and entity references as they are encountered in a left-to-right traversal of the text to be displayed, transforming the string into HTML. An important characteristic of the formatter is that not only can tags, attributes, and entity references be replaced by formatting or other symbols, but they can also be used to suppress arbitrarily large portions of text. Optionally, a matched opening and closing tag pair or a tag’s attribute can be interpreted to indicate that the text between the tags or the attribute’s value be treated as if it were an entity reference (effectively interpreting the opening tag as the start of an entity reference and the closing tag as its end).

Each line of a style sheet has three tab-separated fields, and as each tag is encountered in the text to be displayed, DRAGOMAN searches for a line in the style sheet matching the tag’s tagID in the first field (the indicator):

\[ \text{tagID action startText action endText} \]

Table 1: Equivalences for orthographic normalization

| a | c | i | d | j | k | m | e | t | t | i | c | n | e | f | u | i |

---

6 As stated earlier, the results of searching for phrases will miss instances interrupted by tags.

7 All reported performance statistics are based on single-threaded execution under Ubuntu 16.04 on a shared 2 CPU - Intel E5-2697 v3 Xeon machine with 256GB Ram.
If the encountered tag is a start tag, the first of the two actions is activated, and if it is an end tag, the second is activated. Either or both startText and endText may be empty strings. If the one-character action is a quotation mark, the tag is replaced by the startText or endText, respectively, subject to print suppression; using a minus sign in place of the quotation mark dictates that printing of text be suppressed, a plus sign dictates that printing of text be restarted, an asterisk dictates that the replacement text be printed regardless of print suppression, and an ampersand dictates that the tag content be interpreted as if it were an entity reference. Except for the last of these actions, there is no reliance on tags in the text being properly matched. Tags not matching any indicator in the table are printed as text (subject to print suppression) by replacing the left angle bracket by `<` so that it is interpreted as character data rather than as a tag.

Attributes are handled similarly, except that attributes are identified by the indicator tagID.attrname; the actions (text replacement, print suppression, etc.) occur before and after the attribute value, respectively; and attributes not matching any indicator have their values suppressed by default. Entity references are also handled similarly to tags, except that there is only one action specified per indicator, and it must begin with a quotation mark to indicate text replacement.

A style sheet has the following structure:

- It begins with a plus or minus, signalling whether print is initially on or suppressed.
- After a line with two percent signs, it lists tags or attributes in arbitrary order, with a pair of actions specified for each.
- After another line with two percent signs, it lists entities in arbitrary order, with a single action specified for each.

For the simple style sheet shown in Figure 6, printing is initially on, the content between match tags are displayed in red, d:tract start and end tags are simply ignored (the tags are replaced by empty strings), a line break is inserted after printing the value of the attribute val on a d:tract tag, the printing of any content occurring between greek start and end tags (inclusive) is suppressed, and the entity reference &mdash; is passed on to the HTML renderer as an entity reference, namely &mdash;.

All other tags and entity references are passed to the renderer as character data and any other attributes on the d:tract and greek tags are ignored.

For the Alberti Magni e-corpus, style sheets were created to display matched text in an easily readable form, either before or after the light editing of the works. In addition, a style sheet was created to display text with all tags uninterpreted so that the exact structure is visible. Printing the same paragraph with each of these style sheets is shown in Figure 7. Two additional style sheets were created for displaying large portions of the documents. The first of these suppresses the printing of most text fields but preserves section headings and titles, with suitable inclusion of markup for unordered lists to display a text’s structure, and displays a red asterisk for every match (so that each asterisk appears underneath the heading for the section within which the match occurs), as illustrated by the output fragment in Figure 8; in response to clicking on an asterisk, the system displays the text page on which the corresponding match occurs. The second additional style sheet preserves the single- and two-column formatting of the source texts, as shown in Figure 9.

Like Lector, on which it is based, Dragoman uses a “best-effort” approach to interpreting markup. Unlike CSS [15], it does not rely on the tagged text conforming to well-formed XML in order to display text. Instead, any errors in markup are ignored, relying instead on the user’s interpretation of the rendered display to discover errors in the input text or in the style sheet. This forgiving nature has repeated proven especially useful during periods in a project’s life cycle in which the code for structure recognition (Section 3) is under development or undergoing change.
SUMMA THEOLOGIAE
PARS I

PROLOGUS.
EXPLICIT PROLOGUS.

TRACTATUS I.
DE SCIENTIA THEOLOGIAE.

QUAESTIO I.
An theologam sive scientiam?

QUAESTIO II.
Quid sit theologam secundum definitionem?

QUAESTIO III.
De quo sit theologam ut de subjecto?

