A CLUSTERING APPROACH FOR THE UNSUPERVISED RECOGNITION OF NONLITERAL LANGUAGE

by

Julia Birke
B.A., McGill University, 1996

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the
School
of
Computing Science

SIMON FRASER UNIVERSITY

Summer 2005

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

I further grant permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection.

I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying, publication or public performance of this work for financial gain shall not be allowed without my written permission.

Licence for use of multimedia materials:

☐ No separate DVD or CD-ROM material is included in this work. Multimedia licence not applicable to this work.

☐ Public performance permitted:
Multimedia materials that form part of this work are hereby licenced to Simon Fraser University for educational, non-theatrical public performance use only. This licence permits single copies to be made for libraries as for print material with this same limitation of use.

☐ Public performance not permitted:
Multimedia materials that form part of this work are hereby licenced to Simon Fraser University for private scholarly purposes only, and may not be used for any form of public performance. This licence permits single copies to be made for libraries as for print material with this same limitation of use.

Title of Thesis:

A Clustering Approach for the Unsupervised Recognition of Nonliteral Language

Author: _____

Julia Birke

_____

(Date Signed)

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

APPROVAL

Name: **Julia Birke**

Degree: **Master of Science**

Title of Thesis: **A Clustering Approach for the Unsupervised Recognition of Nonliteral Language**

Examining Committee:

Chair: **Dr. Greg Mori**
Assistant Professor of Computing Science

_____

Dr. Anoop Sarkar
Senior Supervisor
Assistant Professor of Computing Science

_____

Dr. Fred Popowich
Supervisor
Professor of Computing Science

_____

Dr. William Dolan
External Examiner
Head of Natural Language Research, Microsoft Research

Date Defended: _____

ABSTRACT

In this thesis we present TroFi, a system for separating literal and nonliteral usages of verbs through unsupervised statistical word-sense disambiguation and clustering techniques. TroFi distinguishes itself by redefining the types of nonliteral language handled and by depending purely on sentential context rather than selectional constraint violations and paths in semantic hierarchies. TroFi uses literal and nonliteral seed sets acquired and cleaned without human supervision to bootstrap learning. We adapt a word-sense disambiguation algorithm to our task and add learners, a voting schema, SuperTags, and additional context. Detailed experiments on hand-annotated data and the introduction of active learning and iterative augmentation allow us to build the TroFi Example Base, an expandable resource of literal/nonliteral usage clusters for the NLP community. We also describe some other possible applications of TroFi and the TroFi Example Base. Our basic algorithm outperforms the baseline by 24.4%. Adding active learning increases this to over 35%.

DEDICATION

To my family: Hanns Peter, Barbara, and Lisa.

"We struggle with the complexities and avoid the simplicities."

~ Norman Vincent Peale (1898-1993)

ACKNOWLEDGEMENTS

I offer my enduring gratitude to Drs. Anoop Sarkar, Fred Popowich, Bill Dolan, and Dan Fass for all their help, input, and support. Many thanks also to the ever-friendly School of Computing Science staff, particularly Heather Muckart, to the Library staff, particularly Penny Simpson, and to the many wonderful people I have met at SFU over the years. Special thanks to Stas Bekman and Chris Demwell for their Perl and command-line pointers. Finally, I am eternally grateful to my family and friends for their unwavering patience and moral support.

This thesis makes use of the following resources and software for the creation and preprocessing of input data:

The 1987-89 Wall Street Journal (WSJ) Corpus Release 1.

Part of the Penn Treebank Project.

The Penn Treebank Project annotates naturally-occuring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information -- a bank of linguistic trees. We also annotate text with part-of-speech tags, and for the Switchboard corpus of telephone conversations, dysfluency annotation. We are located in the LINC Laboratory of the Computer and Information Science Department at the University of Pennsylvania.

All data produced by the Treebank is released through the Linguistic Data Consortium.

SuperTagger

Copyright (C) 1997 B. Srinivas

A tool for tagging sentences with SuperTags (elementary trees)

This software comes with ABSOLUTELY NO WARRANTY. This is free software.

If you add any new features to this tool or make any improvements, we would like to have access to these versions.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1.0 or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

If you have comments or questions, contact the author by email at srini@research.att.com, or by regular mail at

The Xtag Project
Institute for Research in Cognitive Science
University of Pennsylvania
3401 Walnut St., Suite 400C
Philadelphia, PA 19104-6228
USA

SEND BUG REPORTS to srini@research.att.com

Tagger

SOFTWARE LICENSE

MXPOST and MXTERMINATOR

Developed by Adwait Ratnaparkhi
University of Pennsylvania
Dept. of Computer and Information Science
200 South 33rd Street
Philadelphia, PA. 19104

LICENSE INFORMATION

Adwait Ratnaparkhi ("Owner") grants to the individual researcher who downloads this software ("Licensee") a non-exclusive, non-transferable run-time license to use the MXPOST and MXTERMINATOR software ("Software"), subject to the restrictions listed below under "Scope of Grant."

SCOPE OF GRANT

The Licensee may:

- use the Software for educational or research purposes;
- permit others under the Licensee's supervision at the same site to use the Software for educational or research purposes;
- copy the Software for archival purposes, provided that any such copy contains all of the original proprietary notices.

The Licensee may not:

- use the Software for commercial purposes;
- allow any individual who is not under the direct supervision of the Licensee to use the Software;
- redistribute the Software;
- copy the Software other than as specified above;
- rent, lease, grant a security interest in, or otherwise transfer rights to the Software;
- remove any proprietary notices or labels accompanying the Software;

DISCLAIMER

The Owner makes no representations or warranties about the suitability of the Software and Linguistic Resources, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. The Owner shall not be liable for any damages suffered by Licensee as a result of using, modifying or distributing the Software or its derivatives.

CONSENT

By downloading, using or copying the Software, Licensee agrees to abide by the intellectual property laws, and all other applicable laws of the U.S., and the terms of this License. Ownership of the Software shall remain solely with the Owner.

TERMINATION

The Owner shall have the right to terminate this license at any time by written notice. Licensee shall be liable for any infringement or damages resulting from Licensee's failure to abide by the terms of this License.

WordNet

WordNet 2.0 Copyright © 2003 by Princeton University. All rights reserved.
THIS SOFTWARE AND DATABASE IS PROVIDED "AS IS" AND PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PRINCETON UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
The name of Princeton University or Princeton may not be used in advertising or publicity pertaining to distribution of the software and/or database. Title to copyright in this software, database and any associated documentation shall at all times remain with Princeton University and LICENSEE agrees to preserve same.

Perl Implementation
Lingua::Wordnet by Dan Brian

Wayne Magnuson English Idioms Sayings & Slang

Prairie House Books
Box 84007 Market Mall, Calgary, Alberta, Canada T3A 5C4, Phone +1 403 202-5438, FAX +1 403 202-5437, Email phbooks@telusplanet.net
A printed version of these idioms is available as ISBN 1-895012-09-0 (**New:** 4th printing now available) and a CD-ROM version as ISBN 1-895012-19-8.

Conceptual Metaphor Home Page WWW Server

Andrew Saulters' English Verbs

Andrew Saulters
36996 Georgia Tech Station
Atlanta, GA 30332-1710

The Porter Stemming Algorithm

Perl Implementation
http://www.tartarus.org/~martin/PorterStemmer/perl.txt

TABLE OF CONTENTS

Approval ..... **ii**

Abstract ..... **iii**

Dedication ..... **iv**

Acknowledgements ..... **v**

Table of Contents ..... **ix**

List of Figures ..... **xii**

List of Tables ..... **xiv**

1 Introduction **1**

1.1 Motivation 1
1.2 Contribution 4
1.3 Organization 9

2 A Definition of Terms **11**

2.1 Literal 12
2.2 Nonliteral 14
2.2.1 Metaphor 16
2.2.2 Metonymy and Synecdoche 21
2.2.3 Irony 22
2.2.4 Idioms and Phrasal Verbs 23
2.2.5 Anomalous Semantic Relations 24
2.3 TroFi's Choice 25
2.4 Summary 26

3 Metaphor & Metonymy Processing Review **27**

3.1 Rule-based Systems 28
3.1.1 Russell 29
3.1.2 Fass 29
3.1.3 Martin 30
3.1.4 Narayanan 31
3.2 Dictionary-based Systems 32
3.2.1 Dolan 33
3.2.2 Zernik & Dyer 34
3.3 Corpus-based Systems 35
3.3.1 Murata et al. 36
3.3.2 Nissim & Markert 37
3.3.3 Mason 38
3.4 Metaphor Processing Future 39

4 Word-Sense Disambiguation & Clustering Review 41
4.1 Word-sense Disambiguation Methodologies 41
4.1.1 Resnik 42
4.1.2 Yarowsky 42
4.1.3 Karov & Edelman 43
4.1.4 Beeferman et al. 44
4.2 Clustering Methodologies 44
4.2.1 Lee & Pereira 45
4.3 Summary 46

5 TroFi Overview 47

6 The Data 55
6.1 Data Sources 55
6.1.1 The Wall Street Journal Corpus 55
6.1.2 WordNet 57
6.1.3 Wayne Magnuson English Idioms Sayings & Slang 58
6.1.4 The Conceptual Metaphor Home Page 59
6.1.5 Target Word List 59
6.2 Original Set and Feedback Set Creation 60
6.2.1 Basic Original Sets and Feedback Sets 60
6.2.2 Variations on the Feedback Sets 62
6.3 Summary 72

7 Models & Algorithms 73
7.1 The Unsupervised Algorithm 73
7.1.1 The Basic Algorithm 74
7.1.2 Enhancements 92
7.2 Active Learning 102
7.3 Iterative Augmentation 106
7.4 Summary 108

8 Core Experiments & Results 109
8.1 Evaluation Criteria and Methodology 109
8.2 Baseline 111
8.3 Experiment 1: Sum of Similarities vs. High Similarity 111
8.4 Experiment 2: Comparison of Learners 113
8.5 Experiment 3: Effects of Voting 117
8.6 Experiment 4: Effects of Using SuperTags 119
8.7 Experiment 5: Effects of Increasing Context 122
8.8 Summary 126

9 Active Learning Experiments & Results 129
9.1 Experiment 1: Setting the Parameters 130
9.2 Experiment 2: Timing 132
9.3 Experiment 3: Random Comparison 136
9.4 Benefits of TroFi with Active Learning over Manual Clustering 139
9.5 Summary 143

10 Building the TroFi Example Base 144

11 Conclusion 149

11.1 Future Work 150

11.1.1 Core Algorithms 150

11.1.2 Active Learning 152

11.1.3 TroFi Example Base 152

11.1.4 Applications 153

11.2 Summary and Contributions 159

Appendices 161

Appendix A 161

Bibliography 169

LIST OF FIGURES

Figure 2-A Anatomy of "All the world's a stage" 17
Figure 7-A Iterative Updating 76
Figure 7-B WSM for *grasp*: Initial 82
Figure 7-C Original SSM for *grasp* : 1st Iteration....84
Figure 7-D WSM for *grasp*: 1st Iteration 85
Figure 7-E SSMs for *grasp*: 1st Iteration 85
Figure 7-F WSM for *grasp*: 2nd Iteration....86
Figure 7-G WSM for *grasp*: 4th Iteration 89
Figure 7-H SSMs for *grasp*: 4th Iteration 89
Figure 7-I SSMs for *grasp* extended: 6th Iteration 91
Figure 7-J Final Cluster Attraction Using Highest Similarity vs. Sum of Similarities....94
Figure 7-K Final Cluster Attraction Using Learner A and *Hypothetical* Learner A 97
Figure 7-L Final Cluster attraction using Learner C^1 99
Figure 7-M Voting Results 101
Figure 8-A Highest Similarity and Sum of Similarities Comparison 112
Figure 8-B No Scrub and Learner Comparisons....114
Figure 8-C Simple Voter and Weighted Voter Comparisons 118
Figure 8-D Best Voter and Baseline Comparison....119
Figure 8-E Plain vs. SuperTags using Learner A Doubled Schema....120
Figure 8-F Weighted Voters Comparison for SuperTags....121
Figure 8-G Plain, SuperTag, and Baseline Comparison 122
Figure 8-H Context Comparisons 124
Figure 8-I Original Context Only and Hybrid Context Comparison 125
Figure 8-J Baseline and Hybrid Context Comparison....126
Figure 9-A Parameter Evaluation on Held-Out Set of "fill"....131
Figure 9-B Comparison of Models for *When* to Send Sentences to Human....133
Figure 9-C Optimal Active Learning and Baseline Comparison....136
Figure 9-D Random and TroFi-Selected Comparison 138
Figure 9-E TroFi and Manual Accuracy for Same Human Effort....141
Figure 9-F Human Effort Required to Attain 64.9% Accuracy....142
Figure 10-A Words from Literature using Optimal TroFi Model....146
Figure 10-B All 50 Target Words Results 147

Figure 11-A Separating Literals and Nonliterals 155
Figure 11-B Reclustering across the Literal/Nonliteral Divide 156
Figure 11-C Storing Phrases in the Example Base 157
Figure 11-D Example Base in a Machine Translation System 158

LIST OF TABLES

Table 7-A SSMs for *grasp*: Initial.....83
Table 8-A Usage Counts per Word.....110
Table 9-A Active Learning Parameter Combinations.....130
Table 9-B Results of Manually Clustering 30% of the Original Sentences.....140
Table 10-A Target Words Selected from the Literature.....145

1 INTRODUCTION

1.1 Motivation

Machines can speak. They can understand. In the movies.

Realistically speaking, of course, we are still far removed from even out-dated futuristic machines like R2-D2 and George Jetson's trusty robotic maid Rosie. Part of the reason for this is the inability of robots, and, for that matter, all systems based on Natural Language Processing (NLP) techniques, to deal appropriately with nonliteral language. Why, even Rosie, asked to bring something to drown George's sorrows, might bring a bucket of water instead of a bottle of whisky. And such problems are certainly not restricted to science fiction.

Our local telephone company, for example, recently replaced some of its customer service representatives with a dialogue system. The public reaction has not been particularly favorable, and many people do their best to trick the system into passing the call to an actual human. This ambivalence might arise from the fact that people are still uncomfortable interacting verbally with machines and the fact that current dialogue systems really are not all that clever.

As more and more companies move to dialogue systems and other automated customer relationship management (CRM) systems, it is likely that people will start to feel less ill-at-ease around them. However, increased use will translate into increased frustrations if these systems continue to misunderstand the needs of the users.

One of the quickest ways to derail a conversation with a dialogue system is to throw some nonliteral language at it. It is easy to imagine that a history-driven system might wander down the completely wrong path if it interprets nonliteral language literally, resulting in endless

frustrations for the user. Let us take as an example the following actual conversation with a web-based dialogue system:

> User: Can you tell me where I can grab a quick bite?
>
> System: Wherever you are right now.
>
> User: No, no. I mean, where can I get a bite to eat?
>
> System: Oh I get it. Thanks for explaining that. Have you tried the personal ads?
>
> User: What for? I want something to still my hunger.
>
> System: I haven't heard anything like that before: unknown person said he wan't (sic.) something to still his or her hunger.
>
> User: My tummy is growling!
>
> System: Have you told many people that your tummy is growling?
>
> User: Of course not. Can you recommend a restaurant?
>
> System: Have you tried looking in the Open Directory? I'll ask around and get back to you.

As we can see, it is only by resorting to a literal question that we get a reasonable reply. However, if dialogue systems are to become genuinely useful, they must be able to deal with any sort of input the user throws at them. In addition, dialogue systems will seem far more human and approachable if they themselves use nonliteral language when appropriate.

So why is it so important for Natural Language Processing (NLP) systems – which encompass not only dialogue systems, but also paraphrasing and summarization, language generation, machine translation, information extraction, and a number of other applications – to be able to process such a tiny, specialized part of language?

As it turns out, human language is filled to overflowing with nonliteral language. It infiltrates not only high literature, but also newspaper articles, technical reports, Master's theses, and everyday speech. Until the advent of computer systems, the hunt for nonliteral language, such as metaphors, idioms, and other tropes, and the desire to understand how human beings process them was important for research into human language and thought. Now that growing numbers of NLP tasks are filling the marketplace, being able to recognize and process nonliteral language has become even more important.

Many incomprehensible or irrelevant responses to queries, as well as many unfortunate translations, can be avoided if the NLP system in question includes a method for processing nonliteral language. Let us look at another example.

With the recent growth in bioinformatics, software is having to extract all sorts of information out of medical notes, reports, etc. It is difficult to imagine how a computer would tell the difference between the following two instances of the phrase "pain in the neck":

1. ER report states that she went to hospital for physical therapy due to **pain in the neck**.
2. Doctor firing patient. States: abusive; general **pain in the neck**.

Several researchers – such as Fass (1997), Martin (1990, 1992), and Russell (1976), to name just a few – have taken an interest in the processing of nonliteral language in the past. Unfortunately, most of these systems require a tremendous hand-coding effort and a large amount of processing time, and even then they tend to apply only to a limited domain. Interest in the field seems to be growing again now, with more of an emphasis on automating the learning process (e.g. (Mason 2004)).

The major question for all sorts of NLP systems tends to be: *rule-based* or *statistical*? Most of the metaphor processing systems developed to date have used some sort of rule-based methodology, giving very good results on the set of data they were developed for. What seems to be called for in many situations dealing with large amounts of data, on the other hand, is something that is not necessarily 100% but can be thrown at any amount of material in any domain. This calls for a statistical approach.

The main shortcoming of statistical approaches is the necessity of a training set of some sort. If the algorithm is supervised – which is still likely to achieve better results than an unsupervised algorithm – the training set must be manually annotated. This presents us with a bottleneck similar to writing rules: the algorithm can learn to generalize once the training set is in place, but the initial human effort required is prohibitive, particularly with something as

ubiquitous[1] as nonliteral language. Manually annotating large amounts of training data is time-consuming and error-prone, particularly because it is often extremely difficult to decide whether a word is being used literally or nonliterally, even for humans. Still, several annotation efforts are currently under way (Markert & Nissim, 2002; Semino & Steen, 2001). Another problem with this approach is that nonliteral language is creative and people are apt to come up with novel expressions on a daily basis. Also, for supervised learning algorithms trained on such annotated texts, there might not be enough examples of different nonliteral usages to allow the system to learn anything about them. Finally, it is desirable to have annotated sets in a myriad of languages, which would require a massive international annotation effort. In conclusion, completely unsupervised algorithms may prove a little too unpredictable; manually annotating training data, a little too painful.

Perhaps what is needed is a *tool* to help the human build a type of literal/nonliteral *example base*. Such a collection of literal and nonliteral sentences could be used not only as a training set for a variety of statistical algorithms, but also as a resource for other nonliteral language research. It could further prove a useful resource for any number of NLP applications, including dialogue systems, machine translation systems, and information extraction systems.

1.2 Contribution

In this thesis we present TroFi, a system for separating literal and nonliteral usages of verbs through unsupervised statistical word-sense disambiguation and clustering techniques. We provide a brief summary in the following paragraph and then elaborate on these points in the rest of this section.

TroFi distinguishes itself by redefining the types of nonliteral language handled and by depending purely on sentential context rather than selectional constraint violations and paths in

[1] Allusion to *The Ubiquity of Metaphor*, the anthology containing (Newmark, 1980).

semantic hierarchies. This work is useful for various NLP applications and the science of lexical semantics. We adapt a word-sense disambiguation algorithm to our task and add learners, a voting schema, SuperTags, and additional context. Detailed experiments on hand-annotated data and the introduction of *active learning* and *iterative augmentation* allow us to build the TroFi Example Base, an expandable resource of literal/nonliteral usage clusters for the NLP community. We also describe some other possible applications of TroFi and the TroFi Example Base. Our basic algorithm outperforms the baseline by 24.4%. Adding active learning increases this performance gain to over 35%.

We now examine the contributions made by this thesis to the field of nonliteral language processing in more detail. We consider the main contribution to be a process and algorithms for creating the type of literal/nonliteral example base described in Section 1.1. The second is an actual example base for 50 target words. The third is a new approach to the nonliteral language processing problem.

Before we begin, let us look at a concrete example of our goal. Consider the following two sentences:

> NONLITERAL: Mr. Rowland “touched on the matter” of a possible stake purchase again last week, according to Sir Michael.
>
> LITERAL: “People in Washington touch cheek-to-cheek quite often," says Effi Barry, wife of the city's mayor, Marion Barry Jr.

This is a tiny snippet of the entry for “touch” in the TroFi Example Base. As humans, we can see a distinction between the usages of “touch” in these two sentences based on our understanding of the *literal* (see Chapter 2) meaning of “touch”. TroFi finds such distinctions automatically for arbitrary verbs in its capacity as a nonliteral language processing system.

TroFi uses an unsupervised algorithm – with an optional *active learning* component – to separate literal and nonliteral usages of verbs in a corpus. In its most basic form, TroFi is a *reduction*: it reduces a difficult problem – nonliteral language recognition – to one that we know

(more or less) how to solve – word-sense disambiguation. TroFi has at its core an existing similarity-based word-sense disambiguation algorithm[2]. In order to make this algorithm work for nonliteral language recognition, we make one fundamental assumption and then add a number of important enhancements to the base algorithm. The fundamental assumption is that literal and nonliteral usages of a word can be treated simply as two senses of that word, allowing us to reduce the problem of distinguishing between them to one of word-sense disambiguation. In order to make the selected word-sense disambiguation algorithm work with our two new senses, however, we must introduce a number of significant enhancements:

- the use of databases of known metaphors, idioms, and expressions
- the introduction of scrubbing, different learners, and a voting system
- a modification to the way similarity is calculated
- the use of additional features not directly visible in the input, such as SuperTags (Bangalore & Joshi, 1999)
- the use of additional context
- the addition of an active learning component

We discuss these points briefly below. They will be examined in detail in Chapters 6 and 7.

The basic word-sense disambiguation algorithm works by creating a set of examples for each sense of a given word in a machine-readable dictionary (MRD). It then extracts all the sentences containing the word of interest out of a corpus and attempts to attract them to one of the sense sets.

Obviously this will not work for the literal/nonliteral case unless the MRD senses are separated into literal and nonliteral at some point in the process. It is undesirable to have to do this manually since we want to be able to create example bases that cover as many words as possible. The need to keep TroFi unsupervised becomes even more obvious when we consider its

[2] See (Karov & Edelman, 1998). Also Section 7.1.1.1.

potential application to other languages. One would not want to have to employ speakers of each language to sit down and manually separate MRD senses. We tackle the sense separation problem by using information from databases of known metaphors, idioms, and expressions to build the nonliteral sense set. This then allows us to leave all the MRD senses together as a single literal sense set. Unfortunately this set will still contain a number of nonliteral senses.

To counteract this problem, we introduce the notion of *scrubbing*: using certain properties of the input data to determine that a given sense set, word, or *feature set*[3] should be moved from the literal sense set to the nonliteral sense set or vice versa, or removed altogether. Different scrubbing algorithms yield different results for different words, so in order to get the best performance possible, we produce a number of *learners*, each of which is scrubbed differently. We also establish a voting schema that takes into account the results of all the learners to make a decision about each target-word sentence.

In the original algorithm each target-word sentence is attracted to the single sentence to which it shows the highest similarity. This works well on a collection of small, internally homogeneous sense sets. It works less well on a large binary division where each sense set contains any number of sub-senses. For this reason we introduce the notion of *sum of similarities*, where we take into account the combined similarities shown by all sentences in the sense sets.

The TroFi algorithm allows for the use of all sorts of features in its sense sets. For example, we augment a basic *bag-of-words* approach with syntactic structure. The novelty of our approach is that we do not just use simple *n*-grams; rather, we use SuperTag/lexeme combinations. These include both syntactic and lexical information in a format that is simple yet informative. In terms of features, we also improve results by expanding the context used to include both the sentence preceding and the sentence following the target-word sentence.

[3] See Section 6.2.

Through all these enhancements we are able to produce results that are, on average, 16.9% higher than the core algorithm and 24.4% higher than the baseline.

Since TroFi is meant to *help* the human build an example base, we introduce an optional *active learning* component that allows the human to get involved in further improving the results. TroFi sends sentences it is not sure about to the human, and by agreeing to do up to 30% of the work[4], the human can help improve the results by another 11 or 12%. This may not seem like much for 30% of the effort, but we must keep in mind that TroFi does not always send the full 30%. We did some calculations based on the number of sentences that are sent to the human on average (see Section 9.4) and found that, for the same amount of effort, where a purely manual process would yield an average accuracy of about 21.7%, TroFi attains about 64.9%. Stated another way, to reach the same accuracy obtained by using TroFi, a manual process would require approximately 35% more effort.

The first contribution of this thesis, the TroFi algorithm, allows us to produce the second contribution, the TroFi Example Base. The TroFi Example Base currently consists of literal and nonliteral cluster of sentences from the Wall Street Journal Corpus for 50 target words. It was built using a process called *iterative augmentation* (see Section 7.3). This example base, which can be expanded using the TroFi algorithms, is meant to serve both as a resource for further research and as training data for other statistical algorithms.

As a final contribution, this thesis suggests a new way of approaching the problem of nonliteral language in NLP. It does not profess to be a metaphor or metonymy processing system. It will neither tell the difference between different types of nonliteral language, nor will it provide an interpretation of what a metaphor or metonymy might mean[5] or even how it may have been derived. There is a lot of work in this area (Nissim & Markert, 2003; Dolan, 1995;

[4] Often less is required.
[5] However, see Section 11.1.4.1 for a possible method of interpretation through literal/nonliteral sentence alignment.

Fass, 1997; Martin, 1990; Mason, 2004), and in future work we hope to show that the unsupervised approach presented here could be used as input for nonliteral language interpretation approaches.

We do, however, suggest that it may be worthwhile too take a step back and look at the nonliteral language recognition problem as one that could be approached using a more *brute force* methodology. The motivation for this is two-fold: first, with ever-decreasing hardware limitations, it is possible to make large, statistically-based solutions workable[6]; second, in real-world applications people are often willing to trade off high-maintenance perfection for easy-to-implement flexibility and scalability. Our approach meets the simplicity criterion in that it does not require explicit metaphor maps, reams of linguistic rules, or even the calculation of distances between nodes in a semantic hierarchy (see Chapter 3). In terms of flexibility and scalability, we purposely applied TroFi to real-world data rather than carefully collected example sets and still obtained reasonable results. The success of this initial TroFi implementation suggests that a scalable literal/nonliteral recognition system applicable to any domain and any language, and requiring minimal human effort, is both a worthwhile and attainable goal.

1.3 Organization

Below we provide an overview of the organization of this thesis.

Chapter 1 Introduction – In this chapter, we provide a discussion of the motivations behind TroFi, an examination of the contributions made by the work, and an overview of the organizational structure of the thesis.

Chapter 2 A Definition of Terms – In this chapter, we define those terms that are most likely to be contentious, namely *literal* and *nonliteral*. Other terminology is defined throughout the thesis on a need-to-know basis.

[6] Case in point: The IBM models for statistical machine translation.

Chapter 3 Metaphor & Metonymy Processing Review – In this chapter, we provide a literature review of certain past and current metaphor and metonymy processing systems.

Chapter 4 Word-Sense Disambiguation & Clustering Review – In this chapter, we discuss systems and algorithms designed to solve the problems to which we are trying to reduce the nonliteral language recognition problem.

Chapter 5 TroFi Overview – In this chapter, we provide a brief overview of TroFi.

Chapter 6 The Data – In this chapter, we discuss various data sources and how those data sources are moulded into suitable input for TroFi.

Chapter 7 Models & Algorithms – In this chapter, we provide a thorough discussion of the algorithms employed by TroFi together with an illuminating extended example.

Chapter 8 Core Experiments & Results – In this chapter, we look at the experiments performed to evaluate the basic unsupervised TroFi algorithm.

Chapter 9 Active Learning Experiments & Results – In this chapter, we discuss the experiments performed to evaluate TroFi's active learning component.

Chapter 10 Building the TroFi Example Base – In this chapter, we discuss the construction of the TroFi Example Base.

Chapter 11 Conclusion – In this chapter, we summarize the findings of the thesis and provide suggestions for future work.

Appendix A TroFi Pseudo-code – The appendix contains the pseudo-code for many of the TroFi algorithms.

2 A DEFINITION OF TERMS

TroFi is not a *metaphor* processing system. It does not claim to interpret *metonymy* and it will not tell you what a given *idiom* means. Well one may ask, then, what exactly does it do?

In essence, TroFi attempts to separate literal usages of words[1] from nonliteral ones. This is not as easy as it may sound. As part of TroFi's evaluation criteria, we had to manually annotate a collection of sentences as *literal* or *nonliteral*. It is *extremely* difficult. Everyone can probably remember high-school English: the teacher desperately trying to teach an often bored class how to recognize a metaphor. Those were the simple beginnings. Figurative language is actually much more complex than the basics we were taught in school. In short, it is difficult even for humans – never mind a machine – to clearly distinguish between literal and nonliteral usages. Certainly there are distinctions that are easy to make: "he was forced to eat his spinach" is obviously literal; "he was forced to eat his words" is obviously nonliteral. Another example is: "the sponge absorbed the water" (literal) vs. "the company absorbed the loss" (nonliteral). But what about "the black hole absorbed the light"? Some usages seem to sit on a sort of *figurative continuum*: they start out as nonliteral, but over time they become such an integral part of everyday speech that we begin to think of them as literal. For example, how should we classify "the final decision rests with the examining committee"?

Note that we are not trying to solve the problem of the literal/nonliteral continuum in this thesis. We are simply trying to see whether we can make a binary distinction between usages that seem more literal or standard and usages that seem more nonliteral or nonstandard. In doing so, we end up flushing all the different subtypes of nonliteral language into the same bucket.

[1] In this thesis we will focus on verbs only.

