# Improving Document Ranking for Long Queries with Nested Query Segmentation

Rishiraj Saha Roy[1], Anusha Suresh[1], Niloy Ganguly[1], Monojit Choudhury[2], Deepak Shankar[3], and Tanwita Nimiar[3]

[1] Indian Institute of Technology Kharagpur, Kharagpur, India – 721302.
{rishiraj, anusha.suresh, niloy}@cse.iitkgp.ernet.in
[2] Microsoft Research India, Bangalore, India – 560027.
monojitc@microsoft.com
[3] R. V. College of Engineering, Bangalore, India – 560059.
{deepak.shankar94, ntanwita21}@gmail.com

**Abstract.** Past work on query segmentation has exclusively focused on flat or non-hierarchical segmentation, where query words are simply partitioned into non-overlapping contiguous chunks of words. Such an approach suffers from the problem of granularity, and consequent difficulties in IR application. Here, we explore nested or hierarchical query segmentation, where segments are defined recursively as consisting of contiguous sequences of segments or query words, as an effective and powerful alternative representation of a query. We design a lightweight and unsupervised nested segmentation scheme, and propose how to use the tree arising out of the nested representation of a query to improve retrieval performance. We examine several aspects of the IR application framework and show that nested segmentation can be suitably exploited for the re-ranking of documents leading to significant gains over baselines that include the state-of-the-art flat segmentation strategies, and can provide benefits when combined with the latest term proximity model.

## 1 Introduction

Query segmentation [1–3] is one of the first steps towards query understanding where complex queries are partitioned into semantically coherent word sequences. Recent studies [2, 3] have shown that segmentation can potentially lead to better IR performance. Till date, almost all the works on query segmentation have dealt with *flat* or *non-hierarchical segmentations*, as shown below:

```
windows xp home edition | hd video | playback
```

where pipes (|) represent flat segment boundaries. In flat segmentation, it is hard to specify the appropriate *granularity* or the expected length of the segments. For example, human annotators (or algorithms) that prefer shorter segments may split the first segment into `windows xp` and `home edition`, while others may choose not to break the sequence `video playback`. This leads to confusion amongst annotators leading to low inter-annotator agreement for manually labeled segments [1, 4], and in turn, makes evaluation of query segmentation a difficult problem [3]. Instead of evaluating against human annotated data, a better approach to evaluation of segmentation, therefore, is to use segmented queries directly in IR or some similar application and calculate the respective performance gains. To this end, Saha Roy et al. [3] have recently proposed a framework which estimates the potential of a segmentation strategy in improving IR. While their oracle-based approach might be useful in evaluating the potential of an algorithm, it does not tell us how exactly one can use the segmentation information for harnessing this potential during the retrieval process.

Nevertheless, the issue of an "ideal" granularity creates confusion in a retrieval-based setting. First, whether longer or shorter segments should be preferred purely depends on the query and document pair in question during the search process. Hence, a flat segmentation algorithm consistently adopting either of the two strategies (long or short segments) will fail in several contexts. Next, when the generated segment (say, `windows xp home edition`) is matched only partially in the document (say, as `office xp home edition` or `windows xp pro edition`), a flat segmentation algorithm relying on exact (or approximate) string matching fails to understand that the latter case is much more relevant than the former. These difficulties of granularity associated with flat segmentation can be effectively addressed if we allow nesting or embedding of segments inside bigger segments. For instance, instead of a flat segmentation, our running example query could be more meaningfully represented as follows:

```
(((windows xp) home) edition) ((hd video) playback)
```

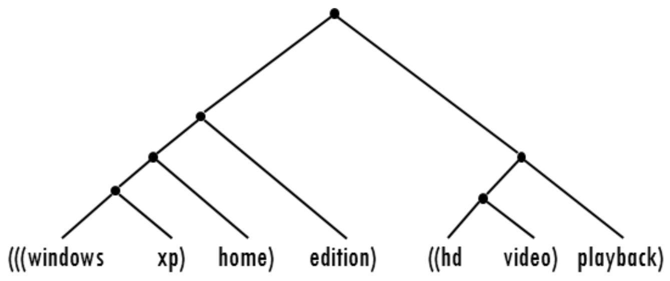


**Fig. 1.** Nested segmentation tree.

Here, the atomic segments – `windows xp` and `hd video`, are progressively joined with other words to produce larger segments – `windows xp home`, `windows xp home edition`, and `hd video playback`. We shall refer to this process as *nested* (or *hierarchical*) *query segmentation*. The hierarchy in this form of syntactic analysis is better visualized through a *nested segmentation tree* as shown in Fig. 1. It is intuitive from this representation that `windows xp` and `hd video` are non-negotiable (atomic units) when it comes to matching within documents, and the strength of ties between word pairs can be said to weaken as they move farther in terms of the (unique) path through the tree. This observation forms the basis of our re-ranking scheme that addresses the issue of non-exact segment matching in documents.

In this work, we develop an algorithm and an evaluation methodology for nested query segmentation that can actually be used to apply nested segmentation to IR (unlike the oracle-based framework proposed by Saha Roy et al. [3]). Our nested segmentation algorithm is based on some very simple yet powerful local statistical and linguistic information. Through a detailed evaluation of the various aspects involved and using two different datasets, we demonstrate that nested segmentation is not only a more informative representation of the query, but also can be exploited to gain better IR performances especially for slightly long queries ($\geq 3$ words). Note that nested segmentation (or *chunking*), which is a very intuitive representation of natural language (NL) sentences [5] and more specifically phrase structure grammar, has hardly been used for representing queries. A possible reason for the low attention paid to this problem could be that the deduction of hierarchical structure in NL sentences heavily relies on accurate part-of-speech (POS) tagging of the words and an underlying grammar. More importantly, such an analysis adds a non-trivial runtime overhead during query processing. Furthermore, there is no prevalent notion of grammatical syntax for Web search queries which could provide a sound basis for a hierarchical query structure.

**Approach.** In absence of linguistic cues, we adopt a purely statistical approach. The intuitions behind our approach are as follows. State-of-the-art flat segmentation algorithms involve a word association score optimization over all the words of the query, and hence flat segments contain vital information that should be utilized effectively. Our objective is to discover more detailed query structure by finding interesting relationships *within* flat segments, and *between* different flat segments. Structure *within* flat segments is determined by an exhaustive search over lower order constituent $n$-grams, and such an approach is feasible in this context because the lengths of flat segments rarely exceed five words. The relative strengths of bigrams straddling flat segment boundaries is exploited in inferring the relationships *between* different flat segments. Relevant bigram statistics, again, are already available. These strategies help us discover the hierarchical structure within a query, which is subsequently harnessed during document re-ranking. This document re-ranking strategy, in turn, is our instrument for directly applying nested segmentation to improve result quality.

**Contributions.** This paper is the first to harness the power of deep query structure through nested segmentation and use it to improve ranking. A highlight of our approach is a principled way of dealing with cases where certain words of a query (segment) are absent in the documents, i.e., an exact match of a segment is not found. Specifically, in this research, keeping the above perspectives in mind, we (a) develop an *unsupervised and lightweight* technique for nested segmentation that uses query logs as the only resource; (b) design a *deterministic* document re-ranking strategy exploiting the nested representation of the query; (c) demonstrate that the use of nested segmentation can lead to significant improvement in document re-ranking over the state-of-the-art flat segmentation strategies; and *(d)* we also extend the framework proposed in Saha Roy et al. [3] to nested segmentation and show that nested segmentation indeed has a much higher potential for IR improvement than its flat counterpart. To ensure reproducibility of results, we are sharing the nested segmentations of our queries along with the complete generation and IR evaluation code[4].

**Organization.** Sec. 2 discusses current techniques in flat query segmentation, their limitations, and the corresponding benefits of nested segmentation. Basic concepts and necessary terminology are defined in Sec. 3. Sec. 4 presents our algorithm for generating nested segmentations. Next, in Sec. 5, we discuss the technique for using nested segmentation to improve result ranking. We describe datasets used in Sec. 6. Sec. 7 describes the experimental results and observations. In Sec. 8, we report the method for extension of the evaluation framework by Saha Roy et al. [3] to nested segmentation. Sec. 9 reviews research on proximity and dependence models, indirectly related to this work. Sec. 10 concludes the paper by summarizing our contributions and highlighting promising future research directions.

## 2 Issues with Flat Segmentation

Query segmentation was proposed by Risvik et al. [6], where the authors used frequencies and mutual information of $n$-grams learnt from query logs to come up with meaningful segmentations for queries. The next ten years saw a plethora of work on segmentation using diverse resources like Wikipedia titles [4], click-through data [2] and query logs [7], while using distinct algorithmic approaches like eigenspace similiarity [8], conditional random fields [9] and expectation maximization [4, 2]. Query segmentation has also been applied to domains other than Web search, like patent search [10] and product search [11, 12]. Unsupervised methods [13, 4, 8, 14–16, 2, 7, 3, 12] have outnumbered supervised techniques [17, 18, 9, 19, 20], as the latter relies on human annotations for training which is quite expensive to obtain in large volumes. Moreover, it is always difficult to achieve good query coverage from various domains in the datasets used for supervised learning [12]. Bergsma and Wang [17] had released a set of 500 human annotated queries sampled from the 2006 AOL query log [21], which facilitated comparison of various segmentation algorithms based on matching metrics. Subsequently, Hagen et al. [16] created a much larger and cleaner dataset of 53, 437 queries[5] (*Webis-QSeC-10*) which was accompanied by ten segmentation annotations each, collected through the Amazon Mechanical Turk[6] crowdsourcing platform. Currently, $\simeq 10\%$ of this dataset is publicly available. Subsequent research has identified several limitations and issues with validation against human

[4] `http://cse.iitkgp.ac.in/resgrp/cnerg/qa/nestedsegmentation.html`, Accessed 24 Oct 2015

[5] `http://www.webis.de/research/corpora`, Accessed 24 Oct 2015

[6] https://www.mturk.com/mturk/welcome, Accessed 24 Oct 2015

segmentations, and an alternative and more appropriate framework for evaluating query segmentation has been proposed [3].

## 2.1 Limitations of flat segmentation

There are two important conceptual deficiencies of flat segmentation [16, 2, 7, 3, 4]: its definition and its use in IR. These two issues are, in fact, very closely related because it seems impossible to posit a definition of a segment without an IR model in place. More often than not, the definition of segmentation is presented vaguely as grouping of "semantically meaningful units" together [4]. Ultimately, it is the segmentation strategy that provides an implicit definition of the concept. Needless to say, such definitions and guidelines leave out scope for a subjective interpretation of segments leading to low inter-annotator agreement on manually annotated queries ($\simeq 58 - 73\%$ on most metrics [4]). See [1, 3] for issues with evaluation against human annotations. However, the problem is deeper than just being an outcome of imprecise definition. Rather, it stems from the fact that the notion of segments cannot be defined in the absence of an IR model – because unlike NL, there are no cognitive correlates of segments, like phrases or clauses, in queries. At best, an annotator can be asked to group multiword (named) entities together, which drastically reduces the scope of segmentation and makes it equivalent to the problem of (named) entity identification in queries.

Second, there is no clear consensus on the best use of segmentation in retrieval or ranking models, although there have been proposals such as the use of dependence models [18], language models [2] and double quotes [1, 3]. A commonly assumed restrictive principle in this context is that the words of the same segment must appear adjacent to each other in the document. This has resulted in the use of double quotes (as operators to ensure exact matches) to surround segments in several experimental frameworks [17, 1, 3]. However, putting quotes around all segments degrades performance [16]. While use of quotes for certain segments yields better results [3], detection of these segments at runtime is still a hard task. Finally, the use of exact segment matching leaves the following important question unanswered: how does one deal with the situation when the exact segment is only partially found in the document? A "segment found/not found" type of binary scoring would not be the best choice, as we have seen through our running example that some of the words may be entirely replaceable (`edition`) while others are not (`windows`). Current proposals of using flat segmentation for IR [18, 2, 1, 3] do not provide guidelines for handling such cases explicitly. Quoting-based strategies severely limit the scope of segmentation and effectively narrow it down to multiword entity detection.

## 2.2 Advantages of nested segmentation

The aforementioned problems are manifestations of the deeper issue of granularity at which segmentation needs to be done, i.e., whether to prefer longer or shorter segments, and whether this choice is context-sensitive. These problems vanish if we allow hierarchical or nested segmentation, where the human annotator or the algorithm is allowed to mark meaningful units in a hierarchical fashion and is required to go as deep as possible preserving the semantics of the query [22]. This will result in multi-level segmentation where at the lowest level, we will have multiword expressions for which quoting or exact matching would make sense during retrieval (e.g., `windows xp` and `hd video` in Fig. 1), whereas at higher levels it would make more sense to semantically interpret the unit and employ less strict matching where the terms are expected to be closer in a relevant document but not necessarily adjacent to each other (e.g., between `hd video` and `playback`).

Nesting is conceptually identical to hierarchical chunking [5] or phrase structure parsing of NL sentences. Chunking involves breaking NL sentences into syntactic units, and has significant benefits in parsing and understanding of NL text. Nesting has been widely used in NL to better represent internal structure within a bigger chunk. For example, a complex noun phrase `((a flight) (from Indianapolis) (to Houston))` can be parenthesized by marking smaller units. Thus, similarly nesting query segments can effectively resolve the problem of granularity. In the context of queries, a straightforward algorithm for nested segmentation would be to continue splitting a query or segments until certain boundary conditions are met. However, we show that this approach overlooks the rich local structures present in the query which can be used to customize nested segmentation.

Huang et al. [23] introduce a simple algorithm for hierarchical query segmentation as an application for Web scale language models. However, they do not suggest how nested segmentation can be used in IR. It is worth mentioning that term proximity models [24] and term dependence models [25], which are based on the fundamental assumption that certain query terms are expected to occur in close proximity in the relevant documents, are obliquely related to the concept of segmentation, because terms that are within a segment or appear closer in a segmentation tree are expected to appear closer in the relevant documents. We borrow some of these ideas to build our re-ranking framework. In our results, we also show the potential benefits of combining nested segmentation with the state-of-the-art term proximity model [26].

## 3 Terms and definitions

We now formally define the types of segmentation and the different distances which will be used to build up the algorithms and re-ranking models in the subsequent sections.

