Phrase Query Optimization on Inverted Indexes

Avishek Anand
Ida Mele
Srikanta Bedathur
Klaus Berberich

MPI–I–2014–5–002

Authors' Addresses

Avishek Anand
L3S Research Center
Hannover, Germany

Ida Mele
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Srikanta Bedathur
IIIT Delhi
New Delhi, India

Klaus Berberich
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Abstract

Phrase queries are a key functionality of modern search engines. Beyond that, they increasingly serve as an important building block for applications such as entity-oriented search, text analytics, and plagiarism detection. Processing phrase queries is costly, though, since positional information has to be kept in the index and all words, including stopwords, need to be considered.

We consider an augmented inverted index that indexes selected variable-length multi-word sequences in addition to single words. We study how arbitrary phrase queries can be processed efficiently on such an augmented inverted index. We show that the underlying optimization problem is $\mathcal{NP}$-hard in the general case and describe an exact exponential algorithm and an approximation algorithm to its solution. Experiments on ClueWeb09 and The New York Times with different real-world query workloads examine the practical performance of our methods.

Keywords

Phrase Queries, Query Optimization

Contents

1 Introduction

As the scale of document collections such as the Web grows, more effective tools to search and explore them are needed. Phrase queries are one such tool. Typically expressed by enclosing a multi-word sequence with quotation marks (e.g., "president of the united states") they enforce that the search engine returns only documents that literally contain the quoted phrase. Our focus in this work is on supporting phrase queries more efficiently.

Phrase queries are supported by all modern search engines and are one of their advanced features most popular with human users, accounting for up to 5% of query volume [18]. Even when unknown to the user, phrase queries can still be implicitly invoked, for instance, by means of query-segmentation methods [10, 13]. Beyond their use in search engines, phrase queries increasingly serve as a building block for other applications such as (a) *entity-oriented search and analytics* [5] (e.g., to identify documents that refer to a specific entity using one of its known labels), (b) *plagiarism detection* [19] (e.g., to identify documents that contain a highly discriminative fragment from the suspicious document), (c) *culturomics* [16] (e.g., to identify documents that contain a specific n-gram and compute a frequency time-series from their timestamps).

Positional information about where words occur in documents has to be maintained to support phrase queries, which leads to indexes that are larger (e.g., [7] report a factor of about 4× for the inverted index) than those required for keyword queries. Also, all words need to be considered in the case of phrase queries, as opposed to keyword queries for which stopwords can be ignored. Consequently, phrase queries are substantially more expensive to process, since more data has to be obtained from the index.

The problem of substring matching, which is at the core of phrase queries, has been studied extensively by the String Processing community. However, the solutions developed (e.g., suffix arrays [15] and permuterm indexes [9]) are designed for main memory and cannot cope with large-scale document collections. Solutions developed by the Information Retrieval community [20,

22] build on the inverted index, extending it to index selected multi-word sequences, so-called *phrases*, in addition to single words.

In this work, we follow the general approach of augmenting the inverted index with selected multi-word phrases. Given such an index, we focus on the problem of *phrase-query optimization*, that is, determining for a given phrase query an optimal set of terms to process it. Consider, as a concrete example, the phrase query "we are the champions" and assume that all bigrams have been indexed alongside single words. Here, the space of possible solutions includes among others {we, are, the, champions} and {we are, are the, champions}. Identifying a cost-minimal set of indexed terms is the problem addressed in this work. Existing work [22] has addressed this problem only heuristically. In contrast, we study its hardness and devise an exact exponential algorithm and an approximation algorithm to its solution.

Contributions that we make in this work thus include

- we study the problem of *phrase-query optimization*, establish its $\mathcal{NP}$-hardness, and describe an exact exponential algorithm as well as an $O(log\, n)$-approximation algorithm to its solution;
- an *experimental evaluation* on ClueWeb09 and a corpus from The New York Times, as two real-world document collections and different workloads denoting the three kinds of tasks where phrase queries are applicable, comparing our approach against state-of-the-art competitor and establishing its efficiency and effectiveness.

Organization. The rest of this paper is organized as follows. Chapter 2 introduces our formal model. The augmented inverted index is described in Chapter 3. Chapter 4 deals with optimizing phrase queries. Chapter 5 describes our experimental evaluation. We relate our work to existing prior work in Chapter 6 and conclude in Chapter 7.

2 Model

We let $\mathcal{V}$ denote the *vocabulary* of all words. The *set of all non-empty sequences of words* from this vocabulary is denoted $\mathcal{V}^+$. Given a word sequence

$$\mathbf{s} = \langle\, s_1,\, \ldots,\, s_n \,\rangle \in \mathcal{V}^+$$

we let $|\,\mathbf{s}\,| = n$ denote its length. We use $\mathbf{s}[i]$ to refer to the word s_i at the i-th position of $\mathbf{s}$, and $\mathbf{s}[i..j]$ $(i \leq j)$ to refer to the word subsequence $\langle\, s_i,\, \ldots,\, s_j \,\rangle$.

Given two word sequences $\mathbf{r}$ and $\mathbf{s}$, we let $pos(\mathbf{r}, \mathbf{s})$ denote the set of positions at which $\mathbf{r}$ occurs in $\mathbf{s}$, formally

$$pos(\mathbf{r}, \mathbf{s}) = \{\, 1 \leq i \leq |\mathbf{s}| \mid \forall\, 1 \leq j \leq |\mathbf{r}| : \mathbf{s}[i + j - 1] = \mathbf{r}[j] \,\} \ .$$

For $\mathbf{r} = \langle\, ab \,\rangle$ and $\mathbf{s} = \langle\, cabcab \,\rangle$, as a concrete example, we have $pos(\mathbf{r}, \mathbf{s}) = \{\, 2, 5 \,\}$. We say that $\mathbf{s}$ *contains* $\mathbf{r}$ if $pos(\mathbf{r}, \mathbf{s}) \neq \emptyset$. To ease notation, we treat single words from $\mathcal{V}$ also as word sequences when convenient. This allows us, for instance, to write $pos(w, \mathbf{s})$ to refer to the positions where w occurs in $\mathbf{s}$.