The rationale for this approach is twofold. First, many metaphor/metonymy/etc. processing systems to date have approached the problem from the bottom up – trying to figure out the low-level details and building systems up from there. This may be scientifically sound, but it does not scale. Second, it is worth questioning whether making fine-grained distinctions between types of nonliteral language is actually helpful at the automatic language processing level, particularly since such a detailed approach can quickly run into the *knowledge acquisition bottleneck* – i.e. having to annotate thousands of sentences manually and potentially not being able to find enough relevant examples of each type. Perhaps a simple distinction between language manageable by a regular NLP system and language requiring special treatment would be sufficient in many cases.

As we have suggested, distinguishing between literal and nonliteral *usages* is non-trivial. We will find that distinguishing between the literal and nonliteral *definitions* is not trivial either. Let us begin with a definition of *nonliteral*: "not literal; using figures of speech - figurative" (WordWeb Online, 2005). Next we have a definition of *literal*: "Conforming or limited to the simplest, nonfigurative, or most obvious meaning of a word or words." (American Heritage Dictionary of the English Language, 2005) One paragraph in, and already we are going around in circles. We will attempt to clarify matters in the following sections.

2.1 Literal

For the purposes of this thesis, we wish to define *nonliteral* as anything that deviates from the literal usage. To do so, we will need to define exactly what we mean by *literal* and what we mean by *deviate*. It turns out that this is actually quite difficult to do.

We have already provided a simple definition of *literal* from the American Heritage Dictionary of the English Language (2005). We provide a slightly expanded version here:

> literal: Being in accordance with, conforming to, or upholding the exact or primary meaning of a word or words.

> literal: Conforming or limited to the simplest, nonfigurative, or most obvious meaning of a word or words.

According to this, a literal meaning is the "primary" or "most obvious" meaning of a word. But what does that mean? Lexical semanticists would likely tie the definition of literal to the *selectional restrictions*, or, more loosely, *preferences*, of a word. The argument is that words *select for* certain types of arguments. For example, the word "drink" typically selects for an animate subject and a liquid object. Thus a car drinking gasoline (Fass, 1997), for example, would violate the selectional restrictions. However, "drink" in the sense of "consume" might well select for an inanimate object, which would mean that our gasoline-guzzling SUV is not actually violating a selectional restriction. It may be violating a *selectional preference*. The question then becomes, what makes one set of arguments preferable to another? This may have to do with any number of factors, including the history of the word, the closeness of a particular sense to physical reality, psychological motivations, the frequency of usage, etc. This opens up a huge field of argument that goes far beyond the scope of this thesis. Just as an example, though, we want to point out that frequency of usage is contradicted fairly quickly as a possible motivating factor when we observe that, at least in the Wall Street Journal, cash is absorbed far more readily than water.

Searching for definitions of *literal* in the metaphor processing literature, we find very little. The most illuminating that we do find is provided by Fass (1997, p. 26). He presents the following list of possible definitions:

> 1. *Conventional literality* in which ordinary conventional language is contrasted with poetic usage, exaggeration, irony, indirect speech acts, and so forth.
>
> 2. *Subject-matter literality* in which certain expressions are the ones ordinarily used to talk about a particular topic or domain.
>
> 3. *Nonmetaphorical literality*, or directly meaningful language, in which one word or concept is never understood by means of a second word (or concept), hence this precludes the use of metaphor and metonymy.
>
> 4. *Truth-conditional literality* in which language is used to refer to existing objects in the actual world and can be judged true or false.
>
> 5. *Context-free literality* in which the literal meaning of an expression is its meaning in a 'null' context.

For our purposes, we will regard the literal meaning of a given word to be the sense – together with its selectional restrictions – that appears to be closest to the above types of literality. We will define *nonliteral* usage primarily as a *deviation* from this literal sense. The notion of *deviation*, together with an expanded definition of nonliteral language, is explored in Section 2.2.

2.2 Nonliteral

The preceding discussion on literality may have conveyed the notion that deviation from literal usage to convey a nonliteral meaning is caused simply by violating the selectional restrictions of a word. As pointed out by Fass (1997) and Hahn & Markert (1999), selectional restriction violations on their own are insufficient to explain all occurrences of nonliteral language. For example, as Hahn and Markert point out, there is no selectional restriction violation in the sentence "I like Chaucer." Now let us look at the sentence in context: "I like Chaucer. The Canterbury Tales is one of my favourite books." We can see that the intended meaning of "Chaucer" – i.e. the works of Chaucer – is not contained in the generally accepted meaning of Chaucer, the man. Hahn and Markert (1999) present a "formal notion of deviance" based on such "categorization conflicts".

Another potential area of deviation is *assertions*, as discussed in (Fass, 1997). The idea is that some words carry special meanings that they can assert onto other words in the sentence, and that those assertions can be violated. For example, the verb "caress" connotes gentleness or tenderness. Consequently, the statement "he caressed his cat brutally" seems distinctly odd, even though there is no selectional restriction violation.

Fass also suggest *contextual inappropriateness* as a way to recognize nonliteral usage. He gives the example by Mark Johnson, that the phrase "all men are animals" may be interpreted literally in the context of a biology class and nonliterally in the context of a bad date.

Although we will not go into all the technical details of deviation, we will consider all the aforementioned types of deviation from the literal sense to be part of our definition of *nonliteral*. We will now look at some more surface-level definitions of nonliteral in order to gain an insight into specific figurative phenomena that our definition should cover.

WordWeb Online (2005) defines *nonliteral* as follows:

> Adjective: nonliteral
>
> (used of the meanings of words or text) not literal; using figures of speech
> - figurative
> See also: analogical, extended, metaphoric, metaphorical, metonymic, metonymical, poetic, rhetorical, synecdochic, synecdochical, tropical

We can get a slightly more detailed explanation if we dig down into the *tropical* part of the definition – not pineapples and bananas, but rather *tropes*.

From the Poetry Glossary (2005):

> The intentional use of a word or expression figuratively, i.e., used in a different sense from its original significance in order to give vividness or emphasis to an idea. Some important types of trope are: antonomasia, irony, metaphor, metonymy and synecdoche. Sidelight: Strictly speaking, a trope is the figurative use of a word or expression, while figure of speech refers to a phrase or sentence used in a figurative sense. The two terms, however, are often confused and used interchangeably. (See also Imagery)

From Wikipedia (2005):

> A **trope** is a rhetorical figure of speech that consists of a play on words, i.e. using a word in a way other than what is considered its literal or normal form. The other major category of figures of speech in the scheme, which involves changing the *pattern* of words in a sentence.
>
> Trope comes from the Greek word, *tropos*, which means a "turn", as in *heliotrope*, a flower which turns toward the sun. We can imagine a trope as a way of turning a word away from its normal meaning, or turning it into something else.
>
> A large number of tropes have been identified, among them:
>
> metonymy as in *association*.
>
> irony as in *contraries*.
>
> metaphor as in *comparatives*.
>
> synecdoche as in the distribution of the *whole* into the *part*.

From WordNet (2005):

> The *noun* trope has one meaning:
>
> Meaning #1: language used in a figurative or nonliteral sense
> Synonyms: figure of speech, figure, image

As we can see, the definition of *trope* is far reaching – especially the WordNet definition, which brings our definition of *nonliteral* full circle. In the same way, what we are expecting TroFi to recognize as nonliteral is far reaching – hence the name TroFi: *Tro*pe *Fi*nder. In fact, we extend our definition of *nonliteral* even more to include phrasal verbs, idioms, and other expressions where the meaning of the whole is not the sum of the parts. In summary, TroFi tries to distinguish between literal usages and usages that deviate from them. We now take a closer look at the forms those deviations may take.

2.2.1 Metaphor

One of the most popular targets for automatic processing is *metaphor*. The Columbia Electronic Encyclopedia, Sixth Edition, (2005) defines metaphor as:

> **metaphor** [Gr.,=transfer], in rhetoric, a figure of speech in which one class of things is referred to as if it belonged to another class. Whereas a simile states that A is like B, a metaphor states that A is B or substitutes B for A. Some metaphors are explicit, like Shakespeare's line from As You Like It: "All the world's a stage." A metaphor can also be implicit, as in Shakespeare's Sonnet LXXIII, where old age is indicated by a description of autumn:
>
> That time of year thou mayst in me behold
> Where yellow leaves, or none, or few, do hang
> Upon those boughs which shake against the cold,
> Bare ruined choirs, where once the sweet birds sang.
>
> A dead metaphor, such as "the arm" of a chair, is one that has become so common that it is no longer considered a metaphor.

For the purposes of this discussion, we will ignore extended metaphors like the Shakespeare sonnet above. We will, however, examine in more detail the metaphor "all the world's a stage." This is fairly easy to recognize as a metaphor and to analyze. The world is

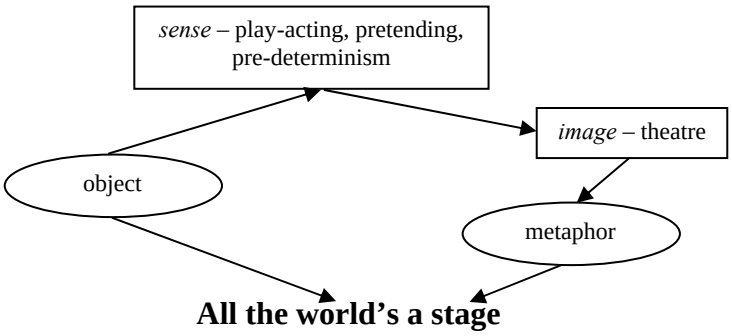
obviously *not* a stage, in the literal sense of the word, so we must be dealing with some sort of substitution or *domain transfer*.

Metaphors are founded on a similarity of qualities between two domains, a *source* domain and a *target* domain – here, the stage and the human condition. It must be possible to perceive the similarity, no matter how subtle, otherwise there is no basis for a metaphor. A framework for decomposing metaphors is provided by Peter Newmark (1981). According to (Newmark, 1981, p. 299), the components of a metaphor are:

> *object* – "the item that is described by the metaphor"
>
> *image* – "the item in terms of which the object is described" (sometimes also called the *vehicle*)
>
> *sense* – "[the point] which shows that the particular aspects of the object and the image are similar" (sometimes also called *tenor*)
>
> *metaphor* – "the words taken from the image"

Given these definitions, we can decompose Shakespeare's "all the world's a stage" metaphor as in Figure 2-A.

Figure 2-A Anatomy of "All the world's a stage"



Source: based on the concept by Newmark (1981)

One way for an automatic system to interpret metaphor would be to somehow figure out the source and target domains and find some reasonable connection – some shared sense –

between them. As with interlingua systems, this approach runs into very complex knowledge representation issues, which are as yet not scalable to large-scale use.

There are a number of different types of metaphors, one of which is those metaphors that start out as metaphors but then become accepted usage, like "the arm" of a chair. Newmark (1981, p. 299) lists five different kinds[2]:

1. dead (fossilized)
e.g. "the eye of a needle"; "they are transplanting the community"
2. cliché
e.g. "filthy lucre"; "they left me high and dry"; "we must leverage our assets"
3. standard (stock)
e.g. "plant a kiss"; "lose heart"; "drown one's sorrows"
4. recent
e.g. "kill a program"; "he was head-hunted"; "spaghetti code"
5. original (creative)
e.g. "A coil of cord, a colleen coy, a blush on a bush turned first men's laughter into wailful mother" (James Joyce); "The teacher mowed the child's essay" (Julia Birke)

Dead or *fossilized* metaphors are word usages that may have been novel and metaphorical historically, but which have now become so common that we see them as just another word sense. It seems to be a general trend with metaphors – especially if the metaphor relies on a novel usage of a single word – that if they are used consistently by a large part of the population over a long period of time, they slowly become part of the accepted usages of a word. Either that or they become *clichés*.

Clichés could be looked at as over-used metaphors. Sometimes they involve a metaphor based on a novel meaning of a single word, as described above, but usually they involve unique expressions or *catch phrases*. Such expressions do get absorbed into popular culture and become widely used, but they tend not to become fossilized metaphors for several reasons:

- they are often part of the slang of a given generation and thus receive lower status in the language

[2] Examples collected primarily by Julia Birke. Peter Newmark must not be blamed for them.

- they are generally whole expressions, not just single words
- they tend not to fill a *gap*[3] in the language

Very close in nature to *clichés* are *standard* or *stock* metaphors. *Idioms* belong to this category. *Standard* metaphors are often seen as expressions where the meaning of the whole is not equal to the sum of its parts.

Recent metaphors, as the name suggests, are metaphors that have just recently been accepted into popular usage. They often include current slang, as well as metaphors created to describe some new technology or phenomenon for which no adequate terminology existed previously. Naturally, much of the language surrounding computers falls into this category.

The last type of metaphor covers unique, never-before-heard expressions born of truly creative minds. These are the metaphors generally found in literature and marketing material. They are known as *original* or *creative* metaphors.

As mentioned earlier, different types of metaphors present different challenges to an automatic metaphor processing system and are dealt with – or ignored – accordingly. In the following paragraphs we will examine how each type has been treated by metaphor processing systems in the literature, and how it will be handled by the TroFi system. As discussed previously, we are trying to keep TroFi's handling of all nonliteral language, not just metaphors, as general as possible.

Dead metaphors are problematic because the line between dead and alive may be fuzzy. The judgment calls may be different from person to person and situation to situation, so it is challenging to get high accuracy on dead metaphors. Due to the difficulty of distinguishing between recent metaphors and dead ones, most metaphor processing systems end up dealing with

[3] By *gap* we mean a new concept in the language for which there is no word. For example, many of the computer terms we use, like "killing" a process, were borrowed from another domain to describe a hitherto unknown concept. Predictably, computer metaphors will one day become fossilized.

at least some fossilized metaphors. TroFi makes up its own mind about whether a metaphor is dead yet or not, categorizing it as literal or nonliteral based on available information.

Clichés and *idioms* can be treated as a single phenomenon by an automated system since in both cases we are generally looking at phrases, and often the meaning of the whole is not equal to the meaning of the parts. Most NLP systems, especially machine translation systems, deal with this problem by storing lists of idioms/clichés in lookup tables. In a dialogue system, appropriate responses for listed figurative phrases can thus be produced; in a translation system, the proper phrasal translation for each idiom/cliché can be found. TroFi applies the same methodology to idioms and clichés as to other nonliteral language. Since TroFi looks at the *usage* of the verbs, taking into account not only the arguments – which can often be ambiguous, as in "the horse kicked the bucket" – but also the remaining sentential context, idioms and clichés become just one more form of nonliteral language. (See also Section 2.2.4.)

Recent metaphors receive a fair amount of attention in metaphor processing. They are generally the most unambiguously classifiable as metaphors, and it is often easier to see in them the source and target domains which some metaphor processing systems depend on. Often they have already been manually recorded as examples in the *conceptual metaphor* collection initiated by Lakoff and Johnson (1980).

The term *conceptual metaphor* merits a little more explanation. The idea, as put forth by Lakoff and Johnson (1980) is that metaphors are not just a language phenomenon used to make poetry more interesting, but rather a reflection of the way our whole conceptual system works. Lakoff & Johnson suggest that we experience many aspects of life in terms of some other aspect of life, for example LOVE AS HEAT or ARGUMENT AS WAR. The metaphors we utter are then just a reflection of these internal thought patterns. This study has resulted in an ever-growing collection of conceptual metaphors, and numerous researchers (e.g. (Fass, 1997; Mason, 2004)) have made use of these collections.

Having the conceptual metaphor collections handy is beneficial of course, but even those metaphor processing systems preceding Lakoff, for example (Russell, 1976), use systems of analogy that are arguably conceptual in nature (Martin, 1990). Systems based on conceptual metaphor principles attempt to identify the source of the metaphor and *map* it to its target, generally through a series of semantic connections or an interpretive high-level representation. In other words, these systems are built around extensive metaphor maps (Martin, 1990, 1992; Fass, 1997) or interlingua (Russell, 1976).

TroFi does not make explicit use of the conceptual metaphor categories for its central algorithm, but it does use the example collections in the preprocessing of input data. Again, TroFi treats recent, as well as conceptual, metaphors as just another chunk of nonliteral language.

The class of *original* metaphors defies most metaphor processing systems. Although some fall into a particular conceptual metaphor category and can consequently treated in the regular way, some are more complicated. Furthermore, since they are *original* there are no convenient lists for looking them up. Some systems attempt to deal with original metaphors by examining the nature of the relationships between the words in a sentence (Fass, 1997). Again, TroFi processes *original* metaphors in the same way it processes all other types of metaphors, by looking at the context and trying to determine if there is anything *odd* about the usage of the verb.

2.2.2 Metonymy and Synecdoche

The second most popular figurative language type for automatic processing is *metonymy*. *Synecdoche* can be seen as a subset of *metonymy*. The Columbia Electronic Encyclopedia, Sixth Edition, (2005) defines *metonymy* as:

> metonymy (mĭtŏn'əmē) , figure of speech in which an attribute of a thing or something closely related to it is substituted for the thing itself. Thus, “sweat” can mean “hard labor,” and “Capitol Hill” represents the U.S. Congress.

Synecdoche is defined as:

> synecdoche (sĭnĕk'dəkē) , figure of speech, a species of metaphor, in which a part of a person or thing is used to designate the whole—thus, "The house was built by 40 hands" for "The house was built by 20 people." See metonymy.

As this definition states, *metonymy* refers to the use of one aspect of something to refer to the whole – for example, "England won the World Cup" means "[The team from] England won the World Cup" (Nissim & Markert, 2003, p. 56), and "All high-school students read Shakespeare" means "All high-school students read [plays by] Shakespeare" (example inspired by (Murata et al., 2000)).

Metonymy processing, like *metaphor* processing, tends to consist of first identifying the *metonymy* and then somehow mapping it to its literal reading. TroFi makes no explicit attempts to handle metonymy, but by depending on the input data available for a given word, TroFi may correctly identify a metonymic usage as nonliteral.

2.2.3 Irony

There are some things that are still a little beyond the grasp of computers, and *irony* is one of those things. The Columbia Electronic Encyclopedia, Sixth Edition, (2005) defines *irony* as:

> irony, figure of speech in which what is stated is not what is meant. The user of irony assumes that his reader or listener understands the concealed meaning of his statement. Perhaps the simplest form of irony is rhetorical irony, when, for effect, a speaker says the direct opposite of what she means. Thus, in Shakespeare's *Julius Caesar,* when Mark Antony refers in his funeral oration to Brutus and his fellow assassins as "honorable men" he is really saying that they are totally dishonorable and not to be trusted. Dramatic irony occurs in a play when the audience knows facts of which the characters in the play are ignorant. The most sustained example of dramatic irony is undoubtedly Sophocles' *Oedipus Rex,* in which Oedipus searches to find the murderer of the former king of Thebes, only to discover that it is himself, a fact the audience has known all along.

We can see from this definition that automatically processing irony – requiring, as it does, the correct recognition of underlying speaker intent – might be a little challenging. TroFi does not deal with irony.

2.2.4 Idioms and Phrasal Verbs

We saw in our definition of *metaphor* that *idioms* are often considered standard or stock metaphors. The definition from Wikipedia (2005) below also ties idioms in with conceptual metaphors:

> An **idiom** is an expression whose meaning is not compositional — that is, whose meaning does not follow from the meaning of the individual words of which it is composed. For example, the English phrase *to kick the bucket* means *to die*. A listener knowing the meaning of *kick* and *bucket* will not thereby be able to predict that the expression can mean *to die*. Idioms are often, though perhaps not universally, classified as figures of speech.
>
> …
>
> Idioms typically admit two different interpretations: a literal one and a nonliteral (or figurative) one. Continuing with the previous example, the phrase *to kick the bucket* can, in fact, refer to the act of giving a kick to a bucket, but this interpretation is usually not the intended one when a native speaker uses the phrase. This aspect of idioms can be frustrating for learners of a new language.
>
> Idioms are often colloquial metaphors. The most common ones can have deep roots, traceable across many languages. Many have translations in other languages, some of which are direct. For example, *get lost!* — which means *go away* or *stop bothering me* — is said to be a direct translation from an older Yiddish idiom.
>
> While many idioms are clearly based in conceptual metaphors such as "time as a substance", "time as a path", "love as war" or "up is more", the idioms themselves are often not particularly essential, even when the metaphors themselves are. For example "spend time", "battle of the sexes", and "back in the day" are idiomatic and based in essential metaphors, but one can communicate perfectly well without them. In forms like "profits are up", the metaphor is carried by "up" itself. The phrase "profits are up" is not itself an idiom. Practically anything measurable can be used in place of "profits": "crime is up", "satisfaction is up", "complaints are up" etc. Truly essential idioms generally involve prepositions, for example "out of" or "turn into".

The last few examples in this definition can also be called *phrasal verbs*. Wikipedia (2005) defines *phrasal verbs* as follows:

> In the English language, a **phrasal verb** is a verb combined with a preposition, an adverb, or an adverbial particle, all three of which are uninflected.
>
> A phrasal verb is also called verb-particle construction, verb phrase, multi-word verb, or compound verb. American English expressions are two-part verb or even three-part verb.
>
> ...
>
> Some grammarians claim that only the *figurative, idiomatic* or *metaphorical* usage of the combination should be called a phrasal verb, and that the *literal* use, where both the verb and the preposition are analysed, and both are found to have a *literal* meaning in a phrasal context, should be called *verb and particle* or *verb-particle constructions*.

> Other linguistic experts are of the opinion that all verb-particle constructions in both *literal*, as well a *figurative/idiomatic* use should be called phrasal verb, irrespectively whether they have an individual meaning or not.
>
> Emphasis in *idiomatic* phrasal verbs is put on the analysis to ascertain whether either verb or particle have a meaning. If neither component has a meaning of its own within the context of the sentence, it confirms the idiomaticalness of the whole and all that needs to be noted is whether the idiom is valid and recognised as such.

Because of the non-compositionality and the potential literal reading, idioms present a special challenge for automatic processing systems, partially because there are often no selectional restriction violations.

Since TroFi looks at the context beyond the immediate arguments, it can treat idioms exactly like any other nonliteral language. Phrasal and expression verbs are both a help and a hindrance to TroFi. By *expression verbs*, we mean expressions like "get the picture". As we will see later, phrasal/expression verbs are vital to the automatic preprocessing of the TroFi input data. Unfortunately, the preprocessor cannot tell the difference between truly idiomatic phrasal verbs and the literal verb-particle constructions discussed in the Wikipedia definition, leaving the door open for error. Furthermore, we claim that TroFi can be made to work in any language, but given the dependency of the algorithm on the existence of phrasal verbs, some adjustments would have to be made for languages containing no recognizable phrasal verbs.

2.2.5 Anomalous Semantic Relations

Fass (1997) discusses certain semantic relations classifiable as neither metaphoric nor metonymic. These are called *anomalous semantic relations*. He provides as an example the phrase "the idea drank the heart." He states: "Anomalous relations have neither the inferences of a metonymic relation nor the relevant analogy of a metaphorical relation." (Fass, 1997, p. 120)

Such anomalous relations can cause significant problems for metaphor processing systems, but since TroFi does not attempt to distinguish between different types of nonliterality, it is able to treat these cases like any other input.

2.3 TroFi's Choice

In Section 1.2, we saw an excerpt from the TroFi Example Base. Now that we are more familiar with the definitions of literal and nonliteral being assumed in this thesis, we look at another, this time for the word "drown":

> NONLITERAL: Realism might drown it.
>
> LITERAL: As Teresina, the maiden who is drowned in the Bay of Naples and ends up as a Nereid in the Blue Grotto before being restored to life by her faithful fisherman lover, Gennaro, Linda Hindberg was too inflexible in body and too stolid in personality.

Our knowledge of the lexical semantics of the word "drown", as well as our general world knowledge, allows us to see the literal/nonliteral distinction in these examples. TroFi attempts to make the same distinction by using unsupervised learning techniques. This is done with the understanding that there might be unclear cases which are problematic for TroFi and which deserve their proper attention within the theory of metaphor and metonymy processing. However, it should be possible for a system like TroFi to handle at least the clear-cut distinctions.

An interesting idea to consider is that the user of the TroFi system may be able to calibrate the literal/nonliteral distinction. We explain in Section 1.2 that TroFi works by attracting sentences of interest to either a literal sense set or a nonliteral sense set. Although these sets are automatically constructed (see Chapter 6) with as much adherence as possible to the definitions provided in this chapter, there will inevitably be some question as to the exact location of the literal/nonliteral divide. By being generated in an unsupervised manner, the literal/nonliteral sets will, in a way, influence where that boundary should lie. By then attracting other sentences to these sets, TroFi can help consolidate the literal/nonliteral distinction for a given target word: TroFi has an *opinion*. Of course, the user of the system may have a different opinion, and, through active learning, he/she has a chance to fine-tune the placement of the dividing line between literal/nonliteral as supported by TroFi. If, as suggested by Hahn and Markert (1999), the literal/nonliteral distinction is subjective, this is a valid thing to be able to do.

2.4 Summary

In this chapter we have provided the definitions of *literal* and *nonliteral* that will be assumed in the remainder of this thesis, and we have introduced the idea that TroFi's notion of *literal* and *nonliteral* can be calibrated. We have subsumed a great number of linguistic phenomena under the blanket term nonliteral, but ultimately it all seems to come back to our first definition: "not literal".

In the next chapter, we provide an overview of some of the nonliteral language processing literature relevant to this thesis. Most of the work reviewed concentrates on specific types of nonliteral language, particularly metaphor and metonymy.

3 METAPHOR & METONYMY PROCESSING REVIEW

The foundations of TroFi lie in a rich collection of metaphor and metonymy processing systems. Researchers have tried everything from hand-coded rule-based systems to statistical systems trained on large corpora. Metaphor processing has even been approached with connectionist systems storing world-knowledge as probabilistic dependencies.

Of the rule-based systems, some rely on a type of interlingua (Russell, 1976) to interpret metaphors, while others consist of complicated networks and hierarchies – often referred to as *metaphor maps* – that provide paths between the source and target concepts of a metaphor (e.g. (Fass, 1997; Martin, 1990, 1992)). These approaches are very effective when applied to certain classes of metaphors. Unfortunately, systems of this type have to be largely hand-coded and generally work only for an enumerable set of metaphors or in limited domains. Furthermore, like many rule-based NLP systems, these approaches tend not to be very efficient.

The other two types of systems we will look at in this chapter – dictionary-based systems and corpus-based systems – can be seen as a reaction to the problems encountered by the rule-based systems. Dictionary-based systems use existing machine-readable dictionaries or lexica built from a corpus as one of their primary sources for metaphor processing information. An example of such a system is presented in (Dolan, 1995). Dolan states that metaphor interpretation capabilities are an "emergent property" of extensive lexical knowledge bases (LKBs). Dolan claims that, by looking at paths and path lengths between words, one can "[allow] the lexicon itself to directly determine whether or not a particular meaning extension is licensed in a particular context" (Dolan, 1995, p. 27). Corpus-based systems may also make use of machine-readable dictionaries, but usually not directly for processing metaphor or metonymy. They extract or learn the necessary information from large corpora instead. By doing so, they attempt

to avoid the need for manual annotation or metaphor-map construction. Examples of such systems can be found in (Murata et al., 2000), (Nissim & Markert, 2003) and (Mason, 2004).

We examine our three types of systems in more detail below, followed by some conjectures on the future of metaphor/metonymy processing.

3.1 Rule-based Systems

The systems described in this section all depend on rules to interpret metaphors. That is, their interpretation components consist, in essence, of complicated systems of rules. Russell (1976) uses an interlingua to abstract from sentences and a matrix to find analogies between them; Fass (1997) finds detailed semantic relations between the words of a sentence and interprets metaphors by looking for common ancestors between source and target concepts in an abstraction hierarchy; Martin (1990, 1992) builds explicit representations of conventional conceptual metaphors into his system using extensive metaphor maps and uses these both to interpret metaphors and to learn new metaphorical uses.

Each of the three systems mentioned above deals slightly differently with metaphor recognition. Russell and Fass both make some use of selectional restriction violations, but Russell depends almost exclusively on verbal subcategorization and selectional preferences. Martin, on the other hand, minimizes the recognition phase in his quest to treat all input equally. He leaves the determination of appropriate usage to his interpretation component.

An additional system discussed in this section is the connectionist system of Narayanan (1999). It uses maps similar to those of the other three systems, but with an emphasis on domain-specific world-knowledge stored as probabilistic dependencies.

3.1.1 Russell

The system developed by Russell (1976) makes explicit use of an *interlingua*. She writes, "What is needed is an 'interlingua', which deals with relationships between concepts at the cognitive level." (Russell, 1976, p.9) Her interlingua depends on relationships between conceptual categories developed by Schank (1973) as part of his Conceptual Dependency Theory. In this theory, conceptual dependencies are represented by diagrams with a variety of symbols representing semantic roles, causal relationships, and so on. Russell applies this formalism to verbs. She then stores the representations in a matrix that can be used to map the relationships between the different conceptual categories to which the verbs belong. In other words, the matrix allows Russell to draw analogies between different usages of verbs, which she psychologically motivates as the method humans use to interpret unknown phrases.

It appears that recognition of metaphorically used verbs is achieved through examination of the "NOMINALs" found as the arguments of the verb. This is analogous to the use of selectional constraint violations for recognizing metaphors.

Russell uses the specialized diagrammatic code mentioned above to build her system. This means that every verb in the system must be painstakingly translated into symbols by hand. Then this symbolic description must be coded. The result is an exquisitely detailed characterization of the verb, but it falters on the fact that metaphors depend on more than the immediate arguments of the verb. Additionally, it was suggested by Martin (1992), that Russell's system has to work exceedingly hard to capture the relationships that the conceptual metaphors of Johnson and Lakoff (1980) are able to capture by their very nature.

3.1.2 Fass

Fass (1997) makes use of selectional preferences, as well as *assertions*, in his system of Collative Semantics (CS). He uses them as one part of his metaphor and metonymy processing

system. Fass looks at the relationships between each pair of arguments in a sentence, evaluating each as one of seven semantic relations based on adherence to, and violations of, preferences and assertions. The seven types of semantic relations defined by Fass are *literal*, *metaphorical*, *metonymic*, *anomalous*, *redundant*, *inconsistent* and *novel*. It is the combination of these semantic relations in a sentence that determines whether the sentence as a whole should be seen as literal or metaphorical.