### 3.1 Types of segmentation

**Flat segmentation.** A flat segmentation of an $n$-word query $q = t_1 t_2 \dots t_n$ is defined as the original query $q$ augmented with a set of decisions $b_i$, $i = 1, 2, ..., (n-1)$, where $b_i$ is 1 if there is a segment boundary between query terms

$t_i$ and $t_{i+1}$, and 0 otherwise (boundaries are present before the first and after the last words). Each sequence of terms between two segment boundaries is called a *flat segment*. In general, a flat segment corresponds to a meaningful semantic unit within the query. We define the default flat segmentation for any query as the case when all the query words are in different segments (`windows | xp | home | edition | hd | video | playback`). The assumption in the default case is that "a query is a bag of words" without any linear dependence between successive terms. This is conceptually equivalent to an *unsegmented query*. In this text, *words* and *terms* refer to the same concept.

**Nested segmentation.** A nested segmentation for a query $q$ is defined as a recursive partitioning of $q$ such that each partition is either an indivisible (possibly multiword) unit or another nested segment. The partitions are marked using parentheses in this paper, and so a nested segmentation is represented as a complete parenthesization of the words in $q$. For example, `(((windows xp) home) edition) ((hd video) playback)` is a possible nested segmentation for the corresponding query. The default nested segmentation of a query is the case where each word is in its own segment and there is no further hierarchy. For example, `(windows) (xp) (home) (edition) (hd) (video) (playback)` is the default nested segmentation for our example query. The default assumption here is again that the query is a bag of words without any semantic relations or hierarchy between them, and is conceptually equivalent to an unsegmented query. By convention, parentheses are always present around single words, and at the ends of the query.

Note that this definition does not enforce a strict *binary partitioning* of the query; it is often possible that an atomic unit is composed of more than two words (`bed and breakfast`). Further, the query can also be made of multiple disparate concepts, like `(price comparison) (ps3) (nintendo) (xbox)`, of which more than two elements (`ps3`, `nintendo` and `xbox`) can conceptually be at the same level. This hierarchical partitioning can be obtained through a top-down or a bottom-up approach. In the *top down* approach, one divides the query into multiple parts, and then proceeds to divide each of these parts till atomic units are obtained. In the *bottom up* approach, one first groups the atomic units together, and then successively grows these units to form bigger segments till the entire query is one segment. In our hybrid method (Sec. 4), we start with an input flat segmentation and apply both top-down and bottom-up approaches to find smaller and larger units in the query respectively. The splitting and joining processed involved can be performed without any ordering constraints.

A *nested segmentation tree* is an alternative representation of nested segmentation, where query terms are leaf nodes and every multiword segment is represented by an internal node whose children include *all and only* nodes corresponding to words or other segments that constitute this segment. Fig. 1 graphically illustrates this concept. This tree representation not only provides an intuitive visualization of nested segmentation, but is also useful in defining the topological distance between a pair of words in the query, as described in Sec. 3.2.

## 3.2 Types of distances

**Tree distance.** The tree distance $td(t_1, t_2; n(q))$ between two terms $t_1$ and $t_2$ in $n(q)$, the nested segmentation of a query $q$, is defined as the shortest path (i.e., the number of hops) between $t_1$ and $t_2$ (or vice versa) through the nested segmentation tree for $q$. A tree ensures a unique shortest path between $t_1$ and $t_2$, which is through the common ancestor of $t_1$ and $t_2$. For example, $td($`xp`, `video`; $n(q)$ in Fig. 1$) = 7$. The minimum possible tree distance between two terms is two. We hypothesize that term pairs having low tree distance must appear close together in the document. Note that $td$ between $t_1$ and $t_2$ can vary for the same $q$, depending on $n(q)$.

**Query distance.** The query distance $qd(t_1, t_2; q)$ between two terms $t_1$ and $t_2$ in a query $q$ is defined as the difference between the positions of $t_1$ and $t_2$ in $q$, or equivalently, the number of intervening words plus one. For instance, in our original example, the distance between `xp` and `video` is 4. In special cases when the same word appears multiple times in a query (`johnson and johnson home page`), each term instance is treated as distinct during pairwise term distance comparisons.

**Document distance.** The distance between a pair of words in a document can be considered as the difference in the positions of the two words in the document, or equivalently, the number of intervening words plus one. Since a pair of words can occur multiple times in a given document, the notion of distance, so defined, is ambiguous. Consequently, various proximity heuristics have been proposed in the past to compute the effective distance between two words in a document [27, 24]. These include the minimum, maximum and mean of the distances between all paired occurrences of the two words in the document. Let $t_1$ and $t_2$ be two terms in the query $q$, which are also present (matched) in a retrieved document $\mathcal{D}$. Cummins and O'Riordan [24] have shown that amongst the various proximity heuristics, *minimum distance* has the highest inverse correlation with document relevance, i.e., the lower the minimum distance between $t_1$ and $t_2$ in $\mathcal{D}$, the higher the chances that $\mathcal{D}$ is relevant to $q$. However, past measures do not directly reward a document if it has multiple instances of $t_1$ and $t_2$ occurring within low distances of each other. Let there be $k$ instances of *ordered* pairwise occurrences of $t_1$ and $t_2$ (*ordered* pairs of positions of $t_1$ and $t_2$, ($p_1$, $p_2$) where $p_1 < p_2$) in $\mathcal{D}$ at minimum distances $dist_i = dist_1, dist_2, \ldots, dist_k$, such that the $dist_i$-s are in ascending order. We combine the ideas of minimum distance and multiple occurrences of a term pair to formulate the following definition of accumulative inverse document distance ($AIDD$) for $t_1$ and $t_2$ in $\mathcal{D}$:

$$AIDD(t_1, t_2; \mathcal{D})_{t_1 \neq t_2} = \frac{1}{dist_1} + \frac{1}{dist_2} + \ldots + \frac{1}{dist_k} \qquad (1)$$

By this method, a document with several $(t_1, t_2)$ pairs near to each other will have a high $AIDD$. Using the inverse of individual distances instead of the normal form has the nice property of not being sensitive to very high outlier distances. Taking the inverse makes the quantity very close to zero, mitigating its effect on the final sum. Since our concept is based on minimum distance,

**Table 1.** Recursive joining and splitting of flat segments as a general strategy for nested segmentation. All text except new segment markers are greyed out.

| Step | Structural representation of the query |
|---|---|
| Input flat seg | `windows xp home edition \| hd video \| playback` |
| Parenthesized | `(windows xp home edition) (hd video) (playback)` |
| Split | `((windows xp home) (edition)) (hd video) (playback)` |
| Split | `(((windows xp) (home)) (edition)) (hd video) (playback)` |
| Join | `(((windows xp) (home)) (edition)) ((hd video) (playback))` |
| Join and output | `((((windows xp) (home)) (edition)) ((hd video) (playback)))` |

we do not need a document length normalizer. A threshold on $k$ is nevertheless necessary to avoid considering *all* pairwise distances of $t_1$ and $t_2$, as distant pairs could be semantically unrelated. We study the impact of variation in $k$ in Sec. 7.1. To avoid scoring unrelated occurrences of a term pair, we consider matches only if $(t_1, t_2)$ occur within a given window size, *win*, i.e., we do not use $d_i$ when it exceeds some window *win*.

We compute the pairwise distances using position vectors ($pv$) of $a$ and $b$ in $\mathcal{D}$ [24]. For example, $pv(a) = \{1, 5, 10\}$ and $pv(b) = \{2, 3\}$ mean that $a$ has occurred in positions one, five and ten and $b$ in two and three (in $\mathcal{D}$), respectively. We currently ignore sentence boundaries while computing AIDD. Such a style of computation of pairwise distances can lead to re-counting of specific instances of $a$ and $b$. For example, the three minimum distance pairs in this case would be (1, 2), (1, 3) and (5, 3). Here, with patterns like `"... a a b b b c..."`), one could address the problem by choosing the optimum distance pair ($a$, $b$) using dynamic programming. This entails search in exponential time over the entire document, limited by the number of occurrences of the less frequent word. However, such an approach has been shown to be less effective than the simple case when re-counting is tolerated (see [24] for a more detailed discussion). Moreover, such patterns are quite rare in running text of documents.

For all the above distances, when the same word appears multiple times in a query, each word instance is treated as distinct during pairwise comparisons.

## 4 Algorithm

The goal behind devising a principled nested segmentation strategy is to discover deep syntactic relationships in a query, which are often present *within* a flat segment, and also *between* multiple flat segments. We do not propound simple top-down (begin with the query as a single unit and continue splitting till all units are single words) or bottom-up (begin with each word as a single unit and continue merging till the whole query becomes one unit) approaches for deducing the hierarchical structure in a query because such methods are naïve and do not involve any optimization step over all the words of the query. State-of-the-art flat segmentation algorithms [16, 2, 3] involve principled optimization criteria leading to discovery of flat segments, and a good nesting strategy should exploit this knowledge to the best capacity. There are three primary constraints or features of a query segmentation algorithm that need to be considered before designing an algorithm. First, the accuracy and robustness (i.e., reasonable performance on a wide variety of queries); second, the speed (segmentation is an online process and therefore to be practically useful it must have a very short turnaround time); and third, lack of annotated data. It might be worthwhile to elaborate a little on this last point. It may be argued that if we can get sufficient queries annotated by human experts for nested segmentation [22], the data could be used for supervised learning of nesting algorithms. Indeed, most NL parsing algorithms do rely on supervised learning on human-annotated treebanks. However, there is an important difference between these two cases. NL parsing is guided by an underlying (context-free or phrase structure) grammar which linguists have designed through years of systematic analysis of NLs. The annotators, who are themselves trained linguists, use the knowledge and framework of the grammar to annotate the tree structure for sentences. Likewise, the parsing algorithms search in the space of all possible parse trees that conform to this grammar. Queries do not follow grammatical rules, or at the least no such grammar has been formulated or deciphered till date. Neither do we have annotators who are experts or native speakers of the "query language". Therefore, structural annotation of queries [28–30] has always been subjective, often leading to low inter-annotator agreement. Moreover, creation of annotated data, for example, the treebanks for NLs, takes a tremendous amount of time and effort. It is also not straightforward to ascertain whether NL parsing algorithms can be efficiently adapted for fast online processing.

**Overview.** Most flat segmentation algorithms for queries are based on some kind of word $n$-gram statistics that are either learnt from documents or from query logs [6, 15, 16, 2, 7]. Since computation of $n$-gram statistics does not require annotated data, this turns out to be a scalable and robust approach that is also quite fast in practice [6]. Therefore, we devise nested segmentation strategies that are based on this simple yet powerful philosophy of $n$-grams. Since flat segmentation is a well-researched problem, we develop our algorithm for nested segmentation by starting with a flat segmentation of the query and trying to *split* within a flat segment and *join* adjacent flat segments recursively. Since flat segments are rarely longer than four to five words, nesting can be done rather fast with some clever manipulations of low order $n$-gram statistics ($n = 2, 3$). Thus, in our setup, given a flat segmentation for a query as input, a nesting strategy consists of the following two steps: (a) Split individual flat segments recursively till atomic units are obtained; (b) Join adjacent flat segments recursively till the whole query is one single unit. The split and the join steps are independent of each other and can be performed in any order. This process is illustrated in Table 1 and Fig. 2 with the help of our running example. In particular, we use the state-of-the-art unsupervised algorithm for obtaining the initial flat segmentation [3], with joining and splitting strategies with $n$-gram scores that are described below.

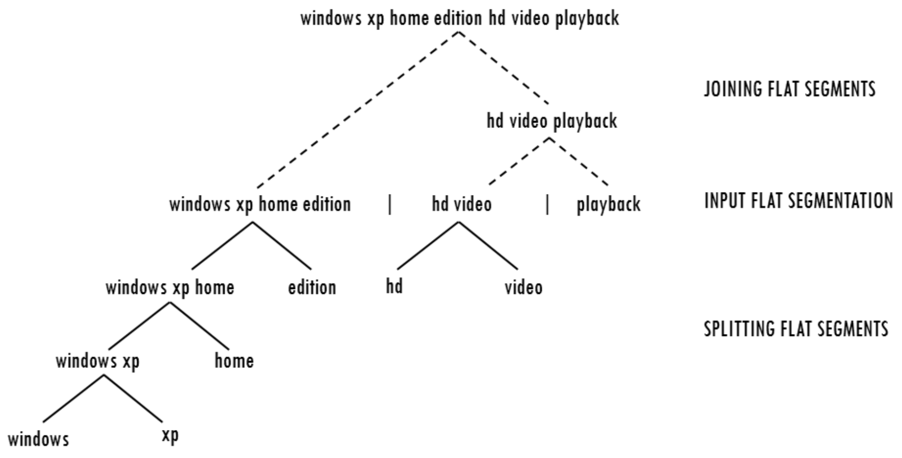


**Fig. 2.** Approach for nested query segmentation.

## 4.1 Splitting flat segments

Our main motivation for designing simple segment nesting strategies stems from the fact that most flat segmentation algorithms compute scores for $n$-grams as a key step of their respective methods (generally $n \leq 5$) [16, 7, 4]. In doing so, most often the scores of the contiguous lower order $n$-grams $(n-1, n-2, \ldots)$ are also known. We exploit these scores to deduce the structure within a flat segment. In this work, we specifically use the state-of-the-art CSR (Co-occurrence Significance Ratio) measure [31] to score $n$-grams. In principle, any word association measure can be used to score $n$-grams for this purpose (see [31] for an in-depth review of word association measures). Let $|n_i|$ denote the length of the $n$-gram $n_i$ in words. We note that for several word association measures, it is *not* necessary that $\text{score}(n_i) < \text{score}(n_j)$ if $|n_i| > |n_j|$, i.e., shorter $n$-grams, even though more frequent, do not necessarily achieve higher word association scores than longer $n$-grams.

