INFO256 Project Report

Implementation and Evaluation of **Xtract** in **WordSeer**

Mosharaf Chowdhury
(21020039)
mosharaf@cs.berkeley.edu

December 15, 2013

1 Introduction

Natural languages are full of word collocations that frequently co-occur and correspond to arbitrary word usages. They appear in both technical and non-technical textual corpora and often have specific significance in individual contexts. Accurately retrieving and identifying collocations from a given corpus in an unsupervised manner is imperative to understanding and automatically generating text related to that corpus.

Identifying collocations, however, is not an easy problem. They vary widely in length, in the words that are involved, and in the relations between the involved words. Moreover, words in a collocation can be adjacent or they can be separated by unrelated words. One straightforward way to retrieve collocations is to find the most frequent N-grams in a corpora. This approach is attractive because many collocations are just adjacent words that frequently appear together. **WordSeer** [1], a text analysis environment for literary corpora from UC Berkeley, uses a variation of the N-gram approach to identify collocations. Consequently, it faces the same shortcomings as any N-gram-based approach would, namely, it cannot identify collocations with non-adjacent words, it has very high recall (and low precision), and it does not provide any functional information about the identified collocations. **Xtract** [2] is a statistical tool (i.e., a collection of algorithms) developed for identifying collocations with high precision and for statistically justifying their significance. It extends the N-gram approach with some statistical filters for better effectiveness.

In this project, we have successfully implemented the **Xtract** toolkit in **WordSeer** and compared its performance with the default N-gram-based collocation retrieval mechanism. Furthermore, we have evaluated and compared the performance of both approaches on two large text corpora: Shakespeare and Abstracts. The former is a literary collection of writings by William Shakespeare, and the latter is a technical corpora with abstracts collected from top HCI conferences. The key findings are the following:

1. **Xtract** is more precise than the N-gram approach.
2. In a show-of-hands poll over a group of 30 experts, **Xtract** outputs were preferred three out of four times. However, in all cases, a sizable fraction of the participants were undecided.

3. **Xtract** is sensitive to its tuning parameters. Identifying the best set of parameters remains an open challenge.

The rest of this report is organized as follows. Section 2 provides a background on collocations as well as the basic N-gram and **Xtract** algorithms that we compare in this report. We discuss the details of our implementation along with the modifications we have made to the algorithms in Section 3. In Section 4, we evaluate the comparative performance and discuss the outcomes of micro-benchmarks. Finally, we conclude in Section 5.

2 Background

In this section, we discuss the characteristics and types of collocations, provide an overview of both the N-gram and **Xtract** algorithms, and characterize these algorithms by the types of collocations they can support.

2.1 Collocations

Collocations are frequent word combinations with the following four properties [2]. They are (i) arbitrary, (ii) domain-dependent, (iii) recurrent, and (iv) cohesive lexical clusters. In addition to the characteristics, Smadja identified three different types of collocations.

1. **Predictive relations** that are pairs of word used together in similar syntactic relations. The two words *may not be adjacent*.
2. **Rigid noun phrases** that are *uninterrupted* sequences of words that cannot be broken into smaller fragments.
3. **Phrasal templates** that consist of idiomatic phrases containing *zero or more empty slots*.

In the rest of this section, we will discuss why the N-gram approach is best suited only for rigid noun phrases (§2.2), whereas **Xtract** can handle all three types of collocations (§2.3).

2.2 The Most Frequent N-Gram Algorithm

The basic N-gram-based collocation identification algorithm is straightforward frequency counting. It has only two steps.

- **Step 1.** Find all N-grams for any $N \geq 1$ consecutive words for the given $query$.
- **Step 2.** Sort the N-grams in the descending order of their frequency.

As a result, the N-gram approach identifies a large number of collocations and the frequency distribution tends to have a long tail. The **WordSeer** implementation performs two additional optimizations.

1. It subsumes the smaller N-grams into larger ones.
2. It reports back only the K topmost N-grams to users. Because of the long tail of the frequency distribution, a small K is often enough. Currently, it uses $K = 100$.