We let the bag of word sequences $\mathcal{C}$ denote our *document collection*. Each document $\mathbf{d} \in \mathcal{C}$ is a word sequence from $\mathcal{V}^+$. Likewise, the bag of word sequences $\mathcal{Q}$ denotes our *workload*. Each query $\mathbf{q} \in \mathcal{Q}$ is a word sequence from $\mathcal{V}^+$.

Using our notation, we now define the notion of *document frequency* as common in Information Retrieval. Let $\mathcal{S}$ be a bag of word sequences (e.g., the document collection or the workload), we define the document frequency of the word sequence $\mathbf{r}$, as the number of sequences from $\mathcal{S}$ containing it

$$df(\mathbf{r}, \mathcal{S}) = |\,\{\, \mathbf{s} \in \mathcal{S} \mid pos(\mathbf{r}, \mathbf{s}) \neq \emptyset \,\}\,| \ .$$

3 Indexing Framework

We build on the *inverted index* as the most widely-used index structure in Information Retrieval that forms the backbone of many real-world systems. The inverted index consists of two components, namely, a *dictionary* $\mathcal{D}$ of *terms* and the corresponding *posting lists* that record for each term information about its occurrences in the document collection. For a detailed discussion of the inverted index we refer to [23].

To support arbitrary phrase queries, an inverted index has to contain all words from the vocabulary in its dictionary (i.e., $\mathcal{V} \subseteq \mathcal{D}$) and record positional information in its posting lists. Thus, the posting $(\, \mathbf{d}_{13}, \langle\, 3, 7 \,\rangle\,)$ found in the posting list for word w conveys that the word occurs at positions 3 and 7 in document $\mathbf{d}_{13}$. More formally, using our notation, a posting $(\, id(\mathbf{d}), pos(w, \mathbf{d})\,)$ for word w and document $\mathbf{d}$ contains the document's unique identifier $id(\mathbf{d})$ and the positions $pos(w, \mathbf{d})$ at which the word occurs.

Query-processing performance for phrase queries on such a *positional inverted index* tends to be limited, in particular for phrase queries that contain frequent words (e.g., stopwords). Posting lists for frequent terms are long, containing many postings each of which with many positions therein, rendering them expensive to read, decompress, and process.

Several authors [8, 20, 22] have proposed, as a remedy, to augment the inverted index by adding multi-word sequences, so-called *phrases*, to the set of terms. The dictionary $\mathcal{D}$ of such an *augmented inverted index* thus consists of *individual words* alongside *phrases* (i.e., $\mathcal{D} \subseteq \mathcal{V}^+$) as terms.

To process a given phrase query $\mathbf{q}$, a set of terms is selected from the dictionary $\mathcal{D}$, and the corresponding posting lists are intersected to identify documents that contain the phrase. Intersecting of posting lists can be done using *term-at-a-time* (TAAT) or *document-at-a-time query processing* (DAAT). For the former, posting lists are read one after each other, and bookkeeping is done to keep track of positions at which the phrase can still occur in candidate documents. For the latter, posting lists are read in parallel and a document, when seen in all posting lists at once, is examined for

whether it contains the query phrase. In both cases, the cost of processing a phrase query depends on the sizes of posting lists read and thus the set of terms selected to process the query. Existing work [22] has addressed this *query-optimization problem*, determining the set of terms that should be used to process a given phrase query, using greedy heuristics. Our approach to address the problem, including its formalization and a study of its hardness, is described in Chapter 4.

4 Query Optimization

We now describe how a phrase query $\mathbf{q}$ can be processed using a given augmented inverted index with a concrete dictionary $\mathcal{D}$. Our objective is thus to determine, *at query-processing time*, a subset $\mathcal{P} \subseteq \mathcal{D}$ of terms, further referred to as *query plan*, that can be used to process $\mathbf{q}$.

To formulate the problem, we first need to capture when a query plan $\mathcal{P}$ can be used to process a phrase query $\mathbf{q}$. Intuitively, each word must be covered by at least one term from $\mathcal{P}$. We capture whether $\mathcal{P}$ *covers* $\mathbf{q}$ using the predicate

$$\begin{aligned} covers(\mathcal{P}, \mathbf{q}) \quad = \quad & \forall\, 1 \leq i \leq |\,\mathbf{q}\,| : \exists\, \mathbf{t} \in \mathcal{P} : \exists\, j \in pos(\mathbf{q}[i], \mathbf{t}) : \\ & \forall\, 1 \leq k \leq |\,\mathbf{t}\,| : \mathbf{q}[i-j+k] = \mathbf{t}[k] \end{aligned}$$

The phrase query $\mathbf{q} = \langle\, abc \,\rangle$, as a concrete example, can thus be processed using $\{\, \langle\, ab \,\rangle, \langle\, bc \,\rangle \,\}$ but not $\{\, \langle\, ab \,\rangle, \langle\, cd \,\rangle \,\}$.

Second, we need to quantify the cost of processing a phrase query using a specific query plan. As detailed above, in Chapter 3, different ways of processing a phrase query (i.e., TAAT vs. DAAT) exist. In the worst case, regardless of which query-processing method is employed, all posting lists have to be read in their entirety. We model the cost of a query plan $\mathcal{P}$ as the total number of postings that has to be read

$$c(\mathcal{P}) = \sum_{t \in \mathcal{P}} df(t, \mathcal{C}) \;.$$

While sizes of positional postings are not uniform (e.g., due to varying numbers of contained positions), suggesting collection frequency as a possibly more accurate cost measure, we found little difference in practice. The sum of document frequencies closely correlates with the response times of our system. This is in line with [14], who found that aggregate posting-list lengths is the single feature most correlated with response time for full query evaluation, as required for phrase queries, which do not permit dynamic pruning.