Lower order $n$-grams are always more frequent than their extensions, and almost always obtain higher scores. Usually, bigrams get the highest scores as unigrams are assigned trivial values. However, this does not mean that flat segments cannot be longer than two words; this is because the final score assigned to a segmentation is a combination of the constituent $n$-gram scores. For example, the bigram `a b` is always more frequent than the trigram `a b c` and hence $\textit{score}(\texttt{a b}) > \textit{score}(\texttt{a b c})$. But it may well happen that in a five-word query `<a b c d e>`, $\textit{score}(\texttt{a b c}) \star \textit{score}(\texttt{d e}) > \textit{score}(\texttt{a b}) \star \textit{score}(\texttt{c d}) \star \textit{score}(\texttt{e})$, where $\star$ is any combination operator, and subsequently the former segmentation emerges as the one with the higher score and `a b c` is grouped as a segment. Techniques resort to suitable normalizations for longer segments to increase their base scores [16], or attempt to match (partially or fully) multiword entities from lists to directly prefer longer chunks [4, 3].

We adopt a simple greedy approach in this research. The $n$-gram that has the highest CSR score within a flat segment (where the number of words in the $n$-gram is less than the number of words in the corresponding flat segment) is immediately grouped together as a unit, i.e. a *sub-segment*. In this work, we restrict $n$ to a maximum of three, i.e. we search for highest scoring bigrams and trigrams exhaustively within a flat segment. We define a sub-segment as a smaller segment created by the division of a larger segment. Recursively, this newly grouped sub-segment's left and right $n$-grams (possibly null) and the sub-segment itself are processed in the same greedy fashion till every string to be processed cannot be divided further. Also, we allow scope for extending a sub-segment by one word from its left or right to incorporate elements. (if the resultant $(n+1)$-gram is the highest scoring candidate), provided it does not interfere with existing segment boundaries. For example, in the flat segment `windows xp home edition`, `windows xp home` has the highest CSR among the five possible $n$-grams (two trigrams and three bigrams). Thus, it is grouped together first. Since `edition` cannot be processed further, we repeat the search within `windows xp home` and group `windows xp` inside it. For the flat segment `the legend of zelda twilight princess`, we have `legend of zelda` grouped first (with `legend of` being grouped inside it in a subsequent step) followed by `twilight princess`. This results in the following embedded structure for the original flat segment: `(the) ((legend of) zelda) (twilight princess)`.

**Optimized Approach (O).** In this approach, every possible way of breaking a flat segment is considered, such that the constituent sub-segments are 1-, 2- or 3-grams only, and the partitioning that leads to the best combined score is selected[7]. These partitions are assumed to be the atomic units of the base flat segment. If a flat segmentation is purely based on an optimal combination of individual segment scores, then each segment, by itself, is an optimal way of combining its constituent words. In such a case, the optimized strategy of splitting would not have any effect on a flat segment. On the other hand, if a flat segment is deduced through matching against a list of named entities or a domain-specific multiword lexicon, getting smaller strings based on the scores is likely. Also note that it is quite possible that the greedy and optimized approaches produce the same final output.

## 4.2 Joining flat segments

Joining flat segments is essential to completing the nested segmentation tree, which in turn ensures a path between every pair of words in the query. At first sight, it seems that for making decisions about joining of two flat segments with $m$ and $n$ words respectively, one needs to have $(m + n)$-gram statistics. However, we found an elegant way to join segments using two simple local statistics explained next.

[7] Addition is the combination operator for the scores owing to the logarithmic space in which they are defined [16, 7].

**Bigram statistics of words at segment boundary.** The bigram at a flat segment boundary, i.e. the last word of a flat segment and the first word of the next flat segment, can be effectively used to take the segment joining decision. In our running example, if we wish to decide whether to join `windows xp home edition` and `hd video`, *or* `hd video` and `playback`, we check the relative order of the scores of the (ordered) bigrams formed by the underlined words only. The bigram with the higher score (in this case `video playback`) dictates which pair should be joined. This process is similarly repeated on the new parenthesized segments obtained until the whole query forms *one unit*. It is not difficult to think of cases where this greedy local and context insensitive approach will fail, and we do not claim that using bigrams only is sufficient in this process. Nevertheless, as we shall see, it works quite well in practice. In this research, we use the well-established concept of pointwise mutual information (PMI) [16] to score bigrams. Let $\mathcal{B} = <w_1\ w_2>$ be a bigram constituted of words $w_1$ and $w_2$. We define PMI($\mathcal{B}$) as follows:

$$\text{PMI}(\mathcal{B}) = \log_2 \tfrac{p(w_1 w_2)}{p(w_1)p(w_2)} \tag{2}$$

where $p(w_1 w_2)$, $p(w_1)$ and $p(w_2)$ refer to the probabilities of occurrence of $\mathcal{B}$, $w_1$ and $w_2$ in the *query log*, i.e. the number of queries each of them are present in, normalized by the total number of queries in the log. Again, for simplicity and for preserving query-specific structures, associated probabilities are computed from query logs only [7].

**Determiners, conjunctions and prepositions.** It often happens that the last (or the first) word in a segment is a determiner, conjunction or preposition (DCP)[8]. In these cases, it is almost always meaningful to combine such a segment with the next segment (or the previous segment) to make a meaningful *super-segment* (a larger segment created by the joining of two smaller segments). Examples are `(bed and) (breakfast)` and `(sound) (of music)`. In our algorithm, we prioritize such cases over the bigram scores during the joining process.

### 4.3 Measures of word association

We need to assign scores to $n$-grams in direct proportion to their internal binding or association strength. We score $n$-grams using the state-of-the-art word association measure CSR (Co-occurrence Significance Ratio) [31]. The measure was parallely adapted for query segmentation in Mishra et al. [7]. The novelty of this technique is that a decision is made on the significance of an $n$-gram only on the basis of the number of queries (in this context) which contain all the terms of the $n$-gram, thus disallowing frequently misleading unigram statistics to interfere with the decision. The CSR for an $n$-gram $\mathcal{N}$, in the context of queries, is defined as (Eq. 3)

$$CSR(\mathcal{N}) = \frac{2(N - E(X)^2)}{k} \tag{3}$$

[8] List used from `http://bit.ly/157MWzP`, Accessed 26 Oct 2015.

where $N$ and $k$ are the numbers of queries in which the words of $\mathcal{N}$ appear in the *correct* order (also, no intervening words) and *any* order (also, any number of intervening words allowed) respectively, and $E(X)$ is the expectation of random variable $X = \sum_i X_i$ which models the number of times the words of $\mathcal{N}$ appear in the correct order in a query, and is computed as $E(X) = \sum_i P_i$. The probability of $[X_i = 1]$ under the bag-of-words null model for queries, $P_i$, is given by (Eq. 4)

$$P_i = \frac{(l_i - n + 1)!}{l_i!} \tag{4}$$

We note that the CSR in this context is computed using only a query log as the input resource. This keeps our method lightweight. Since queries are generally shorter than NL sentences, only bigrams and trigrams are considered. $\mathcal{N}$ is considered to be a statistically significant multiword expression (MWE) if $CSR(\mathcal{N})$ is greater than a significance threshold $\delta$, where $\delta$ is chosen to be $0.6k$ [7]. Note that $\delta$ is specific to every MWE and there is no global threshold. Choosing $\delta$ in such a way allows us to be more selective with the lexicon of MWEs with respect to statistical significance than a global threshold. CSR for unigrams is defined to be zero, since their observed and expected frequencies are equal. CSR uses the *Hoeffding's inequality* to derive its test of word association significance [31, 32, 7].

## 5 Using nested segmentation in IR

Use of flat query segmentation in IR has been based upon the concept of proximity, which states that two words that are in the same segment should also occur within a short distance of each other in relevant documents; whereas words in different flat segments need not necessarily occur close to each other [6]. A stricter but more popularly assumed and experimented version of this hypothesis is that words within a flat segment should occur next to each other exactly in the same order in the relevant document as in the query [4]. This is typically implemented through the use of double quotes around segments, which most search engines interpret as an instruction for exact phrase match. As discussed earlier, this severely limits the scope of query segmentation and often results in misleading conclusions [3]. There is no obvious analogy between quoting of flat segments and that of nested segments, because it is unclear as to which level of nesting the quotes should be applied. More importantly, quoting is against the basic philosophy of nested segmentation because then we are not harnessing the true benefits of the hierarchical representation of the query terms. Nevertheless, see Sec. 8 on how the concept of quoting can be generalized in the context of nested query segmentation.

The flexibility of nested segmentation in comparison to the flat version can be best utilized through more informed matching of query words in the document. The important intuition that we try to leverage here is the fact that while research on term proximity has made successive improvements on *how* to best

**Table 2.** Penalty cases for query word pairs.

| **Tree distance** | **Document distance** | **Penalty** |
|---|---|---|
| Low | Low | Low |
| Low | High | High |
| High | Low | X |
| High | High | X |

X marks represent *don't care* conditions.

match query expressions in the document [24, 26], *which* query expressions matter more for proximity matching is still not very well-answered. We try to close this gap by using the nested segmentation tree. Here, we directly use the tree information to improve document-based term proximity. As an additional step, we use expressions generated by nested segmentation as input to a state-of-the-art term proximity model to reap potential benefits. We now describe these techniques of using nested segmentation for improving IR. Note that our IR setup can be used an evaluation framework for nested query segmentation approaches in the future when there will be multiple competing algorithms at hand.

### 5.1 Re-ranking strategy using tree

Here we define a score *Re-rank Status Value*[9] ($RrSV$) of every document $\mathcal{D}$ that was retrieved and ranked by in response to an unsegmented query $q$. The $RrSV$ for each such document is determined based on the following principle – *A pair of words that have a low tree distance in the nested representation of the query should not have a high document distance.* In other words, while re-ranking a document, the document distance (Eq. 1) between a pair of words should be penalized by a factor *inversely* proportional to their tree distance. We recall that tree distance between two words $a$ and $b$ in a query $q$, $td(a, b; n(q))$ is the path between $a$ and $b$ in the nested segmentation $(n(q))$ tree of $q$, and the document distance between $a$ and $b$ in a document $\mathcal{D}$, $AIDD(a, b; \mathcal{D})$, is defined by Eq. 1. The $RrSV$ for a document $\mathcal{D}$ is thus defined as

$$RrSV_{\mathcal{D}} = \sum_{\substack{t_i, t_j \in q \cap \mathcal{D} \\ t_i \neq t_j \\ td(t_i, t_j; n(q)) < \delta}} \frac{AIDD(t_i, t_j; \mathcal{D})}{td(t_i, t_j; n(q))} \tag{5}$$

where $t_i$-s are query terms matched in the document and $n(q)$ is the nested segmentation for $q$. However, we do not wish to penalize the case when the words are close by in the document and are relatively far in the tree. This is because it is always preferable to have all query words close by in the document [24]. Rather, we want to penalize a document only when specific word pairs (those that have a low tree distance) have high document distance. These situations

[9] The nomenclature is inspired by the Retrieval Status Value ($RSV$) of a document with respect to a query, which is a term that is popular in IR literature [33].

and the corresponding desired penalties are presented in Table 2. This analysis drives us to create a tree distance threshold (cut-off) parameter $\delta$. In other words, if $td(a, b; n(q)) < \delta$, only then is the word pair $a$ and $b$ considered in the computation of $RrSV$. We shall examine the effect of varying $\delta$ in Sec. 7.1.

The set of documents retrieved by a search engine by issuing the unsegmented query will be re-ranked in *descending* order of this $RrSV$. Let the ranks assigned to the document $\mathcal{D}$ by the original ranker and our re-ranking strategy be $R_{orig}(\mathcal{D})$ and $R_{new}(\mathcal{D})$ respectively. Then, according to our strategy, for two documents $\mathcal{D}_1$ and $\mathcal{D}_2$, if $RrSV_{\mathcal{D}_1} > RrSV_{\mathcal{D}_2}$, then $R_{new}(\mathcal{D}_1) < R_{new}(\mathcal{D}_2)$, i.e. $\mathcal{D}_1$ will be ranked higher up in the new ranked list than $\mathcal{D}_2$. The intuition here is that if a document $\mathcal{D}_1$ accumulates a higher value of $RrSV$ than document $\mathcal{D}_2$, then $\mathcal{D}_1$ has a relatively higher number of occurrences of query terms having a low tree distance close together inside its text than $\mathcal{D}_2$.

**Aggregation of original and new ranks** Let each document $\mathcal{D}$ in the initial result list that we wish to re-rank, have an original rank $R_o(\mathcal{D})$. The set of documents that we re-rank are originally retrieved from a collection in response to an unsegmented query using well-established IR ranking principles based on term frequencies and inverse document frequencies, and we wish to give due weight to the old ranks. The new rank obtained by the documents when sorted in descending order of $RrSV$ *only* is $R_n(\mathcal{D})$. We aggregate or fuse these ranks in the following manner to obtain an aggregated score $S_{rank\text{–}agg}$ for every document $\mathcal{D}$ [34] as shown below:

$$S_{rank\text{–}agg}(\mathcal{D}, R_{orig}, R_{new}, w) = \left(w \times \frac{1}{R_{new}(\mathcal{D}) + 1}\right) + \frac{1}{R_{orig}(\mathcal{D}) + 1} \tag{6}$$

where the weight $w$ (assigned to the new rank) is a heuristically tuned scaling factor representing the relative "importance" of the new ranking. The documents are finally ranked in *descending* order of $S_{rank\text{–}agg}$ to produce the final aggregated rank $R_{final}$. Formally, if $S_{rank\text{–}agg}(\mathcal{D}_1) > S_{rank\text{–}agg}(\mathcal{D}_2)$, then $R_{final}(\mathcal{D}_1) < R_{final}(\mathcal{D}_2)$, i.e. $\mathcal{D}_1$ will be ranked higher up in the final aggregated ranked list than $\mathcal{D}_2$ (ties in the final score are broken arbitrarily). Setting $w$ to zero or a very large value nullifies the effects of the new and original ranking respectively. The effect of varying $w$ is examined in Sec. 7.1.

There are several other approaches to rank aggregation and one of several proposed approaches could produce the best results in a given setup. However, that is not the focus of this research and we adopt one of the relatively recent, simple and popular techniques in this work that allows us to tune the effects of the original and new rankings.

**Re-ranking baselines** We now introduce three baselines for comparing the performance of our re-ranking strategy for nested segmentation. Flat segmentation is the first of these baselines, where we extend our notion of using pairwise term proximity to words within flat segments. The other two baselines are natural variants of the re-ranking equation (Eq. 5) that require investigation – one where only document distances are considered, and the other where the tree distance is replaced by the simple query distance.