MEMBRUM I.
De subjecto theologiam sequatur positiones assignat.

MEMBRUM II.
Theologiam sive scientiam unam vel plures? Est. Qua unum sit una, ut una est?

MEMBRUM III.
Utum theologam sit scientia practica vel theoretica?

MEMBRUM IV.
Utum theologam sit scientia universalis, vel particularis?

QUAESTIO IV.
Utum theologam sit scientia ad alios scientias sequens?

Figure 8: Search for "licit": Text suppression and hyperlink generation with a style sheet

Super lucanum (1)
(ed. Breguet, 1894)

/IN SACROSANCTA EVANGELIA LUXULNATA EXPOSITIO.

PROEMIUM BERTI LUCAE IN EVANGELIUM S. PAV.

1. Quoniam quidem multitae consu lat sunt ordinari narrationem que in scriptis completar sunt recurr.

2. Siunt traditaque nobis qui ab inicito ipsi viderant, et ministri Fussi sunt sermonis isti.

3. Visum est et nihil, essecor omnia a principio diligenterus, ex ordine tibi scribera, optime Theophila.

4. Ut cognoscas eorum vestrum, de quibus estudium as, veritatem.

2a/[IN PROEMIUM LUCAE ENARRATIO]

Tuti luui operi Lucas presenti proemium, in quo factum quaque que facienda sunt in proemio egiudicet operis : pointing it ad scribendum cogerent, ad incoherent materiam de quo scribendum est perfectionem auctorit : formam quan tenet in scribendo, et exim. Omnis enim napieris scriptor presentit finem, quem contingere vult scribendo : quem cum contingit, perfectam est scriptum.

Necessitas notum cogeret, est multitud Pomo-Evangelistarum causar propriar

 Damnit in multitudinem et in multas sectas se diversitas multum haeretic.

Judaea, XV, 4 et 5, Volpe Sammonis caputis habent, caudis colligatit, hoc est, intentione veritatis impugnandae.


Figure 9: Using a style sheet that preserves columns

6 RETROSPECTIVE AND FURTHER WORK

With the ubiquity of search engines and web browsers, it may seem that all retrieval applications can be easily addressed with off-the-shelf tools. However, scholars in the humanities have demanding needs for text search and display. Some of these have been addressed by the software developed for the Alberti Magni e-corpus project.\(^8\) In particular, after preparing the scholar’s texts in a suitable XML-tagged form, a system built on top of sgrep for search and Dragoman for display can address many of those needs.\(^9\)

Alternative XML-aware search engines (such as BaseX \([20]\), eXist \([21]\), Wumpus \([3]\), or XQEngine \([16]\)) could equally well have been used in this project, simplifying some solutions but requiring more effort to address other concerns. For example, using an XQuery engine with full-text search capability \([12]\), would allow phrase searches across tags, thus eliminating the shortcoming in using sgrep when confronted by phrase instances that cross page and column breaks and across shifts to and from italic and bold text. However, because pages and columns form a hierarchical structure that is not nested with respect to logical text structures from books and chapters down to paragraphs and sentences, it is necessary to use milestone tags; thus searching within pages or columns or displaying the contents of a single page or column is less convenient with XQuery than it is with sgrep.\(^10\) No matter which engine is chosen, code must be implemented to address the normalization of Latin orthography, and for word-based (as opposed to region-based) systems, Latin-specific stemming rules must be implemented.

In retrospect, would it have been preferable to use XQuery in place of sgrep to serve as the query language for building a search engine for the Alberti Magni e-corpus? Because XQuery is defined by a suite of W3C recommendations \([33]\), adopting that language provides the option of choosing among several well-maintained software implementations and the likelihood that such software will continue to be available for the foreseeable future. XQuery is a far more powerful language than sgrep, which is restricted to identifying and extracting regions from a document in the order that they appear. The computational power of sgrep is therefore more similar to XPath Version 1.0 \([4]\) (but with the addition of intersection and difference of sets and not being restricted to XML) than it is to XQuery. However, the needs of the scholars wishing to access the Alberti Magni e-corpus are for a simple query interface that finds regions of text from a collection of documents and presents matches to queries in the order that they appear in those documents, and this computational requirement does not require the expressive power of XQuery. Were a similar project to re-start today, serious consideration should be given to building the engine with software that supports XPath Version 3.0 or XQuery, but using a simple region algebra such as the one supported by sgrep (or Wumpus, if the corpus is large enough to require an index) should also receive due consideration. The amount of implementation effort to connect the user interface to the search software is likely to be quite similar for both approaches.

