Window Extraction for Information Retrieval

Samuel Huston
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
Amherst, MA, 01002, USA
sjh@cs.umass.edu

W. Bruce Croft
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
Amherst, MA, 01002, USA
croft@cs.umass.edu

ABSTRACT

Proximity-based term dependencies have been proposed and used in a variety of effective retrieval models. The execution of these dependency models is commonly supported through the use of positional inverted indexes. However, few of these models detail how instances of proximate terms should be extracted from the lists of positional data. In this study, we investigate three algorithms for the extraction of windows that span a range of assumptions about the reuse of terms in multiple window instances. We observe that computed collection statistics of unordered windows are significantly affected by the choice of algorithm. We also observe that retrieval efficiency and effectiveness of a state-of-the-art dependence model are not significantly affected by the selection of window extraction algorithm.

1. INTRODUCTION

Dependency retrieval models have been repeatedly shown to improve information retrieval performance over bag-of-words retrieval models [1; 3; 7; 9; 10; 11]. A common approach in these models is to assert that all adjacent pairs of query terms are dependent [1; 7; 9]. Windows of text that contain the identified dependent pairs or sets of terms are identified in each document. Features used in these retrieval models can then be aggregated from the set of windows that are identified in a particular document.

The execution of these dependency models is commonly supported by positional indexes. See Witten et al. [13] for a definition and discussion of this type of index. Positional indexes provide access to the location of each instance of each query term, in each indexed document.

However, very few dependency retrieval models specify precisely how window instances should be identified using these term-position lists. Implicit in these underspecified dependency models is an assumption of how terms that could occur in multiple windows should be treated. In this study, we focus on the extraction of the ordered and unordered windows originally used by Metzler and Croft [7] in the Markov random field model framework. We note that similar windows of are also used in other retrieval models, including: BM25-TP [10], BM25-TP2 [12], and pDFR [9].

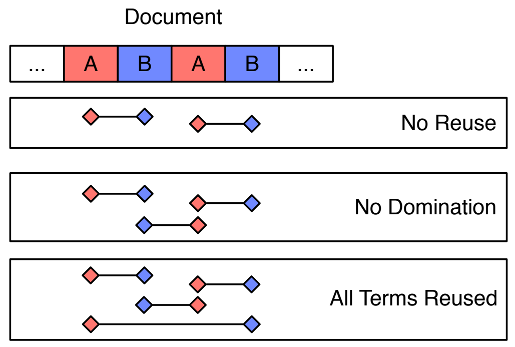

Figure 1: Example extracted sets of unordered windows for each assumption of term reuse.

We discuss three possible assumptions of term reuse. We detail corresponding window extraction algorithms. We discuss the theoretical efficiency bounds of each algorithm, then experimentally test the impact of each algorithm. We observe that there is a significant difference between the collection frequencies of windows extracted by each of these algorithms. For the state- of-the-art sequential dependence model (SDM) [7], we observe that, despite this, there is no difference in retrieval effectiveness or retrieval efficiency, between each of the algorithms.

2. ASSUMPTIONS OF TERM REUSE

As mentioned there is a potentially important efficiency and effectiveness trade-off for the extraction of ordered and unordered windows. Several different algorithms can be devised depending on choices of whether or not to reuse specific term instances in multiple window instances.

We focus on three possible assumptions of term reuse: **no-reuse**, **no-domination**, and **all-terms-reused**. Figure 1 shows an example of the windows extracted using each of these assumptions. In general, the number of window instances extracted under the no-reuse assumption is less than or equal to the number of windows instances extracted under the no-domination assumption. Similarly, the number of instances extracted under the no-domination assumption is less than or equal to the number of instances extracted under the all-terms-reused assumption.

Algorithm Unordered-Windows-No-Reuse

```
 1: INPUT: PostingIterator[ ] itrs
 2: INPUT: $w$, window width
 3: OUTPUT : WindowArray output
 4: while $true$ do
 5:     minPos = minPos($itrs$)
 6:     maxPos = maxPos($itrs$)
 7:     if ($maxPos - minPos$) $< w$ then
 8:         output.add( window($itrs$) );
 9:         for $i = 0$; $i < |itrs|$; $i = i + 1$ do
10:             $itr[i].next()$
11:             if $itrs[i].done()$; return output
12:         end for
13:     else
14:         for $i = 0$; $i < |itrs|$; $i = i + 1$ do
15:             if $itr[i].pos() == minPos$ then
16:                 $itr[i].next()$
17:                 if $itrs[i].done()$; return output
18:             end if
19:         end for
20:     end if
21: end while
```