**Flat segmentation.** This re-ranking technique is based on the notion that words within a flat segment are expected to appear near each other in the relevant documents [4]. Let $q$ be a query that has $p$ flat segments – $S_1$ to $S_p$. The $RrSV$ computation in this case is restricted only to intra-segment term pairs, i.e.,

$$RrSV_{\mathcal{D}} = \sum\nolimits_{k=1}^{p} \sum_{t_i, t_j \in S_k \cap \mathcal{D}, t_i \neq t_j} AIDD(t_i, t_j; \mathcal{D}) \tag{7}$$

We systematically experimented with three flat segmentation algorithms (Hagen et al. [16], Mishra et al. [7] and Saha Roy et al. [3]) where the algorithm by Saha Roy et al. [3] produced marginally better results. Hence, results involving flat segmentations, unless otherwise mentioned refer to the algorithm in Saha Roy et al. [3]. Results for nested segmentation, unless otherwise mentioned, use the corresponding flat segmentation outputs as the start states. Effect of choosing different flat segmentation algorithms is examined in Sec. 7.1.

**Document distances only.** This strategy is based on the principle that proximities between all pairs of query terms are equally important. The re-ranking score is thus simplified as:

$$RrSV_{\mathcal{D}} = \sum_{t_i, t_j \in q \cap \mathcal{D}, t_i \neq t_j} AIDD(t_i, t_j; \mathcal{D}) \tag{8}$$

**Document and query distances.** This method assumes that only terms close by in the query are required to be near each other in the document, and thus takes into account the query distance $qd$. Hence, Eq. 5 is suitably modified as shown below:

$$RrSV_{\mathcal{D}} = \sum_{t_i, t_j \in q \cap \mathcal{D}, t_i \neq t_j} \frac{AIDD(t_i, t_j; \mathcal{D})}{qd(t_i, t_j; q)} \tag{9}$$

### 5.2 Nesting with a term proximity model

Another intuitive way of using nested segmentation for IR would be to integrate the richer query representation with a term proximity models. Very recently, Vuurens and de Vries [26] presented an approach that is more robust than existing models and on an average performed equal or better than existing proximity and dependence models across collections. They analyzed how co-occurring query words can be used to assess document relevance based on their distance in the text, which is used to extend a unigram ranking function with a proximity model that accumulates the scores of all occurring term combinations.

Their method CPE (Cumulative Proximity Expansions) extends the KLD [35] scoring function for query terms with a proximity model that scores every possible combination of two or more query terms, independently. An expansion is any subset of the query terms. For instance, `windows xp`, `windows home`, `xp edition playback`, and so on are expansions generated from our running example query. The computations are done as shown below:

$$PROX(m, \mathcal{D}) = \sum_{q_i \in m} \log(1 + \frac{tf_{m,\mathcal{D}}}{\mu.P(q_i|C)}) \tag{10}$$

$$tf_{m,\mathcal{D}} = \sum_{o \in \#uw(m,\mathcal{D})} \frac{|m|-1}{|o|-1} \tag{11}$$

$$CPE(q, \mathcal{D}) = KLD(q, \mathcal{D}) + \frac{1}{|q|} \sum_{m \in \mathcal{P}_{>1}(q)} PROX(m, \mathcal{D}) \tag{12}$$

where $m$ refers to an expansion of the query terms, $\mathcal{D}$ is the document where term proximities are being analyzed, $q_i$ is a query term in $m$, $\mu$ is the Dirichlet prior parameter, $C$ refers to the document collection, $o$ is an unordered occurrence span of $m$ in $\mathcal{D}$, $\mathcal{P}_{>1}$ refers to the set of all expansions of $q$ which are at least two words long; $|m|$, $|o|$, $|q|$ are the lengths of the expansion, span and query respectively. For overlapping occurrence candidate spans, the shortest or leftmost span is selected.

The default proximity model (Eq. 12) uses the power set of all possible expansions, where stopwords have been removed from the input query string. However, Vuurens and de Vries show that a variation CPES (S = Stopword) outperforms the default CPE model. In CPES, an expansion is valid if it has a stopword with non-stopwords both to its left and right, or if it has a stopword that is at a query boundary (beginning or end).

The nested representation of the query can be easily interpreted to generate a set of such expansions. Specifically, we consider each parenthesized unit in the nested segmentation to be an expansion. Additionally, the rule of conditionally considering stopwords as in CPES can be applied. For CPENS, we do not consider an expansion if it has a stopword without non-stopwords on both sides, while excluding query start or end stopwords. We hypothesize that this strategy would have a comparable IR performance, while bringing about a large decrease in the number of expansions, reducing computational overhead. We refer to such schemes as CPEN (N = nest) and CPENS (CPES stopword heuristic applied). We also explore strategies CPEF and CPEFS (F = flat) for flat segmentation, which are analogous to CPEN and CPENS. For CPEF, the expansions are limited to flat segments. Table 3 shows expansions by the different methods for the query `(((our lady) of) lourdes) ((seven hills) church)` and `our lady | of lourdes | seven hills | church`, where `our` and `of` are stopwords.

## 6 Datasets

We will now describe the datasets that we have used. We divide this section into two parts: (a) data needed for performing nested segmentation of queries, and (b) data needed to apply and evaluate our strategies with respect to IR.

**Table 3.** Expansions by different methods.

| Method | Expansions |
|---|---|
| CPE [26] | `lady lourdes, seven hills, ..., lady lourdes seven, lourdes seven hills, ..., lady lourdes seven hills, lady lourdes hills church, ..., lady lourdes seven hills church`; **Total: 26** |
| CPES [26] | `our lady, lady lourdes, ..., our lady church, lady of lourdes, ..., our lady lourdes hills, lady of lourdes hills, ..., lady of lourdes seven hills, lady of lourdes seven church, ..., our lady of lourdes seven church, our lady of lourdes hills church, ..., our lady of lourdes seven hills church`; **Total: 58** |
| CPEF | `our lady, of lourdes, seven hills, church`; **Total: 4** |
| CPEN | `our lady, our lady of, our lady of lourdes, seven hills, seven hills church`; **Total: 5** |
| CPEFS | `our lady, seven hills, church`; **Total: 3** |
| CPENS | `our lady, our lady of lourdes, seven hills, seven hills church, our lady of lourdes seven hills church` |

## 6.1 For performing nested segmentation

As discussed, our nested segmentation algorithm requires a query log as the only resource, for computing various $n$-gram scores. For our experiments, we use a query log sampled from Bing Australia[10] in May 2010. This raw data slice consists of 16.7$M$ ($M$ = Million) queries (4.7$M$ unique). We subsequently extract 11.9$M$ queries from the raw data such that the queries are composed of ASCII characters only and are of length between two and ten words. The justification for imposing a filter based on query length is as follows. Segmentation of one word queries is not meaningful. Even though nested segmentation is useful only for queries of length three and above, two word queries are often useful for learning scores for significance statistics about bigrams. On the other hand, very long queries (having more than ten words) are typically computer generated messages or excerpts from NL text, and need separate query processing techniques. There are 4.7$M$ unique queries among the extracted 11.9$M$ queries – but in order to preserve log properties arising out of the natural power law frequency distribution of queries, we retain duplicates for all experiments. We use the Porter Stemmer to stem the queries before the computation of the $n$-gram scores. Scoring of $n$-grams was performed using the state-of-the-art word association measure CSR (Co-occurrence Significance Ratio) [31] measure (Sec. 4.3).

**Table 4.** Details of datasets used.

| **Dataset** | **Number of** | **Avg. words** | **Avg. RJs** | **Search** |
|---|---|---|---|---|
| **Name** | **queries** | **per query** | **per query*** | **engine** |
| SGCL12 | 500 | 5.29 | 28.34 | Lucene |
| TREC-WT | 75 | 3.43 | 34.39 | Indri |

* for top 100 results

### 6.2 For re-ranking documents

In order to ensure the replicability of our results, we report our IR evaluation on publicly available datasets only (Table 4) and use open source retrieval systems. Relevant supplementary material and code for this paper have already been shared publicly at `http://cse.iitkgp.ac.in/resgrp/cnerg/qa/nestedsegmentation.html`, Accessed 24 October 2015.

**SGCL12.** Saha Roy et al. [3] released a dataset of 500 queries[11] with associated URLs and relevance judgments (RJs) (approx. 30/query, $0-2$ scale, average rating of three annotators). The corpus contains around $14,000$ Web documents. We shall refer to this dataset as SGCL12 (author last name initials and year). SGCL12 was built for evaluating various flat segmentation algorithms and consists of slightly longer queries (five to eight words) where segmentation is meaningful from an IR perspective. The authors also showed that flat segmentation can potentially lead to substantial nDCG improvement on SGCL12. Hence this dataset is very appropriate for evaluating nested segmentation, and to show improvements over flat segmentation. Note that the queries in the SGCL12 dataset also have flat segmentation annotations from various algorithms and human experts. We use the commercially popular open source Apache Lucene $v3.4.0$[12] (basic TF-IDF retrieval model, used in default configuration; $v3.4.0$ chosen for comparability of certain results with Saha Roy et al. [3]) to search this collection. The first 250 queries were used as the development set for tuning model parameters ($k$, $win$, $\delta$ and $w$) and the last 250 queries were used as the test set.

**TREC-WT.** TREC topics, especially those belonging to the Web Track (WT) and the Million Query Track (MQT) (last held in 2009) are the ideal proxy for real Web search queries. However, the topics of WT are very short (average length of 2.32 words for 2012[13]) and therefore, not very appropriate for evaluation of nested segmentation[14]. The issue with the MQT (2009)[15] is the sparseness of RJs, which is more acute for slightly longer queries. We pulled out the 500 longest queries from the MQT (2009) having at most ten words. 491 of

[10] `https://www.bing.com/?cc=au`, Accessed 29 December 2015.
[11] `http://bit.ly/ZSOybI`, Accessed 27 Oct 2015
[12] `http://lucene.apache.org/core/`, Accessed 24 Oct 2015
[13] `http://trec.nist.gov/data/web2012.html`, Accessed 24 Oct 2015
[14] Nested segmentation can only benefit queries with at least three words.
[15] `http://trec.nist.gov/data/million.query09.html`, Accessed 24 Oct 2015

these queries had no associated RJ. Moreover, of all the queries that have length greater than or equal to five words, only 42 have at least one RJ. Nevertheless, in order to conduct nested segmentation experiments on the widely used TREC data, we accumulated queries from the 2009 to 2012 Web Track (the ideal proxy for Web search queries), and retained the queries that had three or more words (100 queries out of a total of 200)[16]. The highest number of words in this query set is five, even though it would have been better to have longer queries for truly appreciating the benefits of nested segmentation. Relevance judged documents for these queries are present in the ClueWeb09 Category B collection; we used the open source Indri[17] to search this collection through the provided API (used in default configuration) and retrieved the top 100 documents. The queries for which there are no relevant documents in the top 100 results were removed from the dataset. We will refer to the remaining set of 75 queries as the TREC-WT (available at `http://goo.gl/ffgYfi`, Accessed 24 Oct 2015). These queries, on an average, had around 34 RJs within the top 100 results (Table 4). RJs for all TREC-WT queries, downloadable from the respective track websites (*qrels*), have been appropriately collapsed to a 3-point scale (0, 1, 2). 35 queries were used as the development set for tuning model parameters and the remaining 40 queries were used as the test set, and the results are averaged over ten random 35-40 splits (see `http://bit.ly/13StKUN`, accessed 24 Oct 2015, for the 75 query TREC-WT set).

## 7 Experiments and results

In this section, we first report the specifics of our experimental setup and present the detailed results about our re-ranking strategy. In particular, we report the results of the following experiments: (a) effectiveness of nested segmentation over flat segmentation, (b) comparison with past work, (c) effect of query lengths, (d) effect of re-ranking strategies, (e) effect of parameter tuning, (f) effect of algorithmic variants, and (g) comparison with term proximity models.

**Experimental setup.** We used the outputs of three recent flat segmentation algorithms $[16, 2, 7, 3]$ as input to the nested segmentation algorithm and final nested segmentations for these queries were obtained. Documents are retrieved using the unsegmented queries, and subsequently re-ranked using the proposed technique (Sec. 5) and the baselines (Sec. 5.1). Results are compared in terms of *normalized Discounted Cumulative Gain* (nDCG) and *Mean Average Precision* (MAP). nDCG was computed for the top 5, 10 and 20 documents (the Ideal Discounted Cumulative Gain (IDCG), the denominator in nDCG, was computed using the optimal ranking from all judgments for the query). For computing MAP, URLs with ratings $> 0$ were considered as relevant. MAP values are computed on the top-30 documents for SGCL12 and the top-40 documents for TREC-WT (depending upon the approximate pool depth of 28 and 34 respectively (Table 4)). For each dataset, the four parameters (Table 11) were

[16] Nested segmentation only benefits queries $\geq 3$ words long.
[17] `http://www.lemurproject.org/indri/`, Accessed 24 Oct 2015

**Table 5.** Some algorithmic nested segmentations.

| **Seg** | **Query** |
|---|---|
| Flat | `garden city shopping centre \| brisbane \| qld` |
| Nested | `((garden city) (shopping centre)) (brisbane qld)` |
| Flat | `the chronicles of riddick \| dark athena` |
| Nested | `(the ((chronicles of) riddick)) (dark athena)` |
| Flat | `sega superstars tennis \| nintendo ds game` |
| Nested | `((sega superstars) tennis) ((nintendo ds) game)` |
| Flat | `samurai warriors 2 empires \| walk throughs` |
| Nested | `(((samurai warriors) 2) empires) (walk throughs)` |
| Flat | `as time goes by \| sheet music` |
| Nested | `(as (time goes) by) (sheet music)` |

optimized using the grid search technique for maximizing nDCG@10 on the development set and the best set of values were applied on the test set, which are reported in this section. Our proposed re-ranking method is found to be robust to parameter variation, as shown later in the text.

