Shortest-Substring Retrieval and Ranking

CHARLES L. A. CLARKE
University of Toronto
and
GORDON V. CORMACK
University of Waterloo

We present a model for arbitrary passage retrieval using Boolean queries. The model is applied to the task of ranking documents, or other structural elements, in the order of their expected relevance. Features such as phrase matching, truncation, and stemming integrate naturally into the model. Properties of Boolean algebra are obeyed, and the exact-match semantics of Boolean retrieval are preserved. Simple inverted-list file structures provide an efficient implementation. Retrieval effectiveness is comparable to that of standard ranking techniques. Since global statistics are not used, the method is of particular value in distributed environments. Since ranking is based on arbitrary passages, the structural elements to be ranked may be specified at query time and do not need to be restricted to predefined elements.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation* (efficiency and effectiveness)

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Boolean retrieval model, passage retrieval, relevance ranking

1. INTRODUCTION

A major benefit of the Boolean information retrieval model is its ability to specify exact retrieval results. Queries have a well-defined semantics, and only documents matching a query are returned to the user. When there are few matching documents, and the matching documents are short, it is

This work was supported by Communications and Information Technology Ontario, the Natural Sciences and Engineering Research Council of Canada, and the University of Toronto.
Authors' addresses: C. L. A. Clarke, Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario M5S 3G4, Canada; email: clclarke@eecg.toronto.edu; G. V. Cormack, Department of Computer Science, University of Waterloo, 10 King's College Road, Waterloo, Ontario N2L 3G1, Canada; email: gvcormac@plg.uwaterloo.ca.

reasonably easy for a user to identify the documents, or parts of documents, that are relevant. Unfortunately, a query may match many documents, and the matching documents may be long. If a query matches many documents, the documents should be ranked in the order of their expected relevance. If a query matches long documents, the passages that are most likely to be of interest should be identified. In this article, we present novel methods for addressing both of these problems.

A basic Boolean query operates on sets of documents [Salton and McGill 1983]. Initial document sets are specified by keywords and are combined according to the operations in the Boolean query, with `AND` specifying set intersection and `OR` specifying set union. The query

```
"platypus" AND ("aardvark" OR "antelope")
```

matches documents that contain the word "platypus" and at least one of "aardvark" or "antelope". This basic model is often augmented with qualifiers that restrict the Boolean operations to document subfields, such as the title, abstract, or list of authors, as well as with a variety of keyword operators, including operators for specifying proximity, truncation, and concatenation. Augmented Boolean queries of this form were supported by the earliest information retrieval systems and continue to survive in many of the latest digital libraries and World Wide Web search engines.

The Boolean model has been the subject of long-standing criticism as a user interface language for information retrieval systems [Brenner 1996; Cooper 1988; Turtle 1994]. This criticism has primarily focused on the lack of support for relevance ranking and on the difficulties casual users experience when formulating Boolean queries. The alternative, formulating queries as unstructured term vectors and comparing them to documents using similarity measures, provides ranking but requires the exact-match property to be abandoned [Salton and McGill 1983]. Despite criticism, the Boolean model continues to be of both academic and commercial interest and importance [Greiff et al. 1997; Hearst 1996; Lesk 1997; Li and Danzig 1997; Thomas et al. 1999].

A previous article [Clarke et al. 1995a] unified and extended many of the disparate elements of augmented Boolean retrieval languages as part of a simple and expressive algebra for general structured text search. Documents in a database are treated as a single sequence of words, uninterrupted by document boundaries or other structure, with full indexing of word positions. Structural elements, including the start and end positions of sentences, paragraphs, lines, pages, documents, titles, and author lists, are represented by tags indexed between positions in the word sequence. The solution to a query is a set of intervals over this word sequence. Operators are provided for combining and filtering the interval sets. Together with the data representation, the operators provide a closed algebra for search and retrieval of structured text. A particular feature of the approach is its ability to handle heterogeneous structure in a consistent and uniform fashion [Clarke et al. 1995b].

Many techniques for ranking Boolean retrieval results have been proposed. A simple approach is to use the Boolean query to select a document set, and then strip the operators out of the query and use the resulting term vector to rank the document set using the cosine measure or another standard similarity measure [Noreault et al. 1977]. Unfortunately, since the Boolean operators are ignored, term relationships described by the operators are not reflected in the ranking, and the results can be counterintuitive [Bookstein 1978; Radecki 1982].

Fuzzy set theory has been widely applied to the problem [Bookstein 1980; Buell and Kraft 1981; Kerre et al. 1986; Tahani 1978]. In its simplest form, a weight indicates the degree to which a document is "about" a query term. Weights are combined according to the operations in a Boolean query, with `MIN` taking the place of `AND` and `MAX` taking the place of `OR`. This approach has been criticized for producing inappropriate output, since the value of the combined weight is determined solely by the weight associated with a single operand [Kim et al. 1993; Robertson 1978].

Various models for "soft" Boolean operators have been proposed that abandon the exact-match semantics of Boolean retrieval [Greiff et al. 1997; Kim et al. 1993; Lee 1994; Lee et al. 1993; Rousseau 1990; Salton et al. 1983; Waller and Kraft 1979]. The most notable, the *p-norm* model, provides a spectrum of ranking techniques, in which the relationships described by the Boolean operators can be varied in importance [Salton et al. 1983]. At one extreme, Boolean relationships are ignored, and the approach is equivalent to the cosine measure. At the other extreme, the approach is equivalent to the fuzzy set model, with documents matching the Boolean expression exactly. Intermediate between these extremes, the approach has been shown to produce results that can be superior to the cosine similarity measure, the fuzzy set model, and other soft Boolean models. Recently, Greiff et al. [1997] introduced a broad class of soft Boolean operators in the context of the inference network model [Turtle and Croft 1992] that can perform as well as the p-norm operators.

Regrettably, the soft Boolean operators do not obey the properties of standard Boolean algebra, including the distributive and associative laws [Lee 1994; Salton et al. 1983]. As a result, equivalent Boolean expressions can produce different document rankings.

The present article contains two main sections, Sections 2 and 3. The first develops a theory of Boolean queries over arbitrary text intervals, continuing previous research [Clarke et al. 1995a]. The shortest-substring restriction, which is central to the theory, is formally defined, and proofs are given for its basic properties. Definitions for the Boolean operators under the shortest-substring restriction are stated, and the expected commutative, associative, and distributive properties are demonstrated. Two algorithms for implementation are presented; they are evaluated analytically and experimentally.

Section 3 applies the theory to the problem of relevance ranking. Documents, or other structural elements to be ranked, are scored on the basis of the size and number of solution intervals contained within them. Properties

Bells[1]

At[2] six[3] o'[4] clock[5] of[6] an[7] autumn[8] dusk[9]
With[10] the[11] sky[12] in[13] the[14] west[15] a[16] rusty[17] red,[18]
The[19] bells[20] of[21] the[22] mission[23] down[24] in[25] the[26] valley[27]
Cry[28] out[29] that[30] the[31] day[32] is[33] dead.[34]

The[35] first[36] star[37] pricks[38] as[39] sharp[40] as[41] steel[42] —
Why[43] am[44] I[45] suddenly[46] so[47] cold?[48]
Three[49] bells,[50] each[51] with[52] a[53] separate[54] sound[55]
Clang[56] in[57] the[58] valley,[59] wearily[60] tolled.[61]

Bells[62] in[63] Venice,[64] bells[65] at[66] sea,[67]
Bells[68] in[69] the[70] valley[71] heavy[72] and[73] slow[74] —
There[75] is[76] no[77] place[78] over[79] the[80] crowded[81] world[82]
Where[83] I[84] can[85] forget[86] that[87] the[88] days[89] go.[90]

(Sara[91] Teasdale[92])

Fig. 1. Example text. Superscripts indicate word positions.

and parameters of the method are explored. An earlier version of the method was introduced in the context of experimental work for the Fourth Text Retrieval Conference (TREC-4) [Clarke et al. 1995c] and has been used successfully in subsequent TREC experiments [Clarke and Cormack 1996; Cormack et al. 1997; 1998]. The TREC conference papers provide only a brief summary of the method, since they focus primarily on the experimental work. The present article provides a complete description of the method, along with related theory and the details required for efficient implementation. Furthermore, our TREC experiments were based on Boolean queries created manually by our research group. In the present article, a set of independently created test queries is used as the basis for evaluation and comparison.

2. SHORTEST-SUBSTRING RETRIEVAL

For search and retrieval purposes the text is viewed as a string of symbols $c_1 \ldots c_N$ drawn from a *text alphabet* Σ. The short poem [Teasdale 1920] in Figure 1 is used as a recurring example. Here, a reasonable choice for the text alphabet consists of the English words appearing in the text with all characters mapped to lowercase.

$$\Sigma = \{\ \texttt{a}, \texttt{am}, \texttt{an}, \texttt{and}, \texttt{as}, \texttt{at}, \texttt{autumn}, \texttt{bells},$$
$$\texttt{can}, \texttt{clang}, \texttt{clock}, \texttt{cold}, \texttt{crowded}, \texttt{cry}, \texttt{day}, \ldots \}$$

In the example, the superscripts indicate positions in the text sequence. An index function $\mathcal{I}$ maps each symbol in the text alphabet to the set of positions in the database string where the symbol appears ($\mathcal{I} : \Sigma \rightarrow 2^{\{1 \ldots N\}}$). In the example, we have

$$\mathcal{I}(\texttt{"bells"}) = \{1,\ 20,\ 50,\ 62,\ 65,\ 68\}.$$

The discussion of the current section assumes that document boundaries are ignored; multiple documents are treated as if concatenated into a single long document.

The result of a search is represented as a set of ranges or *extents* over the string that forms the database. Each extent is of the form (p, q), where p is the start position of the extent, while q is the end position of the extent.

An extent (p, q) *overlaps* an extent (p', q') if either $p' \leq p \leq q'$ or $p' \leq q \leq q'$ but not both. An extent (p, q) is *nested* in an extent (p', q') if $(p, q) \neq (p', q')$ and $p' \leq p \leq q \leq q'$. If $a = (p, q)$ and $b = (p', q')$ are extents, the notation $a \sqsubset b$ indicates that a nests in b; the notation $a \sqsubseteq b$ indicates that a is *contained in* b—that either a and b are equal or that a nests in b. Extents form a partial order under $\sqsubseteq$.

A solution to a query is a set of extents. Over the database string $c_1..c_N$, the range of the index function $\mathcal{I}$ is limited to N positions. However, there are $O(N^2)$ extents over this same range—every (p, q) such that $1 \leq p \leq q \leq N$. Depending on the query, any of these $O(N^2)$ extents could be a candidate for inclusion in the query's solution set. For example, given a query for a particular word (e.g., `"bells"`), every extent that overlaps an occurrence of the word might reasonably be viewed as a potential member of the solution set, including the extent $(1, N)$ corresponding to the entire database. Similarly, if a query consists of the conjunction of two terms (e.g., `"bells" AND "valley"`) every extent that contains both terms might be considered as a potential solution extent.

Many of these extents overlap and nest. In order to reduce the number of extents that result from a search, we do not allow extents in the solution set to have other solution extents nested within them. However, solution extents are permitted to overlap. The value and significance of this simple reduction rule is demonstrated throughout the remainder of Section 2. In particular, it enables the efficient evaluation of Boolean queries over arbitrary text intervals, while maintaining the standard properties of Boolean algebra and guaranteeing that the resulting solution set must be linear in size. As we shall see, nothing is really lost by this reduction, since the resulting solution set is essentially a compact representation of the full solution set. This approach of eliminating nested extents is termed the *shortest-substring search model*; the resulting set is termed a *generalized concordance list*.