Assembling the above definitions of coverage and cost, we now formally define the problem of finding a cost-minimal query plan $\mathcal{P}$ for a phrase query $\mathbf{q}$ and dictionary $\mathcal{D}$ as the following $\mathcal{NP}$-hard optimization problem

Definition 1 PHRASE-QUERY OPTIMIZATION

$$\arg\min_{\mathcal{P}\subseteq\mathcal{D}} c(\mathcal{P}) \quad s.t. \quad covers(\mathcal{P}, \mathbf{q}) \; .$$

Theorem 1 PHRASE-QUERY OPTIMIZATION *is $\mathcal{NP}$-hard.*

Proof 1 (of Theorem 1) *($\mathcal{NP}$-hardness of* PHRASE-QUERY OPTIMIZATION*) Our proof closely follows Neraud [17], who establishes that deciding whether a given set of strings is elementary is $\mathcal{NP}$-complete. We show through reduction from* VERTEX COVER *that the decision variant of* PHRASE-QUERY OPTIMIZATION *(i.e., whether a $\mathcal{P}$ with $c(\mathcal{P}) \leq \tau$ exists) is $\mathcal{NP}$-complete.*

Let $G(V, E)$ be an undirected graph with vertices V and edges E. We assume that there are no isolated vertices, that is, each vertex has at least one incident edge. VERTEX COVER *asks whether there exists a subset of vertices $VC \subseteq V$ having cardinality $|\,VC\,| \leq k$, so that $\forall\,(u, v) \in E \;:\; u \in VC \vee v \in VC$, that is, for each edge one of its connected vertices is in the vertex cover. We obtain an instance of* PHRASE-QUERY OPTIMIZATION *from $G(V, E)$ as follows:*

- $\mathcal{V} = V \cup E$ *– we introduce a word to the vocabulary for each vertex (v) and edge ($\overline{uv}$) in the graph;*
- $\mathbf{q} = \biguplus_{(u,v)\in E} u\ \overline{uv}\ v$ *– we obtain the query $\mathbf{q}$ as a concatenation of all edge words $\overline{uv}$ bracketed by the words of their source (u) and target (v);*
- $\mathcal{C} = \{\,\mathbf{q}\,\}$ *– the document collection contains only a single document that equals our query;*
- $\mathcal{D} = \mathcal{V} \quad \cup \bigcup_{(u,v)\in E} \{\,\langle\, u\ \overline{uv}\,\rangle\,\} \quad \cup \bigcup_{(u,v)\in E} \{\,\langle\, \overline{uv}\ v\,\rangle\,\}$ *– each vertex word (v) and edge word ($\overline{uv}$) is indexed as well as each combination of edge and source ($u\,\overline{uv}$) and edge and target ($\overline{uv}\,v$).*

This can be done in polynomial time and space. Note also that, by construction, $\forall\,\mathbf{t} \in \mathcal{D} \;:\; df(\mathbf{t}, \mathcal{C}) = 1$ holds. We now show that $G(V, E)$ has a vertex cover with cardinality $|\,VC\,| \leq k$ iff there is a query plan $\mathcal{P}$ with $c(\mathcal{P}) \leq k + |\,E\,|$.

($\Rightarrow$) Observe that VC contains at least one of u or v for each $u\ \overline{uv}\ v$ from the query, which incurrs a cost of $|\,VC\,| \leq k$. We can complement this to obtain a query plan $\mathcal{P}$ by adding exactly one term ($\langle\,\overline{uv}\,\rangle$, $\langle\,u\,\overline{uv}\,\rangle$, or $\langle\,\overline{uv}\,v\,\rangle$) for each $u\ \overline{uv}\ v$ from the query, incurring a cost of $|\,E\,|$. Thus, $c(\mathcal{P}) \leq k + |\,E\,|$ holds.

($\Leftarrow$) Observe that $\mathcal{P}$ must cover each $u\ \overline{uv}\ v$ from the query in one of four ways: (i) $\langle\,u\,\rangle\langle\,\overline{uv}\,\rangle\langle\,v\,\rangle$, (ii) $\langle\,u\,\rangle\langle\,\overline{uv}\,v\,\rangle$, (iii) $\langle\,u\,\overline{uv}\,\rangle\langle\,v\,\rangle$ (iv) $\langle\,u\,\overline{uv}\,\rangle\langle\,\overline{uv}\,v\,\rangle$. Whenever a $u\ \overline{uv}\ v$ from the query is covered as $\langle\,u\,\overline{uv}\,\rangle\langle\,\overline{uv}\,v\,\rangle$, we replace $\langle\,u\,\overline{uv}\,\rangle$ by $\langle\,u\,\rangle$, thus reducing case (iv) to case (ii). We refer to the query plan thus obtained as $\mathcal{P}'$. Note that $c(\mathcal{P}') \leq c(\mathcal{P})$, since all terms have the same cost. $\mathcal{P}'$ contains exactly one term ($\langle\,\overline{uv}\,\rangle$, $\langle\,u\,\overline{uv}\,\rangle$, or $\langle\,\overline{uv}\,v\,\rangle$) for each $u\ \overline{uv}\ v$ from the query, incurring a cost of $|\,E\,|$. Let VC be the set of vertices whose words are contained in $\mathcal{P}'$. We can thus write $c(\mathcal{P}') = |\,VC\,| + |\,E\,|$. VC is a vertex cover, since after eliminating case (iv), each $u\ \overline{uv}\ v$ from the query is covered using either $\langle\,u\,\rangle$ or $\langle\,v\,\rangle$. Thus, $c(\mathcal{P}') = |\,VC\,| + |\,E\,| \leq c(\mathcal{P}) \leq |\,E\,| + k \Rightarrow |\,VC\,| \leq k$.