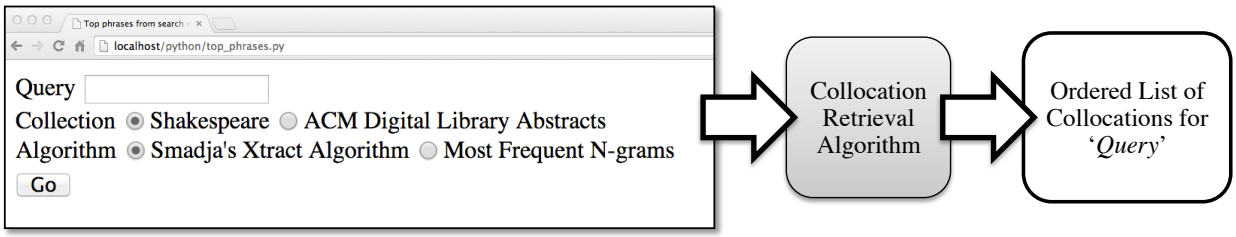


Figure 1: Basic workflow of our implementation. Users provide the $query$, select the corpora they are interested in, and pick the algorithm to use from the GUI. We use the selected algorithm to find the most important collocations and return the results ordered by frequency as a web page.

2.3 The Xtract Lexicographical Tool

Smadja's **Xtract** tool starts with the basic N-gram approach as well. However, to support different collocation types, especially the ones with non-adjacent words, it performs a sequence of statistics filtering in three stages. We provide an overview of the three stages of **Xtract** in the following and refer the reader to the original article [2] for more details.

- **Stage 1: Extracting significant bigrams.** In this stage, **Xtract** identifies adjacent or non-adjacent pairs of bigrams with one word being the given $query$. **Xtract** considers a bigram to be *significant* if it passes three statistical tests (thresholds) in the following order.
  1. k_0: Keep only the most frequent bigrams. Specifically, words that appear around $query$ at least standard deviation more frequently than the average are kept.
  2. U_0: Prefer bigrams that have syntactic preference. The bigram-forming word should be more likely to appear at specific distances (within ± 5) from $query$, i.e., it should have specific syntactic relationship with $query$.
  3. k_1: Choose most likely locations for a bigram. If a bigram-forming word can appear at multiple distances, pick the one with the highest likelihood.
- **Stage 2: From bigrams to N-grams.** In this stage, **Xtract** produces the largest N-word collocations (including wildcards) from bigrams that subsumes all M-word collocations ($M < N$). A word is kept at a certain location of the N-gram (i.e., not converted to wildcard) if its probability of appearing in that location is more than the threshold T.
- **Stage 3: Adding syntax to collocations.** In the final stage, **Xtract** attaches POS tags to the collocations for further subsuming.

3 Implementation Details

We have implemented the N-gram approach as well as **Xtract** in **WordSeer** using `python`, `mysql`, and `apache` web server. The N-gram implementation is less than 200 SLOC in `python`, whereas **Xtract** is implemented in 400 SLOC of `python` (located at `python/wordseer/cluster/xtract.py`).

Characteristics	Shakespeare	Abstracts
Number of Documents	37	6939
Number of Sentences	60219	41109
Number of Words	781720	940006
Number of Unique Words	25282	28737
Number of Unique Lemmas	19824	24360

Table 1: Characteristics of workload.

We have also updated the UI (`python/top_phrases.py`) to accommodate our changes. Please find the updated codebase attached to this submission (Details in Appendix A). Both the Shakespeare and Abstracts corpora had been preloaded into separate `mysql` databases and came with **WordSeer**. We also the lemmatization support from **WordSeer** and POS-tagging features of NLTK.

The basic workflow is quite simple (Figure 1). Users must provide a $query$ in the web interface, select one of the two corpora, and pick either of the compared approaches. We lemmatize the given $query$ and find all lemmatized sentences in the selected corpora that contains the $query$ lemma. Note that the $query$ must be a single word in the current version; we do not support multi-word query because **Xtract**'s behavior is not well-defined in that case.