Particularly interesting for this thesis is that Fass finds the aforementioned semantic relations between words using the different senses of the words themselves and consequently disambiguates those same words using the semantic relations. This supports the close relationship between metaphor processing and word-sense disambiguation assumed by TroFi.

For its metaphor interpretation component, Fass's meta5 system uses abstraction hierarchies to draw analogies between source and target concepts. The literal meanings of metaphors can be discovered by looking for common ancestors between the source and target concepts in the hierarchy. In finding these mappings Fass makes extensive reference to the conceptual metaphor constructs of Johnson and Lakoff (1980).

3.1.3 Martin

The commonality between the above systems is that they contain little explicit knowledge about known metaphors. Fass's system does contain extensive implicit knowledge about how figurative language in general, and conceptual metaphor in particular, works, but it seems to makes no explicit use of existing metaphors. This is where Martin's MIDAS (Metaphor Interpretation, Denotation, and Acquisition System) (Martin 1990, 1992) steps in.

MIDAS, which was developed for use in an expert system for UNIX users, contains hard-coded representations of conventional conceptual metaphors, shown as relationships between concepts. It encodes the relationships in highly detailed metaphor maps and metaphor map

hierarchies. It relies on these representations to interpret input and provide appropriate output. Furthermore, it sets itself apart by being able to learn new usages from the input by analogy to existing metaphors. In addition to conventional metaphors, MIDAS contains representations for handling literal input, including hard-coded expected semantic relations for a given word.

Part of what distinguishes Martin's system from other metaphor processing systems is that it is not purely a metaphor processing system. One of its strengths is that no explicit metaphor recognition phase is required. All input is treated equally. Of course, within the system, all possible interpretations, both literal and metaphorical, must be stepped through in order to interpret an input sentence. This unfortunately requires extensive coding for each possible word or expression, making it difficult to expand the system beyond a specific domain.

Besides the use of existing metaphorical knowledge, the work by Martin most relevant to TroFi is his research into the use of context to recognize and interpret metaphor (Martin, 1994). In this work, Martin examines psychological studies which have shown that the sentences preceding a metaphor may be indicative of either the source or target domains of the metaphor, or may in fact represent another, similar metaphor. The strongest correlation was found with indicators of the target domain in the preceding text. One of TroFi's most successful models depends on this finding.

3.1.4 Narayanan

Narayanan (1999) and other researchers at the Neural Theory of Language (NTL) Research Group at Berkley also make use of Lakoff and Johnson's conceptual metaphor construct, and they too make extensive use of metaphor maps. However, they place a far greater emphasis on domain-specific world knowledge extracted from a corpus. This detailed world knowledge is stored as probabilistic dependencies between variables in a *Belief Net* representing the target domain. The source domain is stored as an "x-schema simulation environment used for

inference" (Narayanan, 1999, p. 123). In addition, there are metaphor maps to map (*project*) source domain information to the target domain Belief Nets. As described in (Narayanan, 1999, p. 121), the system interprets metaphor by generating inferences based on pre-parsed input and is limited to "simple causal narratives in the domains of Politics and Economics."

The system is able to work with fine semantic details like temporal distinctions and speaker intent. Also, it is able to interpret novel expressions of the *original metaphor* variety within the domain programmed into the system. Unfortunately, although many of the rules found in the system can be automatically created using probabilistic analysis of a corpus, the various problems inherent in rule-based systems still prevail: all the Belief Nets, x-schemas, and metaphor maps must be preprogrammed, creating an enormous system that is difficult to expand to other domains and even to other conceptual metaphor types.

Narayanan states: "It is now generally accepted that metaphor interpretation requires the ability to explicitly represent the source and target domains as well as the metaphor maps themselves." (Narayanan, 1999, p. 128) Maybe so, but given the extraordinary computational complexity and the scalability problems of such systems, is it any wonder that some researchers have attempted to break out of this mould to create lighter, more generally applicable systems?

3.2 Dictionary-based Systems

The common thread running through all the rule-based systems outlined above is that they do an extraordinary – and theoretically well-motivated – job on a select subset of English sentences. It is this restriction to a limited domain that prompted Dolan to write: "Previous computational attempts to handle nonliteral word usage have been restricted to 'toy' systems that combine hand-coded lexicons with restricted sets of metaphor types that can be used to sanction specific classes of semantic subcategorization violations. These hand-coded efforts are unlikely to ever scale up to the rigors or real, free text." (Dolan, 1995)

One possible solution is to make more refined *metaphorical* use of the lexicon used by the language processor. Two such methods are described below: Dolan's example-based approach, which uses "a large lexical knowledge base derived from a machine-readable dictionary" (Dolan, 1995), and Zernik & Dyer's phrasal lexicon approach (Zernik & Dyer, 1986).

3.2.1 Dolan

The system developed by Dolan (1995) does away with the need for an intricate, hand-coded metaphor interpretation component like the ones used by the rule-based systems described in Section 3.1. Instead, he automatically derives an extensive lexical knowledge base (LKB) from the Longman Dictionary of Contemporary English (LDOCE). This LKB is a type of semantic hierarchy, much like WordNet, which provides information about semantic relations like hypernymy and holonymy. It also provides selectional preference information in the form of *TypicalObjOf* and *TypicalSubjOf*. Additionally it provides *ranked paths* between words, which allow particular relationships (including metaphorical ones) to be determined.

The beauty of this system is that it does not depend on complex mappings between source and target concepts, and that it is a literal lexicon and metaphor recognition/interpretation system all rolled into one. When input is received, for example a verb and its object, the system attempts to interpret the verb literally using selectional preferences encoded in the LKB. If this proves impossible – i.e. if there is a selectional constraint violation – the system tries to establish a metaphorical relationship between the verb and its object by finding a highly ranked path through the hierarchy between the input object and the object typically found with the input verb. Dolan explains that sometimes, if no direct path can be found, there may still be secondary paths based on path intersection points.

The benefits of Dolan's approach are obvious: the recognition/interpretation component is created largely automatically, and there is no need to code up expensive metaphor maps.

However, the system is limited to finding relationships based on selectional restrictions and semantic relationships between individual words. In this sense it faces the same problem as other systems based on semantic hierarchies: what if there is no direct – or even indirect – semantic relationship to be found using just the arguments of a given word?

3.2.2 Zernik & Dyer

Like Dolan, Zernik & Dyer (1986) deal with metaphorical language in the same way that they deal with literal language, and they use a type of automatically built lexicon to do it. Unlike Dolan, however, they use a phrasal lexicon built using information extracted from a corpus rather than a semantic hierarchy built using information extracted from a machine-readable dictionary.

In the Zernik & Dyer system, metaphor recognition becomes irrelevant, since all possible usages of a word are listed in their phrasal context in the lexicon. Each lexical entry lists the *pattern* to be matched, including required arguments, the *situation* in which one would use this phrase – i.e. the context – and the *concept*, namely the literal meaning of the phrase. If the input phrase happens to be metaphorical, the arguments need simply to be transferred to the literal *concept*. In effect, it functions very much like an example-based machine translation system.

A drawback of this system is that all those lexical entries need to be created, albeit automatically, and stored. One could imagine that storing all those phrases as well as all the additional information would be prohibitively space-consuming. Also, to use the system, all phrases must be matched against the text and the situation evaluated. It further appears that a pre-defined set of situations, such as "$vehicle-collision" or "$fortuitous-encounter" (Zernik & Dyer, 1986, p. 249), must be created in order to be linked to particular phrases. And although the system can handle and learn unknown phrases, it must converse with the user to do so.

The phrasal lexicon layout of Zernik & Dyer's RINA system probably comes closest to how we might build an interpretation system on the TroFi Example Base. We would also want to

create a system where certain uses of a word could be looked up and converted into a literal interpretation if necessary. Like Zernik & Dyer, we populate our database automatically. The major differences, however, are that we do not start with a seed lexicon, we do not need to pre-code a set of situations, and we attempt to keep user interaction optional and to a minimum. Also, rather than building one huge lexicon for the entire English language, TroFi can be tuned to build domain-specific example bases given user-specified corpora and target words.

3.3 Corpus-based Systems

Three of the systems we have seen thus far, two rule-based and one dictionary-based, make use of corpus-based techniques in some capacity. Martin combines context from the corpus with known metaphors to learn previously unseen metaphors for addition to his metaphor maps. Narayanan extracts domain-specific world knowledge from a corpus. Zernik & Dyer make extensive use of the corpus to discern the situations in which a given phrase is typically used. All three of these systems use the corpus to learn, and then convert what they have learned, into rules that must be stored.

Up to this point, metaphor processing systems that make use of the corpus in the statistical sense of corpus-based NLP have been hard to find, probably since something as esoteric and hard-to-define as metaphor seems unlikely to be characterizable using statistical distributions. Still, attempts are beginning to be made to use corpus-based linguistics for processing nonliteral language. A number of approaches have limited themselves to attempting to devise systems for manually annotating nonliteral language (e.g. (Markert & Nissim, 2002; Semino & Steen, 2001)). Most of the recent attempts to apply machine learning methods to the problem have dealt exclusively with metonymy (Nissim & Markert, 2003; Murata et al., 2000), although we did find one project that attempts to discover conceptual metaphors in a collection of corpora by statistical means (Mason, 2004).

We first examine the two approaches to metonymy interpretation – (Nissim & Markert, 2003) and (Murata et al., 2000) – since the methods employed there are similar to the ones used by TroFi for interpreting the broader class of nonliteral language. We then review the work of (Mason, 2004), which explores the discovery of conceptual metaphors in a corpus. The work on supervised metonymy resolution by Nissim & Markert and the work on conceptual metaphors by Mason come closest, to date, to what we are trying to accomplish with TroFi.

3.3.1 Murata et al.

Murata et al. (2000) use example bases derived from corpora to find interpretations for various types of metonymy. In their work on example-based metonymy interpretation in Japanese, Murata et al. attempt to eliminate the necessity for pre-coded metonymy knowledge bases and semantic hierarchies by combining different types of phrases extracted from a corpus. As described in Section 2.2.2, metonymy is a trope where a word associated with a particular entity is made to stand for that entity. The example given in (Murata et al., 2000) (in translation) is “I read Tolstoy”, where “Tolstoy” actually stands for “Tolstoy’s novel” or “the novel of Tolstoy”.

The method of Murata et al. is based on the assumption that a given corpus will contain both examples of the metonymy and examples relating the metonymic word to the entity it stands for. So, in order for the system to interpret “I read Tolstoy” as “I read the novel of Tolstoy”, the corpus must contain numerous examples of “Y of Tolstoy”. From among all the “Y of Tolstoy” examples, the most likely one is chosen based on selectional restrictions of the verb.

It is important to mention this study in relation to TroFi for two reasons. The first is that it shows a method for using only examples extracted from a corpus to interpret a type of nonliteral language. The second is the reason that Murata et al. provide for not extending their method to other types of metaphor: “Metaphor is affected by the context, so metaphor

interpretation is difficult." (Murata et al., 2000) This lends support to our belief that it is important to look beyond the subcategorization frame and selectional restrictions of the verb for metaphor processing.

3.3.2 Nissim & Markert

Nissim & Markert (2003) approach metonymy resolution with machine learning methods, "which [exploit] the similarity between examples of conventional metonymy" (Nissim & Markert, 2003, p. 56). They see metonymy resolution as a classification problem between the literal use of a word and a number of pre-defined metonymy types[1]. They use similarities between *possibly metonymic words* (*PMWs*) and known metonymies as well as context similarities to classify the PMWs.

The main difference between the Nissim & Markert algorithm and the TroFi algorithm – besides the fact that Nissim & Markert deal only with specific types of metonymy and not nonliteral language in general – is that Nissim & Markert use a supervised machine learning algorithm, as opposed to the primarily unsupervised algorithm used by TroFi. Nissim & Markert use a hand-annotated corpus (1000 examples from the BNC) and a decision list classifier for their task. They find that they are able to reduce the context to head-modifier relations and still get comparable results. However, they run into the same problem encountered by TroFi: data-sparseness. Sometimes there is no instance of a particular role-of-head value, for example "subj-of-lose" (Nissim & Markert, 2003), in the training data and so no inferences can be drawn. Nissim & Markert need to be able to extend similarity to other similar heads.

Nissim & Markert discuss using the Karov and Edelman (1998) word-sense disambiguation algorithm[2] to extend similarity to other lexical heads, but they decide to go with a simpler approach involving the integration of a thesaurus. This works well for their particular

[1] The metonymy types used are *place-for-people*, *place-for-event*, *place-for-product*, *mixed*, and *othermet*.
[2] Coincidentally, this is the same algorithm which was chosen as a foundation for TroFi independently of the Nissim & Markert paper.

task since they need only to find other words similar to the main verb. This is not enough for TroFi, since TroFi depends on more than just the arguments of the target verb and using the synonyms of all the words in the context could give us problematic and incorrect similarities.

3.3.3 Mason

Mason (2004) presents work in metaphor processing based on his Ph.D. thesis. He describes CorMet, "a corpus-based system for discovering metaphorical mappings between concepts" (Mason, 2004, p. 23). His system finds the selectional restrictions of given verbs in particular domains by statistical means. It then finds metaphorical mappings between domains based on these selectional preferences. By finding semantic differences between the selectional preferences, it can "articulate the higher-order structure of conceptual metaphors" (Mason, 2004, p. 24), finding mappings like *LIQUID→MONEY*.

Mason's work bears many similarities to TroFi[3]. Its creation is based on similar motivations – i.e. to take a first step towards building a "robust, broadly applicable computational metaphor interpretation system" (Mason, 2004, p. 23). Like TroFi, Mason's CorMet uses WordNet as a primary knowledge source, but unlike TroFi, it actually mines WordNet for selectional preference information. Further it makes use of context in large corpora, although it appears that the domains to be analyzed by CorMet must be pre-defined.

One of the primary differences between TroFi and CorMet is what they are expected to accomplish. CorMet concentrates specifically on conventional conceptual metaphors, attempting to automatically extract these from a corpus and provide them with appropriate labels. It is not really built as a system for recognizing metaphors in the sense of distinguishing metaphorical usages of verbs from literal ones. Mason himself states, "Note that CorMet is designed to detect higher-order conceptual metaphors by finding *some* of the sentences embodying *some* of the

[3] Note that we only became aware of this work well after TroFi had been implemented, so it had no influence on our current functionality.

interconcept mappings constituting the metaphor of interest but is not designed to be a tool for reliably detecting all instances of a particular metaphor." (Mason, 2004, p. 24) TroFi, on the other hand, cannot tell a conceptual metaphor from an idiom, never mind give it a label, but it can recognize all sorts of nonliteral language without even knowing anything about the domain.

3.4 Metaphor Processing Future

We have now walked through the past and present of metaphor/metonymy processing, but what is the future? We have discovered some common threads running through the various methodologies. Most early systems (e.g. (Russell, 1976; Fass, 1997; Martin, 1990)) use large, hand-coded interpretation components. However, other efforts have attempted to eliminate these as they are difficult to maintain and scale to real-world text in disparate domains. They have chosen to rely instead on example-based systems, some (e.g. (Dolan, 1995)) extracting required information out of machine-readable dictionaries and thesauri, others (e.g. (Narayanan, 1999; Murata et al., 2000; Nissim & Markert, 2003; Mason, 2004)) attempting to learn from a corpus. Like current developments in machine translation, example-based systems learning from a corpus with the support of a pre-existing machine-readable dictionary or semantic hierarchy like WordNet appear to be the way of the future.

To date, most research still appears to stick with a particular type of nonliteral language, like metonymy and metaphor – particularly the well-defined and psychologically motivated categories of Lakoff's conceptual metaphor. TroFi appears to be the first system to attempt to define and process the broad, sweeping category of *nonliteral* in a single system.

Another apparent trend is to take a step back from the ambitious task of *interpreting* metonymy and metaphor and to look instead at some new approaches for simply *recognizing* them in some useful way, perhaps working with them in an example-based form later. This is evidenced particularly in (Mason, 2004) and (Nissim & Markert, 2003).

We have already mentioned the similarity of recent developments in metaphor/metonymy processing to developments in machine translation. If we look back to Russell's (1976) use of an interlingua in her metaphor processing system, we can see that the similarities between the two research areas have always been evident. Another area sharing similarities with metaphor/metonymy processing is word-sense disambiguation. For example, both fields have made extensive use of subcategorization frames, selectional restrictions, and paths in semantic hierarchies. It seems logical therefore that the future of metaphor/metonymy processing may be tied not only to advances in machine translation, but also to the state-of-the-art in word-sense disambiguation. Not surprisingly, many of the trends in that field point towards automatically learning from large corpora.

We further explore the similarities between word-sense disambiguation and metaphor/metonymy processing in Chapter 4 through an examination of some word-sense disambiguation approaches that may be adaptable to nonliteral language processing. In addition, we touch briefly on clustering methodologies as related to word-sense disambiguation and nonliteral language processing.

4 WORD-SENSE DISAMBIGUATION & CLUSTERING REVIEW

In Section 3.4, we suggested that nonliteral language processing shares many commonalities with word-sense disambiguation – for example, subcategorization frames, selectional restrictions, and paths in semantic hierarchies. Based on these similarities, we suggest that if we simply regard *literal* and *nonliteral* as two senses of a word, we can reduce the problem of nonliteral language recognition to one of word-sense disambiguation. This means that we should be able to adapt an existing word-sense disambiguation algorithm to our needs.

In Section 4.1, we browse through a number of word-sense disambiguation algorithms in order to find the one most suitable for use in TroFi. Also, since our desired TroFi output is literal/nonliteral clusters, we also take a very brief look at clustering in Section 4.2.

4.1 Word-sense Disambiguation Methodologies

Since word-sense disambiguation is one of the greatest thorns in the side of Natural Language Processing, it has naturally received a great deal of attention in the literature, resulting in a good choice of algorithms for adapting to nonliteral language processing. We discuss a number of approaches in the following sections: a sense disambiguation algorithm based on selectional preference (Resnik, 1997), an unsupervised bootstrapping algorithm for word-sense disambiguation (Yarowsky, 1995), and a similarity-based word-sense disambiguation algorithm (Karov & Edelman, 1998). In addition, we briefly glance at some work on the effects of context on disambiguation (Beeferman et al., 1997). In all cases, the discussion will focus not so much on the details of the work itself but rather on its applicability to TroFi.

4.1.1 Resnik

Resnik (1997) builds his sense disambiguation algorithm on the basis of selectional preference. Recall that selectional preference tends to play a role in nonliteral language processing systems as well. Resnik uses the relative entropy model of (Kullback & Leibler, 1951) to establish *selectional preference strength* of a predicate. This methodology poses two problems for the TroFi approach.

The first difficulty is that once the selectional preferences are found, some method must be deployed to separate the literal ones from the nonliteral ones – either that, or separate them manually. It is worth noting that (Mason, 2004) uses the Resnik algorithm. However, the purpose of Mason's work is to find high-level conceptual metaphors by looking at the differences in selectional restrictions across domains. He does not attempt to separate literal and nonliteral usages as such.

The second difficulty lies in the fact that, by Resnik's own admission, "…although selectional preferences are widely viewed as an important factor in disambiguation, their practical broad-coverage application appears limited…," and "more important is information beyond selectional preference, notably the wider context utilized by Yarowsky." (Resnik, 1997)

4.1.2 Yarowsky

The Yarowsky (1995) algorithm uses an unsupervised bootstrapping method to train itself for word-sense disambiguation tasks. The basic idea is to start with a set of seed collocations for each sense of a given word. All the sentences in the training set can then be tagged according to this basic information. The next step involves training a supervised decision list learner on the tagged set, allowing additional collocations to be learned. The corpus is then retagged with this new information. Any tagged sentences are added to the training set. At this point Yarowsky adds a little twist. He identifies a *one sense per discourse* constraint. This

allows any other sentences containing the target word that are in the same discourse as a previously tagged sentence to be added to the training set. The supervised learner is then retrained on the augmented set, and so on.

Although this algorithm works extremely well for regular word-sense disambiguation, it has a few problems as an algorithm for nonliteral language processing. First, it is very difficult to come up with clearly defined collocation seed sets for the literal/nonliteral distinction. Second, the algorithm requires that additional features be extracted from training examples, but given that nonliteral usages are often unique, this might prove futile. Last, since speakers often mix literal and nonliteral uses of a word within the same discourse, it is hard to place complete faith in the *one sense per discourse constraint*.

4.1.3 Karov & Edelman

A word-sense disambiguation algorithm that seems far more adaptable to the needs of TroFi is the similarity-based word-sense disambiguation algorithm of Karov and Edelman (1998). It requires as input only the sentences containing the target word, some definition of the relevant context (composition of the feature sets), and machine-readable dictionary definitions showing the different senses of the target word. An important point is that the algorithm is extremely flexible in terms of its sense sets. Anything for which one can create a set – for example, *literal* and *nonliteral* – can essentially be regarded as a *sense*. Furthermore, it has the benefit of being able to work with very sparse data using the principle of *transitive similarity*[1]. This is pivotal, since researchers admit that there is a terrible *knowledge acquisition bottleneck* (see Chapter 2) not only in word-sense disambiguation, but also in nonliteral language processing (see (Karov & Edelman, 1998; Nissim & Markert, 2003)). The Karov & Edelman approach appears to lend itself best of the algorithms reviewed to the task of nonliteral language processing. The algorithm, as well as its integration into TroFi, is discussed extensively in Chapter 7.

[1] It is interesting that this same principle emerges in (Dolan, 1995) in the discussion of *secondary paths*.

4.1.4 Beeferman et al.

Although Beeferman et.al.'s *Model of Lexical Attraction and Repulsion* (1997) does not focus specifically on word-sense disambiguation, it does discuss a phenomenon relevant to disambiguation experiments involving a wider context. Their finding is that the predictive power of *context words* decreases relative to their distance from the target word. This means that algorithms depending on this context could be improved by somehow ascribing less relevance to more distant words.

This phenomenon could be modeled in TroFi by manipulating the feature weights (see Section 7.1.1) – for example, adjacent words would be given the highest weight, followed by other words in the same sentence, followed by words in adjacent sentences, and so on. Although this is a possibility, we consider it outside the scope of this thesis and leave it for future work.

4.2 Clustering Methodologies

Looking closely at the description of the Karov & Edelman algorithm in Section 4.1.3, we notice some similarities to clustering. In essence, the sense sets are like seed sets around which we can cluster other sentences, hence our claim that TroFi does literal/nonliteral *clustering*. This opens up the question of whether some other clustering algorithm might not be better suited to the task. In this section we provide an extremely brief overview of clustering and then examine a state-of-the-art clustering algorithm and its applicability to TroFi.

Speaking generally, clustering algorithms group words based on the context in which they occur. This context can consist of something as simple as the preceding word. A count is done on a corpus to determine which words tend to occur with the same *feature* (in this case the preceding word). These are clustered together. Something to keep in mind for the TroFi case is that we want to cluster whole phrases, not individual words, and that we want to use extensive feature sets, the content of which we can, to some extent, control.

The extent to which the content of the features sets can be controlled depends on the type of clustering algorithm chosen. There are two main categories: hierarchical and non-hierarchical. Hierarchical clustering algorithms produce just that – a hierarchy of clusters. Given such a hierarchy, decisions must be made about where the branches should be cut in order to create the most useful clusters. This would be extremely difficult in the case of TroFi, since we approach the clustering problem with no clear idea of where such divisions should fall. Furthermore this would require extensive supervision, which we wish to avoid. Non-hierarchical clustering, on the other hand, requires the construction of *seed clusters* to which other words/phrases can be attracted. In this sense, the Karov & Edelman algorithm (Karov & Edelman, 1998) uses a sort of non-hierarchical clustering technique. However, as opposed to other systems, which generally require seed clusters to be built by hand, the Karov & Edelman algorithm includes a method for building them automatically.

The Karov & Edelman algorithm, with its ability to cluster whole phrases around automatically constructed seed clusters, seems fairly ideal for our nonliteral language recognition purposes. However, for the sake of balance, we do review a pure clustering algorithm below.

4.2.1 Lee & Pereira

Lee and Pereira (1999) use distributional similarity models to build clusters meant to predict unseen events. For example, their algorithm forms clusters of nouns on the basis of the verbs with which they are generally found using complex conditional probability calculations and iterative re-estimation procedures. The defining feature of this clustering algorithm is that it produces greater amounts of generalization than other algorithms, making it attractive for tasks crippled by data sparseness.

In effect, this would make it a good fit for TroFi, except for the fact that TroFi has to be able to cluster whole phrases and use extensive feature lists to do so. The Lee & Pereira

algorithm requires not only the calculation of word and bigram frequencies in a training corpus, it also requires the subsequent calculation of co-occurrence probabilities of word clusters (Dagan, Pereira & Lee, 1994). Furthermore, since this similarity-based measure – like *k-means* and other traditional approaches – clusters only words, not words plus context, it is difficult to provide seed sets of sentences to use as a bootstrap set to jump-start the learning process.

Another problem with using this algorithm for literal/nonliteral clustering is that it is hierarchical and requires some division of the clusters into usable sense sets, either manually or by training on annotated data. That this is a problem is evidenced in (Dagan, Lee & Pereira, 1997) in which (Dagan, Pereira & Lee, 1994)[2] is applied to word-sense disambiguation. In this paper, the authors shy away from performing an experiment on real-world sense-distinctions due to the need for manually annotated training sets.

The clustering algorithm of Lee & Pereira (1999) is the state-of-the-art for being able to handle unseen examples through similarity-based reasoning. However, the Karov & Edelman (1998) algorithm also allows for generalization to unseen examples, and, on the level of being able to handle a wide variety of contexts and not requiring manually annotated training data, the Karov & Edelman algorithm is certainly preferable for our purposes.

4.3 Summary

In this chapter we provided a brief overview of word-sense similarity algorithms and a cursory look at clustering methodologies. We concluded that the Karov & Edelman algorithm (Karov & Edelman, 1998) is ultimately most suited to our nonliteral language recognition task.

We examine the Karov & Edelman algorithm rigorously in Section 7.1.1.1, analyzing how it fits into the TroFi system in Section 7.1.1.2. However, before diving into the details, we provide a high-level overview of the whole TroFi system in Chapter 5.

[2] An early version of the algorithm presented in (Lee & Pereira, 1999).

5 TROFI OVERVIEW

In this chapter, we provide a high-level overview of TroFi. We discuss the details in subsequent chapters: data in Chapter 6; algorithms in Chapter 7; results in Chapters 8, 9 & 10.

The view we are adopting for the purposes of this exploration is that we can take an amorphous collection of sentences unified only by a shared verb and cause a meaningful split of this collection into two sets – one where the shared verb is used in a *literal* sense, and another where it is used in a *nonliteral* sense, as defined in Chapter 2.

We could approach the splitting of the original collection in several ways: do it manually; train a classifier; write a large number of rules; cluster based on individual words; collect some seed sets and attract the original sentences to them.

It is this last approach that we adopt for TroFi. Possibly it is not the most precise of the approaches, but it offers the benefits of being extremely flexible and scalable, and it requires comparatively little labour to set up and maintain. We discuss the reasons for our choice in more detail below.

Our first option for splitting the original collection is to do it manually; the second, to train a classifier. Both of these options require manually annotating enormous amounts of data. Manual annotation is an extremely time-consuming and error-prone process. It is particularly difficult to make consistent decisions on something as fuzzy as the distinction between literal and nonliteral. In addition, because language in general – and nonliteral language in particular – is productive, the annotation task would never be done. Finally, every new language of interest would require its own annotation initiative. As we walk through the TroFi process later in this chapter, several opportunities for manual annotation will arise – for example, manually separating

the sense sets of a machine-readable dictionary (MRD). Even in those cases, TroFi does not use manual input for all the reasons just described.

The third option is to write "a large number of rules" – i.e. create a rule-based system. We are including under this heading any system requiring the use of extensive metaphor maps, distances between nodes in semantic hierarchies, and explicitly coded selectional constraints. Like manual annotation initiatives, such systems require a great deal of manual effort to code the rules and link them together. They are expensive to build and maintain and are often limited in size and scope. Although TroFi makes implicit use of various constraint violations and uses the data from a semantic hierarchy (WordNet) as input, it does not make use of any explicit rules. How this works will become clear in Chapters 6 and 7.

Clustering on single words is not workable because nonliteral language tends not to exist as single words. We must be able to cluster whole phrases. And we must have some idea of what kinds of clusters we are trying to build in the first place.

That leaves us with clustering based on attraction to an existing collection of sentences, the approach chosen for TroFi. We automatically gather a set of literal sentences and a set of nonliteral sentences and split our original collection through attraction to these two sets.

A fundamental assumption is that *literal* and *nonliteral* can be regarded simply as two senses of a given word. This allows us to reduce the difficult problem of literal/nonliteral language recognition to one that is much closer to being solved, namely that of word-sense disambiguation. Seeing literal and nonliteral as word senses is supported in the literature by the classification approach to metonymy resolution pioneered by Nissim and Markert (2003) discussed in Section 3.3.2. They also use *literal* as one of their classes, but they take the task one step further by subdividing the *metonymy* class into a number of known metonymy types.

We must state up front that TroFi currently deals only with verbs, although the algorithm can easily be extended to other parts of speech. Limiting the implementation to verbs follows the example set by many metaphor interpretation systems detailed in the literature, especially those that rely on selectional restriction violations to indicate the presence of a metaphor. The core algorithm employed by TroFi allows for extremely flexible features sets, and there is nothing that prevents the same set of features from being used when the target word is not a verb, but rather a noun or an adjective. Being able to easily extend the system to nouns and adjectives is important for cases where there is nothing nonliteral about the use of the verb at all – for example, "You're the cat's meow," or "He has a luminous disposition." There is nothing nonliteral about *are* and *have* in these sentences, and trying to recognize them as such would prove futile. However, these sentences are undeniably nonliteral, and a literal/nonliteral clustering system ought therefore to be able to handle them. TroFi could be made to accept nouns and adjectives as target words with only minor adjustments. No changes to the core algorithms would be required.