## 7.1 Improvements over flat segmentation

To provide readers with a qualitative feel of nested segmentation outputs on typical queries, we provide some representative nested segmentations generated by our algorithm for SGCL12 queries in Table 5. Tables 6 and 7 presents our *main findings* – the performance of nested segmentation in comparison with unsegmented queries and flat segmentation. Since the TREC-WT dataset was quite small compared to SGCL12, we report average values over ten runs with random train-test splits of 35 and 40 queries respectively, while preserving the query word length distribution. For each algorithm, *Flat* refers to the baseline re-ranking strategy (Sec. 5.1) when applied to the query (flat) segmented by the corresponding algorithm, and *Nested* refers to the proposed re-ranking strategy (Sec. 5.1) when applied to the nested segmentation of the query (Sec. 4) generated when the corresponding flat segmentation was used as the start state. We observe that nested segmentation, when using the proposed re-ranking scheme, significantly outperforms the state-of-the-art flat segmentation algorithms in all the cases. Importantly, improvements are observed for both the datasets on all the metrics. This indicates that one should not consider proximity measures for *only* the pairs of terms that are within a flat segment. Thus, our experiments provide evidence against the hypothesis that a query is similar to a bag-of-segments. We also note that both the flat and nested segmentations perform better than the unsegmented query, highlighting the general importance of query segmentation. *Henceforth in this paper, because of its superior performance over the other flat segmentation methods, we will assume the input flat segmentation for our*

**Table 6.** Performance comparison of flat and nested segmentations on SGCL12 and TREC-WT datasets (Part 1).

| **Dataset** | **Algo** | **Hagen et al. [16]** | | **Li et al. [2]** | |
|---|---|---|---|---|---|
| **SGCL12** | **Unseg** | **Flat** | **Nested** | **Flat** | **Nested** |
| nDCG@5 | 0.6839 | 0.6815 | **0.6982** | 0.6913 | **0.6989** |
| nDCG@10 | 0.6997 | 0.7081 | **0.7262**$^\dagger$ | 0.7144 | **0.7258**$^\dagger$ |
| nDCG@20 | 0.7226 | 0.7327 | **0.7433**$^\dagger$ | 0.7366 | **0.7437**$^\dagger$ |
| MAP | 0.8337 | 0.8406 | **0.8468**$^\dagger$ | 0.8404 | **0.8469**$^\dagger$ |
| **TREC-WT** | **Unseg** | **Flat** | **Nested** | **Flat** | **Nested** |
| nDCG@5 | 0.1426 | 0.1607 | **0.1750**$^\dagger$ | N. A.* | N. A. |
| nDCG@10 | 0.1376 | 0.1710 | **0.1880**$^\dagger$ | N. A. | N. A. |
| nDCG@20 | 0.1534 | 0.1853 | **0.1994**$^\dagger$ | N. A. | N. A. |
| MAP | 0.2832 | 0.2877 | **0.3298**$^\dagger$ | N. A. | N. A. |

The higher value among flat and nested segmentations is marked in **bold**. Statistical significance of nested segmentation (under the one-tailed paired $t$-test, $p < 0.05$) over flat segmentation *and* the unsegmented query is marked using $^\dagger$.
* We are unable to report the performance of Li et al. [2] on TREC-WT due to unavailability of outputs and code, and associated difficulties in reimplementation due to use of proprietary data.

*nested segmentation algorithm as the output by Saha Roy et al. [3].* The algorithm in Saha Roy et al. will be the assumed method for the flat segmentation results, unless otherwise mentioned.

## 7.2 Comparison with past work

For comparing our algorithm with past work, we reimplement the nested segmentation strategy of Huang et al. [23], which is based on SPMI (Segment Pointwise Mutual Information). This is shown in the last column in Table 7. A query (and its segments thereafter) is iteratively split into two halves based on an SPMI threshold until the minimum SPMI reaches a suitably tuned termination threshold. We emphasize again that Huang et al. do not provide any methodology for using nesting for IR. It is observed that their method is outperformed by most nesting strategies on all the metrics. We observed that while the average tree height is 2.96 for our nesting strategy, the same is about 2.23 for Huang et al. (SGCL12). Note that due to the strict binary partitioning at each step for Huang et al., one would normally expect a greater average tree height for this method. Thus, it is the inability of Huang et al. to produce a suitably deep tree for most queries (inability to discover fine-grained concepts) that is responsible for its somewhat lower performance on the metrics. Most importantly, all nesting strategies faring favorably (none of the differences for Huang et al. with other nesting methods are statistically significant) with respect to flat segmentation bodes well for the usefulness of nested segmentation for IR in general.

**Table 7.** Performance comparison of flat and nested segmentations on SGCL12 and TREC-WT datasets (Part 2).

| **Dataset** | **Algo** | **Mishra et al. [7]** | | **Saha Roy et al. [3]** | | **Huang et al. [23]** |
|---|---|---|---|---|---|---|
| **SGCL12** | **Unseg** | **Flat** | **Nested** | **Flat** | **Nested** | **Nested** |
| nDCG@5 | 0.6839 | **0.6977** | 0.6976 | 0.6746 | **0.7000**† | 0.6996 |
| nDCG@10 | 0.6997 | 0.7189 | **0.7274** | 0.7044 | **0.7268**† | 0.7224 |
| nDCG@20 | 0.7226 | 0.7389 | **0.7435** | 0.7321 | **0.7433**† | 0.7438 |
| MAP | 0.8337 | 0.8411 | **0.8481**† | 0.8423 | **0.8477** | 0.8456 |
| **TREC-WT** | **Unseg** | **Flat** | **Nested** | **Flat** | **Nested** | **Nested** |
| nDCG@5 | 0.1426 | 0.1604 | **0.1752**† | 0.1603 | **0.1767**† | 0.1746 |
| nDCG@10 | 0.1376 | 0.1726 | **0.1882**† | 0.1707 | **0.1884**† | 0.1845 |
| nDCG@20 | 0.1534 | 0.1865 | **0.2000**† | 0.1889 | **0.2010**† | 0.1961 |
| MAP | 0.2832 | 0.3003 | **0.3284**† | 0.3007 | **0.3296**† | 0.3263 |

The higher value among flat and nested segmentations is marked in **bold**. Statistical significance of nested segmentation (under the one-tailed paired $t$-test, $p < 0.05$) over flat segmentation *and* the unsegmented query is marked using †.

**Table 8.** Break-up of nDCG gains (over unsegmented query) by length (SGCL12).

| **Nested (flat) segmentation for SGCL12** | | | | |
|---|---|---|---|---|
| **Length** | **#Q** | **#Gain Q** | **#Gain Q%** | **A. G.** |
| 5 | 387 | 235(193) | 60.72(49.87) | +0.1103(+0.0887) |
| 6 | 91 | 58(42) | 63.74(46.15) | +0.1006(+0.0772) |
| 7 | 14 | 9(6) | 64.29(42.86) | +0.1401(+0.1166) |
| 8 | 8 | 4(3) | 50.00(37.50) | +0.0414(+0.1061) |

## 7.3 Effect of query length

To understand the potential of nested segmentation, it is important to see for how many queries in each length group it results in improved retrieval performance. In Tables 8 and 9, we report the number of queries of a particular length in our datasets (**#Q**), the number among these **Q** that show a positive gain in nDCG@10 (**#Gain Q**), the associated percentage of queries and the average nDCG@10 gain (A. G.) computed over all queries of a particular length that show performance improvement over the original unsegmented query. We observe that for almost all length groups, nested segmentation improves a strong majority of the queries. The mean improvement is slightly more for queries in the medium length zone (5- and 6-word queries). We found longer queries in our data (like `you spin my head right round right round` and `eternal sunshine of the spotless mind watch online`) contain song lyrics or long named entities that require exact document matches and hence nesting is often not required, and may be detrimental in certain cases. Corresponding figures for flat segmentation (figures in parentheses) are observed to be lower. It would have been ideal if we had a substantial number of queries for each query length in our datasets.

**Table 9.** Break-up of nDCG gains (over unsegmented query) by length (TREC-WT).

| **Nested (flat) segmentation for TREC-WT** | | | | |
|---|---|---|---|---|
| **Length** | **#Q** | **#Gain Q** | **#Gain Q%** | **A. G.** |
| 3 | 52 | 22(21) | 42.31(40.38) | +0.1695(+0.1868) |
| 4 | 14 | 9(9) | 64.29(64.29) | +0.2842(+0.2071) |
| 5 | 9 | 5(4) | 55.56(44.44) | +0.3156(+0.2987) |

**Table 10.** Performance of re-ranking strategies.

| **Dataset** | **SGCL12** | | | **TREC-WT** | | |
|---|---|---|---|---|---|---|
| **Metric** | **Doc** | **Query** | **Tree** | **Doc** | **Query** | **Tree** |
| nDCG@5 | **0.7006** | 0.6963 | 0.7000 | 0.1700 | 0.1665 | **0.1767**$^\dagger$ |
| nDCG@10 | 0.7193 | 0.7255 | **0.7268**$^\dagger$ | 0.1801 | 0.1798 | **0.1884**$^\dagger$ |
| nDCG@20 | 0.7404 | **0.7441** | 0.7433 | 0.1923 | 0.1886 | **0.2010**$^\dagger$ |
| MAP | 0.8398 | 0.8472 | **0.8477**$^\dagger$ | 0.3237 | 0.3189 | **0.3296**$^\dagger$ |

The highest value among the *Doc*, *Query* and *Tree* re-ranking strategies is marked in **boldface**. Statistical significance of the *Tree* strategy under the one-tailed paired $t$-test ($p < 0.05$) over the lowest value among the three is marked using $^\dagger$.

In the current Web search scenario, slightly longer queries are generally harder to solve, with keyword matches retrieving several spurious results. To be specific, the percentage of long queries ($\geq$ 5 words) in our Bing Australia query log is 26.65% (distinct queries only) – a significant number when the total search volume is considered. Thus, we can no longer undermine the impact nested segmentation can have in Web search. In total, while $\simeq$ 49% queries are benefited by flat segmentation for SGCL12 and $\simeq$ 45% for TREC-WT, the numbers rise to $\simeq$ 61% for SGCL12 and $\simeq$ 48% for TREC-WT in case of nested segmentation. Importantly, the mean improvements (over the unsegmented queries) in nDCG@10 for benefited queries are 0.1084 for SGCL12 and 0.2185 for TREC-WT in case of nested segmentation; corresponding values for flat segmentation are lower: 0.0876 (SGCL12) and 0.2053 (TREC-WT). We note that these numbers are similar for the algorithm of [23].

## 7.4 Comparison of re-ranking strategies

Table 10 compares re-ranking strategies, where *Doc* refers to the baseline re-ranking method that uses only document distances (Eq. 8), *Query* refers to the scheme using document and query distances (Eq. 9), *Tree* refers to the proposed re-ranking strategy using the nested segmentation tree (Eq. 5). We observe that scaling *AIDD* by the tree distance generally improves the results over the unscaled version. This shows the importance of the tree distance in bringing out the relationship between query terms. In other words, the nested segmentation tree provides a more principled and meaningful estimation of proximity between

**Table 11.** List of parameters used in re-ranking.

| Notation | Parameter |
|---|---|
| $k$ | No. of minimum distances considered |
| $win$ | Window size |
| $\delta$ | Tree distance cut-off |
| $w$ | New rank weight |

query terms, which can be systematically exploited during re-ranking of documents for significant performance gains. We observed that the number of queries on which *Doc*, *Query* and *Tree* perform the best are 102, 94, 107 (SGCL12, 250 test queries) and 30, 29.7, 30.8 (TREC-WT, 40 test queries, averaged over ten splits) respectively. The numbers do not add up to 250 (SGCL12) or 40 (TREC-WT) because multiple models may produce the best output for the same query. Thus, the *Tree* model helps greater numbers of queries for both datasets.

## 7.5 Parameter tuning

We now systematically study the effect of variation of the four tunable parameters on the re-ranking performance. Table 11 lists the tunable parameters. Variation patterns on the development set of SGCL12 and TREC-WT are reported in Figs. 3 and 4. *Doc* and *Query* refer to the baseline re-ranking strategies using only document, and document and query distances respectively (Sec. 5.1). *Tree* refers to the proposed re-ranking method based on the nested segmentation tree (Sec. 5.1). Wherever applicable, the *Tree* re-ranking model outperforms the *Doc* and *Query* models systematically. From plots (Fig. 3 *(a)* and *(b)*, we see that preferred values of $k$ and $win$ are five and four respectively for SGCL12 (and one and three for TREC-WT), and increasing them further brings semantically unrelated word pair occurrences into the $RrSV$ computations. Figs. 3 show the effect of varying $\delta$ – the tree distance cut-off value; very low $\delta$ essentially means ignoring the tree hierarchy, and thus leads to poor performance for SGCL12. For SGCL12, the result stabilizes for $\delta \geq 5$, and increasing $delta$ further almost has no effect on the results as there are very few word pairs that will have a tree distance greater than five or six for a typical query. Thus, having this parameter in the system setup is optional; but if one chooses to use $\delta$ for finer control on the results, one must be careful as to not set it to a very low value. However, we note that $\delta = 3$ is ideal for TREC-WT, and greater $\delta$ is applicable only for $\simeq 30\%$ of the queries, which are of length greater than three words. Finally, setting the new rank weight $w$ to two is found to be the best for SGCL12. Setting $w$ to zero logically translates to ignoring the new ranking, and would result in the performance of the original query, which is always poorer than when re-ranking is applied (*Unseg* in Tables 6 and 7). Using a large value for $w$ ($\simeq 1,000$) implies ignoring the old ranking. This is found to produce the best results for TREC-WT, emphasizing the importance of our re-ranker. Thus, one

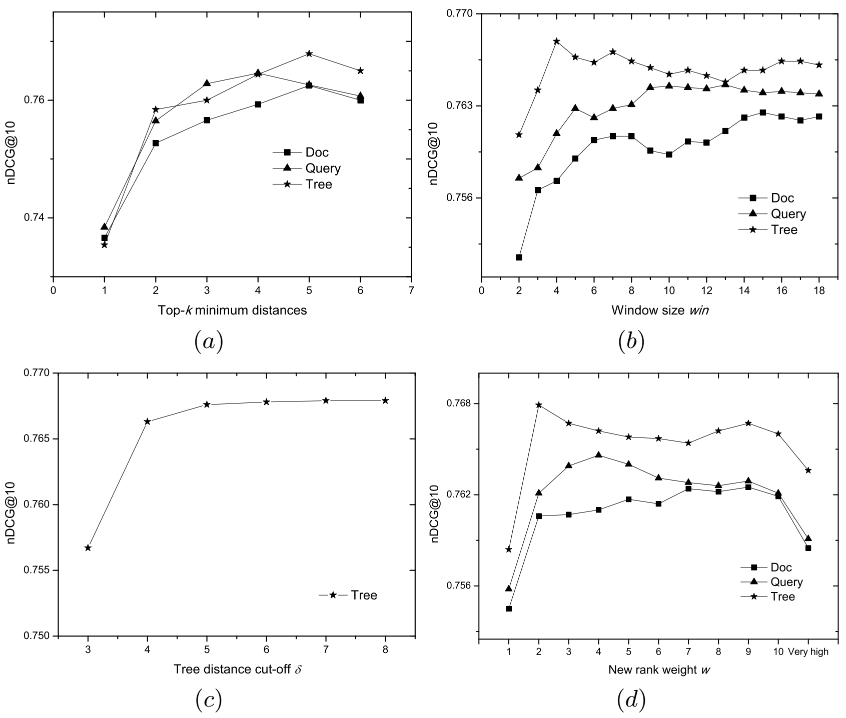


**Fig. 3.** Parameter tuning for SGCL12: (a) $k$ (b) $win$ (c) $\delta$ (d) $w$ (e) $k$ (f) $win$. For examining a particular parameter for a specific re-ranking strategy, others are fixed at the point of global maximum.

should decide on the weights to assign to the original ranker and that derived by the nested representation of the query after an empirical analysis on a relevant tuning set.