Algorithm Unordered-Windows-No-Domination

```
 1: INPUT: PostingIterator[ ] itrs
 2: INPUT: $w$, window width
 3: OUTPUT : WindowArray output
 4: while $true$ do
 5:     minPos = minPos($itrs$)
 6:     maxPos = maxPos($itrs$)
 7:     if ($maxPos - minPos$) $< w$ then
 8:         output.add( window($itrs$) );
 9:     end if
10:     for $i = 0$; $i < |itrs|$; $i = i + 1$ do
11:         if $itr[i].pos() == minPos$ then
12:             $itr[i].next()$
13:             if $itrs[i].done()$; return output
14:         end if
15:     end for
16: end while
```

We now propose three unordered window extraction algorithms that extract sets of window instances matching each of the term reuse assumptions. An unordered window is defined over a set of dependent terms (often a pair), and a window width. Each window instance must include one instance of each term, and the distance from the first term to the last term in the window instance must be less than the width parameter, w.

First, the no-reuse assumption permits each term instance to be present in at most one window instance. Following this assumption, the Unordered-Windows-No-Reuse algorithm details a process that extracts all unordered windows that do not share any term instances. We observe that this algorithm processes each posting list only once, so we can bound the complexity of this algorithm at $\mathcal{O}(\sum_i^n |pos_i|)$ operations.

Second, the no-domination assumption allows a limited amount of reuse of term instances. Observing that each extracted instance covers a specific region of the document, we can omit dominated windows. A window dominates another when both share a starting position, and its ending position is smaller than the dominated window. The Unordered-Windows-No-Domination algorithm details a process that extracts all non-dominated windows. Again, this algorithm only processes each posting list once, so we can bound the complexity at $\mathcal{O}(\sum_i^n |pos_i|)$ operations. In practice, it is reasonable to expect that this algorithm will require slightly longer to execute than the Unordered-Windows-No-Reuse algorithm above, as the term iterators are not moved as rapidly.

Algorithm Unordered-Windows-All

```
 1: INPUT: PostingIterator[ ] itrs
 2: INPUT: $w$, window width
 3: OUTPUT : WindowArray output
 4: $itr_0 = itrs.pop()$
 5: while not $itr_0.done()$ do
 6:     output.addAll(
        ExtractWindows($itrs, itr_0.pos(), itr_0.pos()$))
 7:     $itr_0.next()$
 8: end while
 9: return output;
10: function ExtractWindows(itrs, begin, end)
11:     $itr_i = itrs.pop()$
12:     while not $itr_i.done()$) do
13:         $nBegin$ = min($begin, itr_i.pos()$)
14:         $nEnd$ = max($end, itr_i.pos()$)
15:         if $newEnd - newBegin \leq w$ then
16:             if $itrs$ is empty then
17:                 output.add( window($itrs$) );
18:             else
19:                 output.addAll(
        ExtractWindows($itrs, nBegin, nEnd$))
20:             end if
21:         end if
22:         $itr_i.next()$
23:     end while
24:     $itr_i.reset()$
25:     $itrs.push(itr_i)$
26:     return output;
27: end function
```

Finally, the all-terms-reused assumption permits the reuse of each term instance in all possible window instances. The Unordered-Windows-All algorithm details a recursive algorithm that extracts all possible window instances in the document. Note that this assumption requires that each list of positions must be processed several times to ensure that all windows are extracted. So, the worst case complexity of this algorithm is bounded at $\mathcal{O}(\prod_i^n |itr_i|)$ operations. For longer lists of positions, or larger values of n, this algorithm is likely be considerably less efficient than both of the previous algorithms.

3. EXPERIMENTS

We now empirically evaluate the impact these different assumptions of term reuse have on retrieval performance. We measure three aspects: first, the collection frequency; second, retrieval efficiency of each algorithm; and finally, retrieval effectiveness of each algorithm.

We compare each of these algorithms in the context of the

Collection	# Doc.	# Terms	# Topics
Robust-04	0.5 M	0.252 B	250
GOV2	25 M	22 B	150
ClueWeb-09-B	33 M	26 B	200
ClueWeb-09-A	201 M	123 B	200

Table 1: Summary of data sets used.

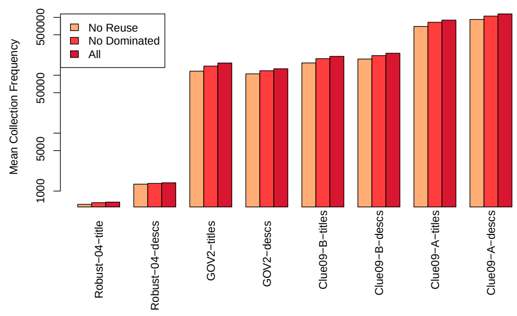

Figure 2: Mean collection frequencies extracted using three window extraction algorithms, for 8 retrieval settings (2 types of queries for 4 TREC collections).

sequential dependence model [7]. That is, all adjacent pairs of query terms are selected to be modeled as an ordered window and as an unordered window. The width of the ordered window is too small to permit any term reuse. So, the focus of the study is on unordered windows. For all experiments, the width of the extracted unordered windows is set to 8.