2.1 Generalized Concordance Lists

A set of nonnesting extents is referred to as a generalized concordance list, or simply *GC-list*, after the earlier *concordance lists* of Burkowski [1992]. In the case of a search for a single word that occurs once in the database, the corresponding generalized concordance list contains a single extent of unit length that begins and ends at the word's position. The index function

$\mathcal{I}$ may be viewed as mapping symbols in the index alphabet onto GC-lists: the elements of the results are interpreted as extents that begin and end at a single position. For example,

$$\mathcal{I}(\text{“bells”}) = \{(1, 1), (20, 20), (50, 50), (62, 62), (65, 65), (68, 68)\}.$$

By viewing the index function as producing extents, it may be augmented to encompass phrase matching. For example,

$$\mathcal{I}(\text{“the valley”}) = \{(26, 27), (58, 59), (70, 71)\}.$$

Similar extensions may be used to support truncation and stemming.

The reduction of a set of extents S to a generalized concordance list may be formalized as a function $\mathcal{G}(S)$:

$$\mathcal{G}(S) = \{a \mid a \in S \text{ and } \nexists\, b \in S \text{ such that } b \sqsubset a\}.$$

A set S of extents is a GC-list if and only if

$$S = \mathcal{G}(S).$$

We next present three simple results that establish the basic properties of GC-lists. These properties will be required for the results and discussion of later sections.

The elements of a GC-list are totally ordered both by their start positions and by their end positions. These total orders are identical; that is, if (p, q) and (p', q') are extents from a GC-list then $p < p'$ if and only if $q < q'$.

THEOREM 2.1 *The elements of a GC-list are identically and totally ordered by their start and end positions.*

PROOF. Let S be a GC-list, with $a = (p, q) \in S$ and $b = (p', q') \in S$. If $p \geq p'$ and $q < q'$, or if $p > p'$ and $q = q'$, then $a \sqsubset b$, and S would not be a GC-list; if $p' \geq p$ and $q' < q$, or if $p' > p$ and $q' = q$, then $b \sqsubset a$, and S would not be a GC-list. Therefore, either (1) $p < p'$ and $q < q'$ (and therefore $a < b$) or (2) $p > p'$ and $q > q'$ (and therefore $a > b$) or (3) $p = p'$ and $q = q'$ (and therefore $a = b$). □

The next result establishes the linearity of solution sets.

THEOREM 2.2 *If a GC-list S represents the solution to a query over the database string $c_1 \ldots c_N$ then $|S| \leq N$.*

PROOF. If $|S| > N$ then two distinct elements of S, (u, v) and (u, w), must share a common start position. Since these elements are distinct, $v \neq w$. If $v < w$ then $(u, v) \sqsubset (u, w)$. If $w < v$ then $(u, w) \sqsubset (u, v)$. In either case S cannot be a GC-list. □

Finally, we show a result used in later proofs:

THEOREM 2.3 *If A is a set of extents with $a \in A$ and $a \notin \mathcal{G}(\mathcal{A})$ then there exists $a' \in \mathcal{G}(A)$ such that $a' \sqsubset a$.*

2.2 Boolean Operators

A natural way of extending the Boolean operators, which are normally interpreted over sets of documents, to arbitrary text intervals is to treat each of the $N(N + 1)/2$ nonempty substrings of the database as a separate document for retrieval purposes. Given a query Q, an extent (p, q) *satisfies* Q if the substring of the database beginning at position p and ending at q would match the query if the substring was treated as a document under the standard set-based Boolean model. More precisely,

(1) an extent (p, q) satisfies a query Q_1 `AND` Q_2 if the extent satisfies Q_1 and satisfies Q_2,

(2) an extent (p, q) satisfies a query Q_1 `OR` Q_2 if the extent satisfies Q_1 or satisfies Q_2, and

(3) an extent (p, q) satisfies a term T if the term occurs in the interval of text corresponding to the extent.

Potentially, a term might be any localized feature of the text, such as a word or phrase, and may be recognized under stemming or truncation.

Once again, the shortest-substring rule may be used to reduce the set of extents that satisfy a query, a set which may be quadratic in size, to a set that is at worst linear in size (Theorem 2). Two operators, "one of" ($\triangledown$), corresponding to `OR`, and "both of" ($\triangle$), corresponding to `AND`, are defined to effect this shortest-substring model of Boolean search.

$$A \triangledown B = \mathcal{G}(\{c \mid \exists a \in A \text{ such that } a \sqsubseteq c \text{ or } \exists b \in B \text{ such that } b \sqsubseteq c\})$$

$$A \triangle B = \mathcal{G}(\{c \mid \exists a \in A \text{ such that } a \sqsubseteq c \text{ and } \exists b \in B \text{ such that } b \sqsubseteq c\})$$

Under the shortest-substring model the query

```
"bells" AND ("sky" OR "valley")
```

is interpreted as

$$\mathcal{I}(\text{"bells"}) \triangle (\mathcal{I}(\text{"sky"}) \triangledown \mathcal{I}(\text{"valley"}))$$

which specifies the set of extents

{(1,12), (12,20), (20,27), (27,50), (50,59), (59,62), (68,71)}

over the text in Figure 1. The extents (35,61), (62,71), (1,90) and many others all satisfy the query, but are eliminated by the shortest-substring rule.

Despite the elimination of many extents, the GC-list resulting from the application of the shortest-substring rule still fully characterizes the set of extents that satisfy a query. An extent satisfies a query if and only if it contains an extent in the GC-list. The GC-list is the smallest set that provides this characterization.

2.3 Properties

The shortest-substring rule eliminates intervals satisfying a query. It is not obvious that this process does not compromise the basic properties of Boolean algebra, particularly associativity and distributivity.

Commutativity follows immediately from the commutativity of "and" and "or" in the definitions of the operators.

THEOREM 2.4 *If A and B are GC-lists, then*

(1) $A \triangle B = B \triangle A$ *and*

(2) $A \triangledown B = B \triangledown A$.

PROOF. Immediate from the definitions. □

In order to establish the associative and distributive properties of the operators, we first establish results concerning the definitions themselves. The definitions may be viewed as consisting of two steps: (1) a predicate that selects from the universe of all extents, followed by (2) an application of the shortest-substring rule (the $\mathcal{G}$ function). In order to discuss each step separately we define two new operators, one for each of the original operators, that represents only the first of these two steps. The operators are

$$A \triangle^* B = \{c \mid \exists a \in A \text{ such that } a \sqsubseteq c \text{ and } \exists b \in B \text{ such that } b \sqsubseteq c\}$$

$$A \triangledown^* B = \{c \mid \exists a \in A \text{ such that } a \sqsubseteq c \text{ or } \exists b \in B \text{ such that } b \sqsubseteq c\}$$

so that

$$A \triangle B = \mathcal{G}(A \triangle^* B)$$

$$A \triangledown B = \mathcal{G}(A \triangledown^* B).$$

The commutativity of the $\triangle^*$ and $\triangledown^*$ operators is immediate from the definitions, as are the following associative and distributive properties:

THEOREM 2.5 *If A, B, and C are sets of extents, then*

(1) $(A \triangle^* B) \triangle^* C = A \triangle^* (B \triangle^* C)$ *and*

(2) $(A \triangledown^* B) \triangledown^* C = A \triangledown^* (B \triangledown^* C)$.

PROOF. Immediate from the definitions. □

THEOREM 2.6 *If A, B, and C are sets of extents, then*

(1) $(A \triangledown^* B) \triangle^* C = (A \triangle C) \triangledown^* (B \triangle^* C)$ *and*

(2) $(A \triangle^* B) \triangledown^* C = (A \triangledown^* C) \triangle^* (B \triangledown^* C)$.

PROOF. Immediate from the definitions. □

The key observation concerning these operators is that their results are insensitive to the application of the $\mathcal{G}$ function to their arguments. The following theorem formally establishes the property.

THEOREM 2.7 *If A and B are sets of extents, then*

(1) $A \triangle^* B = \mathcal{G}(A) \triangle^* B$,

(2) $A \triangle^* B = A \triangle^* \mathcal{G}(B)$,

(3) $A \triangledown^* B = \mathcal{G}(A) \triangledown^* B$, *and*

(4) $A \triangledown^* B = A \triangledown^* \mathcal{G}(B)$.

PROOF.

(1) If c is any element of $A \triangle^* B$ then $\exists a \in A$ and $\exists b \in B$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. By Theorem 2.3, $\exists a' \in \mathcal{G}(A)$ such that $a' \sqsubseteq a \sqsubseteq c$. Therefore $c \in \mathcal{G}(A) \triangle^* B$, implying $A \triangle^* B \subseteq \mathcal{G}(A) \triangle^* B$.

If c is any element of $\mathcal{G}(A) \triangle^* B$ then $\exists a \in \mathcal{G}(A)$ and $\exists b \in B$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. Since $\mathcal{G}(A) \subseteq A$, we have $a \in A$. Therefore $c \in A \triangle^* B$, implying $\mathcal{G}(A) \triangle^* B \subseteq A \triangle^* B$.

(2) Follows from part (1) and the commutativity of $\triangle^*$.

(3) If c is any element of $A \triangledown^* B$ then $\exists a \in A$ such that $a \sqsubseteq c$ or $\exists b \in B$ such that $b \sqsubseteq c$. If $\exists a \in A$ such that $a \sqsubseteq c$ then by Theorem 2.3 $\exists a' \in \mathcal{G}(A)$ such that $a' \sqsubseteq a \sqsubseteq c$. Therefore $c \in \mathcal{G}(A) \triangledown^* B$, implying $A \triangledown^* B \subseteq \mathcal{G}(A) \triangledown^* B$.

If c is any elment of $\mathcal{G}(A) \triangledown^* B$ then $\exists a \in \mathcal{G}(A)$ such that $a \sqsubseteq c$ or $\exists b \in B$ such that $b \sqsubseteq c$. If $\exists a \in \mathcal{G}(A)$ such that $a \sqsubseteq c$ then, since $\mathcal{G}(A) \subseteq A$, we have $a \in A$. Therefore $c \in A \triangledown^* B$, implying $\mathcal{G}(A) \triangledown^* B \subseteq A \triangledown^* B$.

(4) Follows from part (3) and the commutativity of $\triangledown^*$. □

Proofs of associative and distributive properties are now straightforward. These proofs are very similar to one another. The basic proof technique is to first rewrite the left-hand side of each equation as the two-step application of a predicate followed by the function. Then Theorem 2.7 is used to eliminate all but the outermost application of the $\mathcal{G}$ function. The underlying predicate is then manipulated directly. Finally Theorem 2.7 is used to

restore the application of the function throughout the equation, leaving it in the desired form.

THEOREM 2.8 *If A, B, and C are GC-lists, then*

(1) $(A \triangle B) \triangle C = A \triangle (B \triangle C)$ *and*

(2) $(A \triangledown B) \triangledown C = A \triangledown (B \triangledown C)$.

PROOF.