Converting matched text from XML to an HTML display form can also be implemented using XML technology based on W3C recommendations, namely CSS \([15]\). As is true for search, the needs for

---

\(^8\) The software is freely available at https://github.com/fwtompa/e-corpusSE.

\(^9\) One scholar recently wrote: “I am using the engine a lot these days for my research, and it is extraordinarily useful to me. “ Another posted the comment: “The Alberti Magni E-Corpus ... [has] some very nice search features. (Which is important, because Google Books is sometimes really crazy about its word recognition in modern languages, much less in Latin.)” A web site states: “Teresa di Alberto Magno scaricabile in formato pdf e, soprattutto, liberamente ricercabile on line ... si tratta di uno strumento che avrà ancora per molti una sua utilità.”

the Alberti Magni e-corpus do not require the computational power of general transformations. Instead, DRAGOMAN uses only a simple tag interpretation algorithm that is easily explained. The tasks of writing several style sheets and controlling which style sheet to invoke is required, regardless of the implementation technology.

The most difficult part of starting a project with a new corpus is to convert the text into XML that reflects its logical structure, an extremely challenging task when physical layout must be interpreted [8], but also quite challenging when the input is plain text with embedded font information. In most of the text, each feature to be tagged can be recognized fairly easily, but unexpected difficulties arise when the features overlap in unanticipated ways. These problems do not result from the choice of software, but rather they are inherent in working with human-generated data: regardless of the approach taken to recognize structure, every form of feature interleaving that appears in the corpus must be accommodated by the conversion system. In several projects, we have successfully written converters that process the input sequentially in multiple passes. An alternative approach worthy of investigation is to drive the recognition process by a grammar that reflects the structures to be recognized. Unfortunately, developing such grammars for documents that have not yet been transformed into XML may be an equally challenging task due to the presence of many unanticipated exceptions that are commonly present in documents that were written over decades using just pen and paper [34]. Based on his experiences during several projects, Cummings has similarly observed:

"The process of using an incremental ad hoc intermediate format is quite common in such conversions. In many cases it is difficult to know very much about the structure of the document until much of the conversion has been done, hence an incremental approach allows a greater ease of recovery from mistaken conversion steps. However, it highlights a basic problem in such attempts to manipulate legacy files: that until they are in some format for which tools exist, it is difficult to examine or validate their structure." [6]

The specific tag set chosen for the Alberti Magni e-corpus can be improved. It has several idiosyncratic aspects, such as the chosen division names (Figure 2) and the use of <frag> and <cont> tags to represent words split across page and column breaks. If the scholars responsible for managing the e-corpus wish to share the tagged documents with other scholars, they will be well-served to first transform the tagging to be TEI-compliant [29]. Because the corpus has already been structured as well-formed XML, this should be far simpler than applying tags to a new document, and standard XML transformation tools, such as XSLT [17], can surely be used without difficulty. Thereafter, it may be worthwhile to develop a DTD [24] or XSD [22, 25] schema that describes the constraints on document structures found in the collection.

When the Alberti Magni e-corpus project was started, it was intended that the software should be easily applied to any scholar’s corpus with minimal customization. Both grep and DRAGOMAN provide simple mechanisms for adapting the search and display to new corpora. Therefore, the system could easily be used by scholars who work with other document collections. However, the functionality that is provided by the code base that ties these two components together with the query interface is specific to the needs of the Albertus Magnus scholars: the various modes of search, the requirement to display snippets with location information, and the ability to browse documents at most one page at a time may not fulfill all the needs of other scholars. For example, the software provides no ability to rank results, no ability to pose a query in which some terms use normalized spellings and others do not, no ability to compare two documents directly (e.g., to find similar passages, to analyze text differences, or to examine primary and secondary sources together), no ability to link a passage in a document to its critical apparatus (except by proximity), no ability to attach new annotations to a document, and so forth. A more widely acceptable scholars’ search engine still requires much work.

ACKNOWLEDGMENTS

User requirements for the software were developed in consultation with University of Waterloo Professors Delbert Russell (for Campsey) and Bruno Tremblay (for Albertus). The implementation benefitted from assistance with earlier versions from two (then undergraduate) students: Jingchi (Evan) Chen (January 2004-April 2004 for the Campsey project) and Shadab Ul Rashid (May 2008-December 2008 for the Albert Magni e-corpus project) and from repeated discussions with Andrew Kane, who also provided valuable feedback to improve this paper. Facilities were provided by the University of Waterloo, and financial support for the students was provided by NSERC, the Natural Sciences and Engineering Research Council of Canada, and by CFI, the Canada Foundation for Innovation.

REFERENCES