For each of these experiments, we use 4 TREC collections: Robust-04, GOV2, Clueweb-09-Cat-B and Clueweb-09-Cat-A. See Table 1 for the details of each collection. A total of 600 unique topics are available for these collections. Note that the topics for Clueweb-09-B are the same as the topics available for ClueWeb-09-A. Each topic has two formulations, a short title query, consisting of 1-4 terms, and an longer description query of between 5 and 30 terms. In accordance with previous research on the ClueWeb-09 collection [4], we have removed all documents with a fusion-spam-score lower than 60 from these collections. We implement each of the above window extraction algorithms using the Galago retrieval framework [1].

3.1 Results

We start by measuring the collection frequency of unordered windows for each collection, and query set. Figure 2 shows the mean collection frequency of unordered windows of width 8, as extracted by each of the three algorithms, for each TREC collection. All adjacent query terms from each of the topics for each collection are used as the input dependent query terms.

As expected, this data shows that the UNORDERED-WINDOWS-NO-REUSE algorithm identify fewer window instances than the UNORDERED-WINDOWS-NO-DOMINATION algorithm, and both identify fewer window instances that the UNORDERED-WINDOWS-ALL algorithm.

The observed differences in collection frequency between the three algorithms are relatively small. The frequencies of `no-reuse` extracted windows are around 10% lower than the `no-domination` extracted windows, and the frequencies of `all-term- reuse` windows are around 7% lower than `no-domination` extracted windows, across all collections and both types of queries.

Next, we investigate how retrieval effectiveness is affected by each algorithm. For this experiment, we use optimized the parameters for SDM, for each collection using a coordinate ascent approach described by Metzler [8].

Figure 3 shows how each algorithm effects retrieval effectiveness, as measured using mean average precision. We can clearly see that there is almost no change in effectiveness for each window extraction algorithm.

We observe similar results for other retrieval metrics, including nDCG@20, P@20, and ERR@20. For each metric, the Fisher randomization test was applied to all pairs of results, for each collection and query type. No significant differences were observed ($\alpha = 0.05$).

Finally, we measure retrieval efficiency of each algorithm. Given the complexity of each algorithm, we expect to observe that the `no-reuse` algorithm will execute faster than the `no-domination` algorithm, which will execute faster than the `all` algorithm. For this experiment, we implement similar algorithms for the extraction of ordered windows. Even though the output of the ordered window extraction algorithms remains identical in all cases for SDM, the algorithm

[1]`http://lemurproject.org/galago.php`

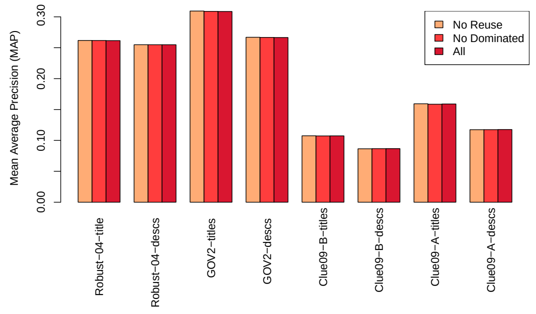

Figure 3: Mean average precision of SDM, using each of the three assumptions of term reuse, in 8 retrieval settings.(2 types of queries for 4 TREC collections).

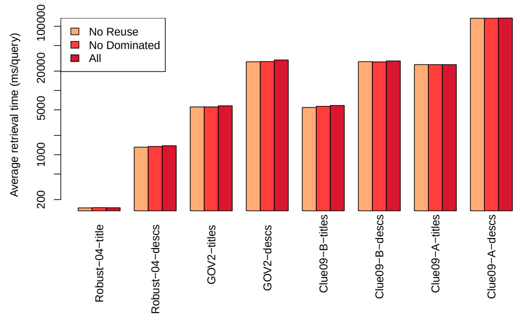

Figure 4: Average time to retrieve the top 1000 documents, using SDM, with each of the three assumptions of term reuse, in 8 retrieval settings.(2 types of queries for 4 TREC collections).

may have some impact on retrieval efficiency. Through these modifications, we aim to maximize any observed time differences between the assumptions.

Figure 4 shows the retrieval efficiency of each algorithm. The retrieval efficiency of each algorithm is measured as the time to retrieve the top 1000 documents, averaged over 5 repeated runs of all queries, for each tested collection, and type of query.