(1)
$$\begin{aligned}
(A \triangle B) \triangle C &= \mathcal{G}(\mathcal{G}(A \triangle^* B) \triangle^* C) && \text{(Definitions)} \\
&= \mathcal{G}((A \triangle^* B) \triangle^* C) && \text{(Theorem 2.7, part (1))} \\
&= \mathcal{G}(A \triangle^* (B \triangle^* C)) && \text{(Theorem 2.5, part (1))} \\
&= \mathcal{G}(A \triangle^* \mathcal{G}(B \triangle^* C)) && \text{(Theorem 2.7, part (2))} \\
&= A \triangle (B \triangle C) && \text{(Definitions)}
\end{aligned}$$

(2)
$$\begin{aligned}
(A \triangledown B) \triangledown C &= \mathcal{G}(\mathcal{G}(A \triangledown^* B) \triangledown^* C) && \text{(Definitions)} \\
&= \mathcal{G}((A \triangledown^* B) \triangledown^* C) && \text{(Theorem 2.7, part (3))} \\
&= \mathcal{G}(A \triangledown^* (B \triangledown^* C)) && \text{(Theorem 2.5, part (2))} \\
&= \mathcal{G}(A \triangledown^* \mathcal{G}(B \triangledown^* C)) && \text{(Theorem 2.7, part (4))} \\
&= A \triangledown (B \triangledown C) && \text{(Definitions)} \quad \square
\end{aligned}$$

THEOREM 2.9 *If A, B, and C are GC-lists, then*

(1) $(A \triangledown B) \triangle C = (A \triangle C) \triangledown (B \triangle C)$ *and*

(2) $(A \triangle B) \triangledown C = (A \triangledown C) \triangle (B \triangledown C)$.

PROOF.

(1)
$$\begin{aligned}
(A \triangledown B) \triangle C &= \mathcal{G}(\mathcal{G}(A \triangledown^* B) \triangle^* C) && \text{(Definitions)} \\
&= \mathcal{G}((A \triangledown^* B) \triangle^* C) && \text{(Theorem 2.7, part (1))} \\
&= \mathcal{G}((A \triangle^* C) \triangledown^* (B \triangle^* C)) && \text{(Theorem 2.6, part (1))} \\
&= \mathcal{G}(\mathcal{G}(A \triangle^* C) \triangledown^* \mathcal{G}(B \triangle^* C)) && \text{(Theorem 2.7, parts (3) and (4))} \\
&= (A \triangle C) \triangledown (B \triangle C) && \text{(Definitions)}
\end{aligned}$$

$$
\begin{aligned}
(2)\quad (A \triangle B) \triangledown C &= \mathcal{G}(\mathcal{G}(A \triangle^* B) \triangledown^* C) && \text{(Definitions)} \\
&= \mathcal{G}((A \triangle^* B) \triangledown^* C) && \text{(Theorem 2.7, part (3))} \\
&= \mathcal{G}((A \triangledown^* C) \triangle^* (B \triangledown^* C)) && \text{(Theorem 2.6, part (1))} \\
&= \mathcal{G}(\mathcal{G}(A \triangledown^* C) \triangle^* \mathcal{G}(B \triangledown^* C)) && \text{(Theorem 2.7, parts (1) and (2))} \\
&= (A \triangledown C) \triangle (B \triangledown C) && \text{(Definitions)} \quad \square
\end{aligned}
$$

Multiplicative and additive identities are

$$\mathbf{T} = \{(1,1), (2,2), \ldots\}$$

$$\mathbf{F} = \{\}$$

T is the set of all extents with unit length; **F** is the empty GC-list. It is straightforward to show the following:

$$A \triangle \mathbf{T} = A$$

$$A \triangle \mathbf{F} = \mathbf{F}$$

$$A \triangledown \mathbf{T} = \mathbf{T}$$

$$A \triangledown \mathbf{F} = A$$

For any two GC-lists A and B, define $A \leq B$ if and only if for every $a \in A$ there exists $b \in B$ such that $b \sqsubseteq a$. The properties proved in this section demonstrate that GC-lists form a distributive lattice [MacLane and Birkhoff 1967] under this partial order, with meet operator $\triangle$, join operator $\triangledown$, greatest element **T**, and least element **F**.

2.4 Computing Solution Sets

By Theorem 2.1, the elements of a GC-list are totally ordered. This total order may be used as the basis for indexing into GC-lists, which in turn forms the basis of an efficient algorithm for solving Boolean queries under the shortest-substring search model. The algorithm may be viewed as a generalization of the "index-skipping" algorithms used to optimize the evaluation of Boolean queries under the standard set-based model [Moffat and Zobel 1996; Turtle and Flood 1995].

The algorithm is based on two *access functions* defined over the database string $c_1..c_N$. The values computed by the access functions are defined as follows:

$$
r(S, k) = \begin{cases} q & \text{if } \exists (p, q) \in S \text{ such that } k \leq p \\ & \quad \text{and } \nexists (p', q') \in S \text{ such that } q' < q \text{ and } k \leq p' \\ N + 1 & \text{if } \nexists (p, q) \in S \text{ such that } k \leq p \end{cases}
$$

and

$$l(S, k) = \begin{cases} p & \text{if } \exists (p, q) \in S \text{ such that } k \geq q \\ & \text{and } \nexists (p', q') \in S \text{ such that } p' > p \text{ and } k \geq q' \\ 0 & \text{if } \nexists (p, q) \in S \text{ such that } k \geq q \end{cases}$$

where S is a GC-list, while k is a position in the database string. In these definitions, the extents $(0,0)$ and $(N + 1, N + 1)$ essentially act as "end of file" markers for GC-lists.

Informally, the function $r(S, k)$ returns the end position of the first extent in S that starts at or after position k, and $l(S, k)$ returns the start position of the last extent in S that ends at or before position k. Over the text in Figure 1, the solution to $\mathcal{I}(\text{"bells"})$ is the set of extents

$$\text{S} = \{(1,1), (20,20), (50,50), (62,62), (65,65), (68,68)\}.$$

Over this set, $r(S, 18) = 20$, $l(S, 64) = 62$, and $r(S, 69) = N + 1 = 93$.

We extend these access functions from GC-lists to queries in the following way: if Q is a query, and S is its solution set, then $r(Q, k) = r(S, k)$ and $l(Q, k) = l(S, k)$. For example, $l(\mathcal{I}(\text{"sky"}), 11) = 0$.

For an arbitrary query Q and position k it is possible to compute values for $r(Q, k)$ and $l(Q, k)$ using a simple bottom-up procedure. For a $\triangledown$ or $\triangle$ operation, the values of the access functions may be computed using recursive calls to the access functions for the operands. At the lowest level, standard inverted-list data structures may be used to build an efficient implementation of the access functions for the index function $\mathcal{I}$.

For the purposes of the theoretical analysis in this article, the implementation of the access functions for $\mathcal{I}$ is assumed to be based on the simplified file structures illustrated in Figure 2. A *dictionary* lists the symbols from the database alphabet and provides pointers into a *postings* file that lists the positions in the database string where the symbol occurs. The dictionary is sorted by the lexicographical order of the symbols in the database alphabet. For each symbol, the associated list of database positions—its *postings list*—is sorted in increasing order. The dictionary is maintained in memory, and a binary search is used to locate the entry for a particular symbol in $O(\log|\Sigma|)$ time, where $|\Sigma|$ is the size of the text alphabet. This dictionary look-up is a one-time cost incurred when the query is parsed and its internal representation is built. The postings file is held on disk. When the postings list for a symbol is required, it may be read into memory with a single disk operation, if it will fit; otherwise, it may be searched on disk. In any case, a binary search then implements the access functions with $O(\log m)$ efficiency, where m is the length of the postings list.

In general, the entire dictionary and the postings lists required to solve a query cannot always be held in available memory, and the total number of

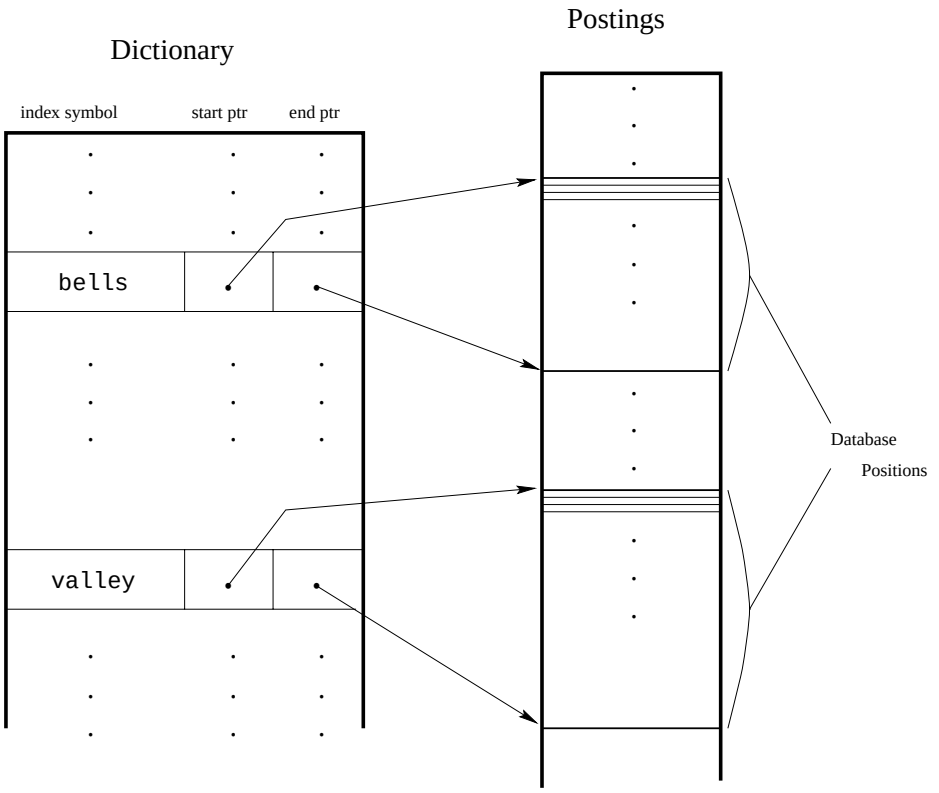

Fig. 2. Simplified file structures.

disk read operations becomes an important consideration in the implementation of the access functions. The performance figures reported later in the article are measured on a system that uses a modified organization of the basic inverted-list file structures designed to reduce the number of read operations required to implement the access functions [Clarke et al. 1994]. If the postings list for a symbol is sufficiently small, the organization allows the full postings list to be found and brought into memory with a single disk operation, including the dictionary search. If the postings list is large, the organization permits direct indexing to the position in the postings list that contains the solution to the access function, avoiding an on-disk binary search that may involve many disk seeks. The organization also permits dynamic insertions and deletions of index information, and can be used to implement the access functions for phrases.

For the $\triangledown$ and $\triangle$ operations, implementation of the access functions is defined by the equations of the following theorem:

THEOREM 2.10 *If A and B are GC-lists, then*

(1) $r(A \triangle B, k) = \max(r(A, k), r(B, k))$,

(2) $r(A \triangledown B, k) = \min(r(A, k), r(B, k))$,

(3) $l(A \triangle B, k) = \min(l(A, k), l(B, k))$, *and*

(4) $l(A \triangledown B, k) = \max(l(A, k), l(B, k))$.

PROOF. We provide details only for Eqs. (1) and (2). The details for Eqs. (3) and (4) are similar. For convenience, $(N + 1, N + 1)$ and $(0,0)$ are treated as members of all GC-lists involved, simplifying the definitions of r and l.

Let $q_A = r(A, k)$ and $q_B = r(B, k)$. By the definition of r, $\exists(p_A, q_A) \in A$ such that $k \leq p_A$ and $\nexists(p'_A, q'_A) \in A$ such that $q'_A < q_A$ and $k \leq p'_A$. Similarly, $\exists(p_B, q_B) \in B$ such that $k \leq p_B$ and $\nexists(p'_B, q'_B) \in B$ such that $q'_B < q_B$ and $k \leq p'_B$.

Proofs for Eqs. (1) and (2) now proceed separately:

Proof for Eq. (1). Let $q = r(A \triangle B, k)$. By the definition of r, $\exists(p, q) \in A \triangle B$ such that $k \leq p$ and $\nexists(p', q') \in A \triangle B$ such that $q' < q$ and $k \leq p'$. We now show that $q = \max(q_A, q_B)$.