Our overall algorithm entails the systematic exploration of certain *variations*; for example, using other word association measures for splitting or joining, using an optimized splitting strategy instead of a greedy one, or whether the preference to DCP is required. We experimented with 16 such variations and found the version reported in this paper to the best among these; however, no statistically significant difference was observed among quite a few strategies. The results are omitted here due to the paucity of space.

## 7.6 Investigation of variations in algorithm

Our overall algorithm entails the systematic exploration of certain variations. First, instead of a greedy approach, one can opt for an optimized strategy to split a flat segment. In this approach, every possible way of breaking a flat segment is considered, such that the constituent sub-segments are 1-, 2- or 3-grams

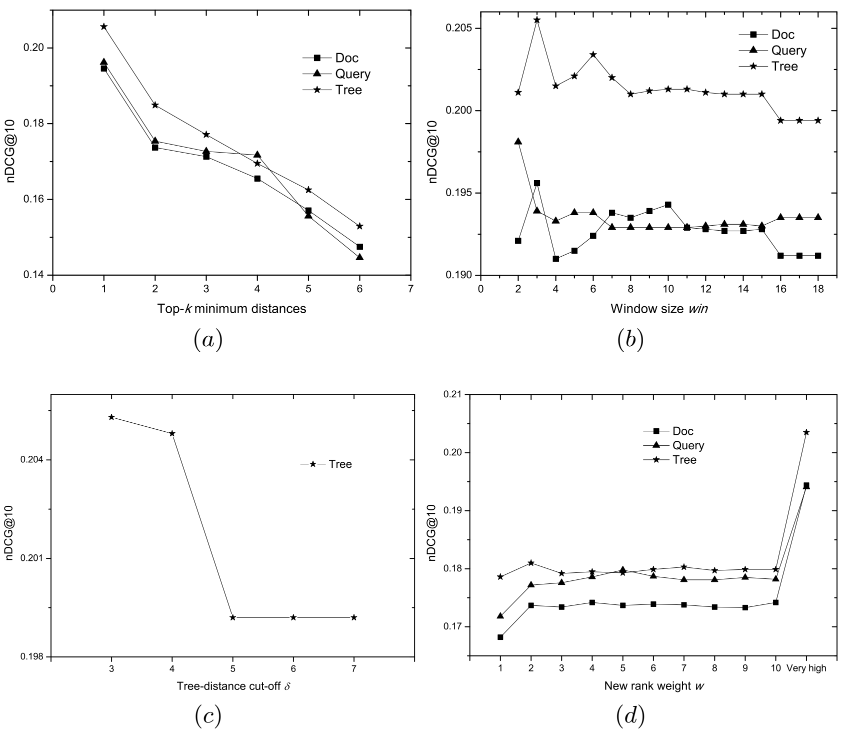


**Fig. 4.** Parameter tuning for TREC-WT: (a) $k$ (b) $win$ (c) $\delta$ (d) $w$. For examining a particular parameter for a specific re-ranking strategy, others are fixed at the point of global maximum.

only, and the partitioning that leads to the best combined score is selected[18]. These partitions are assumed to be the atomic units of the base flat segment. If a flat segmentation is purely based on an optimal combination of individual segment scores, then each segment, by itself, is an optimal way of combining its constituent words. In such a case, the optimized strategy of splitting would not have any effect on a flat segment. On the other hand, if a flat segment is deduced through matching against a list of named entities or a domain-specific multiword lexicon, getting smaller strings based on the scores is likely. Note that it is quite possible that the greedy and optimized approaches produce the same final output.

Second, one can use CSR for scoring bigrams for *joining* smaller segments instead of MI. This gives rise to two choices in the joining phase. Third, the definition of MI can be appropriately extended to score $n$-grams when $n > 2$ [6], and can thus be used during the *splitting* process instead of CSR. This gives to

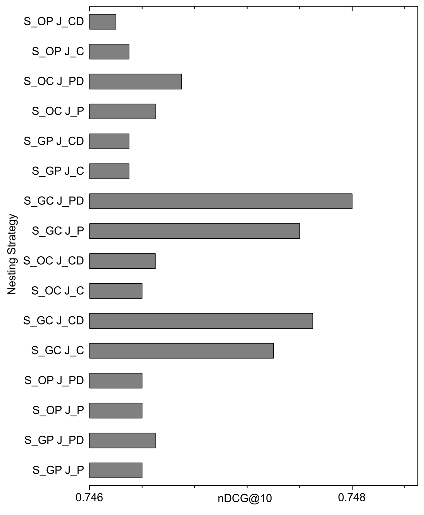
[18] Addition is the combination operator for the scores owing to the logarithmic space in which they are defined [16, 7].

**Table 12.** Nested segmentation strategies that have been developed.

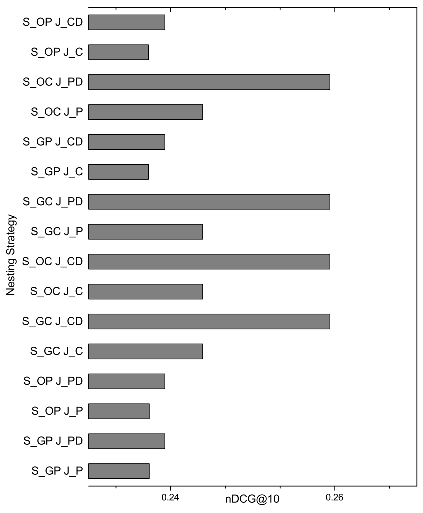
| **Strategy label** | **Strategy details** |
|---|---|
| $S_{GM}J_M$ | Greedy (G) splitting (S) with MI (M), Joining (J) with MI |
| $S_{GM}J_{MD}$ | Greedy splitting with MI, Joining with MI and preference to DCP |
| $S_{OM}J_M$ | Optimized (O) splitting with MI, Joining with MI |
| $S_{OM}J_{MD}$ | Optimized splitting with MI, Joining with MI and preference to DCP |
| $S_{GC}J_C$ | Greedy splitting with CSR (C), Joining with CSR |
| $S_{GC}J_{CD}$ | Greedy splitting with CSR, Joining with CSR and preference to DCP |
| $S_{OC}J_C$ | Optimized splitting with CSR, Joining with CSR |
| $S_{OC}J_{CD}$ | Optimized splitting with CSR, Joining with CSR and preference to DCP |
| $S_{GC}J_M$ | Greedy splitting with CSR, Joining with MI |
| $S_{GC}J_{MD}$ | Greedy splitting with CSR, Joining with MI and preference to DCP |
| $S_{GM}J_C$ | Greedy splitting with MI, Joining with CSR |
| $S_{GM}J_{CD}$ | Greedy splitting with MI, Joining with CSR and preference to DCP |
| $S_{OC}J_M$ | Optimized splitting with CSR, Joining with MI |
| $S_{OC}J_{MD}$ | Optimized splitting with CSR, Joining with MI and preference to DCP |
| $S_{OM}J_C$ | Optimized splitting with MI, Joining with CSR |
| $S_{OM}J_{CD}$ | Optimized splitting with MI, Joining with CSR and preference to DCP |

two choices during the splitting phase. Fourth, *joining* segments may be purely on the basis of bigram scores and DCP (preference to determiners, conjunctions and prepositions during joining, Sec. 4.2) need not be considered during the merging process. This leads to two more choices during the joining phase. We shall systematically represent and refer to these nested segmentation strategies as $S_{UV}J_{XY}$, where $U$ is $G$ or $O$ for greedy and optimized approaches for splitting flat segments respectively, $V$ and $X$ are $C$ or $P$ respectively for CSR and MI scores for splitting and joining respectively. $Y$ is $D$ if DCP is considered during joining, else null. Thus, $S_{GC}J_M$ refers to the case where greedy splitting is done using CSR scores, and joining is done using MI scores without considering DCP. If DCP is considered, the corresponding representation will be $S_{GC}J_{MD}$. In this manner, we have $2 \times 2 \times 2 \times 2 = 16$ combinations in all for nested segmentation (greedy or optimized splitting, splitting or joining with MI or CSR scores and optional preference to DCP) for each input flat segmentation. Choice of these different nesting strategies is examined here.

The best performance of these variants are computed by appropriately tuning model parameters on the development set, and reported in Fig. 5. We observe that the proposed nesting strategy $S_{GC}J_{PD}$ outperforms all the other nested segmentation strategies for both SGCL12 and TREC-WT (there are three other strategies with comparable performance for TREC-WT). In general, it is observed that during splitting, greedy approaches work better. This is due to the fact that the greedy approaches are almost always able to split a multiword segment further leading to deep nested structures that are more informative. On the other hand, while joining, giving preference to DCP turns out to be a better choice. Interestingly, MI scores are more useful for joining segments and CSR scores are better at splitting segments. This also falls in line with the assumptions underlying the usage of these scoring methods; the concept of MI is more



(*a*)



(*b*)

**Fig. 5.** Performance examination of algorithm variations on both datasets: (a) SGCL12 (b) TREC-WT.

**Table 13.** Examination of named entities.

| Method | SGCL12 | TREC-WT |
|---|---|---|
| No nesting in NE | **0.7617** | 0.1569 |
| Full Nesting | 0.7611 | 0.1569 |

Higher values in columns marked in **bold**.

meaningful for examining relative strengths of pairs of words only, and thus has been more frequently used for marking segment breaks (and hence non-breaks) by observing MI scores of adjacent word pairs [13, 16]. In contrast, CSR is aimed at grading how well a group of words gel together as an expression. However, we observe that differences between strategies are not statistically significant, which highlights the flexibility of the algorithm outline. Henceforth in this text, a nested output will refer to the $S_{GH}J_{PD}$ scheme on the flat segmentation produced by Saha Roy et al. [3] (if not mentioned otherwise).

**Effect of named entities.** It becomes evident from the superior performance of the tree distance that whenever possible, giving more weight to exact matches of query word sequences in documents leads to better performance. To strengthen this hypothesis, we examined the effect of *not applying* nesting to named entities (NE) where exact matches are almost always preferable. In this setup, all nodes of a named entity, like `lord of the rings`, are assumed to be on the same tree level. Hierarchical structure is deduced in the remain-

**Table 14.** Performance of SDM and FDM.

| Data | SGCL12 | | | TREC-WT | | |
|---|---|---|---|---|---|---|
| Model | Doc (DD) | Query (DQ) | Tree (DT) | Doc (DD) | Query (DQ) | Tree (DT) |
| SDM | **0.7449** | **0.7449** | 0.7451 | 0.1962 | **0.2026** | **0.2091** |
| FDM | 0.7410 | 0.7429 | **0.7467** | **0.2010** | 0.2010 | 0.2023 |

Higher values in columns (each dataset) marked in **bold**.

ing parts of the query (like a preceding `how to view` and succeeding `movie trailer online`) as usual. We manually marked the named entities in all the 575 queries of our datasets and the remainder of the query was segmented using the $S_{GH}J_{MD}$ strategy. Results show that indeed, treating named entities as a single unit with no deeper structure, does lead to slightly better nDCG@10 (**0.7617** from 0.7611 for SGCL12, no difference for TREC-WT since only two queries have a different nested segmentation when NEs are treated as a single unit). We note that a large majority (390 out of 500) of the queries in SGCL12 have at least one named entity of at least three words.

**SDM and FDM.** Metzler and Croft [25] propose that terms in a query can either be independent of each other (IDM, the independence model), adjacent words may depend on each other (SDM, the sequential dependence model) or all words may depend on each other (FDM, the full dependence model). The concept has been very popular since then and we evaluate each re-ranking strategy aligned to these ideas. For each strategy, the pairwise proximity measures are computed both for adjacent query term pairs only (SDM), and for all term pairs (FDM). The unsegmented query corresponds to an IDM. Table 14 shows the comparison between SDM and FDM.

A very important observation is that neither model gives the best results in all cases. This means that assuming dependence between *all* pairs of query terms is not always meaningful, and spurious linkages should not be considered during the ranking process. On the other hand, it need not imply that adjacent term pairs are only those that matter. We believe that the ideal dependence model for queries is somewhere in between SDM and FDM, and term pairs need to be rewarded or penalized on something more than just query distance. We note that the SDM for **Query** is the same as the SDM for **Doc** since the $qd$ is 1 for SDM. The SDM for the **Hyb** is the same as the SDM for **Tree** because, in essence, the former is given by $(|DD - qd| + 1)/td$; when $qd = 1$ (SDM), this reduces to $|DD|/td$ (**Tree** strategy), where $DD$ is the document distance.