Next, we apply the selected algorithm as described in §2.2 and §2.3, subsume results, remove results with all stop words, order them by frequency, and send back the results as an webpage to the user. Recall that the $query$ and the relevant sentences were lemmatized early on the process; hence, our output collocations appear in lemmatized form as well. Wildcards in **Xtract** output are represented by asterisks.

4 Evaluation

We evaluated the performance of **Xtract** and compared it with the N-gram approach using two large text corpora. In the following, we describe our methodology of evaluation followed by the results from macro- and micro-experiments.

4.1 Methodology

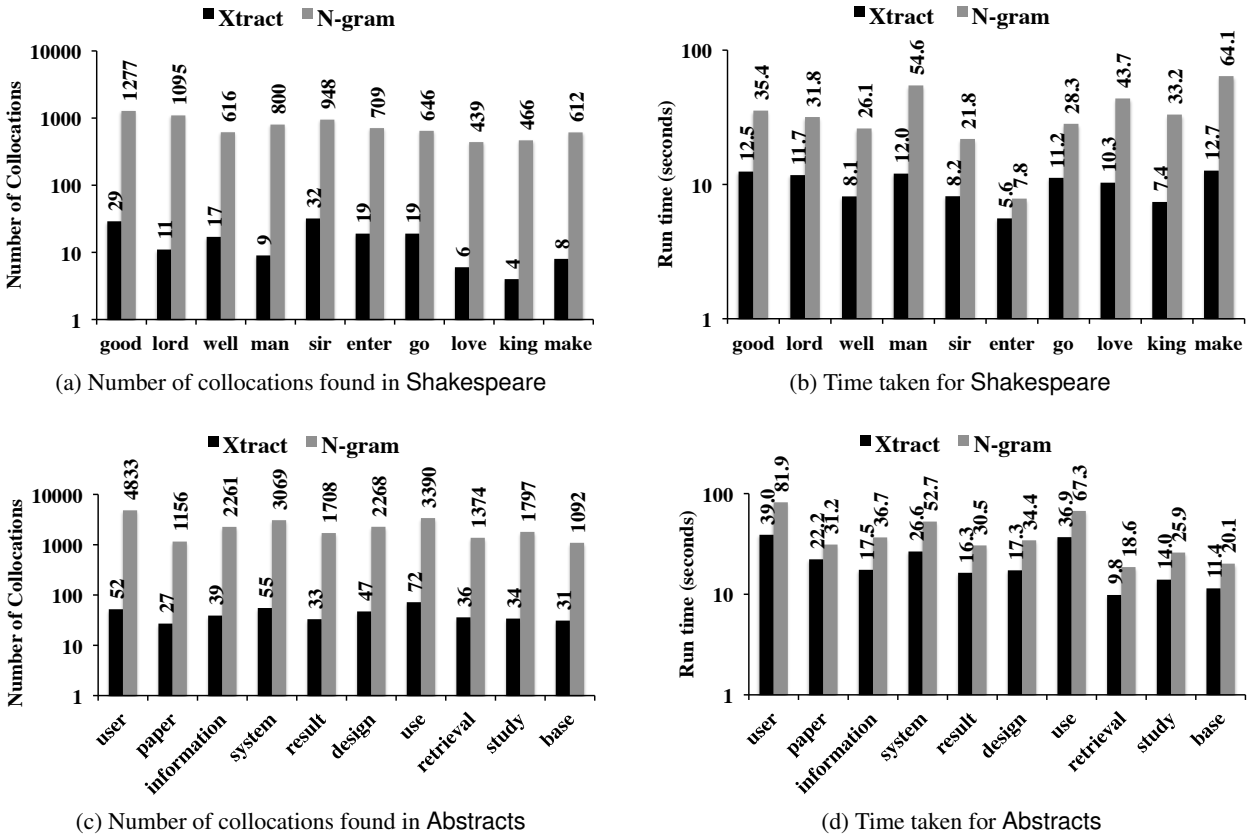
Workload We used two large text corpora, Shakespeare and Abstracts, for evaluation. The former is the complete literary collection of William Shakespeare, and the latter is collection of article abstracts collected from top HCI conferences. Table 1 summarizes some characteristics of the two corpora.