Assume $q > \max(q_A, q_B)$. Since $k \leq p_A$ and $k \leq p_B$, $(k, \max(q_A, q_B)) \in A \triangle^* B$. By Theorem 2.3, $\exists(p', q') \in A \triangle B$ such that $(p', q') \sqsubseteq (k, \max(q_A, q_B))$, implying $q' < q$ and $k \leq p'$, a contradiction. Therefore $q \leq \max(q_A, q_B)$.

Now assume $q < \max(q_A, q_B)$, and without loss of generality assume $q_A = \max(q_A, q_B)$. Since $(p, q) \in A \triangle B$, $\exists(p'_A, q'_A) \in A$ such that $(p'_A, q'_A) \sqsubseteq (p, q)$. Thus, $\exists(p'_A, q'_A) \in A$ such that $q'_A \leq q < q_A$ and $k \leq \min(p_A, p_B) \leq p \leq p'_A$, which cannot hold, since $q_A = r(A, k)$. Therefore $q = \max(q_A, q_B)$.

Proof for Eq. (2). We begin by showing that $A \triangledown B \subseteq A \cup B$. First, assume $\exists c \in A \triangledown B$ such that $c \notin A \cup B$. Since $c \in A \triangledown B$, either $\exists a \in A$ such that $a \sqsubseteq c$ or $\exists b \in B$ such that $b \sqsubseteq c$. If $a \in A$ and $a \sqsubseteq c$ then $a \in A \triangledown^* B$ and $c \notin \mathcal{G}(A \triangledown^* B)$; if $b \in B$ and $b \sqsubseteq c$ then $b \in A \triangledown^* B$ and $c \notin \mathcal{G}(A \triangledown^* B)$. In either case, $c \notin A \triangledown B$, a contradiction. Therefore $A \triangledown B \subseteq A \cup B$.

Let $q = r(A \triangledown B, k)$. By the definition of r, $\exists(p, q) \in A \triangledown B$ such that $k \leq p$ and $\nexists(p', q') \in A \triangledown B$ such that $q' < q$ and $k \leq p'$. We now show that $q = \min(q_A, q_B)$.

Assume $q < \min(q_A, q_B)$, and without loss of generality assume $(p, q) \in A$. Since $q < q_A$ and $k \leq p$, $q_A \neq r(A, k)$, a contradiction. Therefore $q \geq \min(q_A, q_B)$.

Now, assume $q > \min(r(A, k), r(B, k))$, and without loss of generality assume $q_A = \min(r(A, k), r(B, k)) < q$. Since $(p_A, q_A) \in A$, $(p_A, q_A) \in A \triangledown^* B$. By Theorem 2.3, $\exists(p', q') \in A \triangledown B$ such that $(p', q') \sqsubseteq (p_A, q_A)$. There are now two cases: (1) if $p \leq p'$ then $(p', q') \sqsubset (p, q)$ and $(p, q) \notin A \triangledown B$, contradicting the defining property of GC-lists; (2) if $p > p'$ then

$q' \leq q_A < q$ and $k \leq p_A \leq p'$, implying $q \neq r(A \triangledown B, k)$, a contradiction. Therefore $q = \min(q_A, q_B)$. □

The two access functions r and l may be used in concert to compute the solution set for a query Q. We define a third access function τ that indexes into a GC-list and returns a complete extent, rather than a start or end position. If S is a GC-list, we define the value returned by the access function over the database string $c_1..c_N$ as follows:

$$\tau(S, k) = \begin{cases} (p, q) & \text{if } \exists (p, q) \in S \text{ such that } k \leq p \\ & \text{and } \nexists (p', q') \in S \text{ such that } q' < q \text{ and } k \leq p' \\ (N + 1, N + 1) & \text{if } \nexists (p, q) \in S \text{ such that } k \leq p \end{cases}$$

The access function τ returns the first extent from S that ends at or after position k. Once again, we extend the access function from GC-lists to queries, by defining $\tau(Q, k) = \tau(S, k)$, where S is the solution set for query Q. The relationship between the three access functions is given in the following theorem:

THEOREM 2.11 *Let S be a GC-list. Let $v = r(S, k)$. If $v \neq N + 1$, let $u = l(S, v)$; otherwise, let $u = N + 1$.*

$$\tau(S, k) = (u, v)$$

PROOF. If $r(S, k) = v = N + 1$ then $\nexists (p, q) \in S$ such that $k \leq p$. Therefore $\tau(S, k) = (N + 1, N + 1)$. If $r(S, k) = v \neq N + 1$ then $\exists (p, q) \in S$ such that $k \leq p$ and $\nexists (p', q') \in S$ such that $q' < q$ and $k \leq p'$, where $q = v$. Now, $(p, q) \in S$ with $p \leq v$. Assume $\exists (p', q') \in S$ such that $p' > p$ and $v \geq q'$. Since $v = q$, $(p', q') \sqsubset (p, q)$, contradicting for S the defining property of GC-lists. Therefore $p = l(S, v) = u$. Now, $(u, v) \in S$ with $k \leq u$ and $\nexists (p', q') \in S$ such that $q' < v$ and $k \leq p'$. Therefore $\tau(S, k) = (u, v)$. □

The algorithm of Figure 3 generates all elements of the GC-list representing the solution to a query Q. The elements are generated in order of increasing positions, using a call to the τ access function to generate each element. The τ access function is implemented in terms of the r and l access functions using the relationship given in Theorem 2.11. The r and l access functions are implemented using the recursive equations of Theorem 2.10 and the inverted-list file structures of Figure 2. As each extent is generated it may be output or stored for further processing. The efficiency of the algorithm is the subject of the next section.

For example, to compute the solution set for the query

```
ℐ("bells") △ (ℐ("sky") ▽ ℐ("valley"))
```

over the text of Figure 1, the algorithm first evaluates

```
(u, v) ← τ(Q, 1);
while u ≠ N + 1 do
  Generate (u, v);
  (u, v) ← τ(Q, u + 1);
end while;
```

Fig. 3. Algorithm for generating the solution extents for query Q.

$$\tau(\mathcal{I}(\text{“bells”}) \triangle (\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”})), 1).$$

The call is evaluated using the implementation of τ from Theorem 2.11. A call is made to the r access function, which is evaluated using the equations of Theorem 2.10 and the inverted-list file structures:

$$\begin{aligned} v &= r(\mathcal{I}(\text{“bells”}) \triangle (\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{valley})), 1) \\ &= \max(r(\mathcal{I}(\text{“bells”}), 1), r(\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”}), 1)) \\ &= \max(r(\mathcal{I}(\text{“bells”}), 1), \min(r(\mathcal{I}(\text{“sky”}), 1), r(\mathcal{I}(\text{“valley”}), 1))) \\ &= \max(1, \min(12, 27)) \\ &= \max(1,12) \\ &= 12 \end{aligned}$$

To complete the evaluation of the τ access function, a call is made to the l access function:

$$\begin{aligned} u &= l(\mathcal{I}(\text{“bells”}) \triangle (\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”})), v) \\ &= l(\mathcal{I}(\text{“bells”}) \triangle (\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”})), 12) \\ &= \min(l(\mathcal{I}(\text{“bells”}), 12), l(\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”}), 12)) \\ &= \min(l(\mathcal{I}(\text{“bells”}), 12), \max(l(\mathcal{I}(\text{“sky”}), 12), l(\mathcal{I}(\text{“valley”}), 12))) \\ &= \min(1, \max(12, 0)) \\ &= \min(1,12) \\ &= 1 \end{aligned}$$

Finally, from Theorem 2.11, we have

$$\tau(\mathcal{I}(\text{“bells”}) \triangle (\mathcal{I}(\text{“sky”}) \triangledown \mathcal{I}(\text{“valley”})), 1) = (u, v) = (1,12).$$

The algorithm generates (1,12) and next computes

$$\tau(\mathcal{I}(\text{“bells”})\triangle(\mathcal{I}(\text{“sky”})\triangledown\mathcal{I}(\text{“valley”})), u + 1)$$

$$= \tau(\mathcal{I}(\text{“bells”})\triangle(\mathcal{I}(\text{“sky”})\triangledown\mathcal{I}(\text{“valley”})), 13).$$

2.5 Efficiency

This section examines the theoretical efficiency of the algorithm in Figure 3.

A query Q is a Boolean combination of terms $T_1, \ldots, T_n$, whose postings lists have lengths $m_1, \ldots, m_n$. Under the implementation assumptions discussed in the previous section, the l and r access functions for term T_i require $O(\log m_i)$ time. For simplicity, the analysis will ignore the lengths of individual postings lists in the evaluation of the access functions and for all terms will use the length of the longest postings list, $m = \max_{1 \le i \le n} m_i$. For any of the terms in Q, a call to an access function requires $O(\log m)$ time.

THEOREM 2.12 *Let Q be a query consisting of a Boolean combination of terms $T_1, \ldots, T_n$. Using the equations of Theorem 2.10, a call to an access function for Q makes n calls to access functions for its terms.*

PROOF. The proof proceeds by induction on the number of terms, n. If Q consists of a single term, the access function for Q is a direct call to the corresponding access function for the term. If $n > 1$, then Q is either of the form $A\triangledown B$ or $A\triangle B$ where A is a Boolean combination of $n_A < n$ terms and where B is a Boolean combination of $n_B < n$ terms, with $n_A + n_B = n$. An access function for Q makes one call to an access function for A and one call to an access function for B. By the induction hypothesis, the call to the access function for A requires n_A calls to the access functions for its terms, and the call to the access function for B requires n_B calls to the access functions for its terms. Therefore, a call to an access function for Q requires $n_A + n_B = n$ calls to the access functions for its terms. □

Using the relationship defined by Theorem 2.11, a call to the τ access function requires at most $2n$ calls to access functions for the terms. Since an access function call for a term requires $O(\log m)$ time, a call to the τ access function requires $O(n \log m)$ time.

THEOREM 2.13 *Let Q be a query consisting of a Boolean combination of terms $T_1, \ldots, T_n$. Let S be the GC-list forming the solution to Q. The algorithm of Figure 3 requires $O(n|S|\log m)$ time to compute S.*

PROOF. To compute each of the $|S|$ extents in the solution, the algorithm makes a single call, requiring $O(n \log m)$ time, to the τ access function for Q. □

As the theorem indicates, the time to solve a query depends on the size of its solution set, which in turn depends on both the structure of the query and the distribution of the terms within the database. The theorem has positive implications for queries with long postings lists but having a small solution set, such as the conjunction of a common term (e.g., "the") with a rare term (e.g., "aardvark"). On the other hand, a disjunction of terms "the" OR "aardvark" has a solution set equal in size to the sum of the lengths of the postings lists for the terms. In this case, the algorithm's $O(|S| \log m)$ time complexity becomes problematic, since a simple merge of the postings lists would produce the solution set in $O(|S|)$ time.

As the next theorem demonstrates, in no case will the size of the solution set $|S|$ exceed the sum of the lengths of the postings lists for the terms in the query $M = \sum_{i=1}^{n} m_i$. Section 2.6 presents a modified version of the algorithm that requires $O(M)$ time to compute the solution set. The two approaches are compared afterward in Section 2.7.

THEOREM 2.14 *Let Q be a query consisting of a Boolean combination of terms $T_1, \ldots, T_n$, whose postings lists have sizes $m_1, \ldots, m_n$. Let S be the GC-list forming the solution to Q. Let $M = \sum_{i=1}^{n} m_i$.*

$$|S| \leq M$$

PROOF. If $(p, q) \in S$ then p indicates the location of an occurrence of a term from the query. That is, $\exists T_i$, $1 \leq i \leq n$, such that $(p, p) \in \mathcal{I}(T_i)$. For otherwise $(p + 1, q)$ would satisfy the query, and (p, q) could not be an element of the GC-list forming the solution to Q.