4.1 Optimal Solution

If an optimal query plan $\mathcal{P}^*$ exists, so that every term therein occurs exactly once in the query, we can determine an optimal query plan using dynamic programming based on the recurrence

$$\textsc{Opt}(i) = \begin{cases} df(\mathbf{q}[1..i], \mathcal{C}) & : \mathbf{q}[1..i] \in \mathcal{D} \\ \min\limits_{j<i} \left(\textsc{Opt}(j) + \min\limits_{\substack{k \leq j+1 \,\wedge \\ \mathbf{q}[k..i] \in \mathcal{D}}} df(\mathbf{q}[k..i], \mathcal{C}) \right) & : \text{ otherwise} \end{cases}$$

in time $\mathcal{O}(n^2)$ and space $\mathcal{O}(n)$ where $|\,\mathbf{q}\,| = n$. $\textsc{Opt}(i)$ denotes the cost of an optimal solution to the prefix subproblem $\mathbf{q}[1..i]$ – once the dynamic-programming table has been populated, an optimal query plan can be constructed by means of backtracking. In the first case, the prefix subproblem can be covered using a single term. In the second case, the optimal solution combines an optimal solution to a smaller prefix subproblem, which is the optimal substructure inherent to dynamic programming, with a single term that covers the remaining suffix.

Theorem 2 *If an optimal query plan $\mathcal{P}^*$ for a phrase query $\mathbf{q}$ exists such that*

$$\forall\, \mathbf{t} \in \mathcal{P}^* \ :\ |\,pos(\mathbf{t}, \mathbf{q})\,| = 1\ ,$$

then $c(\mathcal{P}^*) = \textsc{Opt}(|\,\mathbf{q}\,|)$, that is, an optimal solution can be determined using the recurrence Opt.

Proof 2 (of Theorem 2) *Observe that Theorem 2 is a special case of Theorem 3 for $\mathcal{F} = \emptyset$. We therefore only prove the more general Theorem 3.*

It entails that we can efficiently determine optimal query plans for phrase queries with no repeated words.

Corollary 1 *We can compute an optimal query plan for a phrase query* $\mathbf{q}$ *in polynomial time and space, if*

$$\forall\, 1 \leq i \leq |\,\mathbf{q}\,| \;:\; |\,pos(\mathbf{q}[i], \mathbf{q})\,| = 1\ .$$

In practice this special case is important, since a large fraction of phrases queries in typical workloads do not contain repeated words.

Otherwise, when there is no optimal query plan $\mathcal{P}^*$ according to Theorem 2, dynamic programming can not be directly applied, since there is no optimal substructure. Consider, as a concrete problem instance, the phrase query $\mathbf{q} = \langle\, abxayb \,\rangle$ with dictionary $\mathcal{D} = \{\, \langle\, a \,\rangle, \langle\, b \,\rangle, \langle\, x \,\rangle, \langle\, y \,\rangle, \langle\, ab \,\rangle \,\}$ and assume $df(\mathbf{t}, \mathcal{C}) > 1$ for $\mathbf{t} \in \{\, \langle\, a \,\rangle, \langle\, b \,\rangle, \langle\, x \,\rangle, \langle\, y \,\rangle \,\}$ and $df(\langle\, ab \,\rangle, \mathcal{C}) = 1$. Here, the optimal solution $\mathcal{P}^* = \{\, \langle\, a \,\rangle, \langle\, b \,\rangle, \langle\, x \,\rangle, \langle\, y \,\rangle \,\}$ does not contain an optimal solution to any prefix subproblem $\mathbf{q}[1..i]$ $(1 < i < |\,\mathbf{q}\,|)$, which all contain the term $\langle\, ab \,\rangle$.

However, as we describe next, an optimal query plan can be computed, in the general case, using a combination of exhaustive search over sets of repeated terms and a variant of our above recurrence.

For a phrase query $\mathbf{q}$ let the set of repeated terms be formally defined as

$$\mathcal{R} = \{\, \mathbf{t} \in \mathcal{D} \;\mid\; |\,pos(\mathbf{t}, \mathbf{q})\,| > 1 \,\}\ .$$

Let further $\mathcal{F} \subseteq \mathcal{R}$ denote a subset of repeated terms. We now define a modified document frequency that is zero for terms from $\mathcal{F}$, formally

$$df'(\mathbf{t}, \mathcal{C}) = \begin{cases} 0 & : \quad \mathbf{t} \in \mathcal{F} \\ df(\mathbf{t}, \mathcal{C}) & : \quad \text{otherwise} \end{cases}$$

and denote by $\textsc{Opt}'$ the variant of our above recurrence that uses this modified document frequency.

Algorithm 1 considers all subsets of repeated terms and, for each of them, extends it into a query plan for $\mathbf{q}$ by means of the recurrence $\textsc{Opt}'$. The algorithm keeps track of the best solution seen and eventually returns it. Its correctness directly follows from the following theorem.

Theorem 3 *Let $\mathcal{P}^*$ denote an optimal query plan for the phrase query $\mathbf{q}$ and $\mathcal{F} = \{\, \mathbf{t} \in \mathcal{P}^* \;\mid\; |\,pos(\mathbf{t}, \mathbf{q})\,| > 1 \,\}$ be the set of repeated terms therein, then*

$$c(\mathcal{F}) + \textsc{Opt}'(|\,\mathbf{q}\,|) \leq c(\mathcal{P}^*)\ .$$

Algorithm 1: PHRASE-QUERY OPTIMIZATION

Input: Phrase query $\mathbf{q}$, dictionary $\mathcal{D}$
Output: Cost *optCost* of optimal query plan

```
1 R = { t ∈ D : |pos(t, q)| > 1 }
2 optCost = ∞
3 for F ∈ 2^R do
4     cost = c(F) + Opt'(|q|)
5     if cost < optCost then
6         optCost = cost
7 return optCost
```