**The SPMI algorithm.** For comparing our algorithm with past work, we reimplement the nested segmentation strategy of Huang et al. [23] (Sec. 2.2), which is based on SPMI (Segment Pointwise Mutual Information). A query (and its segments thereafter) is iteratively split into two halves based on an SPMI threshold until the minimum SPMI reaches a termination threshold. We emphasize again that Huang et al. do not provide any methodology for using nesting for IR. While evaluating their algorithm, Huang et al. observed that the anchor

**Table 15.** Comparison with Huang et al. [23].

| Dataset | SGCL12 | | | TREC-WT | | |
|---|---|---|---|---|---|---|
| Metric | Proposed Algo | Huang et al. (Anno) | Huang et al. (IR) | Proposed Algo | Huang et al. (Anno) | Huang et al. (IR) |
| nDCG@10 | **0.7284** | 0.7224 | 0.7240 | 0.1884 | 0.1845 | **0.1918** |
| MAP | **0.8481** | 0.8456 | 0.8461 | 0.3296 | 0.3263 | **0.3368** |

The highest values in rows (for each dataset) are marked in **bold**.

text language model, obtained using the Microsoft Web $n$-gram Services[19], performed better than title, body and query models. Hence, we choose the anchor text language in our experiments as well, for fairness and comparability. The tunable parameter in their algorithm is the SPMI termination threshold, say $\alpha$, which is required for stopping further nesting. As suggested in their paper, $\alpha$ needs to be tuned by optimizing one of the three matching metrics (*Exact match, Cover, Violation*) against manual annotations. Since the authors do not specify the best of these metrics, we choose to maximize *Exact match.* For this purpose, we ask three human annotators $A$, $B$ and $C$ to discover and annotate important phrasal segments from the queries of SGCL12 and TREC-WT [23]. The annotators were Computer Science undergraduate and graduate students between $22 - 28$ years of age, each issuing around $20 - 30$ Web queries per day. Using this policy, we observed a slightly poorer performance of their algorithm with respect to our proposed strategy. Subsequently, for fairness, we also tuned $\alpha$ to maximize the nDCG@10 value on the development set. Results are presented in Table 15. Values obtained for the three annotators were quite close to each other, and hence only their average is reported. Tuning $\alpha$ using manual annotations and nDCG@10 is indicated by *Anno* and *IR* respectively.

Our algorithm is slightly superior to Huang et al. on both nDCG@10 and MAP on SGCL12, while being slightly inferior on nDCG@5 and nDCG@10 (SGCL12). We recollect that the SPMI threshold for Huang et al. was chosen so as to maximize nDCG@10, and hence the lower IR performance is not due to the choice of an unsuitable threshold. We observed that while the average tree height is 2.96 for our method, the same is about 2.23 for Huang et al. (SGCL12). Note that due to the strict binary partitioning at each step for Huang et al., one would normally expect a greater average tree height for this method. Thus, it is the inability of Huang et al. to produce a suitably deep tree for most queries (inability to discover fine-grained concepts) that is responsible for its somewhat lower performance on the metrics. Huang et al., however, perform better on TREC-WT on both the metrics. More importantly, both nesting strategies faring favorably (none of the differences are statistically significant) bodes well for the usefulness of nested segmentation for IR in general. The tree height distributions for the two algorithms are given in Table 16 (IR optimization for $\alpha$ in Huang et al.).

**Table 16.** Height distributions for nested segmentation tree. Values denote the numbers of queries for each algorithm that attain a tree height equal to the column headers.

| Dataset | Algorithm | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| SGCL12 | Proposed | 0 | 99 | 327 | 71 | 3 |
| | Huang et al. | 59 | 292 | 124 | 23 | 2 |
| TREC-WT | Proposed | 15 | 46 | 13 | 1 | 0 |
| | Huang et al. | 37 | 30 | 7 | 1 | 0 |

**Table 17.** Nesting expansions in proximity model for SGCL12.

| Algorithm | nDCG@5 | nDCG@10 | nDCG@20 | MAP | $\mu$ (nDCG) | $\mu$ (MAP) |
|---|---|---|---|---|---|---|
| KLD | 0.7268 | 0.7412 | 0.7553 | 0.7462 | 19 | 19 |
| CPE | 0.7563 | 0.7719 | 0.7808 | 0.7583 | 19 | 19 |
| CPES | **0.7627** | **0.7784** | **0.7895** | **0.7617** | **19** | **19** |
| CPEF | 0.7107 | 0.7324 | 0.7525 | 0.7512 | 220 | 1100 |
| CPEN | 0.7124 | 0.7326 | 0.7524 | 0.7530 | 150 | 1100 |
| CPEFS | 0.7297 | 0.7489 | 0.7665 | 0.7501 | 140 | 1000 |
| CPENS | 0.7368 | 0.7568 | 0.7504 | 0.7533 | 140 | 1000 |

The highest value in a metric column is marked in **boldface**.

## 7.7 Proximity model with nesting

We reimplemented the method proposed in Vuurens and de Vries [26], and evaluated the CPE, CPES and CPEN approaches on our datasets (Sec. 5.2). The parameter $\mu$ (Eq. 12) was tuned separately for each method and each metric separately on the development set. As shown in the results in Tables 17 and 18, we found CPEN, CPENS, CPEF and CPEFS to be performing somewhat poorer than CPE and CPES on the SGCL12 and TREC-WT datasets. This motivated us to look into the distribution of improvements. We report the percentages of queries where either CPES or CPEN produce the best results, and where the two methods are equal in performance (to the fourth decimal place) in Table 19.

We find that CPEN is better than or comparable in performance to CPES for a large chunk of the queries for both datasets (47.6% for SGCL12 and 87.8% for TREC-WT on nDCG@10). Moreover, the major gain of CPES comes from only around $25 - 30$ queries (out of 250) for SGCL12. So in general it is not that CPEN comes up with meaningless expansions which makes the CPEN aggregate numbers lower than CPES. Rather, CPES comes up with some good expansions which makes CPES better. It is the cumulative effect of all the expansions which is pulling the nDCG up. The query `((bach flower) remedy) (for kids)` is a classic example, where we cannot go beyond 0.56, whereas all the expansions in CPES together take it to 0.86. Similar observations were made for TREC-WT queries and documents. We believe that appropriately weighting expansions during the final scoring holds the key to performance, while keeping the required number of expansions down.

[19] `http://bit.ly/bFKSxz`, Accessed 27 Oct 2015

**Table 18.** Nesting expansions in proximity model for TREC-WT.

| Algorithm | nDCG@5 | nDCG@10 | nDCG@20 | MAP | $\mu$ (nDCG) | $\mu$ (MAP) |
|---|---|---|---|---|---|---|
| KLD | 0.2006 | 0.2150 | 0.3170 | 0.2823 | 1 | 1 |
| CPE | 0.2159 | 0.2476 | 0.3101 | **0.3139** | 1 | 1 |
| CPES | **0.2170** | **0.2481** | **0.3168** | 0.3136 | 1 | 1 |
| CPEF | 0.1552 | 0.1913 | 0.2664 | 0.2562 | 1 | 1 |
| CPEN | 0.1524 | 0.1916 | 0.2647 | 0.2556 | 1 | 1 |
| CPEFS | 0.1529 | 0.1882 | 0.2655 | 0.2547 | 1 | 1 |
| CPENS | 0.1524 | 0.1895 | 0.2647 | 0.2550 | 1 | 1 |

The highest value in a metric column is marked in **boldface**.

**Table 19.** Best method analysis (percentages).

| Dataset | SGCL12 | | | TREC-WT | | |
|---|---|---|---|---|---|---|
| Metric | CPEN | Same | CPES | CPEN | Same | CPES |
| nDCG@5 | 23.2 | 38.4 | 38.4 | 7.0 | 87.3 | 5.8 |
| nDCG@10 | 30.4 | 17.2 | 52.4 | 11.5 | 76.3 | 12.3 |
| nDCG@20 | 32.8 | 3.2 | 64.0 | 27.8 | 49.5 | 22.8 |
| MAP | 25.6 | 35.2 | 39.2 | 37.0 | 34.0 | 29.0 |

The final important item to note is that CPEN generated a far lower number of expansions for a query than CPE or CPES. This is important because each expansion has a non-trivial processing cost (Eq. 12). We report aggregate-level reductions in Table 20, averaged over queries with a particular query length (in words). The last three columns report the average number of expansions generated by the corresponding methods. For SGCL12, we observe that CPEN lowers the number of proximity expansions by $65 - 80\%$ for CPE and by $76 - 84\%$ for CPES. The corresponding numbers for TREC-WT were between $35-65\%$ (CPE) and $46 - 70\%$ (CPES). The percentage reductions for TREC-WT are smaller owing to the shorter query lengths. Thus, integration of nested segmentation with term proximity models based on expansion span analysis in documents can potentially lead to comparable performance with large benefits in terms of computational cost.

**KLD as the baseline instead of Lucene.** We tried another variation of combining the term proximity model and nested segmentation. Here, we used the KLD score as the baseline instead of Lucene. The intuition behind this experiment was to have a stronger and more replicable baseline score. Specifically, we did not re-rank the pages retrieved by Lucene, but ordered pages based on the *sum* of their KLD score and the proximity expansion score as defined in Vuurens and de Vries [26]. In this strategy, only the expansions generated by a nested segmentation were considered for proximity scoring. The proximity score was multiplied by a tuning factor $\alpha$. The obtained results are shown in Tables 21 and 22. The strategy proposed here is referred to as *KLD + ProxNest* in the tables. The best $\alpha$ was found to be 1 and 10 for SGCL12 and TREC-WT respectively. The other optimal parameter values were found to be $k = 3, win = 5, \delta = 4, w = 10$ (SGCL12) and $k = 1, win = 3, \delta = 3, w = 10$ (TREC-WT). Here we find that

**Table 20.** Numbers of proximity expansions.

| SGCL12 | | | | TREC-WT | | | |
|---|---|---|---|---|---|---|---|
| **Length** | **CPE** | **CPES** | **CPEN** | **Length** | **CPE** | **CPES** | **CPEN** |
| 5 | 13.33 | 16.26 | **3.83** | 3 | 2.75 | 3.29 | **1.77** |
| 6 | 24.42 | 30.46 | **4.85** | 4 | 7.07 | 8.29 | **2.43** |
| 7 | 30.29 | 38.00 | **5.93** | 5 | 7.00 | 10.11 | **3.33** |
| 8 | 19.75 | 44.50 | **6.75** | 6 | N. A. | N. A. | N. A. |

The minimum value in each row for each dataset is marked in **bold**.

**Table 21.** KLD with nested segmentation and proximity model (SGCL12).

| Metric | KLD | CPE | CPES | KLD + ProxNest |
|---|---|---|---|---|
| nDCG@5 | 0.7071 | 0.7396 | **0.7453** | 0.7389 |
| nDCG@10 | 0.7217 | 0.7499 | 0.7561 | **0.7579** |
| nDCG@20 | 0.7383 | 0.7642 | 0.7742 | **0.7761** |
| MAP | 0.7462 | 0.7564 | **0.7611** | 0.7520 |

The highest value in a row is marked in **boldface**.

*CPES* and *KLD + ProxNest* perform the best on two out of four metrics each on both the datasets. The difference is not statistically significant in any of the cases. Overall, we find that we cannot significantly advance the state-of-the-art over the current term proximity model, but the proposed techniques for leveraging nested segmentation show promise.

**Summary of results.** We now summarize our main findings from this paper: (a) Nested segmentation significantly outperforms state-of-the-art flat segmentation baselines when using segment structure to re-rank documents based on term proximity; (b) nested segmentation improves performance for a majority of the queries, for both datasets; (c) distances in the nested segmentation tree are more effective at re-ranking than using only document and query distances; (d) exhaustive experimentation with parameter variation shows systematic consistency of tree distance-based re-ranking over other models; (e) comparison with previous work [23] shows that the proposed nesting algorithm is better; (f) incorporating expansions generated by the nested representation of the query can potentially improve performance for the state-of-the-art term proximity model [26]; (g) exploration of fifteen algorithmic variations of our method for generating nested segmentations shows that the proposed technique produces the best results.

## 8 Oracle-based Evaluation

Recently, Saha Roy et al. [3] proposed a simple and effective evaluation framework for (flat) query segmentation that is based on the observation that all flat segments need not match exactly in the relevant documents. Therefore, in this framework (henceforth referred to as the SGCL12 framework), partially quoted versions of a query are generated by quoting and unquoting each flat segment in-

**Table 22.** KLD nested segmentation and proximity model (TREC-WT).

| Metric | KLD | CPE | CPES | KLD + ProxNest |
|---|---|---|---|---|
| nDCG@5 | 0.2006 | 0.2159 | **0.2170** | 0.2040 |
| nDCG@10 | 0.2150 | 0.2476 | **0.2481** | 0.2402 |
| nDCG@20 | 0.3170 | 0.3101 | 0.3168 | **0.3475** |
| MAP | 0.2823 | 0.3139 | 0.3136 | **0.3143** |

The highest value in a row is marked in **boldface**.

dependently[20]. The performance of the best partially quoted query (as reflected by a metric, like nDCG or MAP) is taken to be the *potential* of the segmentation, which could be achieved if one knew which segments should have been quoted for a given query. Therefore, this framework is based on a hypothetical *oracle* that supposedly predicts the best quoted version of a segmented query.

In this research, we extend the SGCL12 framework for nested segmentation algorithms so that we can likewise estimate their potential as can be achieved through quoting. In order to do so, we generalize the concept of all possible quoted versions for nested segmentation as follows. For a given nested segmentation tree of query $q$, we generate all possible quoted versions by quoting the phrases (corresponding to the subtrees) rooted at each internal node independently. For example, quoting nodes 3 and 6 (Fig. 1) would lead to the following query: `"windows xp home" edition "hd video" playback`. However, since nested quoting is meaningless, two internal nodes $x$ and $y$ (say 3 and 4) cannot be quoted simultaneously if one is the ancestor (or descendant) of the other. Following this simple constraint, all possible quoted versions can be generated for any given segmentation tree, such as those shown in Table 23 generated from the tree in Fig. 1. These quoted versions are then used in an IR engine to retrieve documents. The final oracle score (on any IR metric) for the nested segmentation is defined to be the *maximum* score over all the partially quoted versions. Here, score can be measured by any standard IR evaluation metric.

**Results and observations.** We evaluate the above idea using the sixteen nesting strategies on each of the three algorithmic variants of flat segmentation examined [16, 7, 3]. Retrieval performance is tested using SGCL12, using the Lucene (ver. 3.4.0) search system. In Table 24, we report a representative set of oracle scores (nDCG@10) averaged over the 500 queries in the test set, for flat segmentation strategies vis-à-vis their corresponding nested versions. We present results for nesting scheme $S_{GH}J_{PD}$, which, like Sec. 7, yields the highest nDCGs (all nesting schemes obtain higher scores than flat segmentations). In all rows of Table 24, we observe statistically significant improvements over the oracle score of the flat segmentations. Since all possible partially quoted versions of a nested segmentation already contains all possible quoted versions generated from the base flat segmentation, by definition the oracle score of the nested segmentation should be greater than or equal to that of the corresponding flat segmentation

[20] Please see Table 1 in Saha Roy et al. (2012) [3] for an example.