Thus, if $|S| > M$ then two distinct elements of S, (p, q') and (p, q'') must share a common start position p. If $q' < q''$ then $(p, q') \sqsubset (p, q'')$, and S is not a GC-list; if $q' > q''$ then $(p, q'') \sqsubset (p, q')$, and S is not a GC-list; if $q' = q''$ then the elements are not distinct. Therefore $|S| \leq M$. □

2.6 Computing Solution Sets by Scanning

The algorithm of Figure 3 generates the set of solution extents in the order of their position in the database. When generating the solution set, consecutive calls to the τ access function reference increasing positions in the database string. That is, if $\tau(Q, k)$ and $\tau(Q, k')$ are consecutive calls made by the algorithm, each generating an element of the solution set, then $k < k'$. In turn, calls to the l and r access functions made during the evaluation of the r access function (using the relationships defined in Theorems 2.10 and 2.11) reference increasing positions in the database string from one call to the τ access function to the next. At the lowest level, elements of the postings lists are accessed in increasing order as the elements in the solution set are generated.

These observations may be exploited to develop an $O(M)$ variant of the algorithm based on a sequential scan of the postings lists for the terms. For each term T in the query, we maintain three position pointers: $T.l\_current$, $T.r\_current$, and $T.r\_previous$. The postings list is treated as a data stream, supporting two operations: $next()$ and $reset()$. The $next()$ operation returns elements of the postings list in database order, one per call. The $reset()$ operation restarts the data stream at the beginning of the postings list. When a stream is exhausted, the $next()$ operations returns $N + 1$ until the stream is $reset()$. Two separate streams are maintained for each query term, one for the use of the l access function, with operations $T.l\_next()$ and $T.l\_reset()$ and one for the use of the r access function, with operations $T.r\_next()$ and $T.r\_reset()$.

At the start of the algorithm, all pointers are initialized to 0, and all streams are $reset()$. The l and r access functions are implemented as follows:

```
l(T, k) ≡
    while k > T.l_current do
        T.l_current ← T.l_next();
    end while;
    return T.l_current;


r(T, k) ≡
    while k > T.r_current do
        T.r_previous ← T.r_current;
        T.r_current ← T.r_next();
    end while;
    return T.r_previous;
```

Since each iteration of the main loop advances at least one stream, and since the total length of the posting lists is M, the time complexity of the resulting algorithm is $O(M)$.

2.7 Performance

Shortest-substring retrieval has been implemented in the context of the MultiText information retrieval system[1] using a combination of the indexing algorithm defined in Section 2.4 (Figure 3) and the scanning algorithm defined in Section 2.6.

Under the index organization used by the MultiText system, postings lists are organized into fixed-length blocks. Several short postings lists may be placed into a single block, while larger postings lists are split across multiple blocks. If a postings list is split across multiple blocks an in-memory map maintains the value of the first element of the postings list appearing in that block.

[1]http://multitext.uwaterloo.ca

Within blocks the scanning algorithm is used to advance through the postings list, and between blocks the indexing algorithm is used to skip blocks when possible. The positions referenced by the r and l access functions are used to guide this process. At the start of query processing, postings lists smaller than a fixed threshold are read into memory in their entirety. Postings lists larger than this threshold are accessed a block at a time, using the in-memory map to skip blocks that are not required to compute the query solution set. In the previous section, we observed that as solutions are generated by the algorithm, successive calls to access functions for terms return increasing database positions. When skipping blocks, this observation is used to reduce the range of the binary search, by using the position returned by the previous search as a lower bound.

A set of queries were developed to act as a synthetic benchmark to evaluate the performance of the MultiText implementation and to explore the possible benefits of the indexing algorithm over the scanning algorithm. For comparative purposes, the MultiText system was modified to compute solution sets using the scanning algorithm only. We compare the standard indexing/scanning version of the MultiText system with this modified version that uses the scanning algorithm only. The evaluation is based on the corpus used for the TREC-4 "adhoc" task (TREC disks 2 and 3) [Harman 1995], which consists of approximately 2GB of text in SGML format. The methodology is similar to that used by Moffat and Zobel [1996] to evaluate a self-indexing strategy for compressed inverted indices.

One hundred distinct passages of one thousand terms each were selected randomly from the corpus. The passages were stripped of punctuation, markup, stopwords, and terms containing numeric and other nonalphabetic characters. Following the default convention of the MultiText system, characters were normalized to lowercase. Repeated terms were removed, leaving only the first occurrence of each term. Finally, the resulting term sequences were truncated to one hundred terms each.

A query generated by taking the conjunction of the first n terms of one of these term sequences is guaranteed to have at least one solution extent, contained in the original passage. Generating a query from each of the term sequences in this fashion yields a set of one hundred queries, each consisting of the conjunction of n unique terms and having at least one solution extent. By varying the number of query terms n from one to hundred, the performance of the system over a range of query sizes may be evaluated and the effects of indexing may be assessed.

The first two curves in Figure 4 (curves (A) and (B)) show the performance of the standard indexing/scanning algorithm and the scanning-only algorithm. The figure plots query execution time, averaged over the hundred queries, against query size. For both algorithms the average query execution time grows approximately linearly with the number of query terms, but with a growth rate three times faster for the scanning-only algorithm.

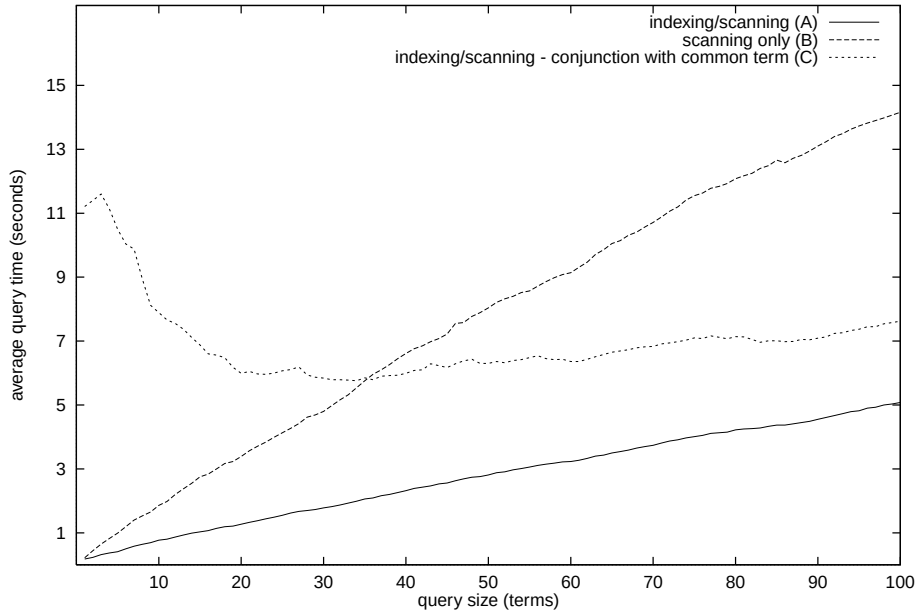

Fig. 4. Performance comparison: scanning algorithm versus indexing/scanning algorithm.

The benefits of indexing are more pronounced for queries with long postings lists and a small solution set, such as the conjunction of a common term with a rare term. For example, the query

```
"the" AND "aardvark"
```

runs in five seconds using the index/scanning algorithm, but requires more than three minutes using the scanning-only algorithm. The final curve in Figure 4 (curve (C)) shows the effects of a common query term on execution time for the indexing/scanning algorithm. Here, the common term `"the"`, which constitutes more than 5% of the database, is added to each query. Over a portion of this curve the execution time actually improves with the query size, as the size of the solution set decreases. The corresponding curve for the scanning-only algorithm is not plotted on the graph, since the execution times are well over three minutes.

3. SHORTEST-SUBSTRING RANKING

Under limited circumstances, the interpretation of Boolean queries under the shortest-substring model may have direct application without any further elaboration. In particular, the ability to specify and retrieve arbitrary passages, rather than entire documents or other predefined structural elements, may be of value if documents are long or if the predefined structural elements are not appropriate to the task at hand. Nevertheless, the theory is best viewed in the context of a broader retrieval model. Previous research introduced elements of the theory as a facet of a general retrieval language for structured text [Clarke et al. 1995a; 1995b]. In this section, the theory forms the basis for a relevance-ranking technique.

3.1 Method

Under the shortest-substring search model, the result of a query is the set of shortest extents that satisfy the specified Boolean predicate. Each extent in the solution set represents an interval in the text. Each document in the database may contain one or more of these solution intervals. Ranking is based on two assumptions:

Assumption A. The shorter a solution extent, the greater the likelihood that an interval of text containing the extent is relevant.

Assumption B. The more solution extents contained within a document, the greater the likelihood that the document is relevant.

The first assumption suggests a basis for ranking individual solution extents; the second suggests a basis for ranking documents in terms of the solution extents contained within them. Both assumptions are superficially reasonable. However, a problem remains in utilizing these assumptions to produce a single, combined score for a document.

Consistent with Assumption B, a score for a document might be obtained by summing individual scores for the solution extents contained within it. Consistent with Assumption A, the score for an extent (p, q) might be based on the inverse of its length $q - p + 1$:

$$\text{Score of } (p, q) \propto \left(\frac{1}{q - p + 1}\right)^{\alpha}$$

with a *falloff parameter* $\alpha > 0$

During preliminary trials of the technique it was observed that if the length of an extent was below a threshold of a dozen or so words, Assumption A no longer appeared to hold as strongly and all extents appeared to be a more or less equally good indicator of likely relevance. As a result, a *cutoff parameter* $\mathcal{K} > 0$ was added to the scoring function:

$$I(p, q) = \begin{cases} \left(\dfrac{\mathcal{K}}{q - p + 1}\right)^{\alpha} & \text{if } q - p + 1 \geq \mathcal{K} \\ 1 & \text{if } q - p + 1 \leq \mathcal{K} \end{cases}$$

For any extent (p, q), we have $0 < I(p, q)$. If solution extents $(p_1, q_1) \ldots, (p_N, q_N)$ are contained in a document, the score for the document is

$$\sum_{i=1}^{N} I(p_i, q_i).$$

The preliminary trials suggested that the scoring method is relatively insensitive to variations in the cutoff $(\mathcal{K})$ and falloff (α) parameters; values

of $\mathcal{K} = 16$ and $\alpha = 1$ were found to produce acceptable results and are used as the default values. The effects of varying these parameters will be considered in Section 3.2.

Shortest-substring retrieval computes the set of solution extents for a specified query. The ranking procedure then filters this set of extents against a list of extents representing document boundaries before computing the final scores and generating the ranking. Evaluation of a query proceeds in five steps:

(1) Determine the set of solution extents.

(2) For each solution extent, either determine the document extent that contains it, or if it overlaps document boundaries, eliminate it.

(3) For each document extent containing one of more solution extents, calculate its score.

(4) Sort the document extents by score.

(5) For the top-ranking documents, translate the document extents into external identifiers.