To re-iterate, we are attempting to reduce the problem of literal/nonliteral recognition to one of word-sense disambiguation. For this reason, TroFi is built on an existing similarity-based word-sense disambiguation algorithm developed by Yael Karov and Shimon Edelman (1998). This algorithm is discussed in Section 7.1.1.1.

The Karov & Edelman algorithm is completely unsupervised and is based on the principle of attraction. Similarities are calculated between sentences containing the word we wish to disambiguate (the *target word*) and collections of sentences called *feedback sets*. In the case of word-sense disambiguation proper, these feedback sets are built around each sense of a word in a machine-readable dictionary or thesaurus, in our case, WordNet. Synonyms of a target word are used as seeds, and sentences containing these seeds are collected from a corpus. For TroFi, we additionally use information from the WordNet definitions and example sentences. A sentence

from the original set is considered to be attracted to the feedback set containing the sentence to which it shows the highest similarity.

In order to make the Karov & Edelman algorithm work as a foundation for literal/nonliteral clustering, a few changes must be made. Most important is the composition of the feedback sets. Using the individual senses from WordNet is insufficient. Since TroFi is an unsupervised algorithm, we have no way of knowing which of these senses might be literal and which nonliteral. Also, there may be any number of nonliteral usages which are not covered by any of the WordNet senses. For this reason we introduce the use of databases of known metaphors, idioms, and expressions (see Section 6.1.4). Critics might say: if such databases exist, why do we need TroFi?

There are several reasons. One is that such databases are unlikely to list all possible instances of nonliteral language. For example, the Berkley Conceptual Metaphor Home Page lists the metaphor, "don't pour your money down the drain," but not "he keeps pouring cash into that failing enterprise." Another reason is that knowing that an expression *can* be used nonliterally does not necessarily mean that you can always tell when it *is* being used nonliterally. For example, let us take the sentence, "*Who* did you run into?!?" Knowing that "run into" can be used idiomatically does not help us to decide whether the addressee just met someone or literally bumped into them while running.

We use the databases of known metaphors, idioms, and expressions to help us redefine our feedback sets. TroFi has only two feedback sets: a nonliteral one built on the aforementioned databases, and a literal one built on the WordNet senses. Unfortunately, we still have a problem. The literal set may still be contaminated by nonliteral senses, resulting in a tug-o'-war between the two feedback sets.

To deal with noise in the data, we introduce a technique called *scrubbing*. We identify problematic senses or words using certain criteria and attempt to eliminate them by moving them

to the opposite feedback set or removing them altogether. Different scrubbing methodologies produce varying results, so we employ a number of them to create different learners. We revisit learners later in this chapter. For details on the construction of learners, see Section 6.2.2.1.

Let us take a step closer and examine the internal structure of a feedback set. As we have mentioned previously, a feedback set is a set of sentences from a corpus collected on the basis of *seed* words. The seed words are synonyms of the target word. In addition, our feedback sets contain example sentences from WordNet and from the databases of known metaphors, idioms, and expressions. Each sentence is pared down to a list of *features*. For TroFi, any stemmed (Porter, 1980) noun or verb that is not the target or seed word and that is not in a list of frequent words can be a feature. There is no structure imposed on these features: TroFi uses a *bag-of-words* approach.

The composition of the feedback sets is extremely important. They have more effect than any other component on TroFi's success or failure. For this reason, much of our research has been devoted to improving them and using them to their full advantage. Besides scrubbing, there are two other enhancements to the feedback sets: the expansion of context and the addition of structural information through the use SuperTags, which encode additional features not directly visible in the input (Bangalore & Joshi, 1999).

The problem with the bag-of-words approach is that structural information is completely ignored. Also, by limiting ourselves to nouns and verbs, we are throwing away potentially valuable prepositions, adverbs, and particles. The addition of SuperTags, from the realm of tree-adjoining grammars, to the feature lists allows us to remedy some of these shortcomings.

The primary benefit of SuperTags as far as TroFi is concerned is that each tag encodes information not only about a word's part of speech, but also about its local syntactic context. In other words, the tag provides information about surrounding words as well. We use a *SuperTag trigram* – to be further discussed in Section 6.2.2.2 – to capture the immediate structure

surrounding our target and seed words. This is an improvement on regular *n*-grams because we are able to capture meaningful syntactic relationship between words, not just their relative locations in the sentence.

The second enhancement concerns the inclusion of additional context. Studies have shown (e.g. (Martin, 1994)) that sources and targets of metaphors (see Section 2.2.1) are often revealed in sentences preceding the actual metaphor. Furthermore, people often explain their statements after they have made them. It is quite possible, therefore, that the relevant features for deciding whether a usage is literal or nonliteral will be not in the sentence containing the target word, but in an adjacent sentence. For TroFi we experiment with including both the sentence preceding and the sentence following the sentence containing the target word. Larger contexts are possible, but the cost of working with such large feature sets becomes prohibitive.

Once we have constructed the feedback sets, we can run the TroFi algorithm.

We mentioned previously that TroFi attracts sentences containing the target word to the feedback sets by calculating similarities as described in (Karov & Edelman, 1998) and Section 7.1.1.1. Two sentences are considered to be similar if they contain similar words and two words are considered to be similar if they are contained in similar sentences. This circularity engenders *transitive similarity*: if sentence A is attracted to sentence B, and sentence B is attracted to sentence C, then sentence A will be attracted to sentence C. This is important because sometimes there are no shared words between a given sentence from the original set (i.e. sentences containing the target word) and any of the sentences in either of the feedback sets. Under normal circumstances such a sentence would never make it into either cluster. By virtue of transitive similarity, however, if this original sentence shares a word with some other original sentence and that sentence is similar to one of the feedback sets, then it will drag the first sentence with it. By allowing us to work with deficient data sources, the algorithm allows us to defeat the *knowledge acquisition bottleneck*.

The likelihood of finding all possible usages of a word in a single corpus is low. Transitivity of similarity, however, allows us to make the most of the available information.

Unfortunately, we are still not able to use the available information to its best advantage. Doing so requires a change to the basic algorithm. Instead of determining attraction based on the highest similarity shown between an original sentence and a single feedback set sentence, we use the sum of all the similarities. Although it is appropriate for fine-grained tasks like word-sense disambiguation to use the single highest similarity score in order to minimize noise, it may be too limiting for a broader task like literal/nonliteral clustering. The literal and nonliteral senses cover a vast number of usages that could well be spread across a number of sentences in the feedback sets. Summing across all the similarities of an original set sentence to the feedback set sentences could therefore give us more persuasive results.

We now return to the learners produced through scrubbing. Due to the different scrubbing methodologies employed, the learners vary in their composition and in the patterns of attraction they produce. Each learner performs better on some target words and worse on others. In order to maximize the potentials of the learners, we allow them to vote using a majority-rules schema with optional weighting. The goal is to capitalize on a learner's positive tendencies while down-playing its negative ones.

There will always be some sentences that TroFi is less certain about than others. The basic TroFi algorithm makes a default decision on these by awarding them to the cluster of the feedback set to which they show the most similarity, no matter how slight. Better results can be achieved by sending these sentences to a human evaluator. For cases where the human is willing to do a certain percentage of the work, we introduce an optional active learning component. This can be regarded either as the human helping the algorithm or as the algorithm helping the human. We prefer the latter view. Allowing TroFi to help with the task of literal/nonliteral clustering greatly reduces the amount of work that would be required for someone to perform this task

completely manually. In addition, having the certainty of a human judgement allows us to add the sentences in question to the feedback sets, potentially improving their attractiveness.

One last idea we wish to introduce in this overview is that of *iterative augmentation*. Once we finish our first TroFi run, we save the clusters and also add the newly clustered sentences to the feedback sets. We then save the feedback sets with all their final similarity scores. In future runs, these feedback sets can be re-used and the old similarity scores used as weights. In this way, useful sentences can be expected to become more attractive over time, gradually improving accuracy on sets of previously unseen sentences.

Over time, TroFi can create, or help to create, extensive literal/nonliteral clusters. These can be used in any number of applications. On the most basic level, they can be used as training data for statistical classifiers. On a more general level, the clusters can serve as a resource for a variety of processes and algorithms requiring either a uniform collection of sentences or many examples of different usages of particular words. Most ambitiously, one could attempt to use the clusters to build a nonliteral language interpretation system.[1]

In this chapter, we provided a brief tour through the entire TroFi system. We now begin a more detailed examination of the various components, starting in Chapter 6 with the data sources used by TroFi and the generation of the input data.

[1] See Section 11.1.4.1.

6 THE DATA

The data is a vitally important part of the TroFi process. It is the data, far more than the clustering algorithm itself, which determines a clean literal/nonliteral separation. It is also in the approach to creating the input data and the usage of that data by TroFi that this thesis makes its greatest *technical* contribution: we are using a known word-sense disambiguation algorithm, but we adapt it to our nonliteral language recognition problem through the creation of different types of feedback sets and modifications to the algorithm to make best use of those feedback sets.

TroFi employs several different data sources. Most importantly, since TroFi falls into the class of corpus-based approaches to metaphor processing, we need a corpus. In addition we need aids for producing the literal and nonliteral feedback sets. The data sources we have settled on are: the Wall Street Journal Corpus (WSJ), WordNet, Wayne Magnuson English Idioms Sayings & Slang, and the Conceptual Metaphor Home Page.

We discuss each of these data sources and their use in TroFi in Section 6.1. Then, in Section 6.2, we discuss the creation of the various types of feedback sets.

6.1 Data Sources

6.1.1 The Wall Street Journal Corpus

We evaluated two corpora for use by TroFi: the Brown Corpus and the Wall Street Journal Corpus (WSJ). The Brown Corpus of Standard American English, consisting of 500 texts of about 2000 words each, is interesting because of the varied domains from which it is drawn. However, although Brown's coverage is broader, we found that the Wall Street Journal Corpus, which leans heavily in the direction of commerce, finance, and economic issues, still contains a

fair amount of domain variation. Also, it is larger and less antiquated than the Brown Corpus. This was our main reason for ultimately choosing the WSJ.

The version of the WSJ being used consists of '88-'89 Wall Street Journal articles and was compiled as part of the Penn Treebank Project. The corpus was tagged using the Adwait Ratnaparkhi tagger and the B. Srinivas SuperTagger. The statistics provided for the corpus are:

- Total number of tokens in corpus: 24,008,639
- Total sentences: 979,309
- Total number of tagged sentences: 968,293
- Total number of types: 195,308
- Total number of types with count <= 4: 124,789

The SuperTagged corpus files have the following format:

```
In//IN//B_Pnxs
a//DT//B_Dnx
move//NN//A_NXN
that//WDT//B_COMPs
would//MD//B_Vvx
represent//VB//B_N0nx0Vnx1
a//DT//B_Dnx
major//JJ//B_An
break//NN//A_NXN
with//IN//B_nxPnx
tradition//NN//A_NXN
…
the//DT//B_Dnx
cherished//JJ//B_An
title//NN//A_NXN
of//IN//B_nxPnx
partner//NN//A_NXN
.//.//B_sPU
...EOS...//...EOS...
...EOS...//...EOS...
```

We convert these files into three different formats – tagged, SuperTagged, and nv (noun/verb) – for use by TroFi. The nv sentences consist of the stemmed nouns and verbs in each sentence with any tokens found in a list of 374 frequent words removed. The frequent word list consists of the 332 most frequent words in the BNC (British National Corpus) plus contractions, single letters, and numbers from 0-10. The resulting *nv* sentences are really the feature sets that will be used by the TroFi algorithm. Below are examples of each of the three formats:

- Tagged:

```
In\/IN a\/DT move\/NN that\/WDT would\/MD represent\/VB a\/DT
major\/JJ break\/NN with\/IN tradition\/NN … the\/DT cherished\/JJ
title\/NN of\/IN partner\/NN \.\/\.
```

- SuperTagged:

```
In\/B_Pnxs a\/B_Dnx move\/A_NXN that\/B_COMPs would\/B_Vvx
represent\/B_N0nx0Vnx1 a\/B_Dnx major\/B_An break\/A_NXN
with\/B_nxPnx tradition\/A_NXN … the\/B_Dnx cherished\/B_An
title\/A_NXN of\/B_nxPnx partner\/A_NXN \.\/B_sPU
```

- nv:

```
tradit … titl partner
```

TroFi uses a randomly selected 10% of the corpus to create its first-run clusters. Additional sentences for *iterative augmentation* (see Chapter 10) are drawn from the remaining 90%. Initial development of the TroFi algorithms was carried out using the Brown Corpus, so no separate development set from the WSJ was needed. Furthermore, the set of target words used for the experiments in Chapters 8 and 9 is augmented by additional target words when we build the TroFi Example Base in Chapter 10, showing that TroFi works equally well on a set of completely unseen target words.

6.1.2 WordNet

The word-sense disambiguation algorithm on which TroFi is based – (Karov & Edelman, 1998) – uses "a machine readable dictionary or a thesaurus". Karov & Edelman state that "the single best source of seed words was WordNet" (Karov & Edelman, 1998, p. 48). We take their advice and use WordNet as our machine-readable dictionary (MRD) because it is freely available and has interfaces in a number of different programming languages[1].

WordNet organizes the entry for a given word into synonym sets or *synsets* based on the different senses of the word. Each synset contains a list of synonyms, a definition, and an example or two. The following is the entry for the verb "bite":

[1] TroFi uses the Perl implementation Lingua::Wordnet by Dan Brian.

1. bite, seize with teeth -- (to grip, cut off, or tear with or as if with the teeth or jaws; "Gunny invariably tried to bite her")

2. bite, sting, burn -- (cause a sharp or stinging pain or discomfort; "The sun burned his face")

3. bite -- (penetrate or cut, as with a knife; "The fork bit into the surface")

4. sting, bite, prick -- (of insects, scorpions, or other animals; "A bee stung my arm yesterday")

Whereas Karov & Edelman use WordNet only for finding synonyms to use as seeds for feedback sets, we go a few steps further. We also turn the WordNet definitions and example sentences into feature lists and, most importantly, use the characteristics of the synonym lists, definitions, and examples sentences to refine our feedback sets. This process of feedback set refinement is called *scrubbing*. We discuss *scrubbing* further in Section 6.2.2.1.

6.1.3 Wayne Magnuson English Idioms Sayings & Slang

Wayne Magnuson English Idioms Sayings & Slang is an electronic collection[2] of idioms, sayings, and slang. It lists thousands of terms and expressions and gives a definition and an example for each. This data source is attractive for use by TroFi due to its size – over 6300 expressions – and its similarity to WordNet in terms of content and formatting. Like in WordNet, there are definitions and examples that we can use for building feature lists. Below is an example of a Wayne Magnuson entry for the target word “climb”:

climb the walls . . feel upset or stressed . . On the first day of school , the teacher was climbing the walls .

The main weakness of this data source is a dearth of recent and original metaphors. To remedy this shortcoming, we seek out the WWW Server of the Conceptual Metaphor Home Page.

[2] This collection has also been published by Prairie House Books (ISBN 1-895012-09-0).

6.1.4 The Conceptual Metaphor Home Page

The WWW Server of the Conceptual Metaphor Home Page, administered by the University of California, Berkley, provides access to the database of conceptual metaphors compiled by George Lakoff, "the Father of the Conceptual Metaphor".

The conceptual metaphor collection is organized by metaphor type, for example:

> Understanding is Digestion: "It'll take some time to digest that information."
> Intense Emotions are Heat: "The crowd was all fired up."

For TroFi, the metaphor types are irrelevant. Of course this does not mean that much of the nonliteral language that TroFi discovers will not be of a particular metaphor type. It simply means that we are not concerned in this thesis with using or producing specific labels ourselves. TroFi simply trawls a list of examples compiled from the Conceptual Metaphor WWW Server to find sentences containing a given target word.

In the remainder of this thesis, we often refer to the Conceptual Metaphor list combined with the Wayne Magnuson collection as *the database of known metaphors, idioms, and expressions.*

6.1.5 Target Word List

The *target word list* is the root of the TroFi Example Base. These are the words for which sentences are extracted from the WSJ and for which TroFi distinguishes between literal and nonliteral usages. In the current implementation we restrict the target list to verbs. With verbs there is less likely to be interference between nonliterals or confusion about where exactly in the sentence the nonliteral lies. However, TroFi is not inherently limited to verbs. It could easily be adapted for other parts of speech.

The target word list was derived as follows. We started with a list of English verbs (9091 of them) compiled by Andrew Saulters at Georgia Tech. Next we pruned this using an available

list of the 332 most frequent words in the British National Corpus. We then counted how many times each verb appears in our database of known metaphors, idioms, and expressions. We also counted how many times each word appears in the Brown Corpus. Finally, we automatically selected those words which appeared at least once in the idiom/metaphor lists and more than twice in Brown. The resulting list of 142 words was augmented with the following hand-selected list to bring the total to 150:

die	drown	go	grasp
guard	pass	plant	plow

In addition, a few words likely to cause an overflow of examples were manually removed or replaced. We then extracted all the sentences containing these words from the 10% slice of the WSJ Corpus described in Section 6.1.1 and manually annotated them for selection and testing purposes. We further pared our list of words down to 50 by first choosing 25 words that have been discussed or mentioned in the metaphor processing literature and by then subjectively selecting another 25 based on apparent variety of nonliteral usages and availability of nonliteral feedback set data.

6.2 Original Set and Feedback Set Creation

In this section, we discuss the use of the data sources described in Section 6.1 to create the original and feedback sets for the TroFi algorithm. We first discuss the basic selection of these sets and then examine in detail the additional types of feedback sets we create to help with the literal/nonliteral clustering task.

6.2.1 Basic Original Sets and Feedback Sets

The basic premise of the TroFi algorithm is the attraction of sentences containing a specific target word – the original set – to either the literal or the nonliteral feedback set.

Logically, as with hand-selected training sets in other statistical algorithms, the composition of the feedback sets is paramount.

To build the original set we follow the data set creation methodology employed by Karov and Edelman (1998) for their word-sense disambiguation algorithm. We extract from the corpus all the *examples* or *contexts* of the *target word* – i.e. all the sentences containing the word we are trying to disambiguate.

The next step is to create a *feedback set* for each sense of the target word by collecting examples containing *seed words* for each sense. To gather the seed words, we look up the target word in an MRD or a semantic hierarchy – in our case, WordNet. For each sense of the target word, we extract a seed set of "contextually similar" words (Karov & Edelman, 1998). According to Karov & Edelman these are words that are likely to be found in contexts similar to those of the target word when it is used in a particular sense. Feedback sets extracted using these seed words ideally contain contexts relevant only to one particular sense of the target word. For example, a synonym of the target word "die" is "perish". It is expected that sentences containing the word "perish" will most often exhibit a context befitting the "death" sense of "die". Unfortunately, nonliteral readings of a word are often transferred to synonyms of that word. For example, it is not unlikely that one would hear both "his dreams died" and "his dreams perished." However, we proceed on the assumption that such transferred nonliteral readings are less frequent. For example, one is more likely to hear the nonliteral expression "the engine died" than the slightly odd "the engine perished."

Karov & Edelman build a feedback set on each of the synsets for a given target word by using all the synonyms from that synset as seeds. Unfortunately, this is not appropriate for our purposes since we require not individual sense feedback sets, but rather a literal feedback set and a nonliteral feedback set. In theory one could divide the WordNet synsets into literal and nonliteral, but this would add an element of supervision that we wish to avoid. Although it would

solve numerous problems for the small collection of target words explored in this thesis, it would not scale well to larger collections or to an implementation of TroFi in a foreign language.

We approach the task of building the nonliteral feedback sets on the synonyms and examples from our database of known metaphors, idioms, and expressions (see Section 6.1.4). The literal feedback sets are taken from the synonyms and examples of the WordNet synsets. In addition to using synonyms as seeds, we convert the examples in both the data sources into additional feature lists. They often contain some of the most typical contexts for a given target word and thus add a strong initial basis for attraction.

We now have both a literal and a nonliteral feedback set, but, unfortunately, since the literal set contains *all* the WordNet senses, it contains the nonliteral ones as well. This issue turns out to be one of the largest problems for TroFi, and we attempt to address it with a technique called *scrubbing*. We discuss *scrubbing* and other feedback set enhancements in Section 6.2.2.

Each sentence in the original and feedback sets is pared down to a set of *features*. Anything deemed useful to the disambiguation task can be declared a feature, and features can be weighted according to their expected contribution. Features can be anything from simple nouns and verbs to *n*-grams and syntactic structures. For the basic algorithm, we use just the nouns and verbs in each sentence that are not also found in the frequent word list (see Section 6.1.1). We later attempt to improve the feature lists by adding extra context (see Section 6.2.2.3) and information about the syntactic frames of the verbs encoded in SuperTags (see Section 6.2.2.2).

6.2.2 Variations on the Feedback Sets

In the previous section we lamented the fact that our literal feedback sets may be contaminated by nonliteral WordNet senses. In the same way, the nonliteral feedback sets may contain words and phrases that cause problems by overlapping with words and phrases from the

literal feedback set. This type of feedback set *noise* is the greatest threat to TroFi accuracy, so we must attempt to remedy the situation.

We have already discussed why we cannot manually separate literal and nonliteral WordNet synsets: it would be fine for a few target words, but it would not scale well to large collections. Additionally, it would become an even more difficult chore if we were running TroFi in a language other than English. Trying to clean the feedback sets after they have been created would be far more onerous still. TroFi generates hundreds, even thousands, of feedback set sentences. Cleaning these by hand would be prohibitive in terms of both time and sanity. This means that we must somehow clean up the feedback sets automatically.

Implementing an automatic process is also important for maintaining TroFi's status as an unsupervised algorithm. Doing manually clean-up would be akin to creating training data by hand – which is precisely what we are trying to avoid.

We refer to our automatic cleaning process as *scrubbing*. In effect, we attempt to *scrub* the dirt (or *noise*) out of the feedback sets. More precisely, *scrubbing* describes the process of moving or removing particular synsets, words, or feature sets from the input data or from the generated feedback sets. Decisions are made based on specific properties of the synsets and/or words, or on the overlap of certain features between one feedback set and the other.

We experimented with a number of different scrubbing methods. Since different scrubbing methodologies produce different results, we employ a variety of strategies and refer to the results as *learners*. These learners are an important enhancement to the basic TroFi algorithm. We discuss the creation of learners in the Section 6.2.2.1.

The feedback set feature lists can be improved further by augmenting them with structure in the form of SuperTags and by increasing their size with additional context. These enhancements will be discussed in Sections 6.2.2.2 and 6.2.2.3, respectively.

6.2.2.1 Learners

Before we can leap into a discussion of scrubbing and the construction of different learners, we must first acclimatize ourselves to the basic feedback-set-building algorithm. We examine it in detail in this section and refer back to it often in our subsequent discussions of the individual learners. Pseudo-code is available in Appendix A.

We begin with the nonliteral feedback set. We first find all entries containing a given target word in our database of known metaphors, idioms, and expressions. We select all the nouns and verbs that are not in our list of frequent words from the definitions (where available) and add them to the nonliteral seed list. We also add them to a *scrubber* set – a collection of words that we will later use to scrub the literal set. Next we convert all the example sentences into feature sets and add them to our nonliteral feedback set. We also add the words from these sentences to the scrubber set. Finally, we select from the WSJ Corpus sentences containing the seed words we have collected, convert them into feature lists, and add them to the nonliteral feedback set.[3]

To create the literal feedback sets, we take the synonyms from each WordNet synset containing a given target word and add them to the set of literal seeds. Note that since WordNet synonyms may be single words or whole expressions, our seed lists may contain not only single verbs, but also phrasal and expression verbs. Next we convert the example sentences and definitions into feature sets and add them to the literal feedback set. We gather all the words from all the feature sets together into a scrubber. Finally we select the rest of the literal feedback set from the WSJ Corpus using our seed list. If a given seed contains a particle or noun, sentences containing first the verb and then the particle or noun as the next or next-next word are selected.

[3] Note that we strip the seed words out of the corpus feature sets so that they do not cause unhelpful commonalities between sentences.

If we were doing scrubbing, we would have used the collected scrubbers at various points throughout the above algorithm. We discuss the scrubbing process in general in the following paragraphs and then devote a section to the creation of each of the learners.

Scrubbing is founded on a few basic principles. The first is that the contents of the database of known metaphors, idioms, and expressions are just that – known. Consequently we take them as primary and use them to scrub the WordNet synsets. The second is that phrasal and expression verbs are often indicative of nonliteral uses of verbs – i.e. they are not the sum of their parts – so they can be used as catalysts for scrubbing. The third is that content words appearing in both feedback sets will cause a tug-o'-war, a situation we want to avoid. The fourth is that our scrubbing action can take a number of different forms: we can choose to scrub just a word, a whole synset, or even an entire feature set. In addition we can either move the offending item to the opposite feedback set or remove it altogether. Some of the principles described here require further explanation.

By phrasal/expression verbs we mean verbal units consisting of more than one word. We take these as indicators of nonliteral senses because they are very often not the sum of their parts. For example, in telling someone to "throw it away," one would not want to be taken literally. Scrubbing on the basis of phrasal/expression verbs can cause problems for literal synsets containing phrasal verb synonyms – for example, "set on" is a synonym of "attack" in the perfectly good literal sense of assailing someone physically. Also, many nonliteral synsets contain no phrasal/expression verbs – for example, the synset of "absorb" that contains the sample sentence "her interest in butterflies absorbs her completely" contains no phrasal or expression verbs. To counteract the first problem, we build more than one learner. To counteract the second problem, we additionally use overlapping words as indicators for scrubbing.

We define overlapping words as content words (in our case, nouns and verbs) in the synsets that can also be found in the relevant entries of the database of known metaphors, idioms,

and expressions, and vice versa. For example, if the word "squander" is in our scrubber for the target word "blow", then we scrub the synsets containing that word. The reasoning here is simple: we know that "squander" is a nonliteral meaning of "blow" because our database of known metaphors, idioms, and expressions tells us so. Hence, a synset listing "squander" as a synonym is most likely describing a nonliteral sense.

Based on the above discussion, it would seem logical that we would cover the most ground by using both phrasal/expression verbs and overlapping words as scrubbing indicators, rather than just one or the other. We have confirmed this intuition experimentally. In addition to looking simply at a choice between both, either/or, or neither, we experimented with assigning a value to each indicator and adding them up to find the *scrubbability factor* of each synset. If a synset's scrubbability factor was above a certain threshold it was scrubbed. Informal experiments revealed that this method provided no significant improvement over the simple approach.

We identify problem synsets, words, and feature sets using the above indicators. We must then decide whether to move or remove them. Our primary motivation is to remove contaminants, and either of these actions will accomplish that. A secondary motivation is to try to fortuitously augment the nonliteral sets. Sometimes the nonliteral feedback sets are extremely weak, and moving content over from the literal set can help give them a much needed boost. Unfortunately, by doing so, we risk introducing noise.

We experimented with a number of these options to produce a whole complement of learners. There is safety in numbers, and we hope that if we create a collection of learners, they will be able to balance out each other's imperfections. Each learner is characterized by its *scrubbing profile*:

INDICATOR :

the linguistic phenomenon that triggers the scrubbing :

phrasal/expression verbs, overlap (words appearing in both sets)

TYPE :

the kind of item to be scrubbed :

word, synset, feature set

ACTION :

the action to be taken with the scrubbed item :

move, remove

6.2.2.1.1 Learner A

The scrubbing profile of Learner A is:

INDICATOR : phrasal/expression words AND overlap

TYPE : synset

ACTION : move

To build Learner A, we start by creating the nonliteral feedback set as described in our general discussion above. When we get to the literal feedback set, instead of sending all the synsets directly to the literal side, we use our scrubbing indicators to pick out synsets that should cross over to the nonliteral side. For Learner A, we are looking at the synonym lists only: the chosen synsets are those whose synonym lists contain either expression/phrasal verbs or scrubber words collected during the creation of the nonliteral feedback set. These indicators suggest to TroFi that the synset is potentially encoding a nonliteral sense.

Once we have decided the fate of each synset, we proceed normally, collecting seed words and examples for the literal set from the literal synsets and additional seed words and examples for the nonliteral set from the nonliteral synsets. Once we have built both feedback sets, we do a final pass to remove any feature sets that have inadvertently ended up in both sets.[4]

[4] Sentences from the WSJ Corpus can end up in both sets if there happen to be identical seed words, or if a sentence contains a seed word both from both the literal list and the nonliteral list.

6.2.2.1.2 Learner B

The scrubbing profile of Learner B is:

INDICATOR : phrasal/expression words AND overlap
TYPE : synset
ACTION : remove

The only difference between Learner A and Learner B is that instead of moving problematic synsets from the literal set to the nonliteral set, we remove them altogether. This saves us from accidentally contaminating the nonliteral set. However, it does mean that we are throwing away information that could have been used to pad out sparse nonliteral sets.

6.2.2.1.3 Learner C^1

The scrubbing profile of Learner C^1 is:

INDICATOR : overlap
TYPE : word
ACTION : remove

Learner C^1 is constructed on the notion that moving or removing whole synsets might be overkill. Hence, we do not interfere with the synsets, but rather remove overlapping words from the final literal and nonliteral feedback set feature lists.

We build the feedback sets without any special interventions and only then scrub both sets. From the literal feedback set sentences, we remove any scrubber words we collected while building the nonliteral set. If we find such an overlapping word, we add it to the scrubber we collected while building the literal set so that it will be scrubbed from the nonliteral set as well. We do this because if there is overlap, the offending word, although potentially useful to one set, could just as easily wreak havoc. For example, imagine that we get overlap on the word “wind” for the target word “blow”. The literal set speaks of the “wind blowing” while the nonliteral set tells us that the “winds of war are blowing”. Since we are using a bag-of-words approach, it is

easy to see how the word “wind” is going to cause problems, especially if we leave it in just the nonliteral set. We do lose important information by removing it from the literal set, but that is why we have more than one learner.