We observe almost no difference in efficiency between the `no-reuse` and `no-domination` algorithms. A small difference ($< 2\%$) is observed between the `no-domination` and `all` algorithms in some cases. The standard deviation of the reported mean processing times is computed to be less than 2% of the mean processing time for each collection.

4. RELATED WORK

While the experiments presented in this study focus on the unordered windows used in SDM, these results are applicable to several other effective dependency retrieval models. For example, BM25-TP [10], and pDFR [9] both use similar unordered window features, but do not explicitly assert how windows should be extracted from positional data, or whether extracted windows may or may not share term instances. However, as the width of the unordered windows in these retrieval models are not larger than in SDM, we expect to observe little difference in retrieval performance between the three tested assumptions of term reuse.

There are a small number of dependency models that specify how to extract window instances from positional data. First, a variant of BM25-TP [3] specifies how unordered windows can be efficiently extracted from positional data. We note that this algorithm produces a set of windows that is similar to those produced using the assumption of no-term-reuse. Second, BM25-Span [11] defines a text span as the output of the provided span extraction algorithm.

Lv and Zhai [6] proposes an alternative approaches to incorporating term proximity into the evaluation of a document. The positional language model scores documents at a specific location in the document, term instances are then propagated to the desired scoring position. So, this model does not require windows to be identified in documents.

An alternative to extracting windows from sets of positional data, is to directly index the frequency or score of the required windows. Heuristic approaches to indexing term dependencies have been shown to improve retrieval efficiency [2; 5]. However, we note that these indexes also rely on some unstated assumptions of term reuse.

5. CONCLUSIONS

Retrieval models that use windows to measure term dependence have not specified exactly how term occurrence is counted in these windows. In this study, we have defined how window instances can be counted and investigated possible trade-offs between retrieval efficiency and retrieval effectiveness present in the extraction of window instances for dependency models. We have identified three reasonable assumptions of term reuse that could allow for different retrieval performance. We present three corresponding algorithms that extract the appropriate set of unordered windows from lists of positional data for a given document.

We empirically tested the use of these algorithms in a retrieval system. From the observed data, there is no clear reason to prefer any of these window algorithms. We observe that the collection frequency of the window increases as more term reuse is permitted. However, we do not observe any significant changes in retrieval effectiveness or retrieval efficiency.

It is reasonable to expect that larger window widths will increase the differences between the algorithms. In future work, we plan to investigate the impact these assumptions have on retrieval performance for retrieval models that use larger window widths.

Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #CNS-0934322. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

[1] Michael Bendersky, Donald Metzler, and W. Bruce Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM WSDM*, pages 31–40, 2010.

[2] Andreas Broschart and Ralf Schenkel. High-performance processing of text queries with tunable pruned term and term pair indexes. *ACM Trans. Inf. Syst.*, 30(1):5:1–5:32, March 2012.

[3] Stefan Büttcher, Charles L. A. Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proc. of the 29th ACM SIGIR*, pages 621–622, 2006.

[4] Gordon V. Cormack, Mark D. Smucker, and Charles L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[5] S. Huston, A. Moffat, and W.B. Croft. Efficient indexing of repeated n-grams. In *Proc. of the 4th ACM WSDM*, pages 127–136. ACM, 2011.

[6] Y. Lv and C.X. Zhai. Positional language models for information retrieval. In *Proc. of the 32nd ACM SIGIR*, pages 299–306. ACM, 2009.

[7] D. Metzler and W.B. Croft. A markov random field model for term dependencies. In *Proc. of the 28th ACM SIGIR*, pages 472–479, 2005.

[8] Donald Metzler. Using gradient descent to optimize language modeling smoothing parameters. In *Proc. of the 30th ACM SIGIR*, pages 687–688, 2007.

[9] Jie Peng, Craig Macdonald, Ben He, Vassilis Plachouras, and Iadh Ounis. Incorporating term dependency in the dfr framework. In *Proc. of the 30th ACM SIGIR*, pages 843–844, 2007.

[10] Yves Rasolofo and Jacques Savoy. Term proximity scoring for keyword-based retrieval systems. In *Proc. of the 25th ECIR*, pages 207–218, 2003.

[11] Ruihua Song, Michael J. Taylor, Ji-Rong Wen, Hsiao-Wuen Hon, and Yong Yu. Viewing term proximity from a different perspective. In *Proc. 30th ECIR*, pages 346–357, 2008.

[12] Krysta M. Svore, Pallika H. Kanani, and Nazan Khan. How good is a span of terms?: exploiting proximity to improve web retrieval. In *Proc. of the 33rd ACM SIGIR*, pages 154–161, 2010.

[13] I.H. Witten, A. Moffat, and T.C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.