**Table 23.** Quoted versions for tree in Fig. 1. Only greyed versions are generated by flat segmentation.

| |
|---|
| `"windows xp home edition hd video playback"` |
| `"windows xp home edition" "hd video playback"` |
| `"windows xp home" edition "hd video playback"` |
| `"windows xp" home edition "hd video playback"` |
| `windows xp home edition "hd video playback"` |
| `"windows xp home edition" "hd video" playback` |
| `"windows xp home" edition "hd video" playback` |
| `"windows xp" home edition "hd video" playback` |
| `windows xp home edition "hd video" playback` |
| `"windows xp home edition" hd video playback` |
| `"windows xp home" edition hd video playback` |
| `"windows xp" home edition hd video playback` |
| `windows xp home edition hd video playback` |

**Table 24.** Mean oracle scores (nDCG@10).

| **Base Seg Algorithm** | **Flat Seg (Source: [3])** | **Nested Seg ($S_{GC}J_{MD}$)** |
|---|---|---|
| Hagen et al. [16] | 0.7670 | **0.7763***∗* |
| Li et al. [2] | 0.7560 | **0.7733***∗* |
| Mishra et al. [7] | 0.7510 | **0.7738***∗* |
| Saha Roy et al. [3] | 0.7680 | **0.7766***∗* |

Higher values in rows marked in **bold**.
$^*$ marks statistically significant improvement of nested segmentation over flat segmentation under $t$-test and Wilcoxon Signed Ranked Test; null hypothesis rejected if $p < 0.05$.

algorithm. Nevertheless, it is not necessary that the oracle score for nested segmentation will be *significantly higher* than flat segmentation. Thus, these results further indicate the superiority of nested segmentation and its stronger IR potential over flat segmentation. We also note that on SGCL12, quoting can take the nDCG@10 to a maximum of 0.832 [3]. With our current nesting algorithms, we are thus able to cut down the gap between the state-of-the-art and the best achievable performance (using quotes) by about $\simeq 13\%$.

## 9 Related Research

In essence, the nested segmentation tree specifies a complete term dependence structure, and suggests that term pairs having a low tree distance should be in close proximity in the document. Our research, thus, lies along the confluence of ideas from proximity, dependence models for IR, and query segmentation. In this section, we present a brief review of proximity and dependence models, which though are not directly related to our work, can potentially benefit nested segmentation-based retrieval strategies. Work on segmentation has been reviewed in Sec. 2.

**Term Proximity Models.** The notion that document relevance is directly improved by query terms appearing close to each other has its roots in the NEAR operator in the Boolean retrieval framework [36] by which a user can specify that two query terms should occur near each other in the document. It has fueled a plethora of research on term proximity over the years, and primarily involves incorporating proximity heuristics into a traditional retrieval model to show a performance improvement. Tao and Zhai [27] systematically explored the proximity measures that had been proposed till date, and found that the minimum document distance between *any* two terms of a query is best correlated with relevance. They also make the important conclusion that any naïve combination of the existing ranking function with a proximity measure is unlikely to be fruitful. Cummins and O'Riordan [24] further propose more heuristics and show that ideally the minimum distance between *all* pairs of terms should be examined. They also propose that any particular measure is often unlikely to give the best overall results, and that the optimal combination needs to be learnt from the data. The proximity concept has also been generalized to term sequences rather than pairs only [37–39], which has brought with it new challenges like assigning relative weights to such sequences [37]. In fact, flat segmentation strategies roughly fall under this philosophy, with the underlying assumption that proximities (or more strictly, adjacencies) are important only within flat segments. We have shown in our experiments that such a model is easily outperformed, and the tree-based model suggests which of the long range dependencies are crucial to query semantics.

**Term Dependence Models.** Traditional retrieval models like BM25 assume independence between query terms, even though the idea that certain dependencies are important for efficient retrieval is hardly new, including ideas based on tree structures [40]. Gao et al. [41] propose a language model-based framework for predicting term dependencies by assuming that a query is generated from a document in two stages; first the linkages are formed and then the terms that satisfy the linkage constraints. Metzler and Croft [25] propose that term dependencies in a query can be classified into a sequential dependence model (SDM), where adjacent terms are related, and a full dependence model (FDM) where all the terms are inter-dependent. Their results show that significant improvements in IR are possible by formally modeling dependencies and the FDM outperforms the SDM on most corpora. Concepts of term dependence [25] have also been found useful in query segmentation by Bendersky et al. [18] and relatively newer retrieval models [42]. The nested segmentation tree based retrieval is much less computationally intensive than Gao et al. [41] and more informed than Metzler and Croft [25]. The tree not only encodes the term dependencies, but also provides an effective way of weighting long range dependencies in search queries.

## 10 Conclusions and Future Work

The primary contribution of this paper lies in proposing a strategy to use nested segmentation of Web search queries for improving IR performance. We have shown that the tree structure inherent in the hierarchical segmentation can be used for effective re-ranking of result pages ($\simeq 7\%$ nDCG@10 improvement over unsegmented query for SGCL12 and $\simeq 40\%$ for TREC-WT). Importantly, since $n$-gram scores can be computed offline, our algorithms have minimal runtime overhead. The only resource used for performing nested segmentation is a query log, which is always available to search engines. Thus, we believe that they can be practically useful for large-scale Web search systems. While the concept of flat query segmentation has been around for more than a decade, there is very little work that show a significant IR benefit of directly applying the process. Therefore, it has been a long standing debate whether query segmentation is at all useful in practice for IR. Our results clearly demonstrate that hierarchical segmentation can bring in substantial IR gains for slightly long queries. The re-ranking strategy proposed can be used as an evaluation framework for nested segmentation strategies in the future. Finally, we show that incorporating expressions generated by nested segmentation into the latest term proximity model can bring about a significant reduction in computational overhead, while improving IR performance in some cases.

This research is a first systematic exploration of nested query segmentation, that raises several open questions and can be extended in several ways. From an IR perspective, it could be useful to assign weights to term pairs before their tree distance is considered, and before being used as a candidate proximity expansion. The nesting algorithm and the allied document re-ranking strategy can be improved through more sophisticated data-driven approaches. In fact, nested query segmentation can be viewed as the first step towards query parsing, and can lead to a generalized query grammar. We believe that our findings can make a major impact on query understanding if effort is appropriately channelized along these avenues.

## Acknowledgments

The first author was supported by Microsoft Corporation and Microsoft Research India under the Microsoft Research India PhD Fellowship Award.

## References

1. Hagen, M., Potthast, M., Beyer, A., Stein, B.: Towards optimum query segmentation: In doubt without. In: CIKM '12. (2012) 1015–1024
2. Li, Y., Hsu, B.J.P., Zhai, C., Wang, K.: Unsupervised query segmentation using clickthrough for information retrieval. In: SIGIR '11. (2011) 285–294
3. Saha Roy, R., Ganguly, N., Choudhury, M., Laxman, S.: An IR-based evaluation framework for web search query segmentation. In: SIGIR '12. (2012) 881–890

4. Tan, B., Peng, F.: Unsupervised query segmentation using generative language models and Wikipedia. In: WWW '08. (2008) 347–356
5. Abney, S.: Prosodic structure, performance structure and phrase structure (1992)
6. Risvik, K.M., Mikolajewski, T., Boros, P.: Query segmentation for web search. In: WWW '03 Posters. (2003)
7. Mishra, N., Saha Roy, R., Ganguly, N., Laxman, S., Choudhury, M.: Unsupervised query segmentation using only query logs. In: WWW '11. (2011) 91–92
8. Zhang, C., Sun, N., Hu, X., Huang, T., Chua, T.S.: Query segmentation based on eigenspace similarity. In: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers. ACLShort '09, Stroudsburg, PA, USA, Association for Computational Linguistics (2009)
9. Yu, X., Shi, H.: Query segmentation using conditional random fields. In: Proceedings of the First International Workshop on Keyword Search on Structured Data. KEYS '09, New York, NY, USA, ACM (2009)
10. Ganguly, D., Leveling, J., Jones, G.J.: United we fall, divided we stand: A study of query segmentation and prf for patent prior art search. In: Proceedings of the 4th Workshop on Patent Information Retrieval. PaIR '11, New York, NY, USA, ACM (2011) 13–18
11. Kiseleva, J., Guo, Q., Agichtein, E., Billsus, D., Chai, W.: Unsupervised query segmentation using click data: preliminary results. In: WWW '10, ACM (2010) 1131–1132
12. Parikh, N., Sriram, P., Al Hasan, M.: On segmentation of ecommerce queries. In: Proceedings of the 22nd ACM International Conference on Conference on Information and Knowledge Management. CIKM '13, New York, NY, USA, ACM (2013) 1137–1146
13. Jones, R., Rey, B., Madani, O., Greiner, W.: Generating query substitutions. In: Proceedings of the 15th international conference on World Wide Web. WWW '06, New York, NY, USA, ACM (2006)
14. Brenes, D.J., Gayo-Avello, D., Garcia, R.: On the fly query segmentation using snippets. In: Proceedings of the First Spanish Conference on Information Retrieval. CERI '10, Madrid, Spain, The First Spanish Conference on Information Retrieval (2010)
15. Hagen, M., Potthast, M., Stein, B., Braeutigam, C.: The power of naive query segmentation. In: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. SIGIR '10, New York, NY, USA, ACM (2010)
16. Hagen, M., Potthast, M., Stein, B., Bräutigam, C.: Query segmentation revisited. In: WWW '11. (2011) 97–106
17. Bergsma, S., Wang, Q.I.: Learning noun phrase query segmentation. In: EMNLP-CoNLL '07. (2007) 819–826
18. Bendersky, M., Croft, W.B., Smith, D.A.: Two-stage query segmentation for information retrieval. In: SIGIR '09. (2009) 810–811
19. Zhang, W., Cao, Y., Lin, C.Y., Su, J., Tan, C.L.: An error driven approach to query segmentation. In: Proceedings of the 22nd international conference on World Wide Web companion. WWW '13 Companion, Republic and Canton of Geneva, Switzerland, International World Wide Web Conferences Steering Committee (2013)
20. Zhang, W., Cao, Y., Lin, C.Y., Su, J., Tan, C.L.: Learning a replacement model for query segmentation with consistency in search logs. In: Proceedings of the Sixth International Joint Conference on Natural Language Processing, Nagoya, Japan, Asian Federation of Natural Language Processing (October 2013)

21. Pass, G., Chowdhury, A., Torgeson, C.: A picture of search. In: Proceedings of the 1st international conference on Scalable information systems. InfoScale '06, New York, NY, USA, ACM (2006)
22. Ramanath, R., Choudhury, M., Bali, K., Saha Roy, R.: Crowd Prefers the Middle Path: A New IAA Metric for Crowdsourcing Reveals Turker Biases in Query Segmentation. In: ACL '13. (2013) 1713–1722
23. Huang, J., Gao, J., Miao, J., Li, X., Wang, K., Behr, F., Giles, C.L.: Exploring web scale language models for search query processing. In: WWW '10. (2010) 451–460
24. Cummins, R., O'Riordan, C.: Learning in a pairwise term-term proximity framework for information retrieval. In: SIGIR '09. (2009) 251–258
25. Metzler, D., Croft, W.B.: A markov random field model for term dependencies. In: SIGIR '05. (2005) 472–479
26. Vuurens, J.B., Vries, A.P.: Distance Matters! Cumulative Proximity Expansions for Ranking Documents. Inf. Retr. **17**(4) (August 2014) 380–406
27. Tao, T., Zhai, C.: An exploration of proximity measures in information retrieval. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '07, New York, NY, USA, ACM (2007)
28. Bendersky, M., Croft, W.B., Smith, D.A.: Structural annotation of search queries using pseudo-relevance feedback. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management. CIKM '10, New York, NY, USA, ACM (2010)
29. Bendersky, M., Croft, W.B., Smith, D.A.: Joint annotation of search queries. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. HLT '11, Stroudsburg, PA, USA, Association for Computational Linguistics (2011)
30. Sarkas, N., Paparizos, S., Tsaparas, P.: Structured annotations of web queries. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10, New York, NY, USA, ACM (2010)
31. Chaudhari, D.L., Damani, O.P., Laxman, S.: Lexical co-occurrence, statistical significance, and word association. In: EMNLP '11. (2011) 1058–1068
32. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301) (1963)
33. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
34. Agichtein, E., Brill, E., Dumais, S.: Improving web search ranking by incorporating user behavior information. In: SIGIR '06. (2006) 19–26
35. Zhai, C., Lafferty, J.: A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst. **22**(2) (April 2004) 179–214
36. Keen, E.M.: The use of term position devices in ranked output experiments. J. Doc. **47**(1) (March 1991)
37. Bai, J., Chang, Y., Cui, H., Zheng, Z., Sun, G., Li, X.: Investigation of partial query proximity in web search. In: Proceedings of the 17th international conference on World Wide Web. WWW '08, New York, NY, USA, ACM (2008)
38. He, B., Huang, J.X., Zhou, X.: Modeling term proximity for probabilistic information retrieval models. Inf. Sci. **181**(14) (July 2011)
39. Song, R., Taylor, M.J., Wen, J.R., Hon, H.W., Yu, Y.: Viewing term proximity from a different perspective. In: Proceedings of the IR research, 30th European conference on Advances in information retrieval. ECIR'08, Berlin, Heidelberg, Springer-Verlag (2008)

40. Yu, C.T., Buckley, C., Lam, K., Salton, G.: A generalized term dependence model in information retrieval. Technical report, Cornell University (1983)
41. Gao, J., Nie, J.Y., Wu, G., Cao, G.: Dependence language model for information retrieval. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '04, New York, NY, USA, ACM (2004)
42. Peng, J., Macdonald, C., He, B., Plachouras, V., Ounis, I.: Incorporating term dependency in the dfr framework. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '07, New York, NY, USA, ACM (2007)