6.2.2.1.4 Learner C^2

The scrubbing profile of Learner C^2 is:

INDICATOR : overlap
TYPE : feature set
ACTION : remove

While experimenting with Learner C^1, we noticed that the feature sets containing the overlapping words often contained other words likely to cause false attraction. In order to annihilate this potential threat, we created Learner C^2, which simply removes the whole feature set if an overlapping word is found.

Running some tests on the different effects of these two learners, we found that we obtained slightly better results using Learner C^2. For this reason – and due to the overhead of adding extra learners – we decided to use only Learner C^2 for the final version of TroFi.

6.2.2.1.5 Learner D

The scrubbing profile of Learner D is:

INDICATOR : n/a
TYPE : n/a
ACTION : n/a

Learner D is the baseline – no scrubbing. We simply use the basic algorithm described at the beginning of Section 6.2.2.1.

6.2.2.2 SuperTags

Thus far, our feature lists have used a *bag-of-words* approach: we have collected a bunch of words, but have imposed no structure on them. This means that we have been discarding syntactic information about subjects and objects. Furthermore, by using only verbs and nouns, we have been ignoring potentially important information held in other words in the sentence. These ignored words include prepositions and particles. We have been ascribing great value to these – in the form of phrasal/expression verbs – in the building of our learners, indicating that they should really be incorporated into other parts of the algorithm as well.

We attempt to include some of this important information through the use of *SuperTags* from the realm of tree-adjoining grammars. The benefit of *SuperTags*[5] is that they encode a great deal of syntactic information in a single tag. In addition to a word's part of speech, they also encode information about its location in a syntactic tree. In other words, from the SuperTag of a given word, we learn something about the surrounding words as well. We experimented with a number of different SuperTag combinations: just the SuperTag of the target verb; a *trigram* incorporating the preceding and following SuperTags; a combination tag mixing actual words in with the tags. Since TroFi attempts to establish similarities and differences between sentences, the critical part of choosing a SuperTag combination is to find something that will not create too many – quite possibly false – similarities. Using just the SuperTag of the target word, for example, creates a feature that may be common to many of the sentences and is thus unhelpful. On the other hand, we do not want to create anything too unique for fear of producing no new similarities at all.

After some experiments, we settled on a *SuperTag trigram* composed of the SuperTag of the target word and the SuperTags of the following two words if they contain any actual prepositions, particles, adverbs, or nouns, as well as the words themselves. If the trigram does

[5] We will occasionally refer to these as *xtags*, particularly in graphs and the pseudo-code in Appendix A.

not include any of these additional words, it is not used. To learn about the exact construction of the SuperTag trigrams, please consult the pseudo-code in Appendix A. We provide a couple of examples of SuperTagged sentences and their corresponding feature sets below:

A/B_Dnx person/A_NXN needs/B_nx0Vs1 discipline/A_NXN to/B_Vvx kick/B_nx0Vpls1 a/B_Dnx habit/A_NXN like/B_nxPnx drinking/A_Gnx0Vnx1 ./B_sPU

→

disciplin habit drink kick/B_nx0Vpls1_habit/A_NXN

The/B_Dnx theory/A_NXN filled/A_nx0Vpx1 in/A_PXPnx gaps/B_ARBs left/A_nx0Vpx1 in/A_PXPnx earlier/B_An accounts/A_NXN ./B_sPU

→

theori gap account fill/A_nx0Vpx1_in/A_PXPnx_gap/B_ARBs

It is important to note that we do not produce another learner when we use SuperTags; rather, we produce a whole parallel set of learners. The full complement of learners described in Section 6.2.2.1 is there, only in this case they all include SuperTag trigrams in their feature lists. It is worth noting that the creation of Learners A and B changes somewhat if SuperTags are used. In the original version we only move or remove synsets based on phrasal/expression verbs and overlapping words. If SuperTags are used, we also move or remove feature sets whose SuperTag trigram contains adverbs, particles, or prepositions, as these may indicate phrasal/expression verbs.

6.2.2.3 Context

Sometimes critical disambiguation features are contained not in the sentence with the target word, but in an adjacent sentence. As we all learned during vocabulary lessons in elementary school, people often explain what they mean before or after they use a given word or expression. For example, in the sentence “Member hit the ceiling,” there is nothing to indicate whether “hit the ceiling” is being used literally or nonliterally. The sentence following this sentence, however, provides the necessary context: “Member stated that she broke her thumb while she was cheering for the Patriots at home and hit her thumb on the ceiling.”

We can see that adding context from adjacent sentences to our feature lists may prove beneficial. Unfortunately, there is also the chance that these extra features will generate enough noise to hinder rather than help the clustering process. To add context, we simply group the sentence containing the target word with a specified number of surrounding sentences and turn the whole group into a single feature set. As with SuperTags, we do not create additional learners by using context; rather, we create a full parallel set.

The example below shows the effect of adding context. First we see the simple feature set; second, the feature set with added context:

foot drag/A_Gnx0Vnx1_foot/A_NXN

→

foot everyon mcdonnel dougla commod anyon paul nisbet aerospac
analyst prudentialbach secur mcdonnel propfan model spring count order delta
drag/A_Gnx0Vnx1_foot/A_NXN

In the simple case, it is difficult for TroFi to tell whether the "dragging of the feet" should be taken literally or nonliterally: there is not much to go on. In the context case, however, the additional features will allow TroFi to make a more confident decision.

6.3 Summary

In this chapter we discussed the various data sources used by TroFi, namely the Wall Street Journal Corpus, WordNet, Wayne Magnuson English Idioms Sayings & Slang, and the Conceptual Metaphor Home Page. We also described in detail the construction of the feedback sets, including our scrubbing algorithm and the different learners produced by scrubbing. Finally, we discussed the contribution of SuperTags and additional context to the expansion of the feature sets. In Chapter 7, where we discuss TroFi's various models and algorithms, we will gain further insight into how all this data is actually used.

7 MODELS & ALGORITHMS

We will discuss the algorithms used by TroFi for literal/nonliteral clustering in three main sections:

1. Unsupervised Algorithm
    a. Sum of Similarities vs. Highest Similarity
    b. Learners and Voting
    c. SuperTags
    d. Context
2. Active Learning
3. Iterative Augmentation

7.1 The Unsupervised Algorithm

By now is should be clear that TroFi's primary task is to separate literal and nonliteral usages of a given target word into two distinct clusters based on the surrounding context. We have also mentioned that in order to do this, we have chosen to regard *literal* and *nonliteral* as two distinct senses of the target word, allowing us to reduce our nonliteral language recognition problem to one of word-sense disambiguation.

In order to determine whether word-sense disambiguation algorithms can in fact be applied to nonliteral language recognition, we construct the core TroFi algorithm around a statistical similarity-based word-sense disambiguation algorithm presented in (Karov & Edelman, 1998). We chose the Karov & Edelman approach because certain features of this algorithm make it particularly attractive for the task of nonliteral language recognition:

1. it allows for flexible sense distinctions
2. it provides flexibility in the choice of sentence features
3. it is largely unsupervised
4. its training information can be derived using a machine-readable dictionary (MRD)
5. it deals effectively with very sparse data

Our investigations revealed that the basic Karov & Edelman algorithm, although quite successful as a tool for word-sense disambiguation according to the authors, is, on its own, insufficient for the task of separating literal vs. nonliteral usages. We have thus added various enhancements to the basic algorithm, including a different way of calculating similarity, a variety of learners, a voting system, SuperTag trigrams, and additional context. These enhancements, as well as the basic algorithm, will be discussed in detail in the following sections.

7.1.1 The Basic Algorithm

The algorithm used by Karov and Edelman (1998) for word-sense disambiguation serves as the basis for TroFi's literal/nonliteral clustering algorithm. The reason we use this algorithm as a core and not some other clustering algorithm is that we are not simply trying to cluster words. We are actually trying to cluster predicates, arguments, and adjuncts all at the same time. The Karov & Edelman algorithm allows us to do that.

In the following section we provide a brief summary of the Karov & Edelman algorithm. It will be useful for our subsequent discussion of the TroFi algorithm, which will be illustrated using a sample scenario.

7.1.1.1 The Karov & Edelman Algorithm

The Karov & Edelman (1998) algorithm is built on the idea that two sentences are similar if they contain similar words, and that two words are similar if they are contained in similar sentences. The apparent circularity of this algorithm allows similarities to be found between

sentences that do not share any direct commonalities. The similarities are found by *iteratively updating* two sets of matrices. How exactly this works, is explored below.

The algorithm requires two sets of data: the original set, which contains the example sentences for a given target word, and the feedback sets, the sets of *contextually similar* sentences containing the WordNet synonyms of the target word. These sets are used to populate *similarity matrices*. There are two types: a *Word Similarity Matrix* (WSM), and *Sentence Similarity Matrices* (SSMs). The WSM lists all the words from the union of the original and feedback sets along both the *x* and *y* dimensions. The SSMs always have the original examples along the *y* dimension. In the first iteration of the algorithm – which uses the *Original SSM* – the *x* dimension also lists the original set. In subsequent iterations – which use the *Feedback SSMs* – the *x* dimension is reserved for the feedback set. There is a different Feedback SSM for each sense. These matrices are used to calculate the similarity of words and sentences by *iteratively updating* the WSM using the SSM and vice versa.

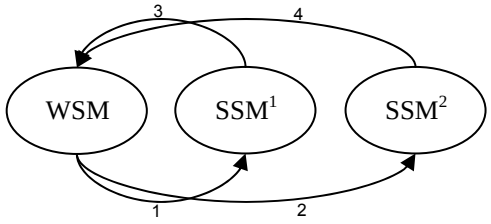
Updating is a mutually recursive process which starts with the WSM initialized to the identity matrix. Looking at similarity on a scale of 0 to 1, this indicates that every word is maximally similar to itself and minimally similar to every other word. Next, the algorithm calls for *iteration*:

1. update the sentence similarity matrices SSM_n^k, using the word similarity matrix WSM_n;
2. update the word similarity matrix WSM_n, using the sentence similarity matrices SSM_n^k.

(Karov & Edelman, 1998), where *n* is the iteration and *k* is the sense of the target word.

To explain this another way: we *start* with the WSM and then update the Original SSM with it. We then update the WSM from the Original SSM. Then we update each of the Feedback SSMs from the WSM. Then we update the WSM from each of the Feedback SSMs in turn. Then we update each of the Feedback SSMs from the WSM. And so on. We illustrate this process in Figure 7-A. The numbered arrows show the order in which updating occurs.

Figure 7-A Iterative Updating



Source: based on the concept by Karov & Edelman (1998)

Iteration continues until the difference in the changes is small enough to meet the stop conditions. We examine the Karov & Edelman *updating* process in more detail below.

The initialized WSM is used to calculate the similarity values of the Original SSM using the following formulas:

$$sim_{n+1}(S_1, S_2) = \sum_{W \in S_1} weight(W, S_1) \cdot aff_n(W, S_2) \quad (1)$$

$$aff_n(W, S) = \max_{W_i \in S} sim_n(W, W_i) \quad (2)$$

Here n refers to the iteration, S_1 is a sentence from the y dimension, S_2 is a sentence from the x dimension, and W is a word.

To find the similarity between two sentences (i.e. $sim_{n+1}(S_1, S_2)$), we simply find the sum of the weighted affinities (i.e. $aff_n(W, S_2)$) of each word W (which *belongs* to S_1) to S_2. The affinity can be easily calculated using the WSM as a lookup table. For a given word W, we look up its similarity to each word W_i in S_2 and take the maximum. The weight (i.e. $weight(W, S_1)$) by which each affinity is multiplied in the calculation of $sim_{n+1}(S_1, S_2)$, is a function of the weights of individual words in a sentence, normalized by the total number of words in the sentence. Once all the similarities have been calculated for the Original SSM, it is used to update the WSM.

The formulas for finding the similarity of two words given an SSM are similar to the sentence similarity formulas described above. There is one fundamental difference, however. Whereas the sentence similarity calculations are concerned with the words W_i that *belong* to a given sentence S (e.g. $W_i \in S_1$), the word similarity calculations are concerned with the sentences S_j that *include* a given word W (e.g. $S_j \ni W_1$). The formulas for calculating word similarity are given below:

$$sim_{n+1}(W_1, W_2) = \sum_{S \ni W_1} weight(S, W_1) \cdot aff_n(S, W_2) \quad (3)$$

$$aff_n(S, W) = \max_{S_j \ni W} sim_n(S, S_j) \quad (4)$$

Here n refers to the iteration, W_1 is a word from the y dimension, W_2 is a word from the x dimension, and S is a sentence.

To find the similarity of two words (i.e. $sim_{n+1}(W_1, W_2)$), we find the sum of the weighted affinities (i.e. $aff_n(S, W_2)$) of each sentence S (which *includes* W_1) to W_2. In the case of word similarity, the affinity can be calculated using the SSM – here, the Original SSM. For a given sentence S, we look up the similarity of S to any S_j that includes W_2 and take the maximum. The weight (i.e. $weight(S, W_1)$) by which each affinity is multiplied in the calculation of $sim_{n+1}(W_1, W_2)$ is a function of the weight of each sentence that includes a given word, normalized by the total number of sentences that include that word. This concludes the first iteration.

The second iteration follows the same procedure as above, but instead of working with the Original SSM, it works with the Feedback SSM of a particular sense. This algorithm converges, and a proof of convergence is given in (Karov & Edelman, 1998).

Once all the matrices have been populated, we can read from them the similarity of each original sentence to each sentence in the feedback set for a particular sense. We define each original sentence with a similarity to a given feedback sentence above a particular threshold as being *attracted* to that feedback sentence. We cluster all original examples that are attracted to a given feedback set.

Karov & Edelman see this as the training phase and then describe how the clusters can be used to disambiguate unseen words. With TroFi we are only interested in finding the clusters.

Let us return for a moment to why the Karov & Edelman algorithm is so suitable for literal/nonliteral clustering, i.e. nonliteral language recognition. The five main features mentioned earlier were flexible sense distinctions, flexibility in choice of sentence features, a largely unsupervised algorithm, training information derivable from a machine-readable dictionary (MRD), and the ability to deal effectively with very sparse data. We now discuss each of these points in turn.

Flexible sense distinctions are important because we define two new senses: *literal*, to cover all the literal senses/usages of the target word, and *nonliteral*, to cover all the nonliteral senses/usages. We build feedback sets for both of our new senses. Ideally, we should be able to define any original sentence attracted to the literal feedback set as being part of the *literal cluster* and any sentence attracted to the nonliteral feedback set as being part of the *nonliteral cluster*.

Because any number of features may be important for distinguishing between literal and nonliteral usages of verbs, our chosen algorithm must give us flexibility for experimenting with different types of sentence features without having to edit the algorithm – or even its implementation. For example, in addition to using the nouns and verbs in a sentence as basic features, we also use SuperTag trigrams (see Section 6.2.2.2).

The third aspect of the Karov & Edelman algorithm that makes it particularly applicable to literal/nonliteral clustering is that it is largely unsupervised, allowing us to avoid the *knowledge acquisition bottleneck* – that is, having to annotate thousands of sentences manually. Anyone who has ever done any manual sense tagging – or even something as "basic" as part-of-speech tagging – knows that this is neither trivial nor pleasant. The difficulties become even more pronounced when the distinction between the senses is necessarily vague, like the literal/nonliteral distinction. It is worth keeping in mind that a linguistic distinction that is difficult for a human to judge is likely to be even more difficult for a machine, so achieving reasonable results by applying an unsupervised algorithm to raw, real-world data, is – to say the least – challenging.

In order to build a sufficient training set for finding sense distinctions, one generally requires large datasets which can be difficult to come by. Luckily the Karov & Edelman algorithm is designed to work with very sparse data. Without this feature, we would stand little chance of making TroFi work at all. The reason is that both the literal and the nonliteral senses encompass a number of sub-senses, and each of these senses can be disambiguated using a large number of features. It is unlikely that examples of all the possible features are going to end up in the feedback sets. Still, we want to correctly cluster literal and nonliteral sentences whose features do not have an exact counterpart in either of the feedback sets. This is where the most intriguing aspect of the Karov & Edelman algorithm comes into play: *transitivity of similarity.*

Transitivity of similarity means that if S_1 and S_2 are found to be similar, and S_2 and S_3 are found to be similar, then S_1 and S_3 will also be similar. The result is that many more original sentences are attracted to the literal and nonliteral feedback sets than would normally be the case. As we will see, this property will become an important feature of the TroFi algorithm.

7.1.1.2 The TroFi Algorithm

Simply put, TroFi takes a collection of sentences containing a particular target verb and splits them into two clusters, one where the target word is used literally, and one where it is used nonliterally. In this section we will describe how exactly it does this with the help of an illustrative example using the target word "grasp". For the pseudo-code version of the algorithm, please see Appendix A.

TroFi uses the Karov & Edelman algorithm described in Section 7.1.1.1 as the clustering component of its nonliteral language recognition algorithm. This is worth keeping in mind during the following discussion.

The basic version of TroFi uses as input an original set and unscrubbed feedback sets (Learner D). The creation of these sets is exhaustively described in Section 6.2. Here we will just provide a brief overview of the resultant input using the word "grasp" as our target word. [1]

We will use the following sentences for the original set:

She grasped her mother's hand.
He thinks he has grasped the essentials of the institute's finance philosophies.
The president failed to grasp KaiserTech's finance quandary.

Once we stem this input and remove anything that is a frequent word or something other than a noun or a verb, we get:

1 L mother hand
2 N essenti institut financ philosophi
3 N president kaisertech financ quandary

The Ls and Ns are testing labels indicating whether the sentence is literal or nonliteral. The numbers allow us to refer to these feature sets in Figure 7-C, Figure 7-E, and Figure 7-H.

[1] Note that this is a highly simplified example. A full example would be too complicated to be illuminating.

Our literal feedback set looks as follows:

His aging mother gripped his hands tightly.

→

L1 mother hand

Finally, our nonliteral feedback set is the following:

After much thought, he finally grasped the idea.

This idea is risky, but it looks like the director of the institute has finally comprehended the basic principles behind it.

Mrs. Fipps is having trouble comprehending the legal straits.

→

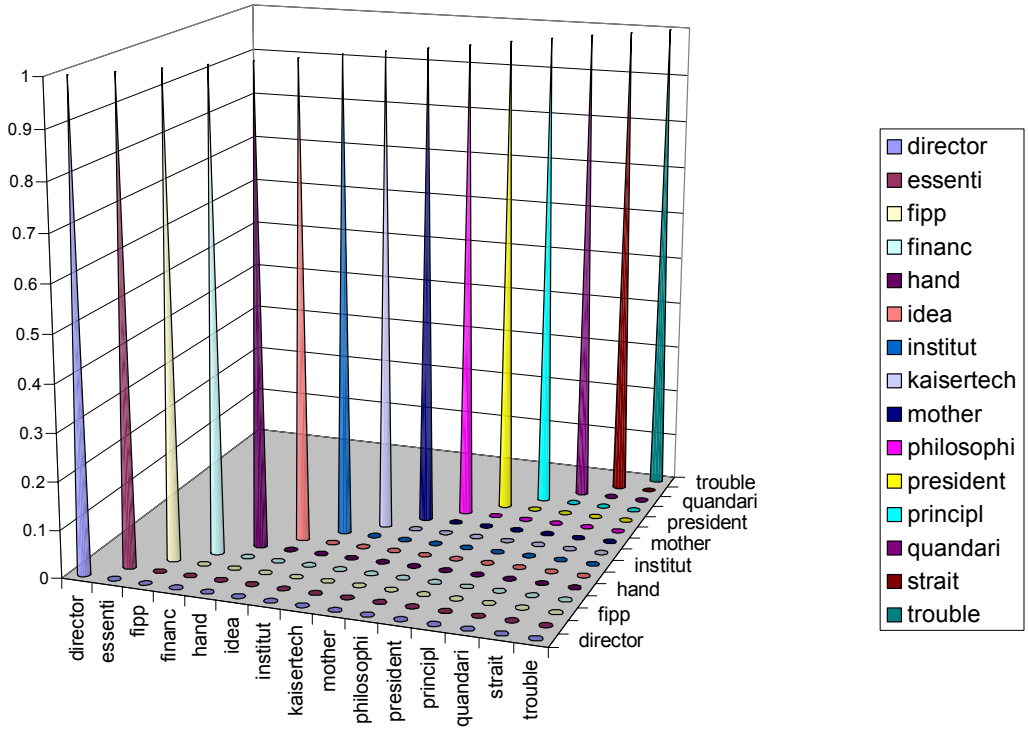
N1 idea

N2 idea director institut principl

N3 fipp trouble strait

We first read all the words in the original set as well as all the words in the feedback sets into a WSM (Word Similarity Matrix), with all the words along both the x and y axes. Since each word is maximally similar to itself the matrix is initialized to the identity matrix. We represent the WSM graphically in Figure 7-B to provide a better overview.

Figure 7-B WSM for *grasp*: Initial



Karov & Edelman do some experimentation with initializing the matrix in such a way that the similarities reflect the length of the path between any two words in WordNet. They find that this obfuscates the issue as it can create dominant similarities between dissimilar words or hold apart words that might otherwise find each other. We can predict by analysis the same sort of behaviour for the TroFi case. For example, although "idea" and "principle" are in the same WordNet tree, namely *cognitive_content*, they are far away from "essentials" which is in the *entity* tree. Of course, although "president" and "director" are in the *entity* tree and we would like to show a similarity between "president", "director", and "essentials", we do not necessarily want the resulting side-effect similarity between "essentials" and "mother" and "hand". Furthermore, "hand" is also under *cognitive_content*, and the path between "hand" and "idea" is shorter than the path between "hand" and "mother". It is worth noting that one might even have to consider disambiguating the words before finding the WordNet distances. This would require a great deal

of additional effort. So we can see that although using WordNet path lengths could definitely be used for nonliteral language recognition work – as studied at length by Dolan (1995) and as suggested by Nissim and Markert (2003) – in the TroFi case it would require additional investigation, experimentation, and analysis that is outside the scope of this thesis.

We next set up: an Original SSM (Sentence Similarity Matrix) with the original sentences (feature sets) both along the x and the y axes; a Literal Feedback SSM with the literal feedback set sentences along the x axis and the original sentences along the y axis; and, a Nonliteral Feedback SSM with the nonliteral sentences along the x axis and the original sentences along the y axis. All these matrices are initialized to 0 as shown in Table 7-A.

Table 7-A SSMs for *grasp*: Initial

Original SSM

	mother hand	essenti institut financ philosophi	president kaisertech financ quandari
mother hand	0	0	0
essenti institut financ philosophi	0	0	0
president kaisertech financ quandari	0	0	0

Literal Feedback SSM

	mother hand
mother hand	0
essenti institut financ philosophi	0
president kaisertech financ quandari	0

Non-Literal Feedback SSM

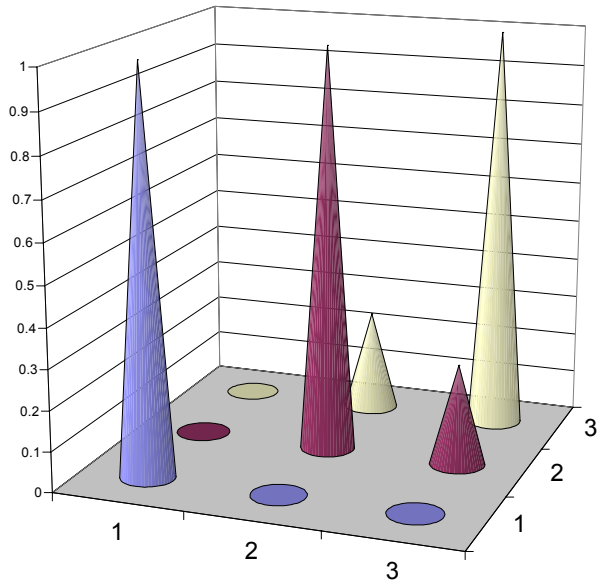
	idea	idea director institut principl	fipp trouble strait
mother hand	0	0	0
essenti institut financ philosophi	0	0	0
president kaisertech financ quandari	0	0	0

We now start the algorithm proper by *updating*[2]. We begin by calculating the similarity of each of the original sentences to itself by summing over the weighted WSM similarities of the words in those sentences. Each sentence will necessarily be equal to itself. We demonstrate on an example. The feature set "mother hand" contains two words. Each of these words will thus carry a weight of 0.5. When we check the WSM, we find that the similarity of "mother" to itself is 1, and that the similarity of "hand" to itself is 1. Thus the similarity of "mother hand" to "mother hand" is $0.5 \cdot 1 + 0.5 \cdot 1 = 1$. Sometimes we will also find similarities between different

[2] A detailed description of this process is given in Section 7.1.1.1.

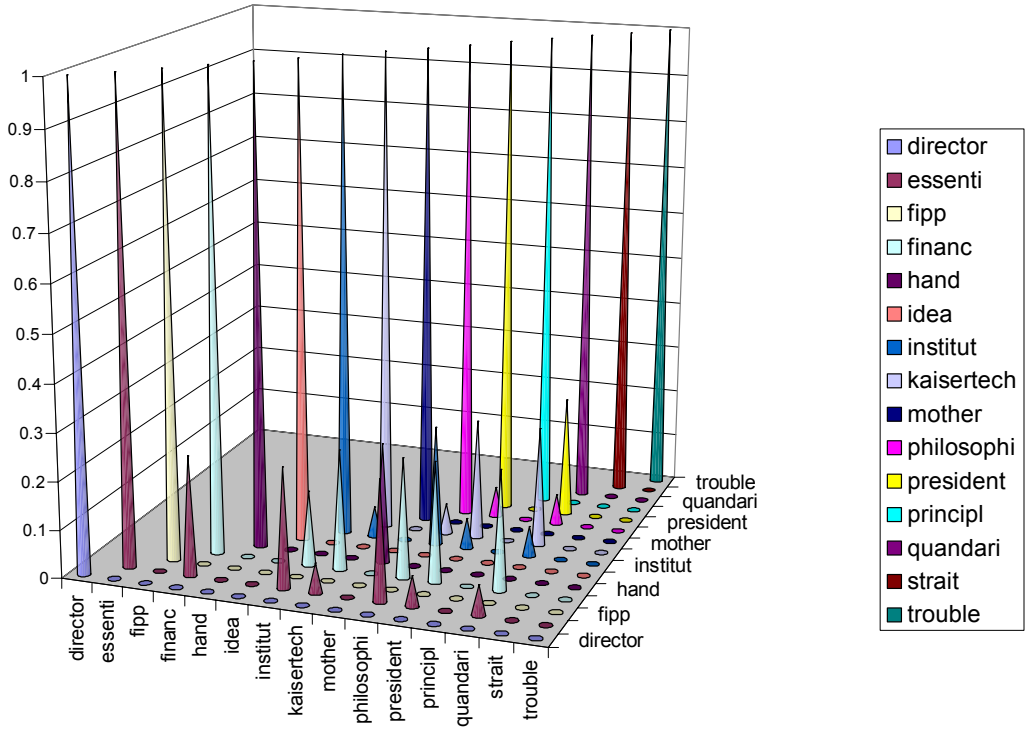
original sentences. Using the same type of calculation as for the "hand mother" example, we find that "essenti institut financ philosophi" and "president kaisertech financ quandari" have a similarity of $0.25 \cdot 0 + 0.25 \cdot 0 + 0.25 \cdot 1 + 0.25 \cdot 0 = 0.25$. The results are shown in Figure 7-C. Recall that the labels in the following graphs refer back to the feature sets on pages 80 and 81.

Figure 7-C Original SSM for *grasp* : 1st Iteration



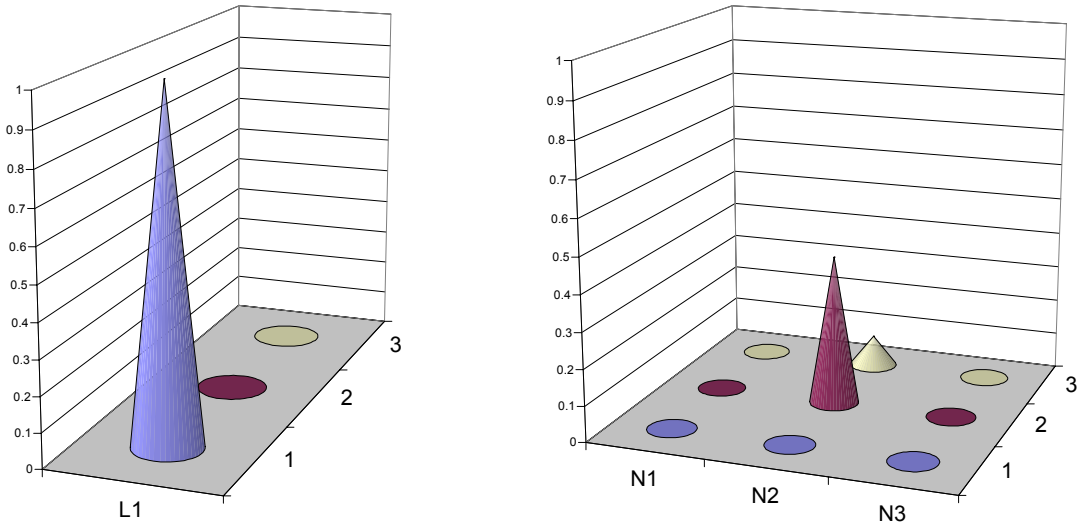
Once we have updated the Original SSM, we update the WSM from the Original SSM. We find the similarities between each pair of words by summing over the weighted SSM similarities of the sentences containing those words. For example, for "financ" and "essenti" we get a similarity of $0.125 \cdot 1 + 0.125 \cdot 1 = 0.25$. The weight of 0.125 is due to the fact that each sentence containing the word "financ" has four words, making for a sentence weight of 0.25 and there are two sentences containing the word "financ", meaning that we must further divide the sentence weight by two, giving us 0.125. Note that we only replace similarities from the previous iteration if the new similarity value is higher. This is why our similarity for "mother" to itself remains at 1 and does not sink to 0.5. The state of the WSM after the first iteration is shown in Figure 7-D.