Stop Words We used the default set of stop words defined in **WordSeer**, which include prepositions, pronouns, determiners, conjunctions, modal and primary verbs, adverbs, and punctuations as stop words in our system.

Queries Unless otherwise specified, we use the words in Table 2 as queries during our evaluation. The words are ordered by their frequencies in each row.

Corpora	Queries
Shakespeare	good, lord, well, man, sir, enter, go, love, king, make
Abstracts	user, paper, information, system, result, design, use, retrieval, study, base

Table 2: Top 10 most frequent words (except stop words) in Shakespeare and Abstracts corpora.



(a) Number of collocations found in Shakespeare

(b) Time taken for Shakespeare

(c) Number of collocations found in Abstracts

(d) Time taken for Abstracts

Figure 2: Comparative evaluation of the number of collocations found and corresponding run time for the N-gram approach and **Xtract**.

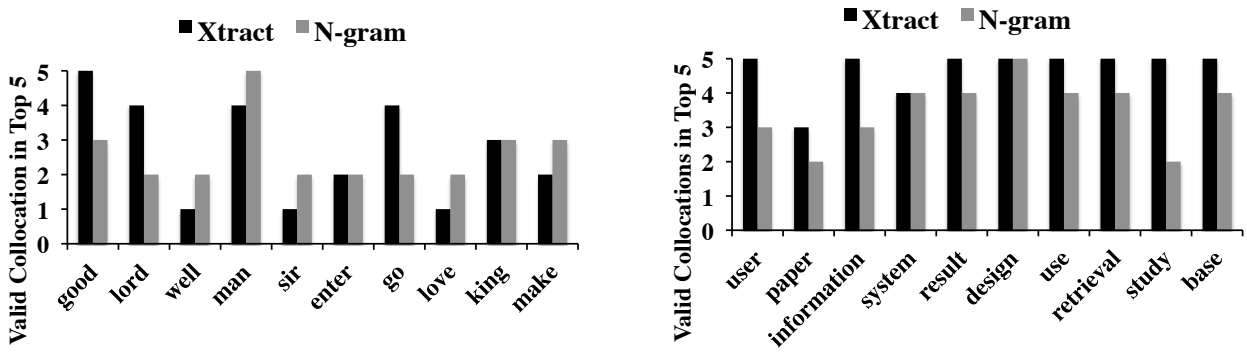
Parameters We set $k_0 = k_1 = 1$, $U_0 = 10$, and $T = 0.75$ for **Xtract** in our experiments. These are default values specified by Smadja [2].

4.2 Collocations Found and Time Taken

We start by discussing the raw characteristics of the two approaches. Figure 2 presents the number of collocations found and the time taken to find them in the two corpora for the *query* words in Table 2.

We observe that **Xtract** finds much fewer number of collocations than the N-gram approach (Figure 2a and Figure 2c). This is expected because the N-gram approach recalls *every* collocation for a given *query* and has an extremely long tail, whereas **Xtract** actively filters out infrequent/uninteresting collocations.

For the N-gram approach, the number of collocations found decreases as the *query* words become less frequent in corresponding corpus (i.e., moving from left to right in the X-axis in Figure 2a and Figure 2c).



(a) Shakespeare corpora

(b) Abstracts corpora

Figure 3: Precisions of the top 5 collocations found by the N-gram approach and **Xtract**.

POS	Count	Xtract Better	N-gram Better	Inconclusive
Noun	11	7	2	3
Verb	7	3	2	1
Adjective	2	1	1	0
ALL	20	11	5	4

Table 3: Comparative performance of **Xtract** and the N-gram approach in terms of precision based on the POS tag of the $query$. This table summarizes Figure 3.

This, however, is not always true for **Xtract**. In the Abstracts corpus, the number of collocations found remains in the same ballpark even though $query$ words become less and less frequent.