Proof 3 (of Theorem 3) *Let $\mathcal{P}^*$ denote an optimal query plan for the phrase query $\mathbf{q}$ and*

$$\mathcal{F} = \{\, \mathbf{t} \in \mathcal{P}^* \mid |\mathit{pos}(\mathbf{t}, \mathbf{q})| > 1 \,\}$$

be the set of repeated terms and $\bar{\mathcal{F}} = \mathcal{P}^* \setminus \mathcal{F}$ be the set of non-repeated terms therein. Without loss of generality, we assume that $\mathbf{q}$ ends in a non-repeated terminal term # having $\mathit{df}(\#, \mathcal{C}) = 0$ – this can always be achieved by "patching" the query. We order non-repeated terms $\mathbf{t} \in \bar{\mathcal{F}}$ by their single item in $\mathit{pos}(\mathbf{t}, \mathbf{q})$ to obtain the sequence $\langle\, \mathbf{t}_1, \ldots, \mathbf{t}_m \,\rangle$ with $m = \left|\, \bar{\mathcal{F}} \,\right|$. We refer to the first position covered by $\mathbf{t}_i$, corresponding to the single item in $\mathit{pos}(\mathbf{t}, \mathbf{q})$, as b_i and to the last position as $e_i = (b_i + |\, \mathbf{t}_i \,| - 1)$.

We now show by induction *that*

$$\textsc{Opt}'(e_i) \leq \sum_{j=1}^{i} \mathit{df}(\mathbf{t}_j, \mathcal{C}) = c(\mathcal{P}^*) - c(\mathcal{F}) \;.$$

($i = 1$) We have to distinguish two cases: (i) $b_1 = 1$, that is, $\mathbf{q}[1..e_1]$ is covered using a single non-repeated term – OPT' selects this term according to its first case. (ii) $b_1 > 1$, that is, there is a set of repeated terms from $\mathcal{F}$ that covers $\mathbf{q}[1..k]$ for some $b_1 - 1 \leq k < e_1$ – OPT' can select the same repeated terms at zero cost and combine it with $\mathbf{t}_1$ that covers $\mathbf{q}[b_1..e_1]$. Thus, in both cases, $\textsc{Opt}'(e_1) \leq \mathit{df}(\mathbf{t}_1, \mathcal{C})$.

($i \to i + 1$) We assume $\textsc{Opt}'(e_i) \leq \sum_{j=1}^{i} \mathit{df}(\mathbf{t}_j, \mathcal{C})$. Again, we have to distinguish two cases: (i) $e_i \geq b_{i+1} - 1$, that is, the term before $\mathbf{t}_{i+1}$ is also a non-repeated term. Thus, our recurrence considers $\textsc{Opt}'(e_i) + \mathit{df}(\mathbf{t}_{i+1}, \mathcal{C})$ as one possible solution. (ii) $e_1 < b_{i+1} - 1$, that is, there is a gap covered by repeated terms between $\mathbf{t}_i$ and $\mathbf{t}_{i+1}$ – OPT' can select the same repeated terms at zero cost and thus considers $\textsc{Opt}'(e_i) + 0 + \mathit{df}(\mathbf{t}_{i+1}, \mathcal{C})$ as one solution. Thus, in both cases, $\textsc{Opt}'(e_{i+1}) \leq \sum_{j=1}^{i+1} \mathit{df}(\mathbf{t}_j, \mathcal{C})$.

The cost of Algorithm 1 depends on the number of repeated terms $|\mathcal{R}|$, which is small in practice and can be bounded in terms of the number of positions in $\mathbf{q}$ occupied by a repeated word

$$r = |\{ 0 \leq i \leq |\mathbf{q}| \mid |pos(\mathbf{q}[i], \mathbf{q})| > 1 \}| \ .$$

For our above example phrase query $\mathbf{q} = \langle abxayb \rangle$ we obtain $r = 4$. Note that $|R| \leq \frac{r \cdot (r+1)}{2}$ holds. Algorithm 1 thus has time complexity $\mathcal{O}(2^{\frac{r \cdot (r+1)}{2}} n^2)$ and space complexity $\mathcal{O}(n^2)$ where $|\mathbf{q}| = n$.

4.2 Approximation Algorithm

Computing an optimal query plan can be computationally expensive in the worst case, as just shown. It turns out, however, that we can efficiently compute an $\mathcal{O}(log\, n)$-approximation, reusing results for Set Cover [21].

To this end, we convert an instance of our problem, consisting of a phrase query $\mathbf{q}$ and a dictionary $\mathcal{D}$ with associated costs, into a Set Cover instance. Let the universe of items $\mathcal{U} = \{ 1, \ldots, |\mathbf{q}| \}$ correspond to positions in the phrase query. For each term $\mathbf{t} \in \mathcal{D}$, we define a subset $S_{\mathbf{t}} \subseteq \mathcal{U}$ of covered positions as

$$S_{\mathbf{t}} = \{ 1 \leq i \leq |\mathbf{q}| \mid \exists j \in pos(\mathbf{t}, \mathbf{q}) : j \leq i < j + |\mathbf{t}| \} \ .$$

The collection of subsets of $\mathcal{U}$ is then defined as $\mathcal{S} = \{ S_{\mathbf{t}} \mid \mathbf{t} \in \mathcal{D} \}$ and we define $cost(S_{\mathbf{t}}) = df(\mathbf{t}, \mathcal{C})$ as a cost function.

For our concrete problem instance from above, we obtain $\mathcal{U} = \{ 1, \ldots, 6 \}$ and $\mathcal{S} = \{ \{ 1, 4 \}, \{ 2, 6 \}, \{ 3 \}, \{ 5 \}, \{ 1 \} \}$ as a Set Cover instance.

We can now use the greedy algorithm that selects sets from $\mathcal{S}$ based on their benefit-cost ratio, which is known to be a $\mathcal{O}(log\, n)$-approximation algorithm [21]. This can be implemented in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space where $|q| = n$.