Figure 7-D WSM for *grasp*: 1st Iteration



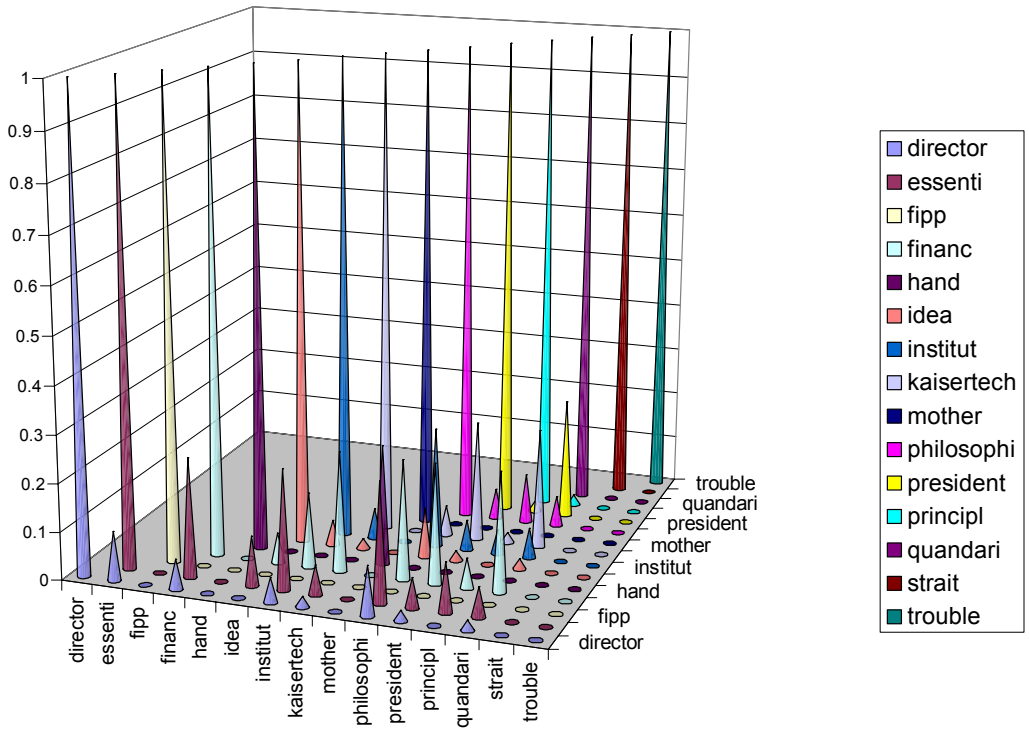
This concludes the first iteration. In subsequent iterations, we first update the Nonliteral Feedback SSM and then the Literal Feedback SSM from the WSM, as shown in Figure 7-E.

Figure 7-E SSMs for *grasp*: 1st Iteration



We then update the WSM, first from the Nonliteral Feedback SSM and then from the Literal Feedback SSM. We can see the results in Figure 7-F.

Figure 7-F WSM for *grasp*: 2nd Iteration



The reason for doing both SSM updates in sequence, rather than following each SSM update with a WSM update, is that we want the literal and nonliteral matrices to stand on an even footing. If we were to update the WSM immediately after updating the Nonliteral Feedback SSM, there would be word similarities available to the Literal Feedback SSM that were not available to the Nonliteral Feedback SSM.

Already after the second iteration we can see the transitivity of similarity at work. If we were to cluster simply based on words in the original sentence appearing somewhere in one of the feedback sets, we would correctly attract the sentence “she grasped her *mother*'s *hand*” to the literal feedback set due to the sentence “his aging *mother* gripped his *hands* tightly.” Also, using

only direct attraction, the sentence “he thinks he has grasped the essentials of the *institute*'s finance philosophies” is correctly attracted to the nonliteral feedback set due to the sentence “this idea is risky, but it looks like the director of the *institute* has finally comprehended the basic principles behind it.” However, using just direct attraction, there would be no way to get the sentence “the president failed to grasp KaiserTech's finance quandary” into the nonliteral cluster because it has no words in common with the nonliteral feedback set. Fortunately it can be *indirectly* attracted to the nonliteral feedback set through its sharing of the word “finance” with an original sentence that *is* attracted to the nonliteral feedback set. The reason for this is as follows: the original feature sets “essenti institut financ philosophi” and “president kaisertech financ quandary” are similar because they both contain the word “financ”. The words “financ” and “institut” are similar because they are contained in similar sentences. Both original sentences are then attracted to the feedback set sentence “idea director institut principl” because they contain similar words – not identical, but similar.

At the end of each iteration, we are able to read from the SSMs the similarity of each original sentence to each feedback sentence. We take the highest of these similarities for each original sentence to a literal feedback set sentence and to a nonliteral feedback set sentence. These similarity values are used for *clustering*.

Clustering refers to the attraction of each original sentence to either the literal or the nonliteral feedback set. The reason we cluster at the end of each iteration rather than after the algorithm has converged has to do with the addition of *learners* (to be discussed in Section 7.1.2.2.1). In order to decide which cluster an original sentence should belong to, we abide by the following decision process: if the similarity to either feedback set is below a given threshold, or the absolute difference between the highest similarity to the literal feedback set and the highest similarity to the nonliteral feedback set is below a given threshold, then no decision can be made at the current iteration and the original sentence is considered to be attracted to the *undecided*

cluster. Otherwise, if the highest similarity of the original sentence to the literal feedback set is greater than the highest similarity of that sentence to the nonliteral feedback set, then the sentence is considered to be part of the *literal* cluster. If, on the other hand, its similarity to the nonliteral feedback set is greater, then we add it to the *nonliteral* cluster. Note that we are *not* at this point pulling any sentence out of the running, so to speak. We are simply making a temporary decision to allow for *voting* between the learners (see Section 7.1.2.2.2).

At the end of each iteration, we calculate the greatest change in similarity values. As long as this value is above a given threshold, we continue iteratively updating. The threshold was set experimentally to ensure that the algorithm stops after a reasonable (optimistically below 12) number of iterations.

Due to the simplicity of our current example, there are only four iterations – three with changes and one to meet the stop condition. Figure 7-G and Figure 7-H show the final states.

Figure 7-G WSM for *grasp*: 4th Iteration

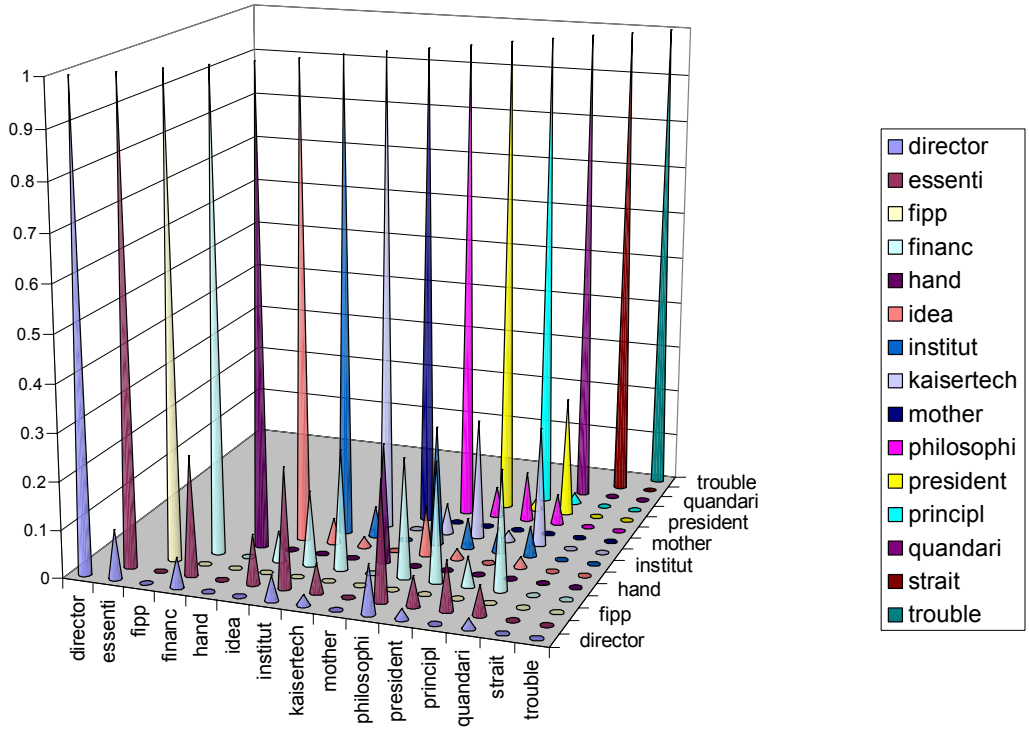
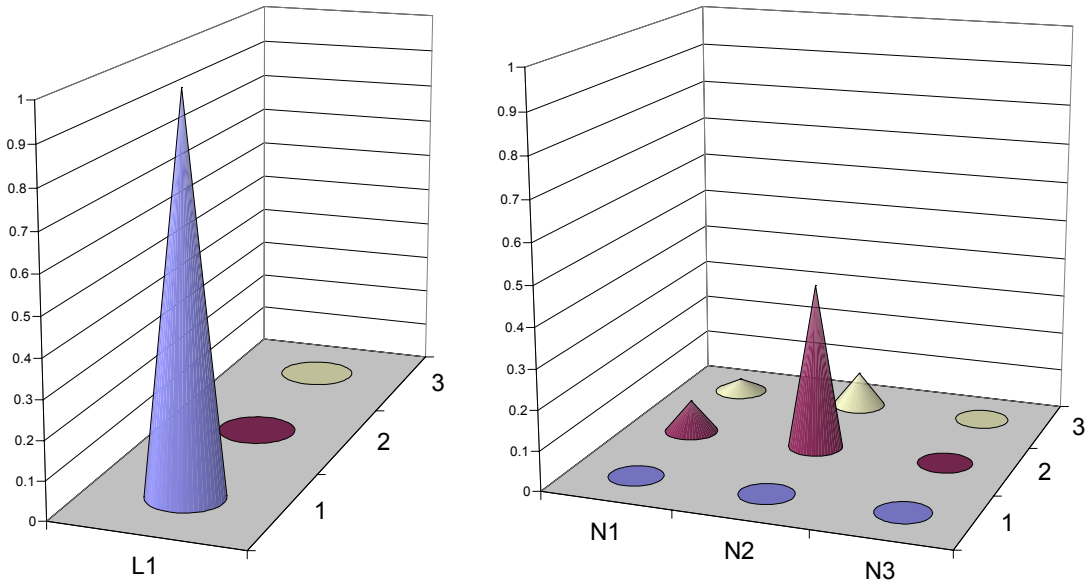


Figure 7-H SSMs for *grasp*: 4th Iteration



As we can see, by the time the program finishes running, we have established the similarity of our two nonliteral original sentences to another of the nonliteral feedback set sentences, namely to “after much thought, he finally grasped the idea”. This fact may not seem particularly significant at the moment because we are only interested in the highest similarity and we have only a few sentences. However, it could have been extremely important had we had many sentences, because through this new similarity we might have found additional similarities. Also, we shall see that the *number* of feedback set sentences to which an original sentence is attracted is highly significant for one of our TroFi enhancements (see Section 7.1.2.1).

We have just described the basic TroFi algorithm. Unfortunately, attraction to the correct feedback set is not quite that simple. Because of feature overlap and insufficient attraction, there will often be sentences that are attracted equally to both sets, as well as sentences that are attracted to neither set. Also, there will always be cases where an original sentence is attracted to the wrong set entirely. For example, let us change the original set to contain the following:

> The girl and her brother grasped their mother's hand.
> He thinks he has grasped the essentials of the institute's finance philosophies.
> The president failed to grasp KaiserTech's finance quandary.
> →
> **1a** L girl brother mother hand
> **2a** N essenti institut financ philosophi
> **3a** N president kaisertech financ quandari

The literal feedback set will now read thus:

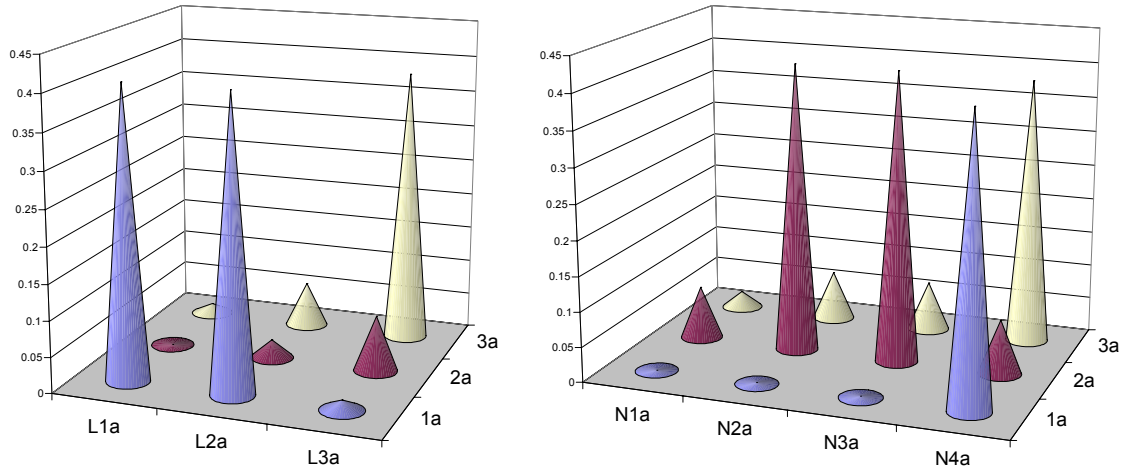
> The man's aging mother gripped her husband's shoulders tightly.
> The child gripped her sister's hand to cross the road.
> The president just doesn't get the picture, does he?
> →
> **L1a** man mother husband shoulder
> **L2a** child sister hand cross road
> **L3a** president

Finally, the makeup of the nonliteral feedback set will be as follows:

> After much thought, he finally grasped the idea.
>
> This idea is risky, but it looks like the director of the institute has finally comprehended the basic principles behind it.
>
> Mrs. Fipps is having trouble comprehending the legal straits of the institute.
>
> She had a hand in his finally fully comprehending their quandary.
>
> →
>
> **N1a** idea
>
> **N2a** idea director institut principl
>
> **N3a** fipp trouble strait institut
>
> **N4a** hand quandari

Note that "picture" does not show up as a feature in the above examples. It is part of the expression verb "get the picture", a synonym of "grasp". As part of a seed, it is omitted from the feature set. The final results of running TroFi on these expanded sets are show in Figure 7-I. The labels in the following graphs refer back to the revised feature sets above.

Figure 7-I SSMs for *grasp* extended: 6th Iteration



Looking at the above graphs it is quite obvious that we got Trouble, we got lots and lots of Trouble, with a capital "T" and that rhymes with "P" and that stands for Precision. Note that since both feedback sets contain the words "president" and "hand", we get a sort of tug-o'-war.

In both the case of “the girl and her brother grasped their mother's hand” and “the president failed to grasp KaiserTech's finance quandary,” we have a tie, meaning that this basic version of TroFi will not be able to add them to either the literal or the nonliteral cluster.

Luckily this is only the basic implementation of TroFi. In the following sections we walk through some methods and algorithms designed to enhance the basic algorithm and hopefully remedy problems like the above.

7.1.2 Enhancements

There are a number of ways in which we can enhance the basic TroFi algorithm, including:

1. Sum of Similarities vs. Highest Similarity
2. Learners and Voting
3. SuperTags
4. Context

We will discuss each of these in turn in the following sections.

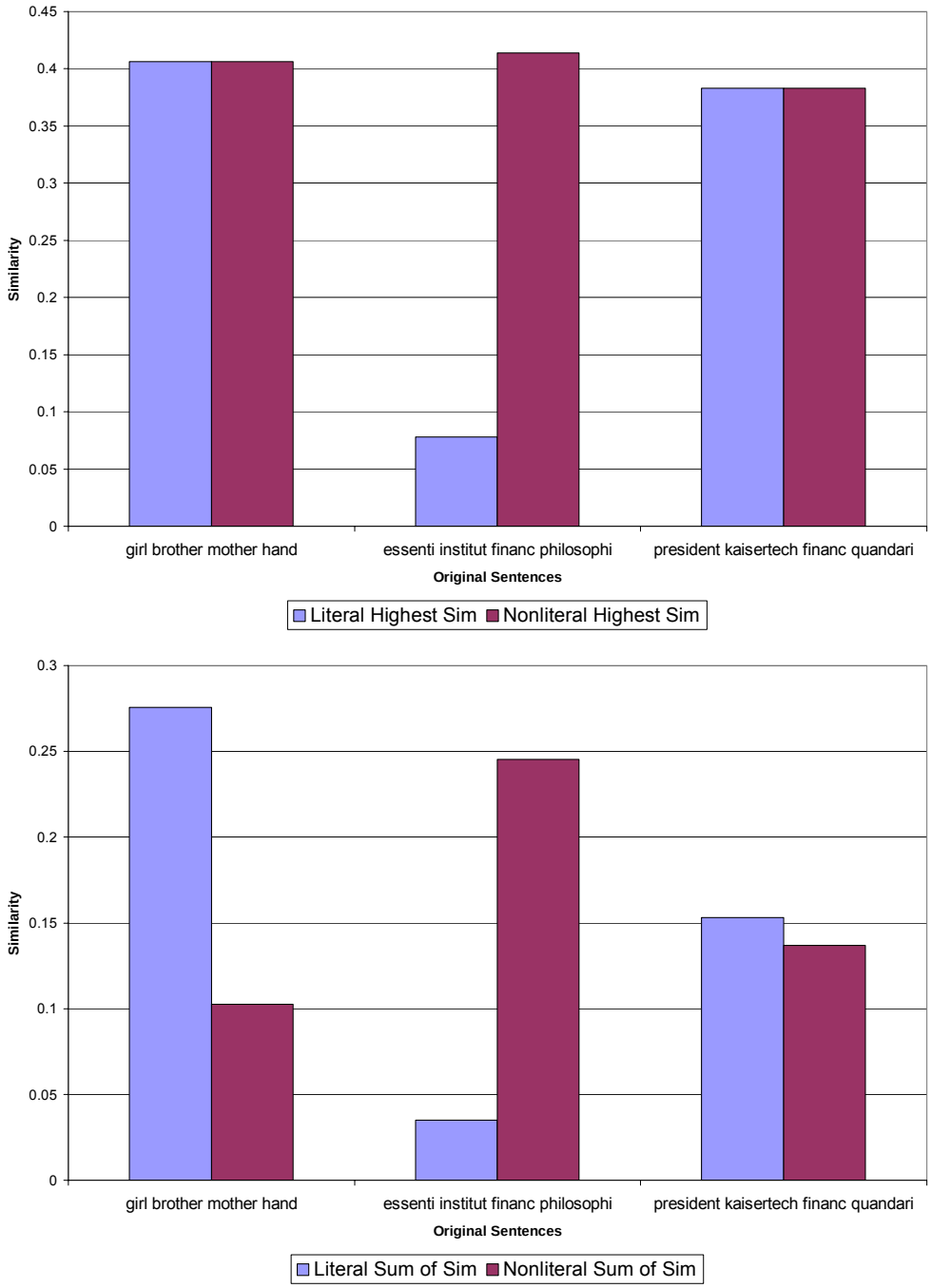
7.1.2.1 Sum of Similarities vs. Highest Similarity

Currently we base the strength of attraction of an original sentence to a feedback set on the highest similarity of the original sentence to any of the sentences in the feedback set. This is designed to eliminate noise: any feedback set sentences showing a spurious similarity to the original sentence will ideally be ignored. This makes sense for homogeneous sense sets with a fairly limited scope. Unfortunately, the literal and nonliteral feedback sets often encompass a large number of senses and a large number of domains. It may not be enough to depend on individual feedback set sentences. One way to remedy the situation is to measure attraction based not on highest similarity values, but rather on sum of similarities values.

Recall that for highest similarity values, at the updating stage, TroFi returns the highest similarity value between the original sentence and any of the feedback set sentences. In order to calculate the *sum of similarities*, TroFi simply sums over all the similarities displayed by each original sentence to any of the feedback set sentences. In order to make this sum comparable to other sums, and to ensure that the individual similarities do not add up to more than 1, we normalize by dividing each individual similarity score – i.e. the similarity of each original sentence to each feedback set sentence – by the total number of sentences in that particular feedback set. We also have to make an adjustment to the similarity threshold below which sentences are sent to the undecided cluster. This threshold is also divided by the total number of sentences in each feedback set. If we did not do this, most sentences would end up in the undecided cluster most of the time.

Summing over similarities will not only ensure that we do not ignore some vital sense information, it will also decrease the incidence of ties between the literal and nonliteral feedback sets. We accept the risk that any attraction, no matter how unfortunate, could affect the results. Figure 7-J shows how our "grasp" example behaves with the sum of similarities enhancement.

Figure 7-J Final Cluster Attraction Using Highest Similarity vs. Sum of Similarities



As we can see, for the second original sentence, “essenti institut financ philosphi”, using sum of similarities gives us the same clustering behaviour as using highest similarity. The sentence is correctly attracted to the nonliteral feedback set in both cases. However, recall that for the first and last original sentences, using highest similarity gave us a tie. Using sum of

similarities, on the other hand, we get the tie-breaker we were looking for – correct for the first sentence, incorrect for the last. The reason for the incorrect decision is that, although we have a direct attraction to both sets because the word "president" is in the literal feedback set and the word "quandary" is in the nonliteral feedback set, the attraction to the nonliteral feedback set is weakened slightly by the fact that there are more sentences in the nonliteral feedback set. This is a difficult problem, but we will attempt to overcome it with some of our other enhancements.

We can see that using sum of similarities definitely has an effect on the clustering results – even if it is not always the effect we want. The effect of the two approaches is likely to vary from target word to target word, depending on the nature of the original sentences and the feedback sets. The choice between using highest similarity and sum of similarities will therefore be made experimentally. The results are discussed in Section 8.3.

7.1.2.2 Learners & Voting

One of the biggest stumbling blocks for TroFi is noise in the feedback sets causing false attraction. To remedy this, we created a number of algorithms for *scrubbing* the noise out of the feedback sets, resulting in a variety of *learners*. (See Section 6.2.2.1 for details.)

One of the issues with learners is that since some scrubbing methods may cause incorrect moves or removals, the benefit of learners is maximized if they are used in combination to enhance each other's strengths and minimize each other's weaknesses. For this reason we have developed a voting schema allowing for a contribution from each learner. We discuss the effects of the individual learners in Section 7.1.2.2.1 and examine the voting system in Section 7.1.2.2.2.

7.1.2.2.1 Learners

In Section 6.2.2.1, we described the creation of the various learners that can be used by TroFi. In the following sections we examine the effect of each learner on the TroFi algorithm. We demonstrate using our running example.

7.1.2.2.1.1 Learner A

Recall that Learner A was produced by moving WordNet synsets from the literal to the nonliteral feedback set on the basis of phrasal/expression verbs and overlapping words. We want to move words and phrases likely to cause false attraction to the opposite feedback set. To see the basic effects this alternative learner has on TroFi, we simply replace the basic learner with Learner A, and re-examine our running example. (See Chapter 8 for detailed results.)

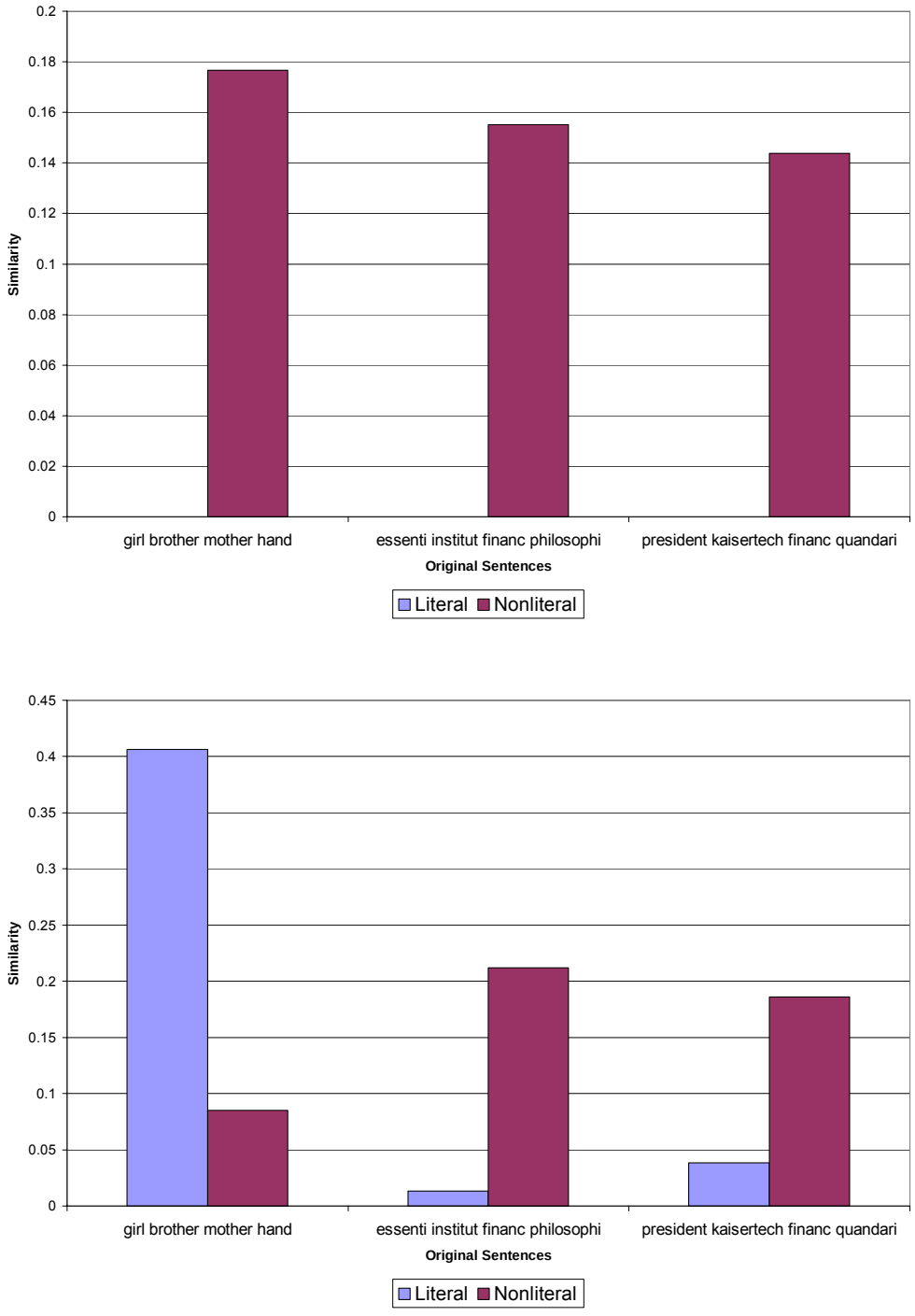
The WordNet entry for "grasp" is:

1. grasp, grip, hold on -- (hold firmly)
2. get the picture, comprehend, savvy, dig, grasp, compass, apprehend -- (get the meaning of something; "Do you comprehend the meaning of this letter?")

To create Learner A, the synsets of "grasp" containing phrasal or expression verbs are moved to the nonliteral set. Unfortunately, both synsets of "grasp" contain phrasal or expression verbs, meaning that our literal feedback set is empty and all the original sentences will be attracted to the nonliteral set. This means that the nonliteral original sentence "president kaisertech financ quandari" is now attracted to the nonliteral set, but so, unfortunately, is our literal example, "girl brother mother hand".

Let us speculate for a moment about what would happen if the first synset did not contain the phrasal verb "hold on". In that case, only the second synset, the one containing the expression verb "get the picture" would be scrubbed. The result would be that our feedback set sentence "president" would end up in the nonliteral feedback set where it could correctly attract "president kaisertech financ quandari". The first graph in Figure 7-K shows the results given by the *actual* Learner A; the second, by the *hypothetical* Learner A.

Figure 7-K Final Cluster Attraction Using Learner A and *Hypothetical* Learner A



In the non-hypothetical case, Learner A allows us to correctly cluster a sentence that the basic learner cannot handle. However, it does cause one of our other sentences to be classified

incorrectly. We must investigate the remaining learners to see if we can reap the advantages of Learner A while cancelling out its negative effects.

7.1.2.2.1.2 Learner B

Learner B is similar to Learner A. The only difference is that rather than moving entire synsets on the basis of phrasal/expression verbs and overlapping words, we simply remove them. This has the effect of scrubbing false attraction from one feedback set without accidentally introducing noise into the other.