The set of extents representing document boundaries is a GC-list. To improve the performance of the ranking process, an algorithm similar to that of Section 2.4 may be used in Step (2) to skip over documents that do not contain solution sets. It is also possible to reduce storage requirements by interleaving the execution of Steps (1)–(3), finding solution extents, and scoring documents one at a time. Under the shortest-substring model, the query

```
"bells" AND ("sky" OR "valley")
```

specifies the set of extents

$$\{(1,12), (12,20), (20,27), (27,50), (50,59), (59,62), (68,71)\}$$

over the poem in Figure 1. Treating each of the three verses of the poem as a separate document eliminates $(27,50)$ and $(59,62)$, which overlap verses, and $(1,12)$, which overlaps the title. Scoring each verse separately, and for the purposes of this example using a cutoff of $\mathcal{K} = 4$ and a falloff of $\alpha = 1$, gives a score for each verse as follows:

$$\textit{first:} \quad I(12,20) + I(20,27) \approx 0.44 + 0.50 = 0.94$$

$$\textit{second:} \quad I(50,59) = 0.40$$

$$\textit{third:} \quad I(68,71) = 1.00$$

The third verse outscores the first, but contains fewer solution extents. Despite the many substrings that satisfy the query, few contribute to the scores.

```
<top>
<num> Number:  218
<desc> Description:
How have mini steel mills changed the steel industry?
</top>
```

Fig. 5. TREC topic 218.

```
<top>
<num> Number:  218
<ques> Question:
"steel mill" AND ("steel industry" OR (steel AND production))
</top>
```

Fig. 6. Query for TREC topic 218.

3.2 Evaluation

Shortest-substring ranking has been implemented in the MultiText system, and its effectiveness has been demonstrated by the MultiText experiments for the TREC series of conferences [Clarke and Cormack 1996; Clarke et al. 1995c; Cormack et al. 1997; 1998]. In those experiments, queries were created by individuals familiar with the MultiText system and were tailored to shortest-substring ranking. For this article, we use a set of independently generated queries to evaluate effectiveness.

The evaluation is based on the TREC-4 "adhoc" test collection [Harman 1995]. This collection consists of a 2GB corpus, a set of 49 statements of user requirements referred to as "topics," and set of relevance judgments relating the two. The corpus comprises roughly half a million documents drawn from a variety of sources and includes newspaper articles, government documents, and patents. The topics are in the form of one or more short statements or questions, expressed in natural language and having an average length of 16 words. There are 85,042 relevance judgments, each relating a specific document and topic. In 6,501 cases the document has been judged "relevant" to the topic; in all other cases, the document has been judged "not relevant."

The evaluation uses a set of Boolean queries developed manually from the TREC-4 topics by FS Consulting as part of their own participation in the conference [Schiettecatte and Florance 1995]. The queries required only a few minor changes in syntax for use with MultiText. Apart from the Boolean operators `AND` and `OR`, the queries use only term truncation and phrases, both of which are supported by the MultiText system. Figure 5 shows a topic from TREC-4 (TREC topic 218); Figure 6 shows the FS Consulting query corresponding to this topic.

The queries were executed on the MultiText system using shortest-substring ranking; the results are shown in Table I. As a group, the queries retrieved 17,581 documents from the TREC-4 corpus. Of these, 1,229 (7%) are relevant, giving an overall recall of 19%. Table I lists precision

Table I. Comparison of Retrieval Effectiveness

Precision at	Shortest-Substring Ranking	Unranked	Okapi	Okapi Ranked Boolean
5 documents	0.449	0.220	0.416	0.433
10 documents	0.402	0.206	0.376	0.412
15 documents	0.396	0.210	0.343	0.380
20 documents	0.362	0.207	0.310	0.346
100 documents	0.162	0.119	0.189	0.161

averaged over the 49 topics at five retrieval levels—5, 10, 15, 20, and 100 documents—and compares the results of shortest-substring ranking with three other retrieval runs.

For the run reported in the column labeled "Unranked," the documents were ordered only by their (arbitrarily assigned) positions in the database. The results of this run are representative of the performance that might be expected from a traditional Boolean information retrieval system. Shortest-substring ranking improves on this run by more than 95% at 10 documents and more than 36% at 100 documents. These improvements are significant (paired t-test, $p < 0.05$). Generally, an unranked query is expected to exhibit the same precision at all retrieval levels. The drop-off in precision at 100 documents is due to the relatively small number of queries that retrieve 100 documents or more. Most of the queries (41 of 49) retrieve 10 documents or more, but only 18 retrieve 100 documents or more. The results reported in the column labeled "Okapi" were generated using the MultiText implementation of the standard Okapi measure [Robertson and Walker 1994; Robertson et al. 1994]. For a document D and a query Q we compute

$$\sum_{t \in D \wedge t \in Q} \log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right)\left(\frac{f_{D,t}}{f_{D,t} + l_D / l_{avg}}\right)$$

where

N = number of documents in the collection

n_t = number of documents containing term t

$f_{D,t}$ = frequency of ocurrence of term t in document D

l_D = length of document D

l_{avg} = average document length.

This formula is equivalent to the Okapi BM11 measure with reasonable choices for the parameter values ($k_1 = 1$ and $k_2 = k_3 = R = r = 0$). The purpose of this run is to provide an independent assessment of the retrieval performance that can be expected using this query set.

Table II. Effect of Shortest-Substring Ranking on Queries with Many Solutions

	Topic Number								
	228	223	240	235	236	245	212	248	230
Documents returned	5819	2915	1642	1060	1060	769	533	448	386
Relevant returned	24	194	126	98	12	13	10	52	29
Unranked precision	0.004	0.067	0.077	0.093	0.011	0.017	0.019	0.116	0.075
Precision at 20	0.100	0.650	0.550	0.800	0.100	0.250	0.000	0.500	0.600

For the Okapi run, the Boolean operators were stripped from the queries, which were then treated as term vectors. To permit a direct comparison, the term truncation and phrase structuring of the original queries were maintained in the term vectors, and stemming was not applied. Shortest-substring ranking slightly outperforms the Okapi measure at 5, 10, 15, and 20 documents. At 100 documents, the Okapi measure slightly outperforms shortest-substring ranking. However, these differences are not significant.

Naturally, the Okapi run includes many documents that do not match the original Boolean query. For the "Okapi Ranked Boolean" run, documents were scored using the Okapi measure but were also filtered against the original query. Only documents matching the original Boolean query were retained for output. In comparison with the Okapi run, filtering improves retrieval performance at 5, 10, 15, and 20 documents. Once again, the precision at 100 documents is influenced by the relatively small number of queries that return 100 documents or more. At all document retrieval levels except 10, shortest-substring ranking very slightly outperforms the filtered Okapi run, but these differences are not significant.

If a query matches only a small number of documents, ranking these documents is of limited value. Shortest-substring ranking appears to be of greatest value when a query matches a great many documents, few of which are relevant. Table II shows the effects of shortest-substring ranking on the nine queries returning the greatest number of documents. For each query, the figure lists the associated topic number, the total number of documents returned by the query, the number of relevant documents returned, the unranked precision, and the precision at 20 documents when the results are ranked using shortest-substring ranking. The unranked precision—the ratio of the number of relevant documents returned to the total number of documents returned—represents the precision that a user could expect at any document level if no ranking were available. For all topics but one (topic 212), precision at 20 documents is substantially greater than the overall precision, often by a factor of 7 or more.

The default values for cutoff ($\mathcal{K} = 16$) and falloff ($\alpha = 1$) were used for the results reported above. Figure 7 shows the effects of varying cutoff with the value of the falloff parameter maintained at $\alpha = 1$; Figure 8 shows the effects of varying falloff with the value of the cutoff parameter maintained at $\mathcal{K} = 16$. The figures plot precision at 10, 20, and 100 documents for a range of parameter values. At the low end of the range, performance improves with higher values of cutoff, supporting our introduction of this

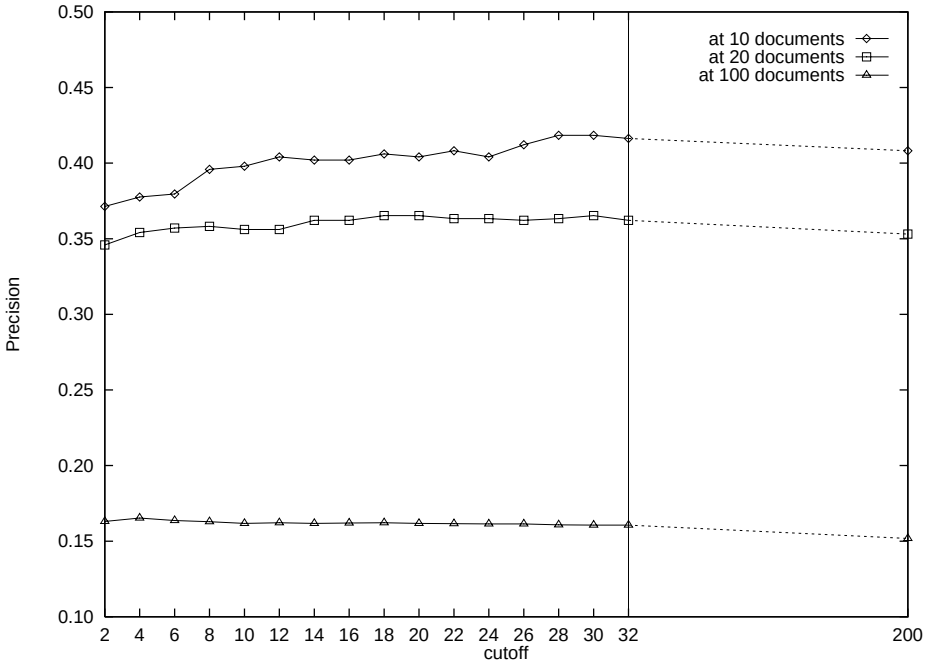

Fig. 7. Effects of varying cutoff ($\mathcal{K}$) with $\alpha = 1$.

parameter. For this test collection, a cutoff value slightly higher than the default and a falloff value slightly lower than the default might be preferable, but generally the results support our preliminary assessment: that the scoring method is relatively insensitive to variations in the values of these parameters.

Shortest-substring ranking is based on the two assumptions (A and B) given at the beginning of Section 3.1. The results above support these assumptions when taken together; we now examine the validity of the assumptions individually. In order to separately test Assumption A we ignore the number of solution extents contained in a document and score it solely on the length of the shortest solution extent contained within it. If (p, q) is the shortest solution extent contained in the document, its score would be $1/(q - p + 1)$. At the other extreme, in order to separately test Assumption B we ignore the length of the solution extents and score a document solely by counting the number of solution extents it contains. If it contains N solution extents, its score would be N.

Table III gives the results for these tests. The first two runs in the table are reproduced from Table I, giving results for the shortest-substring ranking run and the unranked run. For the run labeled "By Length," documents were ranked according to the length of the shortest solution extent contained within them; for the run labeled "By Count," documents were ranked according to the number of solution extents they contain. While these runs improve on the unranked results, shortest-substring

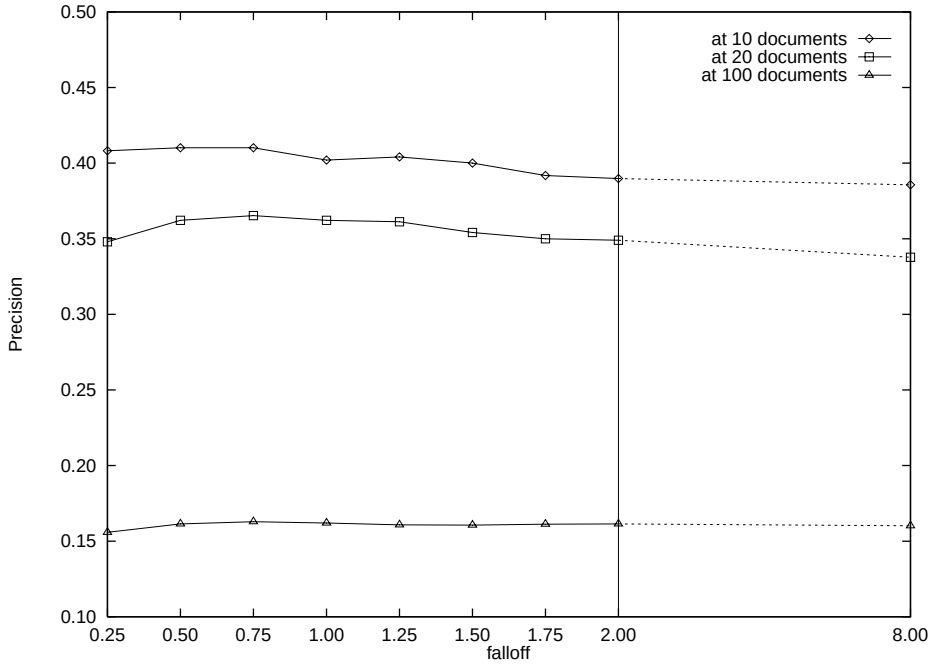

Fig. 8. Effects of varying falloff (α) with $\mathcal{K} = 16$.