Note that, as a key difference to the greedy algorithm described in [22], which to the best of our knowledge does not give an approximation guarantee, our greedy algorithm selects subsets (corresponding to terms from the dictionary) taking into account the number of additional items covered and the coverage already achieved by selected subsets.

5 Experimental Evaluation

We now describe our experimental evaluation. We begin with the description of the datasets, followed by a comparison of the query-optimization methods from Chapter 4.

5.1 Datasets

Document Collections. We use two real-world document collections for our experiments:

- *ClueWeb09-B* [2] (CW) – more than 50 million web documents in English language crawled in 2009;
- *The New York Times Annotated Corpus* [4] (NYT) – more than 1.8 million newspaper articles published by The New York Times between 1987 and 2007.

Both document collections were processed using Stanford CoreNLP [3] for tokenization. To make CW more handleable, we use boilerplate detection as described in [12] and available in the `DefaultExtractor` of boilerpipe [1].

	# Queries	# Distinct	ø Length
YAGO	5,720,063	4,599,745	2.45
MSN	10,428,651	5,627,838	3.58
MSNP	131,857	105,825	3.37
NYTS	1,000,000	970,051	21.66
CWS	1,000,000	929,607	20.55

Table 5.1: Workload characteristics

Workloads. To reflect different use cases including web search, entity-oriented search, and plagiarism detection, we consider four different workloads for our experimental evaluation:

- **MSN** is a query workload made available for research by a commercial web search engine in 2009. It contains queries routed to the US Microsoft search site and sampled over one month (May 2006).
- **MSNP** as the subset of explicit phrase queries from the aforementioned query workload, i.e., queries enclosed in quotes (e.g., "national pandemic influenza response plan").
- **YAGO** is a workload of entity labels from the YAGO2 knowledge base [11]. In its `rdfs:label` relation, it collects strings that may refer to a specific entity, which are mined from anchor texts in Wikipedia. For the entity `Bob_Dylan`, as a concrete example, it includes among others the entity labels "bob dylan", "bob allen zimmerman", and "robert allen zimmerman".
- **NYTS/CWS** as workloads consisting of 1,000,000 sentences randomly sampled from the NYT and CW document collection, respectively.

Table 5.1 reports characteristics of the different workloads. Note that we excluded single-word queries from all workloads. Interestingly, queries are on average shorter in the YAGO workload (2.45 words) than in the web search query workloads. By design, the NYTS and CWS workloads consist of substantially longer phrase queries. We use document frequency as a cost measure for all our experiments. As mentioned earlier, one could use collection frequency instead. In practice, though, the two measures are highly correlated and we did not observe big differences. Also, as a one-time preprocessing performed using Hadoop and made available to all methods, we compute n-gram statistics for the workload and document collection using the method described in [6].

5.2 Experimental Results

The experiment examines the effect that the choice of query-optimization method can have on query-processing performance. We consider three query-optimization methods for this experiment: the greedy algorithm (GRD) from [22], our greedy algorithm (APX) that comes with an approximation guarantee, and our exponential exact algorithm (OPT). GRD considers terms

in increasing order of their document frequency, thus based on their selectivity, and chooses a term if it covers any yet-uncovered portion of the phrase query. Originally designed to deal with bigrams only, we extend GRD to break ties based on term length, and thus favor the longer term, if two terms have the same document frequency.

To compare the three query-optimization methods, we built augmented inverted indexes whose dictionaries include all phrases up to a specific maximum length $l \in \{ 2, 3, 4, 5 \}$. Thus, for $l = 5$, all phrases of length five or less are indexed. This allows us to study the behavior of the methods as more terms to choose from become available.

Table 5.1 reports average query-processing costs for all sensible combinations of our document collections and workloads for different choices of l. When studying the behavior of the different query-optimization methods on an augmented inverted index obtained for a specific choice of l, we exclude queries from all workloads that consist of fewer than l words. This is reasonable, since those queries can simply be processed by looking up the corresponding n-gram and there is nothing to optimize.

As we can see from the table, the approximate solutions (GRD and APX) work well, and their costs are close to the ones obtained with the optimal algorithm (OPT). As expected, the performance of APX, our approximation algorithm, is closer to optimal than the performance of the heuristic GRD. This difference is more pronounced on the NYTS and CWS workloads consisting, by construction, of verbose queries.

Also, as expected, we observe that with increasing l, the number of posting read per query decreases for all query optimizers. The improvements are drastic when considering sentence workloads for all optimizers. For shorter queries there is less room for query optimization, as observed in the YAGO and MSNP workloads. For longer queries, in contrast, there is more room for query optimization and thus an opportunity for OPT and APX to make better choices, as observed on the NYTS/CWS workloads consisting of verbose queries. This is even more noticeable for larger choices of l, as can be seen from the results obtained for $l = 4$ on CWS where OPT processes less than 50% of postings than GRD.

We also observe that a majority of queries does not contain repeated words. Even otherwise the number of repetitions is typically small. This has a favorable impact on the execution time of OPT. We observe that all query-optimization methods perform similarly in terms of execution time. Our optimization methods are thus robust and achieve superior performance to GRD.