For either approach, there is no correlation between the time taken to find the collocations vs. the frequency of $query$ words. In general, our **Xtract** implementation is faster than the default N-gram implementation in **WordSeer**.

4.3 Precision and Recall

While we can easily calculate the number of collocations found and the time taken, it is much harder to calculate the precision and recall of the compared approaches. This is specially harder for the N-gram approach, because of the large number of results returned; manual inspection is almost impossible. Even for the smaller number of results returned by **Xtract** this is not easy.

To keep the problem tractable, we make a couple of compromises. First, we do not try to compute recall and instead focus on precision. Note that the original Xtract paper also presented only the precision scores. Second, we find the precision of the top 5 collocations found by each of the approaches.

Figure 3 presents the results. We see that most of the time **Xtract** has a higher precision. This is specially true for the Abstracts corpora (Figure 3b). To better understand the outcomes, we looked at the POS tags of the $query$ words. We see in Table 3 that **Xtract** performs significantly better when $query$ words are nouns. For verbs and adjective $query$ words, **Xtract** is as good or better than the N-gram approach.

query **Word**	**Corpora**	**Xtract Better**	N**-gram Better**	**Undecided**
sword	Shakespeare	70%	20%	10%
sweet	Shakespeare	30%	10%	60%
design	Abstracts	50%	30%	20%
information	Abstracts	20%	40%	40%

Table 4: Outcome of show-of-hands poll over 30-odd experts. Note that the numbers were guesstimated on the spot; hence, the relative differences are more important than the actual values.

4.4 Show-of-Hands Poll

In addition to performing manual inspection, we also performed four show-of-hands poll among 30-odd experts. We selected two *query* words from each corpora: sword and sweet from Shakespeare, and design and information from Abstracts. By showing the top 10 collocations identified by the compared algorithms, we asked the experts which one they prefer.

Table 4 presents the outcome. We see that three out of four times they selected **Xtract** outcomes. We found that **Xtract** was preferred more in presence non-adjacent collocations; recall that the N-gram approach can only identify collocations with adjacent words.

4.5 Sensitivity of Xtract

All four parameters of **Xtract** – k_0, k_1, U_0, and T – are for filtering out uninteresting collocations. For the best performance, given a corpus, the key is to find the suitable region in the four-dimensional parameter space of **Xtract**. Setting them too low will be too permissive, while setting them too high can be excessively prohibitive. In our evaluation, we did see quite good precision results for **Xtract** using Smadja-suggested threshold values, at least among the top 5 collocations. As a result, we were lucky in not having to manually tune these parameters. There is, however, a non-zero probability that we could further improve our results with a better set of parameter values. We consider the problem of auto-tuning for the best parameter set out of the scope of this project, but we suggest this a possible avenue of future research.

5 Conclusion

Collocations play an important role in understanding as well as generating text in natural language. In this project, we have implemented a statistics-based collocation retrieval tool **Xtract** in **WordSeer** and compared its performance and precision with that of the most-frequent-N-grams approach. We have found that **Xtract** performs well in most cases; however, given its simplicity, the N-gram approach perform reasonably well too. A minor issue in using **Xtract** could be its parameter sensitivity. We recommend further work toward auto-tuning **Xtract** parameters.

Acknowledgements

We would like to specially thank Aditi Muralidharan, the primary author of **WordSeer**, for her continued support on various aspects of this project.

References

[1] Aditi Muralidharan and Marti Hearst. WordSeer: Exploring language use in literary text. In *HCIR*, 2011.

[2] Frank Smadja. Retrieving collocations from text: Xtract. *Computational linguistics*, 19(1):143–177, 1993.

A Layout of the Code Directory

```
- api/
  - python/
    - cluster.py
    - top_phrases.py          // UI
    - wordseer/
      - __init__.py
      - cluster
        - __init__.py
        - k_means.py
        - top_phrases.py      // N-gram implementation
        - xtract.py           // Xtract implementation
      - search.py
      - sentence.py
      - setup.py
      - util.py
      + xmlparse
  + search-with-phrases
  + theme-finder
```