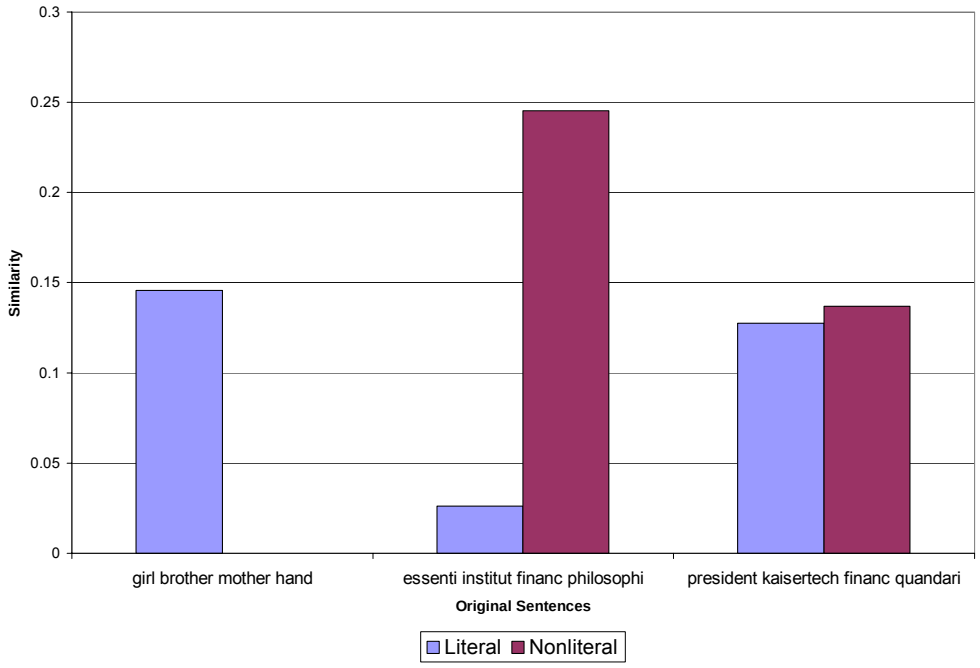
In terms of our "grasp" example, we still end up with an empty literal feedback set, but at least we are not adding noise to the nonliteral feedback set. Perhaps Learner C (C^1 or C^2) will give us what we need.

7.1.2.2.1.3 Learner C^1

The Learners C are a little less drastic than Learners A and B. Rather than moving or removing whole synsets, which, as we have seen, can have hefty effects, for Learners C^1 and C^2 we just remove features or feature sets, respectively, from the final feedback sets on the basis of overlapping words. For Learner C^2, we remove the whole feature set containing an overlapping word. For Learner C^1, we just remove the offending word itself.

Applying Learner C^1 to our running example, we find that the word "hand" is scrubbed out of both feedback sets. Figure 7-L shows the effects on the final results.

Figure 7-L Final Cluster attraction using Learner C^1



We expected that removing the word "hand" from both sets would break any potential ties in the case of the sentence "girl brother mother hand". However, it has also solved the rest of our attraction problems as a side-effect. Due to the transitivity of similarity, changing a single word will often have far-reaching effects.

7.1.2.2.1.4 Learner C^2

For Learner C^2, instead of removing overlapping words, we remove whole feature sets containing those words.

In terms of our "grasp" example, Learner C^2 would cause the feature set containing the word "hand" – namely "child sister hand cross road" – to be completely removed. We have found that, in general, Learner C^2 yields slightly better results than Learner C^1 over a number of target words, so that is the one we use as the TroFi default.

7.1.2.2.1.5 Learner D

Learner D is the *basic* learner we have been referring to during the course of this discussion. It is completely unscrubbed, and hence at the same time unenhanced and unbroken. It serves both as the baseline against which the other learners can be compared and as an important anchor for the voting system.

7.1.2.2.2 Voting System

We now have four learners with four different opinions. The question is, how do we use those opinions to our best advantage? The simplest thing, of course, is to compare them and choose the one exhibiting the best performance. Unfortunately, when we try to do this (see Section 8.4) we find that some learners produce better results for some words and worse results for others. In the end, our experiments found no marked difference between the average accuracies produced by the learners. As previously suggested, we need to be able to take the best from each learner. We accomplish this by letting them vote.

We initially experimented with a variety of different voting schemata – for example, *all agree*, *best 3 of 4*, *at least one* – and used a backoff model to pick a set of results. This became too complex too quickly, the logic was non-trivial to justify, and the results were questionable. Therefore, we decided to go with a simple *majority-rules* schema. We provide the pseudo-code below, followed by a more detailed explanation.

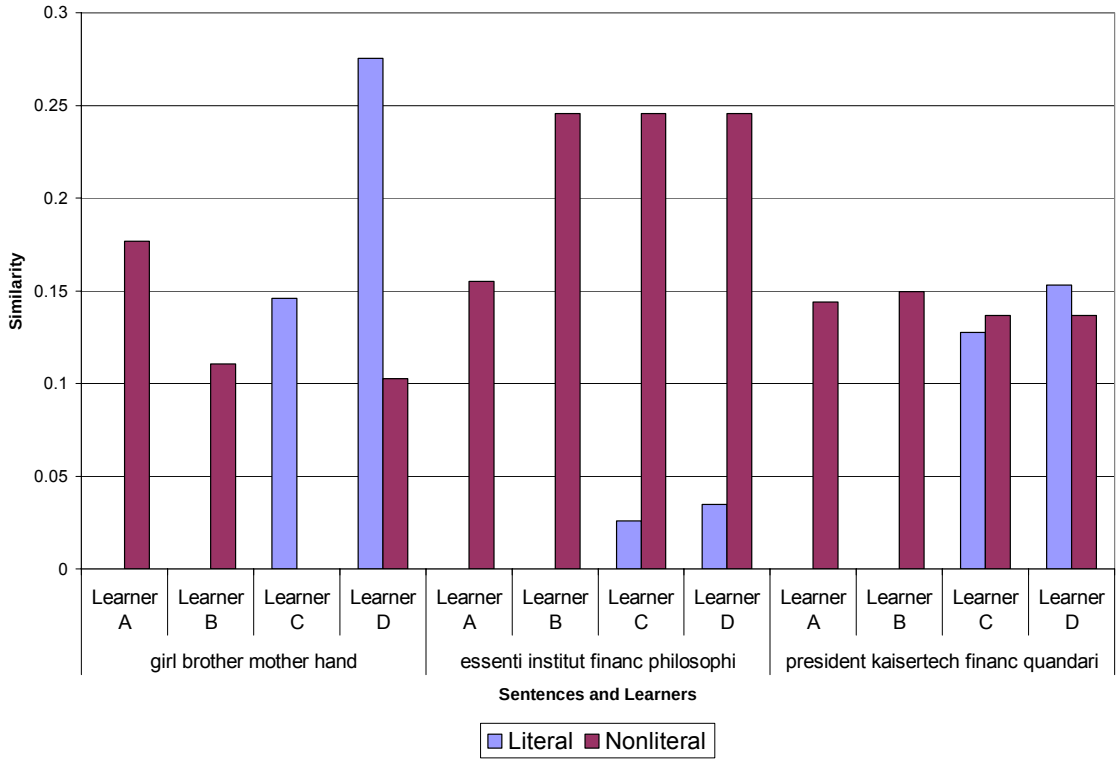
```
&3of4Voter

for each original sentence
       if nonliteral for 3 of 4 learners
              add to nonliteral cluster
       else if literal for 3 of 4 learners
              add to literal cluster
       else
              add to undecided cluster
```

We have gone from the extremely complicated to the extremely simple. If three of the four learners say that a given original sentence should be nonliteral, we put it in the nonliteral

cluster; if three of four say it should be literal, we put it in the literal cluster; if three of four cannot make up their mind or there is a tie, we put it in the undecided cluster.

The voting results are given in Figure 7-M.

Figure 7-M Voting Results



Unfortunately, we have a tie for our hotly contested first example. We could potentially improve results by giving the learners different weights. There are some experiments to this effect in Section 8.5. Also, we still have a few enhancements that might have a positive effect.

7.1.2.3 SuperTags

The addition of SuperTags (see Section 6.2.2.2) requires no special behaviour from TroFi. The SuperTag trigrams are just another feature. The effects of SuperTags on the accuracy of the TroFi output will be more fully explored in Section 8.6. For now, suffice it to say that the

SuperTag trigrams can provide beneficial additional similarity between sentences, sometimes providing information – like prepositions and nominal arguments – that would otherwise be ignored. Like all features, the SuperTag trigrams can be either a benefit or a detriment. In order to limit false attraction we have kept the contents of the SuperTag trigrams rather constrained, as described in Section 6.2.2.2.

7.1.2.4 Context

As discussed in Section 6.2.2.3, the context enhancement simply involves the expansion of the feature sets to include the sentences immediately preceding and following the sentence containing the target word. The purpose of this is to increase the size – and therefore the attractiveness – of our feature sets. Like the SuperTag trigrams, these additional features require no special treatment.

Unfortunately, the context enhancement does face a stumbling block in terms of speed. TroFi currently uses an $O(n^2)$ implementation, where n is the total number of words (with duplicates removed) in the union of the original and feedback sets. This means that increasing the size of the feedback sets by adding extra context slows down processing considerably. The optimization of the algorithm for better scalability is left for future work.

7.2 Active Learning

We have seen in our "grasp" example, and we will see in our experiments in Chapter 8, that there is only so far we can go with the unsupervised algorithm. It would be nice to be able to push the results a little further. For this reason we have added an optional active learning component to TroFi.

One way to look at the TroFi active learning component is as damage control. Early in the classification process, one could attempt to nudge things in the right direction by sending a

certain percentage of the sentences that TroFi is not sure about to a human helper. These sentences could then be added to the appropriate feedback set to provide a certain point of attraction for additional sentences that might not otherwise have been attracted to the correct feedback set.

Another way to look at active learning is that, rather than the human helping TroFi do its automatic clustering, TroFi is helping the human do manual clustering. TroFi takes the process as far as it can, leaving the human with a reduced set of sentences for manual annotation.

Whichever way we look at it, one of the concerns about using active learning is that we do not want to impose too much work on the human. After all, TroFi is supposed to be a labour-saving device. The goal of the active learning component, therefore, must be to minimize effort and maximize impact. Several issues must be addressed in order to accomplish this. First, we must make a decision on how much work, at most, the human should ever have to do. Then we must ensure that we send the most appropriate sentences possible to fill this quota. Finally, we must use the human expertise to our best advantage by applying it at the point in the algorithm where it will have the most beneficial effect.

We experimented with several methods for deciding what sentences would be sent to the human. For a start, they are always sentences from the undecided cluster. As we learned in Section 7.1.1.2, TroFi puts sentences in the undecided cluster if their attraction to either feedback set, or the absolute difference between the attractions to the two feedback sets, falls below a given threshold. We toyed therefore with the idea of sending everything in the undecided cluster to the human and controlling the size of the cluster purely with the similarity threshold setting. We found after some experimentation, however, that we got great variation between different target words, and that we were often sending an unreasonable percentage of the sentences to the human. To counteract this, we added a cap on the percentage of input sentences that TroFi would be allowed to send to the human. We then found that we were not always sending the most

uncertain sentences. Also, we were often not filling the allowable quota. For this reason, we raised the similarity threshold so that practically anything could be sent to the undecided cluster and we imposed an order on these sentences based on similarity values. The result is that those original sentences showing the least similarity to either feedback are sent to the human first.

In order to make certain TroFi settings work, it is imperative that not all the sentences sent to the human have similarity values or absolute differences of zero. The reason is that if nothing is attracted to a given original sentence during the first iteration, moving it into a feedback set is not going to help either. On the other hand, not clustering a sentence counts negatively in the evaluation, and sending a rogue sentence to the human might be its only chance. We solve this dilemma through compromise: we send alternating positive similarity values (starting from the lowest) and zeros.

After TroFi receives a decision from the human, it moves the sentence in question not only into the correct cluster, put also into the corresponding feedback set. Some experimentation revealed, however, that the positive attraction of these sentences was not enough to effectively counteract the existing false attraction generated by feedback set sentences placed in the wrong set right from the start. Thus we tried moving all the feedback set sentences showing a similarity to the human-judged sentence to the *correct* feedback set as well. Unfortunately, with most large sets, this resulted in having to move every single feedback set sentence from one feedback set to the other, meaning that one of the feedback sets ended up empty. We tried various percentages, but finally decided that the negligible potential benefit was not worth the risk.

After deciding *what* sentences to send to the human, we must decide *when* to send them. One possibility is to wait until the last iteration in the hope that TroFi correctly clustered everything else. This immediately destroys any *bootstrapping* possibilities: nothing can be *learned* from the human input. For this reason, we experimented with having TroFi send the sentences to the human earlier. There are several possibilities: send the whole quota right after

the first iteration, send the whole quota at some later iteration, or send a small fraction of the quota at each iteration.

The potential benefit of sending everything after the first iteration is to have at least a small chance of nipping any false attractions in the bud. Also, as mentioned earlier, the algorithms can be expected to learn from the human decisions. Risks include sending sentences to the human prematurely – i.e. before TroFi has had a chance to make potentially correct decision about them. Because of some flipping back and forth of sentences between clusters while the program runs, it is difficult to predict when TroFi will have made its final decision.

Karov & Edelman suggest that nothing really good happens after the third iteration. We have found this to be true in some cases and not in others. The main issue is the similarity values tend to keep increasing with each iteration to eventually converge. This means – particularly since we are using sum of similarities – that eventually the numbers may increase in such a way as to make an original sentence flip from one cluster to the other, or from the undecided set into the literal or nonliteral cluster. We have discovered however that the general *order* of similarity values tends not to change very much, if at all, after the third iteration. This suggests that right after the third iteration may be a good time to send the sentences to the human. Allowing the human to put a hand in at this point could give the algorithm a second chance.

A third approach is to send the quota of sentences to the human in small doses in order to gain some benefit at each iteration – i.e. the certainty measures will change for the better with each bit of human input, so at each iteration more and more appropriate sentences will be sent to the human. This is a type of *bootstrapping*. Ideally, we would get a compounding of benefits. On the other hand, we could also get a compounding of risks.

The risks of the TroFi active learning component are similar to the risks we discovered when looking at using context as an enhancement (see Section 7.1.2.4). The sentences sent to a particular cluster by the human may themselves be correctly classified, but they might contain

misleading features that will cause incorrect attractions. We currently have no way to minimize this risk. Also, as mentioned above, sending sentences to the human in a distributed manner may compound this problem.

We have covered the active learning algorithm loosely in the above paragraphs. A more structured version can be found in the pseudo-code in Appendix A.

Looking back at our "grasp" example, we can see that active learning gives us a chance to save our problematic first example, "the girl and her brother grasped their mother's hand." Voting produces a tie for this sentence (see Figure 7-M), so it ends up in the undecided cluster. Since it is the only sentence there, it is sent to the human, who can identify it as literal and place it into the literal cluster.

In Section 9.2, we will experimentally examine the relative merits of sending all the sentences after the first iteration, after the third iteration, at the end, or distributed across all the iterations. We will also take a look at what happens if we simply select random sentences to send to the human. Finally, we will discuss the value provided to the human by TroFi.

7.3 Iterative Augmentation

Most statistical NLP algorithms include a training phase. We have not yet made any reference to training TroFi. The reason is that we are not specifically trying to train the algorithm to work on unseen examples. Rather, we are trying to separate our input into two distinct sets, which, in theory could then be used, among other things, as training data for a statistical classifier. In a way, we are *creating* training data. Still there may be a place for a sort of training phase in TroFi as well.

We can consider TroFi to have a training phase only insofar as one can refer to learning from previous runs as training. A more appropriate term here may be *iterative augmentation*. We want to be able to add to our literal and nonliteral clusters over time. We would also expect

to be able to use the knowledge we have gained in previous runs, and not start from scratch each time. We can do this by adding our new clusters to the appropriate feedback sets at the end of each run, and saving the results for future runs. In this way we can augment both the clusters and the feedback sets over time; hence the term *iterative augmentation*.[3]

Benefits of iterative augmentation are that it will allow us to compensate for current speed limitations. As we suggested when discussing the addition of context, since the current implementation is $O(n^2)$, where n is the total number of words, drastically increasing the size of the initial input sets will grind the algorithm to a halt. However, working with smaller input sets in an iterative fashion will keep the algorithm from becoming unusable.

When TroFi is run initially, in addition to the clusters, we output a set of sentences, which we will, for the sake of convenience, call a *classifier*. The idea behind this classifier is that we save the information gathered during the initial run in two ways. First, we add the newly classified sentences to their respective feedback sets with a high certainty (weight). Second, we add certainty measures to the actual feedback set sentences by giving them a weight corresponding to their highest similarity to any of the original set sentences plus epsilon. The epsilon value is a small number used to ensure that none of the weights is zero, so that none of the sentences is dropped out of the clustering algorithm altogether. Just because a particular feedback set sentence does not attract any original set sentences in one run does not mean that it will not attract any in future runs. We simply want to make certain that those feedback set sentences that proved useful during the initial runs are given higher importance in subsequent runs. The pseudo-code for building the classifier is provided in Appendix A.

It is not difficult to see the potential pitfalls in the proposed algorithm, namely that incorrect attractions will be compounded in future runs by using the classifier. One way to

[3] Note that active learning (see Section 7.2) is optional for augmenting the classifier. If active learning is used, the decisions about the undecided cluster become more reliable, and so both the augmentation of the literal and nonliteral clusters and the revised versions of the classifiers will be more accurate.

minimize this problem might be to exclude sentences below a given certainty threshold from both the clusters and the classifier. This is left for future work.

Once we have a classifier we can use it in a slightly modified version of the TroFi algorithm. Instead of many learners and a voting system, we use a single learner. Also, instead of building new feedback sets, we use our newly created classifiers. We use a new original set of sentences and, as before, attempt to send each sentence to the appropriate cluster. At the end, we can add the freshly clustered sentences to our classifiers, giving us: *iterative augmentation*.

7.4 Summary

In this chapter we examined TroFi's models and algorithms on hand of an illustrative running example. We began with the core word-sense disambiguation algorithm upon which TroFi is built. We then walked through our collection of enhancements: sum of similarities, learners, voting, SuperTags, and additional context. Subsequently, we moved on to an analysis of the active learning and iterative augmentation algorithms.

In the following chapters we carry out extensive experimentation to determine the effectiveness of the various models and algorithms. We begin with the core algorithms and enhancements in Chapter 8, followed by the active learning component in Chapter 9. Iterative augmentation is further discussed in Chapter 10.

8 CORE EXPERIMENTS & RESULTS

In this chapter, we discuss the evaluation of the core TroFi algorithms. This includes a description of the baseline, a comparison of sum of similarities to highest similarity, a look at variations in learners and voting schemata, and an examination of adding SuperTags and/or additional context to the feature sets. Active learning experiments and results will be covered separately in Chapter 9.

8.1 Evaluation Criteria and Methodology

TroFi's core algorithms and enhancements were evaluated on the 25 target words listed in Table 8-A. (For information on how these words were chosen, please see Section 6.1.5.) The original sets for the target words contain anywhere from 1 to 115 sentences pulled from the test set of the WSJ Corpus. Some of the sets started out larger, but were randomly reduced to approximately 100 sentences. The sets were kept quite small to limit the amount of manual annotation required in preparation for testing. We provide the total number of sentences, plus the manually evaluated literal and nonliteral counts for each target word in Table 8-A. Keep in mind that one objective of this thesis is to deal with real-world data. This implies that the data is often noisy and certainly does not like to split nicely into half literal and half nonliteral. It can even happen that there are either no literals or no nonliterals. This imbalance may affect results.

Table 8-A Usage Counts per Word

	absorb	assault	die	drag	drown	escape	examine	fill	fix	flow
Literal	4	3	24	12	4	24	49	47	39	10
Nonliteral	62	0	11	41	1	39	37	40	16	31
Total	66	3	35	53	5	63	86	87	55	41

	grab	grasp	kick	knock	lend	miss	pass	rest	ride	roll
Literal	5	1	10	11	77	58	0	8	22	25
Nonliteral	13	4	26	29	15	40	1	20	26	46
Total	18	5	36	40	92	98	1	28	48	71

	smooth	step	stick	strike	touch
Literal	0	12	8	51	13
Nonliteral	11	94	73	64	41
Total	11	106	81	115	54

A variety of feedback sets were generated for different experiments. There are from one to around 1500 literal and nonliteral feedback set sentences per word for each of the experiments.

The algorithms were evaluated based on how accurately they clustered the hand-annotated sentences. Included in this evaluation was whether the algorithm managed to cluster the sentences at all. This addition was prompted by the question of what to do with sentences that are attracted to neither cluster or are equally attracted to both. We will call these *unknowns*. Several options came to mind: put all unknowns in the literal cluster; put all unknowns in the nonliteral cluster; ignore unknowns; or, put unknowns in the opposite set from their label – e.g., if the manual label says the sentence is literal, add the sentence to the nonliteral cluster. This approach allows us to evaluate the algorithms more fairly, since any failure to cluster a sentence is seen as an incorrect clustering. The *opposites* method is for evaluation purposes only, of course. In non-test situations we ignore unknowns to keep clusters as pure as possible.

Evaluation results were recorded as recall, precision, and f-score values. Given pre-annotated sentences, TroFi is able to automatically calculate *precision* and *recall* for each set of experimental results. *Literal recall* is defined as the percentage of all the literal original sentences that are correctly attracted to the literal cluster: $(literalsInLiteralCluster/totalLiterals)$. *Literal precision* is the percent of original sentences attracted to the literal cluster that are actually literal: $(literalsInLiteralCluster/sizeOfLiteralCluster)$. *Nonliteral precision* and *recall* are

defined similarly. The recall and precision values are used to calculate the *f-score* for each set, where *f-score* is defined as $(2 \cdot precision \cdot recall)/(precision + recall)$. To combine the literal and nonliteral results into an overall accuracy, we average literal and nonliteral precision and literal and nonliteral recall, and calculate a new f-score from these averages.

In cases where there are only literals in the original set, nonliteral recall is set to 100%. Nonliteral precision is considered to be 100% as long as no literals are attracted to the nonliteral cluster. If so much as one literal sentence is attracted to the nonliteral cluster, nonliteral precision becomes 0%. Literal precision and recall are calculated normally, although literal precision will of course be 100% by default. On the other hand, in cases where there are only nonliterals, literal recall is 100%, and literal precision is either 100% or 0% as defined above.

8.2 Baseline

The baseline accuracy for each target word is calculated using a simple attraction algorithm. Each original set sentence is attracted to the feedback set sentence containing the sentence with which it has the most words in common. This corresponds well to the basic *highest similarity* TroFi algorithm. Sentences that are attracted to neither set, or are equally attracted to both, are placed in the opposite cluster to where their manual label says they should be (see Section 8.1). We provide the pseudo-code for the highest-similarity baseline in Appendix A.

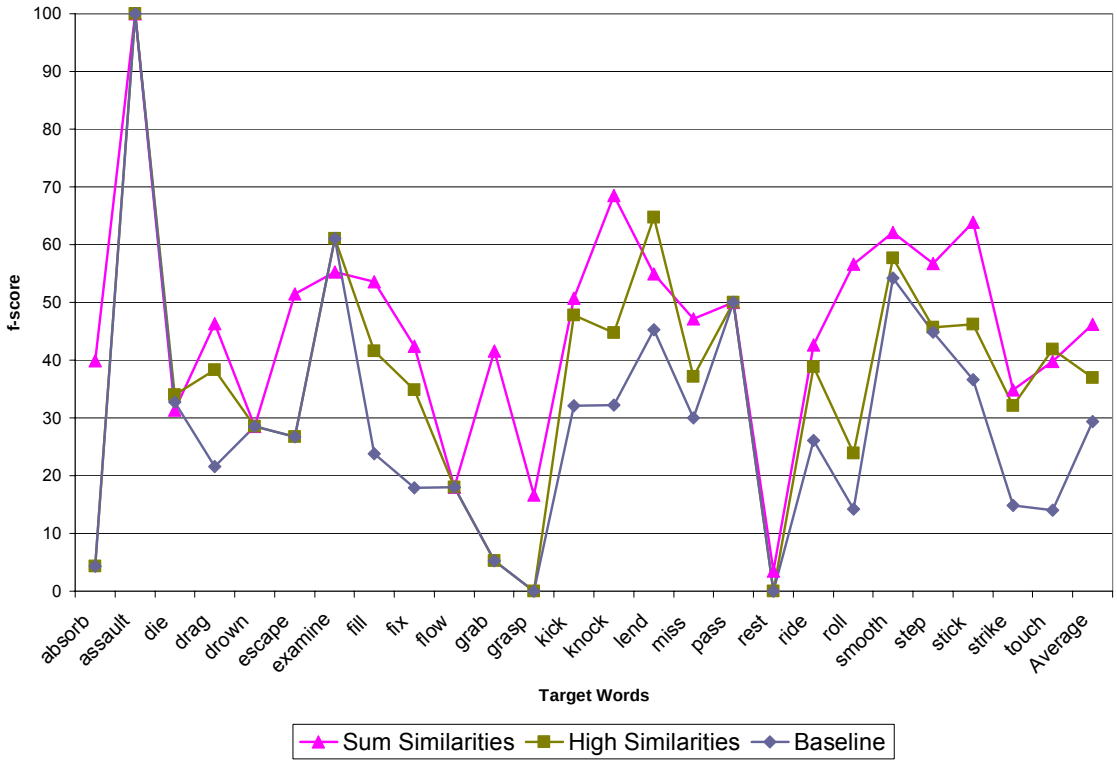
8.3 Experiment 1: Sum of Similarities vs. High Similarity

In our discussion of the TroFi algorithms in Chapter 7, we found that replacing highest similarity measures with sum of similarities measures had good effects on our simple example. In this experiment, we examine whether sum of similarities is more effective in general as well.

TroFi was run on the set of 25 target words first using highest similarity measures and then using the sum of similarities enhancement. Figure 8-A shows the results as compared to the

baseline. Note that although we do calculate the precision, recall, and f-score for both the literal and the nonliteral clusters – and the average – for all of our experiments, we only graph the average f-score for ease of viewing.

Figure 8-A Highest Similarity and Sum of Similarities Comparison



Let us first compare the TroFi highest similarity results to the baseline. We can see that the results produced by TroFi are significantly higher for some of the words, like "lend" and "touch" for example. This can be explained by transitivity of similarity (see Section 7.1.1.1). On average, the most basic TroFi algorithm gives us a 7.6% improvement over the baseline.

Next we examine the sum of similarities results. All the individual target word results except for "examine" sit above the baseline. The reason this point falls below the baseline is that while TroFi can generate some beneficial similarities between words related by context, it can also generate some detrimental ones. This means that when we use sum of similarities, it is

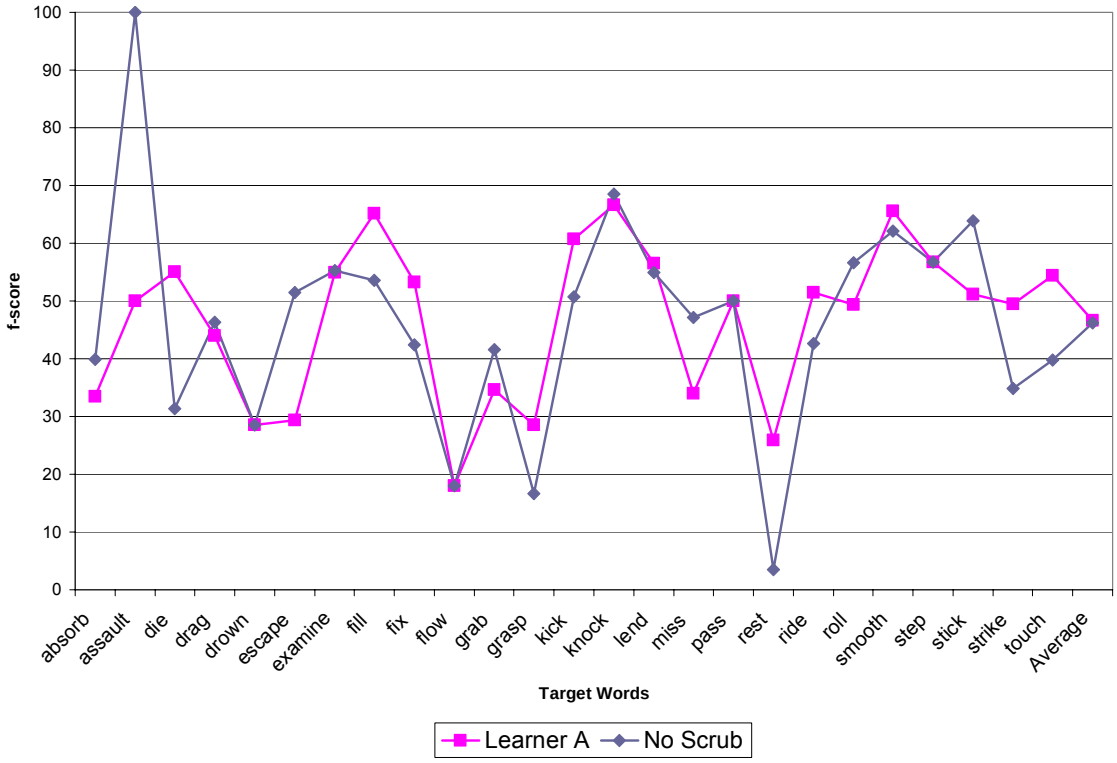
possible, for example, for the transitively discovered indirect similarities between an original nonliteral sentence and all the sentences in the literal feedback set to add up to more than a single direct similarity between the original sentence and a single nonliteral feedback set sentence. In such a case, the original sentence will be attracted to the wrong set. Dips below the baseline cannot happen in the highest similarity case because a single sentence would have to show a higher similarity to the original sentence than that produced by sharing an identical word. This is not possible because word and sentence weights generally prevent transitively discovered similarities from adding up to 1.