	NYT											
	GRD				**APX**				**OPT**			
	$l=2$	$l=3$	$l=4$	$l=5$	$l=2$	$l=3$	$l=4$	$l=5$	$l=2$	$l=3$	$l=4$	$l=5$
YAGO	23,070	18,793	16,978	15,693	22,997	18,775	16,970	15,682	22,854	18,707	16,961	15,674
MSNP	32,466	21,415	19,866	19,734	32,177	21,392	19,857	19,726	31,796	21,333	19,844	19,714
MSN	29,755	22,635	21,596	21,442	29,596	22,621	21,590	21,436	29,300	22,590	21,582	21,427
NYTS	238,270	10,466	1,035	203	228,117	9,833	955	182	217,068	9,135	895	170
	CW											
	GRD				**APX**				**OPT**			
	$l=2$	$l=3$	$l=4$	$l=5$	$l=2$	$l=3$	$l=4$	$l=5$	$l=2$	$l=3$	$l=4$	$l=5$
YAGO	142,451	67,046	58,463	55,651	140,659	66,398	58,374	55,572	138,889	65,938	58,327	55,537
MSNP	281,427	87,705	77,695	95,180	275,841	86,986	77,634	95,145	268,994	86,413	77,552	95,103
MSN	285,823	150,526	163,286	209,294	282,217	150,130	163,257	209,278	277,676	149,829	163,219	209,250
CWS	2,846,711	221,089	83,170	53,795	2,692,296	195,132	43,256	33,167	2,576,273	186,095	41,874	32,252

Table 5.2: Query optimization results (# postings read)

6 Related Work

Williams et al. [22] put forward the *combined index* to support phrase queries efficiently. It assembles three levels of indexing: (i) a *first-word index* as a positional inverted index, (ii) a *next-word index* that indexes all bigrams containing a stopword, and (iii) a *phrase index* with popular phrases from a query log. Its in-memory dictionary is kept compact by exploiting common first words between bigrams. Query processing escalates through these indexes – first it consults the phrase index and, if the phrase query is not found therein, processes it using bigrams and unigrams from the other indexes. Transier and Sanders [20] select bigrams to index based only on characteristics of the document collection. Selecting bigrams makes sense in settings where phrase queries are issued by human users and tend to be short – as observed for web search by Spink et al. [18]. Variable-length multi-word sequences have previously been considered by Chang and Poon [8] in their *common phrase index*, which builds on [22], but indexes variable-length phrases common in the workload.

7 Conclusion

We have presented a comprehensive approach to process phrase queries using an inverted index whose dictionary is augmented by variable-length word sequences. We studied the problem of *query optimization*, to determine cost-efficient plans for phrase queries on a given index. We also proposed solutions to select an optimal (or close to optimal) query plan for processing a phrase query over an augmented inverted index. Lastly, we performed an extensive evaluation of our approaches using real-world datasets.

Bibliography

[1] Boilerpipe
`http://code.google.com/p/boilerpipe/`.

[2] ClueWeb09
`http://corpus.nytimes.com`.

[3] Stanford CoreNLP
`http://nlp.stanford.edu/software/corenlp.shtml`.

[4] The New York Times Annotated Corpus
`http://corpus.nytimes.com`.

[5] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable ad-hoc entity extraction from text collections. *PVLDB*, 1(1):945–957, 2008.

[6] K. Berberich and S. J. Bedathur. Computing n-gram statistics in mapreduce. In *EDBT*, pages 101–112, 2013.

[7] S. Büttcher, C. L. A. Clarke, and G. V. Cormack. *Information Retrieval - Implementing and Evaluating Search Engines*. The MIT Press, 2010.

[8] M. Chang and C. K. Poon. Efficient phrase querying with common phrase index. *Inf. Process. Manage.*, 44(2):756–769, 2008.

[9] P. Ferragina and R. Venturini. The compressed permuterm index. *ACM Transactions on Algorithms*, 7(1):10, 2010.

[10] M. Hagen, M. Potthast, A. Beyer, and B. Stein. Towards optimum query segmentation: in doubt without. In *CIKM*, pages 1015–1024, 2012.

[11] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[12] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *WSDM*, pages 441–450, 2010.

[13] Y. Li, B.-J. P. Hsu, C. Zhai, and K. Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *SIGIR*, pages 285–294, 2011.

[14] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *SIGIR*, pages 621–630, 2012.

[15] U. Manber and E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

[16] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative Analysis of Culture Using Millions of Digitized Books. *Science*, 2010.

[17] J. Neraud. Elementariness of a finite set of words is co-np-complete. *ITA*, 24:459–470, 1990.

[18] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.

[19] E. Stamatatos. Plagiarism detection based on structural information. In *CIKM*, pages 1221–1230, 2011.

[20] F. Transier and P. Sanders. Out of the box phrase indexing. In *SPIRE*, pages 200–211, 2008.

[21] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[22] H. E. Williams, J. Zobel, and D. Bahle. Fast phrase querying with combined indexes. *ACM Trans. Inf. Syst.*, 22(4):573–594, 2004.

[23] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.

Below you find a list of the most recent research reports of the Max-Planck-Institut für Informatik. Most of them are accessible via WWW using the URL `http://www.mpi-inf.mpg.de/reports`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
– Library and Publications –
Campus E 1 4