Table III. Comparision of Retrieval Effectiveness

Precision at	Shortest- Substring Ranking	Unranked	By Length (A)	By Count (B)
5 documents	0.449	0.220	0.371	0.376
10 documents	0.402	0.206	0.339	0.365
15 documents	0.396	0.210	0.317	0.332
20 documents	0.362	0.207	0.308	0.315
100 documents	0.162	0.119	0.157	0.146

ranking improves on all three. For both the "By Length" and "By Count" runs, the improvements over the unranked run are significant (paired t-test, $p < 0.05$). For the shortest-substring ranking run, the improvements over the other three runs are significant except for the "By Length" run at the 100 document level. The results validate the assumptions, but underscore the benefit of using them in combination.

3.3 Properties of the Method

The score for a document does not depend on *idf* (inverse document frequency) or other global statistics, and the factors contributing to a document's score are independent of the characteristics of other documents in the database. This property is of particular benefit in a distributed environment, where documents are scattered across many nodes and where global statistics may be difficult to compute [Callan et al. 1995].

The score for an interval does not depend on *tf* (within-document term frequency) or other document-level statistics. A definition for "document boundary" is used only at query time, not at database build time. As a result, the method may be used to rank any structural element (paragraphs, pages, etc.) that can be defined and represented at query time as a GC-list of extents. The MultiText system can support a query of the form

$$rank\ X\ by\ Y$$

where X and Y are arbitrary queries expressed in the MultiText structured-document query language.

High-scoring solution extents may be used to identify the parts of documents that are the most likely to be of interest. As part of our TREC-5 and TREC-6 experiments we used this property for interactive query refinement, for automatic query expansion, and to facilitate rapid relevance assessment [Clarke and Cormack 1996; Cormack et al. 1997]. With a few exceptions [Kaszkiel and Zobel 1997; Knauss et al. 1995], research into the use of passages in information retrieval has generally assumed that the division of a document into passages must be done a priori when the database is created [Allan 1995; Callan 1994; Hearst and Plaunt 1993; Salton et al. 1993; Wilkinson 1994; Wilkinson and Zobel 1994]. In contrast, shortest-substring retrieval bases the decomposition into passages on the query itself.

The use of term proximity is a distinguishing characteristic of shortest-substring ranking. Presently, the only widespread use of proximity for ranking is the automatic recognition and indexing of phrases [Jing and Croft 1994; Mitra et al. 1997; Salton and McGill 1983]. Proximity also plays a role in local feedback techniques for query expansion [Attar and Fraenkel 1977; Xu and Croft 1996]. Keen [1991; 1992a; 1992b] evaluates the benefits of proximity operators in Boolean systems, and proposes several non-Boolean ranking methods based on term proximity within sentences. Hawking and Thistlewaite [1995; 1996] describe a proximity scoring method similar to shortest-substring ranking. The method uses nested intervals as well as overlapping intervals. Fixed proximity limits are defined as part of the query; intervals longer than this fixed limit are not considered during the ranking process. Implementation is based on text scanning. In contrast, the indexing algorithm of Section 2.4 efficiently implements shortest-substring ranking using standard inverted-list data structures.

Shortest-substring ranking is fast. The evaluation of Section 3.2 was performed on an inexpensive, mid-range personal computer costing less than US $1900 in 1997. The average query execution time was less than two seconds. Many queries executed in less than a second, with a few queries taking several seconds due to phrases containing high-frequency words (e.g., "National Endowment for the Arts").

3.4 MultiText Experiments for TREC

Shortest-substring ranking is the primary ranking method for the MultiText system, and as a result has been used in most MultiText experiments for the TREC conference series [Clarke and Cormack 1996; Clarke et al. 1995c; Cormack et al. 1997; 1998]. MultiText has reported a wide range of experiments at the TREC conferences. While the various conference proceedings should be consulted for the details, we provide a brief summary of selected experiments here.

The method was first used for TREC-4. Searchers created Boolean queries manually from the topics. In accordance with the TREC-4 guidelines for manual query creation, no interaction was used: documents in the target database were not examined by the searchers while the queries were being formulated. After the queries were created, they were executed against the target corpus to generate a ranked list of the top 1000 documents for each query. This set of ranked lists (called a "run" in TREC terminology) was then submitted for evaluation.

Comparison between TREC runs using manually created queries should be undertaken with caution, since queries vary from group to group, and run to run. Nonetheless, shortest-substring ranking demonstrated its potential at TREC-4. Of the 15 groups submitting TREC-4 manual runs for the main TREC experimental task (the "adhoc" task) MultiText had the third best overall performance. Detailed performance figures are given in Table IV. The first column shows the results of MultiText's official TREC-4 adhoc run (`uwgcl1`). In addition to shortest-substring ranking, the run used "tiered" queries to improve recall. Each tiered query consisted of one or more tiers, with each tier consisting of a single Boolean query. The top tier of each query was intended to have relatively high precision and attempted to capture accurately the user requirements embodied in the topic. This top tier was evaluated and ranked first. After the top tier was exhausted, the remaining tiers were used in order, until 1000 distinct documents were generated or until all tiers were exhausted. The second column of Table IV shows the performance of the top tier of each query when used alone, which may be more appropriate for comparison with the FS Consulting queries discussed in Section 3.2. The third column of the table gives the performance of the official run submitted by FS Consulting, using the same queries discussed in Section 3.2. These results may be compared with those of Table I.

When treated as unranked Boolean queries, the top tiers of the MultiText queries had much better recall but worse precision than the FS Consulting queries. The average unranked recall (the average over all queries of the ratio of relevant documents returned to the number of relevant documents for a query) is 0.2511 for the FS Consulting queries and 0.6167 for the top tier of the MultiText queries. The average unranked precision (the average over all queries of the ratio of relevant documents to total documents returned by a query) is 0.2432 for the FS Consulting queries and 0.1644 for the top tier of the MultiText queries. Despite the lower unranked

Table IV. TREC-4 Results

Precision at	uwgcl1	uwgcl1 (top tier only)	fsclt1
5 documents	0.551	0.551	0.392
10 documents	0.504	0.502	0.357
15 documents	0.487	0.472	0.331
20 documents	0.476	0.460	0.312
100 documents	0.341	0.324	0.152

precision, it appears that shortest-substring ranking can take advantage of the higher recall to produce better retrieval performance.

The TREC-5 guidelines for manual queries permitted interactive query creation. The interface used for creating the queries took advantage of the passage retrieval capabilities of shortest-substring ranking for query refinement. In response to an initial query, the user was shown high-ranking passages, which could then be used to identify words and phrases for addition and deletion. Of the 16 groups submitting TREC-5 manual runs, MultiText had the second best overall performance.

For both TREC-6 and TREC-7, passage retrieval was used for interactive query refinement, automatic query expansion, and rapid relevance assessment. In addition, shortest-substring ranking was adapted and extended for use with queries expressed as unstructured term vectors. While the resulting method is generally suitable only for very short queries, those consisting of one to four terms, queries of this length form an important class. Queries of this length are common in interactive environments, such as searching on the World Wide Web; the title fields of the query topics introduced at TREC-6 and TREC-7 were expressly intended for use as very short queries.

Both TREC-6 and TREC-7 provided an opportunity to experiment with larger corpora. Most TREC experiments are conducted on corpora roughly 2GB in size. The TREC-6 Very Large Collection (VLC) experiments [Hawking and Thistlewaite 1997] were conducted on a 20GB corpus; the TREC-7 VLC experiments [Hawking et al. 1998] were conducted on a 100GB corpus. Between TREC-5 and TREC-6, and between TREC-6 and TREC-7, the MultiText system underwent performance enhancement and tuning. By TREC-7, Boolean queries giving excellent retrieval performance could be executed over the 100GB VLC corpus in under a second, with the corpus distributed over four inexpensive personal computers.

4. CONCLUSIONS AND FUTURE WORK

The article introduces a novel interpretation of Boolean queries in information retrieval systems and a new method of relevance ranking arising from this interpretation. The method is both fast and effective. Unlike methods based on soft Boolean operators, the method preserves the exact-match semantics of Boolean retrieval and the standard properties of Boolean algebra. The method is fundamentally a passage retrieval technique; this

feature can be exploited to rank arbitrary document elements defined at query time. Since the method does not depend on global statistics, it easily scales for use in parallel and distributed environments.

Refining and enhancing the retrieval and ranking procedures is part of our ongoing and planned research. Scoring is based on two independently verified assumptions that are intuitively reasonable, but the actual formula is somewhat arbitrary. Other formulae reflecting the assumptions may be developed and evaluated. Ideally, this process would be guided by theory. Formulae that incorporate *tf* and *idf* statistics would compromise beneficial properties of shortest-substring ranking, but these properties would not necessarily be compromised by document length normalization. For a number of reasons, including efficiency and simplicity, ranking is based only on the smallest extents satisfying a query, but it is possible that the method could be extended to take longer extents into consideration. Additional experiments over a variety of test collections could be directed toward optimizing the default values for parameters in the existing formula. Finally, the buffering, caching, and prefetching policies used in the implementation may be further enhanced and tuned.

REFERENCES

ALLAN, J. 1995. Relevance feedback with too much data. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '95, Seattle, WA, July 9–13), E. A. Fox, P. Ingwersen, and R. Fidel, Eds. ACM Press, New York, NY, 337–343.

ATTAR, R. AND FRAENKEL, A. S. 1977. Local feedback in full-text retrieval systems. *J. ACM 24*, 3 (July), 397–417.

BOOKSTEIN, A. 1978. On the perils of merging Boolean and weighted retrieval systems. *J. Am. Soc. Inf. Sci. 29*, 3, 156–158.

BOOKSTEIN, A. 1980. Fuzzy requests: An approach to weighted Boolean searches. *J. Am. Soc. Inf. Sci. 31*, 4 (July), 240–247.

BRENNER, E. H. 1996. Beyond boolean—New approaches to information retrieval. National Federation of Abstracting and Information Services, Philadelphia, PA.

BUELL, D. A. AND KRAFT, D. H. 1981. Threshold values and Boolean retrieval systems. *Inf. Process. Manage. 17*, 3, 127–136.

BURKOWSKI, F. J. 1992. An algebra for hierarchically organized text-dominated databases. *Inf. Process. Manage. 28*, 3 (1992), 333–348.

CALLAN, J. P. 1994. Passage-level evidence in document retrieval. In *Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '94, Dublin, Ireland, July 3–6), W. B. Croft and C. J. van Rijsbergen, Eds. Springer-Verlag, New York, NY, 302–310.

CALLAN, J. P., LU, Z., AND CROFT, W. B. 1995. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '95, Seattle, WA, July 9–13), E. A. Fox, P. Ingwersen, and R. Fidel, Eds. ACM Press, New York, NY, 21–28.