D-66123 Saarbrücken

E-mail: `library@mpi-inf.mpg.de`

MPI-I-2014-5-001	M. Dylla, M. Theobald	Learning Tuple Probabilities in Probabilistic Databases
MPI-I-2013-RG1-002	P. Baumgartner, U. Waldmann	Hierarchic superposition with weak abstraction
MPI-I-2013-5-002	F. Makari, B. Awerbuch, R. Gemulla, R. Khandekar, J. Mestre, M. Sozio	A distributed algorithm for large-scale generalized matching
MPI-I-2013-1-001	C. Huang, S. Ott	New results for non-preemptive speed scaling
MPI-I-2012-RG1-002	A. Fietzke, E. Kruglov, C. Weidenbach	Automatic generation of inductive invariants by SUP(LA)
MPI-I-2012-RG1-001	M. Suda, C. Weidenbach	Labelled superposition for PLTL
MPI-I-2012-5-004	F. Alvanaki, S. Michel, A. Stupar	Building and maintaining halls of fame over a database
MPI-I-2012-5-003	K. Berberich, S. Bedathur	Computing n-gram statistics in MapReduce
MPI-I-2012-5-002	M. Dylla, I. Miliaraki, M. Theobald	Top-k query processing in probabilistic databases with non-materialized views
MPI-I-2012-5-001	P. Miettinen, J. Vreeken	MDL4BMF: Minimum Description Length for Boolean Matrix Factorization
MPI-I-2012-4-001	J. Kerber, M. Bokeloh, M. Wand, H. Seidel	Symmetry detection in large scale city scans
MPI-I-2011-RG1-002	T. Lu, S. Merz, C. Weidenbach	Towards verification of the pastry protocol using TLA+
MPI-I-2011-5-002	B. Taneva, M. Kacimi, G. Weikum	Finding images of rare and ambiguous entities
MPI-I-2011-5-001	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Temporal index sharding for space-time efficiency in archive search
MPI-I-2011-4-005	A. Berner, O. Burghard, M. Wand, N.J. Mitra, R. Klein, H. Seidel	A morphable part model for shape manipulation
MPI-I-2011-4-003	J. Tompkin, K.I. Kim, J. Kautz, C. Theobalt	Videoscapes: exploring unstructured video collections
MPI-I-2011-4-002	K.I. Kim, Y. Kwon, J.H. Kim, C. Theobalt	Efficient learning-based image enhancement : application to compression artifact removal and super-resolution
MPI-I-2011-4-001	M. Granados, J. Tompkin, K. In Kim, O. Grau, J. Kautz, C. Theobalt	How not to be seen inpainting dynamic objects in crowded scenes
MPI-I-2010-RG1-001	M. Suda, C. Weidenbach, P. Wischnewski	On the saturation of YAGO
MPI-I-2010-5-008	S. Elbassuoni, M. Ramanath, G. Weikum	Query relaxation for entity-relationship search
MPI-I-2010-5-007	J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum	YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia
MPI-I-2010-5-006	A. Broschart, R. Schenkel	Real-time text queries with tunable term pair indexes
MPI-I-2010-5-005	S. Seufert, S. Bedathur, J. Mestre, G. Weikum	Bonsai: Growing Interesting Small Trees
MPI-I-2010-5-004	N. Preda, F. Suchanek, W. Yuan, G. Weikum	Query evaluation with asymmetric web services
MPI-I-2010-5-003	A. Anand, S. Bedathur, K. Berberich, R. Schenkel	Efficient temporal keyword queries over versioned text

MPI-I-2010-5-002	M. Theobald, M. Sozio, F. Suchanek, N. Nakashole	URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules
MPI-I-2010-5-001	K. Berberich, S. Bedathur, O. Alonso, G. Weikum	A language modeling approach for temporal information needs
MPI-I-2010-1-001	C. Huang, T. Kavitha	Maximum cardinality popular matchings in strict two-sided preference lists
MPI-I-2009-RG1-005	M. Horbach, C. Weidenbach	Superposition for fixed domains
MPI-I-2009-RG1-004	M. Horbach, C. Weidenbach	Decidability results for saturation-based model building
MPI-I-2009-RG1-002	P. Wischnewski, C. Weidenbach	Contextual rewriting
MPI-I-2009-RG1-001	M. Horbach, C. Weidenbach	Deciding the inductive validity of $\forall\exists^*$ queries
MPI-I-2009-5-007	G. Kasneci, G. Weikum, S. Elbassuoni	MING: Mining Informative Entity-Relationship Subgraphs
MPI-I-2009-5-006	S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, G. Weikum	Scalable phrase mining for ad-hoc text analytics
MPI-I-2009-5-005	G. de Melo, G. Weikum	Towards a Universal Wordnet by learning from combined evidenc
MPI-I-2009-5-004	N. Preda, F.M. Suchanek, G. Kasneci, T. Neumann, G. Weikum	Coupling knowledge bases and web services for active knowledge
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-003	T. Neumann, G. Weikum	The RDF-3X engine for scalable management of RDF data
MPI-I-2009-5-002	M. Ramanath, K.S. Kumar, G. Ifrim	Generating concise and readable summaries of XML documents
MPI-I-2009-4-006	C. Stoll	Optical reconstruction of detailed animatable human body models
MPI-I-2009-4-005	A. Berner, M. Bokeloh, M. Wand, A. Schilling, H. Seidel	Generalized intrinsic symmetry detection
MPI-I-2009-4-004	V. Havran, J. Zajac, J. Drahokoupil, H. Seidel	MPI Informatics building model as data for your research
MPI-I-2009-4-003	M. Fuchs, T. Chen, O. Wang, R. Raskar, H.P.A. Lensch, H. Seidel	A shaped temporal filter camera
MPI-I-2009-4-002	A. Tevs, M. Wand, I. Ihrke, H. Seidel	A Bayesian approach to manifold topology reconstruction
MPI-I-2009-4-001	M.B. Hullin, B. Ajdin, J. Hanika, H. Seidel, J. Kautz, H.P.A. Lensch	Acquisition and analysis of bispectral bidirectional reflectance distribution functions
MPI-I-2008-RG1-001	A. Fietzke, C. Weidenbach	Labelled splitting
MPI-I-2008-5-004	F. Suchanek, M. Sozio, G. Weikum	SOFIE: a self-organizing framework for information extraction
MPI-I-2008-5-003	G. de Melo, F.M. Suchanek, A. Pease	Integrating Yago into the suggested upper merged ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single phase construction of optimal DAG-structured QEPs
MPI-I-2008-5-001	G. Kasneci, M. Ramanath, M. Sozio, F.M. Suchanek, G. Weikum	STAR: Steiner tree approximation in relationship-graphs
MPI-I-2008-4-003	T. Schultz, H. Theisel, H. Seidel	Crease surfaces: from theory to extraction and application to diffusion tensor MRI
MPI-I-2008-4-002	D. Wang, A. Belyaev, W. Saleem, H. Seidel	Estimating complexity of 3D shapes using view similarity
MPI-I-2008-1-001	D. Ajwani, I. Malinger, U. Meyer, S. Toledo	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for finite domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : a large ontology from Wikipedia and WordNet