CLARKE, C. L. A. AND CORMACK, G. V. 1996. Interactive substring retrieval. In *Proceedings of the 5th Text Retrieval Conference* (TREC-5, Gaithersburg, MD, Nov.), E. M. Voorhees and D. K. Harman, Eds. National Institute of Standards and Technology, Gaithersburg, MD, 267–277.

CLARKE, C. L. A., CORMACK, G. V., AND BURKOWSKI, F. J. 1994. Fast inverted indexes with on-line update. Tech. Rep. CS-94-40. Computer Science Dept., University of Waterloo, Waterloo, Canada.

CLARKE, C. L. A., CORMACK, G. V., AND BURKOWSKI, F. J. 1995a. An algebra for structured text search and a framework for its implementation. *Comput. J. 38*, 1, 43–56.

CLARKE, C. L. A., CORMACK, G. V., AND BURKOWSKI, F. J. 1995b. Schema-independent retrieval from heterogeneous structured text. In *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV), 279–289.

CLARKE, C. L. A., CORMACK, G. V., AND BURKOWSKI, F. J. 1995c. Shortest substring ranking MultiText experiments for TREC-4. In *Proceedings of the 4th Text Retrieval Conference* (TREC-4, Washington, D.C., Nov.), D. K. Harman, Ed. National Institute of Standards and Technology, Gaithersburg, MD, 295–304.

COOPER, W. S. 1988. Getting beyond Boole. *Inf. Process. Manage. 24*, 3 (May 1988), 243–248.

CORMACK, G. V., CLARKE, C. L. A., PALMER, C. R., AND TO, S. S.-L. 1997. Passage based refinement. In *Proceedings of the 6th Text Retreival Conference* (TREC-6, Nov.), E. Voorhees and D. Harman, Eds. 303–319.

CORMACK, G., PALMER, C., VAN BIESBROUCK, M., AND CLARKE, C. 1998. Deriving very short queries for high precision and recall. In *Proceedings of the 7th Text Retreival Conference* (TREC-7),

GREIFF, W. R., CROFT, W. B., AND TURTLE, H. 1997. Computationally tractable probabilistic modeling of Boolean operators. In *Proceedings of the 20th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '97, Philadelphia, PA, July 27–31), N. J. Belkin, A. D. Narasimhalu, P. Willett, W. Hersh, F. Can, and E. Voorhees, Eds, ACM Press, New York, NY, 119–128.

HARMAN, D. K., Ed. 1995. *Proceedings of the 4th Text Retrieval Conference*. (TREC-4, Washington, D.C., Nov.). National Institute of Standards and Technology, Gaithersburg, MD.

HAWKING, D. AND THISTLEWAITE, P. 1995. Proximity operators—So near and yet so far. In *Proceedings of the 4th Text Retrieval Conference* (TREC-4, Washington, D.C., Nov.), D. K. Harman, Ed. National Institute of Standards and Technology, Gaithersburg, MD, 131–143.

HAWKING, D. AND THISTLEWAITE, P. 1996. Relevance weighting using distance between term occurrences. Tech. Rep. TR-CS-96-08. Department of Computer Science, Australian National Univ., Canberra, Australia. Available via http://cs.anu.edu.au/techreports/1996/index.html.

HAWKING, D. AND THISTLEWAITE, P. 1997. Overview of the TREC-6 very large collection track. In *Proceedings of the 6th Text Retreival Conference* (TREC-6, Nov.), E. Voorhees and D. Harman, Eds.

HAWKING, D., CRASWELL, N., AND THISTLEWAITE, P. 1998. Overview of TREC-7 very large collection track. In *Proceedings of the 7th Text Retrieval Conference* (TREC-7),

HEARST, M. A. 1996. Improving full-text precision on short queries using simple constraints. In *Proceedings of the 5th Annual Symposium on Document Analysis and Information Retrieval* (Las Vegas, NV, Apr.),

HEARST, M. A. AND PLAUNT, C. 1993. Subtopic structuring for full-length document access. In *Proceedings of the 16th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '93, Pittsburgh, PA, June 27–July), R. Korfhage, E. Rasmussen, and P. Willett, Eds. ACM Press, New York, NY, 59–68.

JING, Y. AND CROFT, W. B. 1994. An association thesaurus for information retrieval. In *Proceedings of the Intelligent Multimedia Information Retrieval Systems* (RIAO '94, New York, NY), 146–160.

KASZKIEL, M. AND ZOBEL, J. 1997. Passage retrieval revisited. In *Proceedings of the 20th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '97, Philadelphia, PA, July 27–31), N. J. Belkin, A. D. Narasimhalu, P. Willett, W. Hersh, F. Can, and E. Voorhees, Eds, ACM Press, New York, NY, 178–185.

KEEN, E. M. 1991. The use of term position devices in ranked output experiments. *J. Doc. 47*, 1 (Mar.), 1–22.

KEEN, E. M. 1992a. Some aspects of proximity searching in text retrieval systems. *J. Inf. Sci. 18*, 2 (1992), 89–98.

KEEN, E. M. 1992b. Term position ranking: some new test results. In *Proceedings of the 15th Annual International ACM Conference on Research and Development in Information Re-*

trieval (SIGIR '92, Copenhagen, Denmark, June 21–24), N. Belkin, P. Ingwersen, A. M. Pejtersen, and E. Fox, Eds. ACM Press, New York, NY, 66–76.

KERRE, E. E., ZENNER, R. B. R. C., AND DE CALUWE, R. M. M. 1986. The use of fuzzy set theory in information retrieval and databases: A survey. *J. Am. Soc. Inf. Sci. 37*, 5 (Sept.), 341–345.

KIM, M. H., LEE, J. H., AND LEE, Y. J. 1993. Analysis of fuzzy operators for high quality information retrieval. *Inf. Process. Lett. 46*, 5 (July 9, 1993), 251–256.

KNAUS, D., MITTENDORF, E., SCHÄUBLE, P., AND SHERIDAN, P. 1995. Highlighting relevant passages for users of the interactive SPIDER retrieval system. In *Proceedings of the 4th Text Retrieval Conference* (TREC-4, Washington, D.C., Nov.), D. K. Harman, Ed. National Institute of Standards and Technology, Gaithersburg, MD.

LEE, J. H. 1994. Properties of extended Boolean models in information retrieval. In *Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '94, Dublin, Ireland, July 3–6), W. B. Croft and C. J. van Rijsbergen, Eds. Springer-Verlag, New York, NY, 182–190.

LEE, J. H., KIN, W. Y., KIM, M. H., AND LEE, Y. J. 1993. On the evaluation of Boolean operators in the extended Boolean retrieval framework. In *Proceedings of the 16th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '93, Pittsburgh, PA, June 27–July), R. Korfhage, E. Rasmussen, and P. Willett, Eds. ACM Press, New York, NY, 291–297.

LESK, M. 1997. *Practical Digital Libraries: Books, Bytes, and Bucks*. Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA.

LI, S.-H. AND DANZIG, P. 1997. Boolean similarity measures for resource discovery. *IEEE Trans. Knowl. Data Eng. 9*, 6 (Nov.-Dec.), 863–876.

MACLANE, S. AND BIRKOFF, G. 1967. *Algebra*. Macmillan, New York, NY.

MITRA, M., BUCKLEY, C., SINGHAL, A., AND CARDIE, C. 1997. An analysis of statistical and syntactic phrases. In *Proceedings of the 1997 Intelligent Multimedia Information Retrieval Systems Conference* (RIAO '97, Montreal, Canada, June), 200–214.

MOFFAT, A. AND ZOBEL, J. 1996. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst. 14*, 4 (Oct.), 349–379.

NOREAULT, T., KOLL, M., AND MCGILL, M. J. 1977. Automatic ranked output from Boolean searches in SIRE. *J. Am. Soc. Inf. Sci. 28*, 6, 333–339.

RADECKI, T. 1982. Reducing the perils of merging Boolean and weighted retrieval systems. *J. Doc. 38*, 1 (Sept.), 207–211.

ROBERTSON, S. E. 1978. On the nature of fuzz: A diatribe. *J. Am. Soc. Inf. Sci. 29*, 6, 304–307.

ROBERTSON, S. E. AND WALKER, S. 1994. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '94, Dublin, Ireland, July 3–6), W. B. Croft and C. J. van Rijsbergen, Eds. Springer-Verlag, New York, NY, 232–241.

ROBERTSON, S. E., WALKER, S., JONES, S., HANCOCK-BEAULIEU, M. M., AND GATFORD, M. 1994. Okapi at TREC-3. In *Proceedings of the 3rd Text Retrieval Conference* (TREC-3, Nov.), 109–126.

ROUSSEAU, R. 1990. Extended boolean retrieval: A heuristic approach?. In *Proceedings of the 13th International Conference on Research and Development in Information Retrieval* (SIGIR '90, Brussels, Belgium, Sept. 5–7), J.-L. Vidick, Ed. ACM Press, New York, NY, 495–508.

SALTON, G. AND MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., Hightstown, NJ.

SALTON, G., ALLAN, J., AND BUCKLEY, C. 1993. Approaches to passage retrieval in full text information systems. In *Proceedings of the 16th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '93, Pittsburgh, PA, June 27–July), R. Korfhage, E. Rasmussen, and P. Willett, Eds. ACM Press, New York, NY, 49–58.

SALTON, G., FOX, E. A., AND WU, H. 1983. Extended Boolean information retrieval. *Commun. ACM 26*, 12 (Dec.), 1022–1036.

SCHIETTECATTE, F. AND FLORANCE, V. 1995. Document retrieval using the MPS information server. In *Proceedings of the 4th Text Retrieval Conference* (TREC-4, Washington, D.C., Nov.), D. K. Harman, Ed. National Institute of Standards and Technology, Gaithersburg, MD, 401–419.

TAHANI, V. 1978. A fuzzy model of document retrieval systems. *Inf. Process. Manage. 12*, 3, 177–187.

TEASDALE, S. 1920. *Flame and Shadow*. Macmillan, New York, NY.

THOMAS, S. W., ALEXANDER, K., AND GUTHRIE, K. 1999. Technology choices for the JSTOR online archive. *IEEE Computer 32*, 2 (Feb.), 60–65.

TURTLE, H. 1994. Natural language vs. Boolean query evaluation: a comparison of retrieval performance. In *Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '94, Dublin, Ireland, July 3–6), W. B. Croft and C. J. van Rijsbergen, Eds. Springer-Verlag, New York, NY, 212–220.

TURTLE, H. AND FLOOD, J. 1995. Query evaluation: Strategies and optimizations. *Inf. Process. Manage. 31*, 6 (Nov.), 831–850.

TURTLE, H. R. AND CROFT, W. B. 1992. A comparison of text retrieval models. *Comput. J. 35*, 3 (June), 279–290.

WALLER, W. G. AND KRAFT, D. H. 1979. A mathematical model for a weighted Boolean retrieval system. *Inf. Process. Manage. 15*, 5, 235–245.

WILKINSON, R. 1994. Effective retrieval of structured documents. In *Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval* (SIGIR '94, Dublin, Ireland, July 3–6), W. B. Croft and C. J. van Rijsbergen, Eds. Springer-Verlag, New York, NY, 311–317.

WILKINSON, R. AND ZOBEL, J. 1994. Comparison of fragmentation schemes for document retrieval. In *Proceedings of the 3rd Text Retrieval Conference* (TREC-3, Nov.), 81–84.

XU, J. AND CROFT, W. B. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '96, Zurich, Switzerland, Aug. 18–22), H.-P. Frei, D. Harman, P. Schaüble, and R. Wilkinson, Eds. ACM Press, New York, NY, 4–11.

Received: March 1998; revised: August 1998; accepted: May 1999