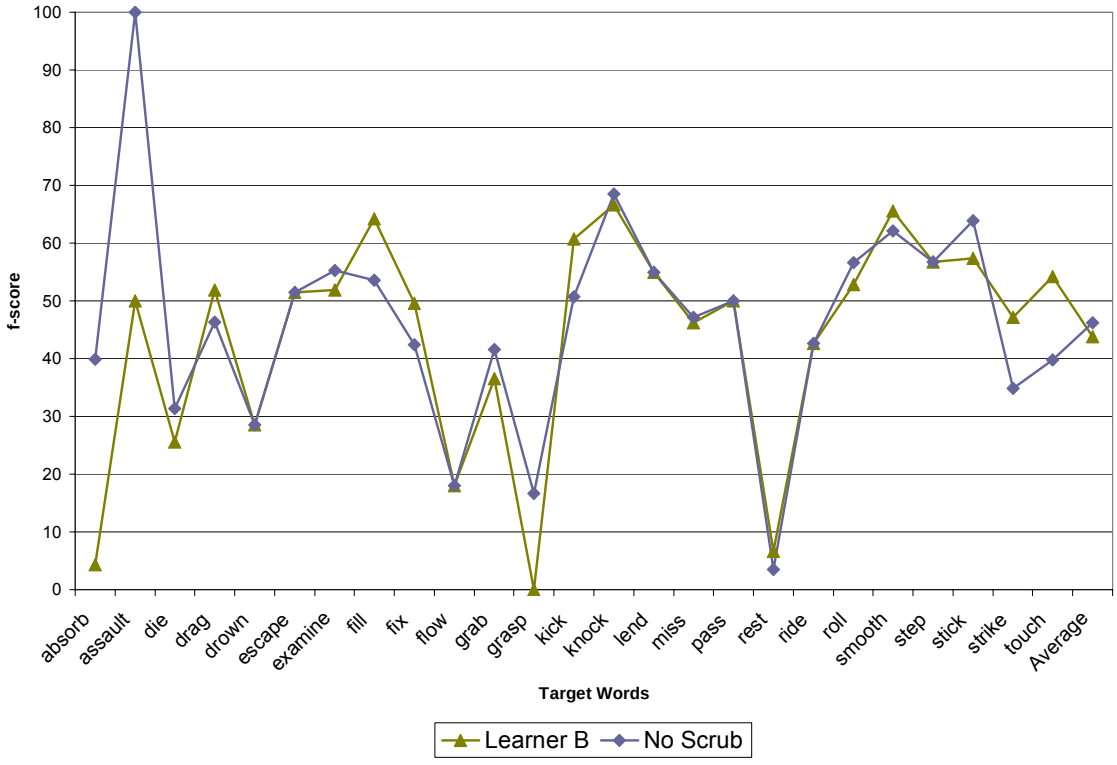
Given the above explanation, we should not be surprised to find that using highest similarity occasionally produces better results than using sum of similarities. However, on average we can expect to get better results with sum of similarities. In this experiment alone, we get an average f-score of 46.3% for the sum of similarities results – a 9.4% improvement over the average high similarity results (36.9%) and a 16.9% improvement over the baseline (29.4%).

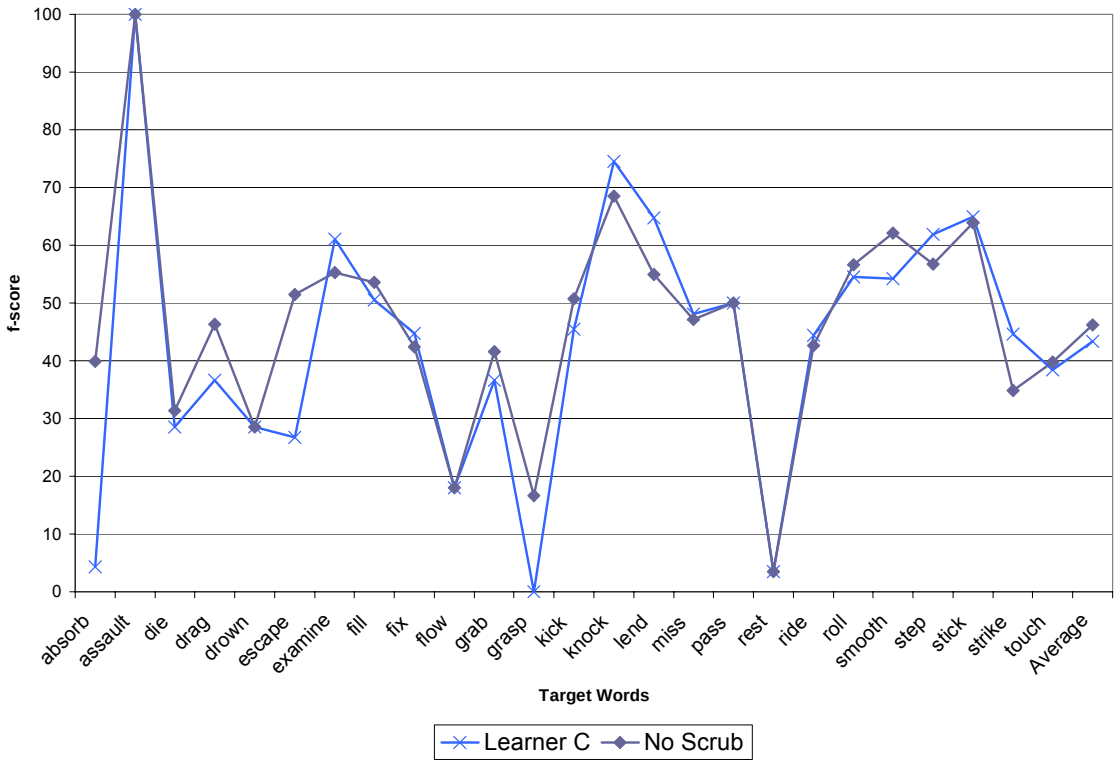
8.4 Experiment 2: Comparison of Learners

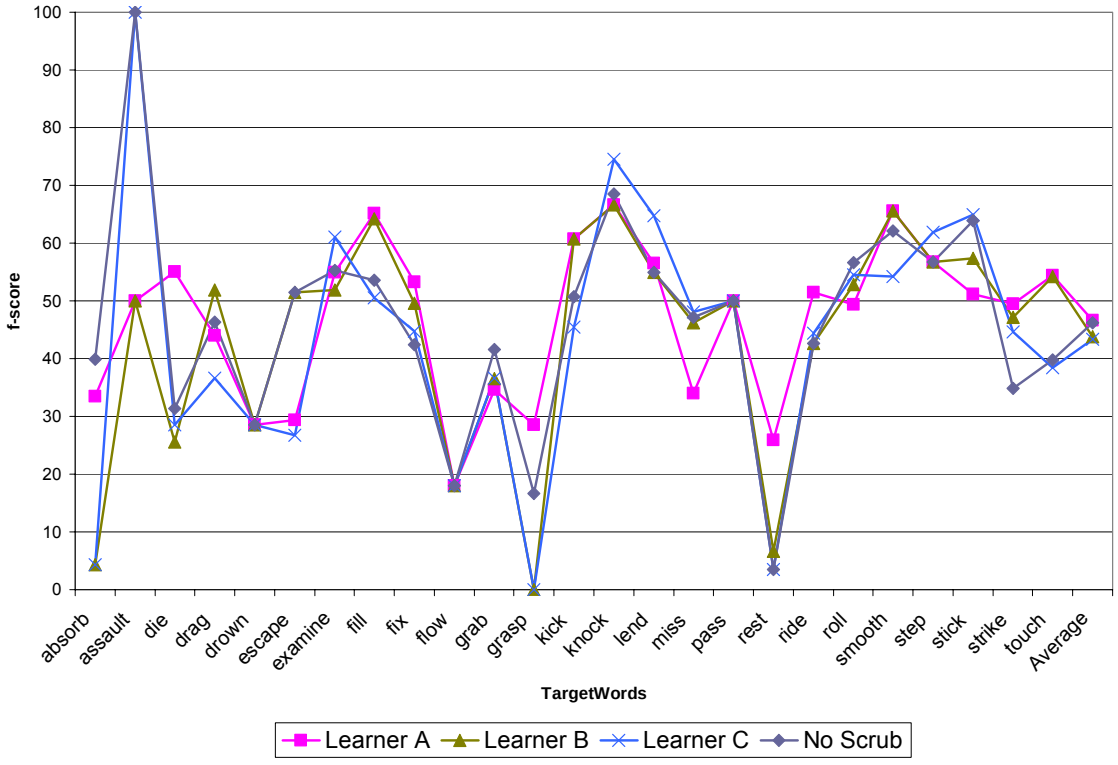
In Sections 6.2.2.1 and 7.1.2.2.1 we discussed in depth the composition and importance of learners. We can see from Experiment 1 in Section 8.3 that spurious sentences or words in a feedback set can have heavy repercussions: not only can they cause direct false attractions, but through transitive similarity their effects can spread out across the rest of the set. In this section, we show the difference that scrubbing can make by comparing Learners A, B, and C to the *no-scrub* learner, Learner D. Note that the highest similarity and sum of similarities results in Section 8.3 were produced using Learner D. We also compare all four learners to each other. The results are shown in Figure 8-B. We examine the effects of superimposing a voting schema in Section 8.5.

Figure 8-B No Scrub and Learner Comparisons






f-score
100
90
80
70
60
50
40
30
20
10
0
absorb
assault
die
drag
drown
escape
examine
fill
fix
flow
grab
grasp
kick
knock
lend
miss
pass
rest
ride
roll
smooth
step
stick
strike
touch
Average
Target Words
Learner C
No Scrub


f-score
100
90
80
70
60
50
40
30
20
10
0
absorb
assault
die
drag
drown
escape
examine
fill
fix
flow
grab
grasp
kick
knock
lend
miss
pass
rest
ride
roll
smooth
step
stick
strike
touch
Average
TargetWords
Learner A
Learner B
Learner C
No Scrub

We begin with Learner A. Recall that we create Learner A by moving whole synsets to the opposite feedback set on the basis of phrasal/expression verbs and overlapping words. As we can see, sometimes Learner A is better and sometimes the no-scrub learner is better, with the final averages being almost the same. We cannot expect Learner A to be uniformly superior across all target words because despite being able to move appropriate synsets to the opposite feedback set, its scrubbing algorithm is quite capable of occasionally moving inappropriate ones.

For Learner B, rather than *moving* the synsets, we simply *remove* them with the hope of eliminating potentially dangerous synsets from one feedback set without accidentally contaminating the other one. A quick glance will tell us that, as could be expected, the gains are not as great as for Learner A, but neither are the losses. Unfortunately, the average Learner B results sit 2.5% below the no-scrub average. We want to be able to eventually balance out the effects of Learners A and B and hopefully also add some slight improvement with Learner C.

For Learner C, rather than scrubbing synsets, we remove sentences from the feedback sets themselves on the basis of overlapping words. We found that removing the whole sentence gave us slightly better results than just removing individual words. As with the other two learners, we have some good, some bad, and some indifferent results, with the average falling 2.8% below the no-scrub results.

To gain some perspective on how the learners might play off against each other to enhance strengths and reduce weaknesses, we must study the graph comparing all four learners in Figure 8-B. The one thing that should be evident is that it would be difficult to claim that any one of the learners is independently superior to any of the others. Although Learner D (no scrub) and Learner A have a higher overall average, we can see that there are words for which they are soundly trumped by Learners B and/or C. To attempt to pull the best out of these variable results, we introduce a voting system.

8.5 Experiment 3: Effects of Voting

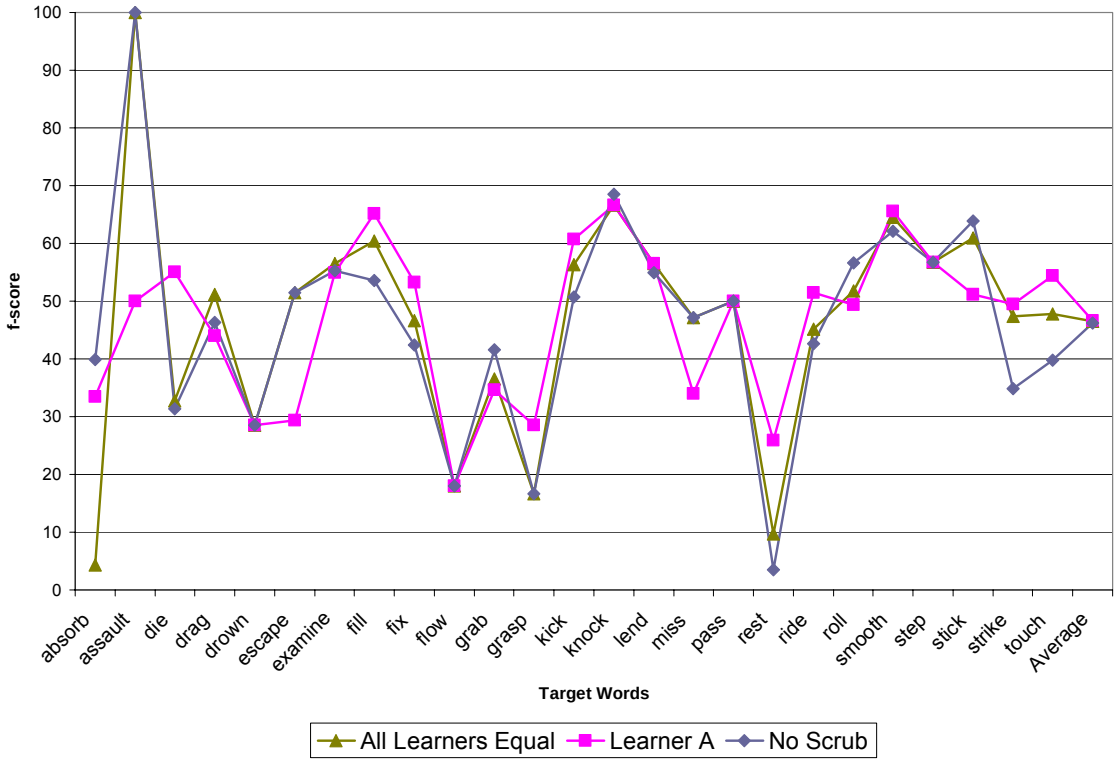
In this section we experiment with a voting system (see Section 7.1.2.2.2) to find a weighted schema that will allow us to eke out the best possible performance from our four learners. As described in Section 7.1.2.2.2, we begin with a basic majority-rules schema. If three learners decide that a given original sentence belongs in a particular cluster, they will override the fourth. If there is a tie, the fate of the sentence stays undecided until a default decision can be made or the sentence is sent to a human evaluator through active learning.

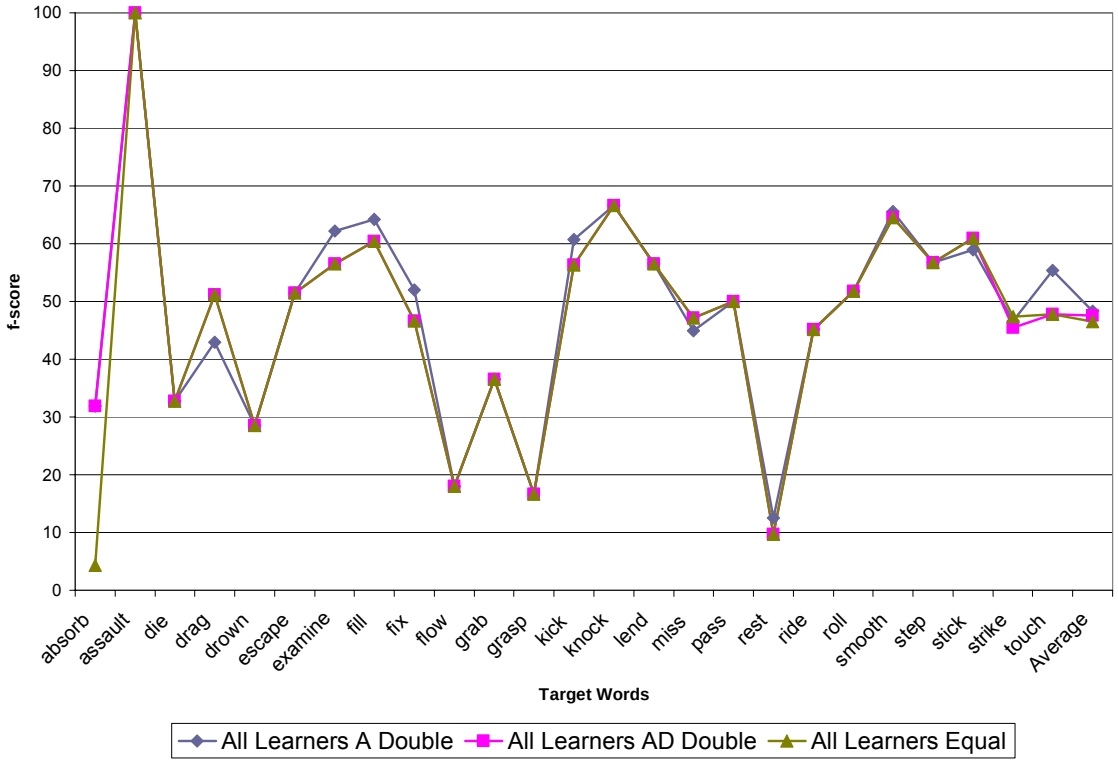
The first graph in Figure 8-C shows the results of voting when all learners carry an equal weight. These results are ever so slightly above the no-scrub learner and ever so slightly below our best individual learner, Learner A. We introduce *weighting* to try to improve on these results.

We begin by adding extra weight to our two best-performing learners, Learners A and D. We do this by doubling the number of votes allotted to Learners A and D and awarding the win based on four votes out of six. Thus, either Learners A and D have to agree, Learners A, B, and C have to agree, or Learners B, C, and D have to agree.

A quick glance at Figure 8-C shows us that the only reason for our slight improvement – 1% – on the final average is that we managed to counteract that low score obtained by "absorb" when all learners are equal. We attempt to do better by giving a higher weight only to the strongest learner, Learner A. For this schema, Learner A gets two votes and a decision is based on three votes out of five. Although the average advantage of doubling Learner A is minimal – 1.8% above weighting all learners equally; 1.7% above Learner A alone – we do get some nice improvements across the target words, with only one word – "drag" – seriously dragging us down.

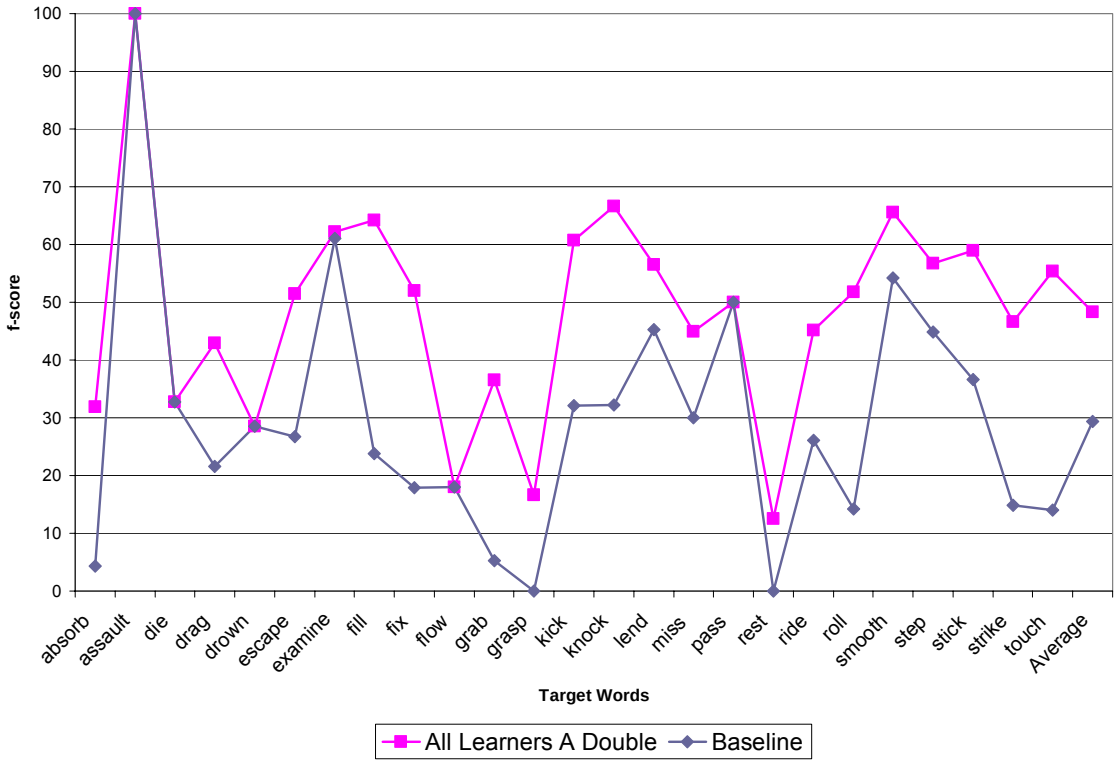
Figure 8-C Simple Voter and Weighted Voter Comparisons





Of course, with the averages being so close and the results depending so much on individual target words, it is difficult to justify calling any of the voting schemata the undisputed winner. We will see the truth of this even more in the following sections when we start examining the augmentation of feature sets. Still, if we compare our current state-of-the-art to where we started – see Figure 8-D – we can see that we are now solidly above the baseline.

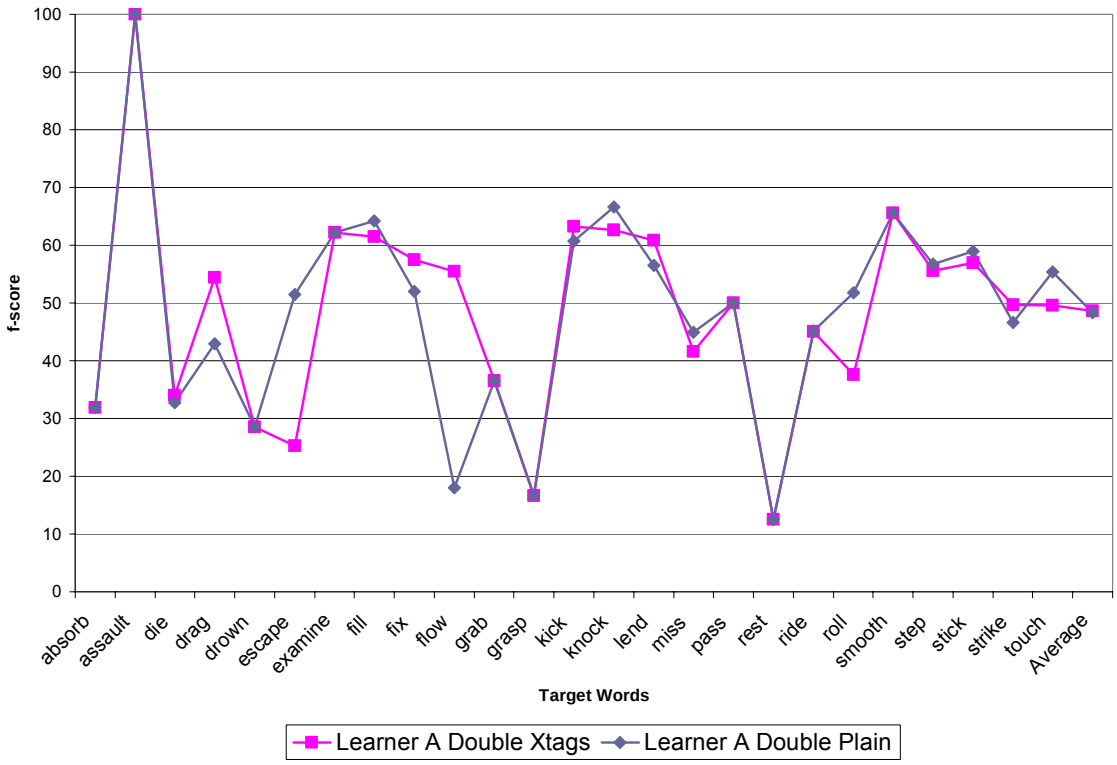
Figure 8-D Best Voter and Baseline Comparison



8.6 Experiment 4: Effects of Using SuperTags

In Sections 6.2.2.2 and 7.1.2.3, we discuss augmenting our bag-of-words feature sets with SuperTags. In the following experiments we examine the effects of this enhancement. We begin by comparing the SuperTag results with the plain results, using the *Learner A doubled* voting schema for both. The results are shown in Figure 8-E.

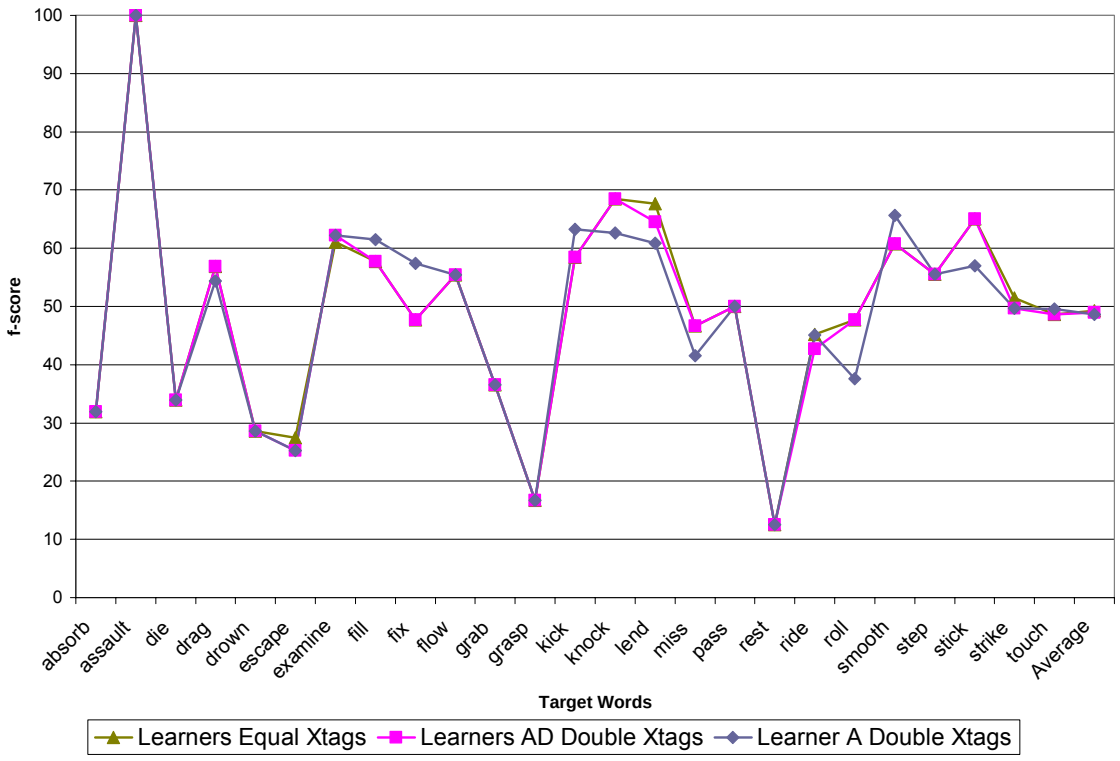
Figure 8-E Plain vs. SuperTags using Learner A Doubled Schema



We can see that adding SuperTags to the feature lists has a definite effect, but it his hard to say whether the overall effect is positive or negative. The SuperTag average is slightly higher, but only by 0.2%. There is a sizable impact on individual target words, however. Adding SuperTags can dramatically improve the results (e.g. “flow) but it can also make them drastically worse (e.g. “escape”). Ideally we would like to make a decision about whether to use SuperTags on a case-by-case basis. Unfortunately this would reduce scalability: we need one setting for the whole system.

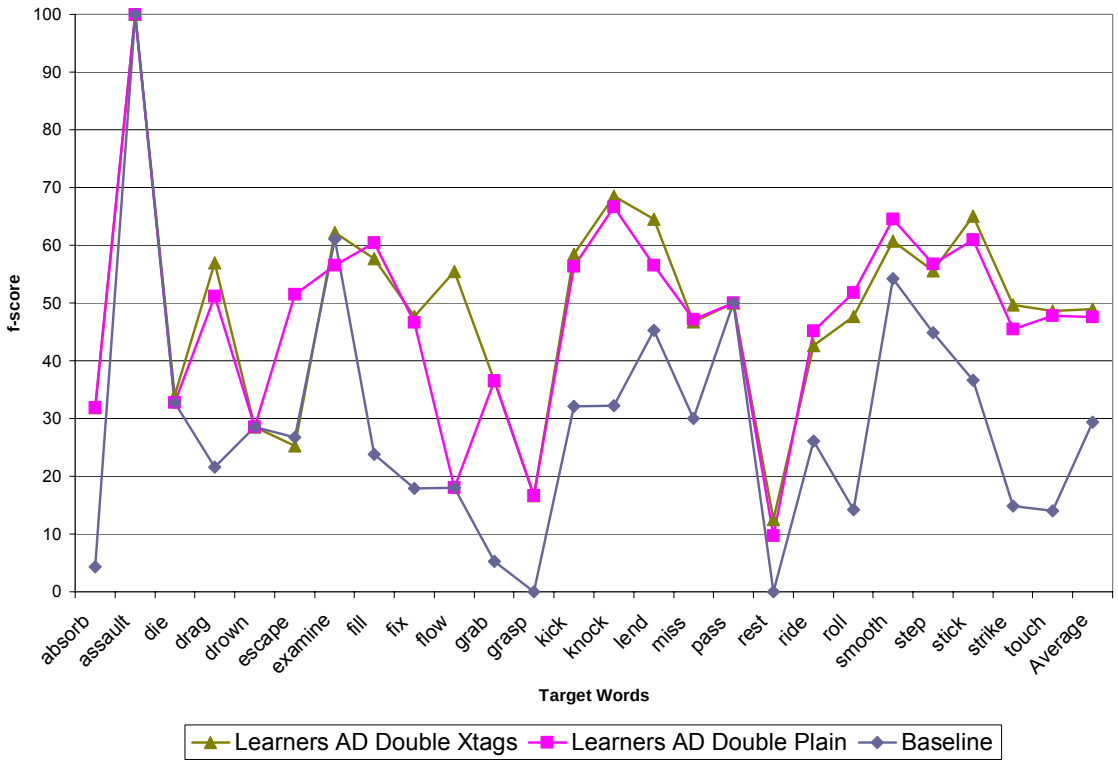
We mentioned in Section 8.5 that changing the feature sets could affect the voting schema results. We re-run the final voting schema experiment using SuperTags (see Figure 8-F). Recall that for the plain feature sets, the best results were achieved by doubling the vote of Learner A only; the next, by doubling Learners A and D; the lowest, by having all learners equal.

Figure 8-F Weighted Voters Comparison for SuperTags



It is difficult to tell from the graph, but comparing the numerical results we find that for SuperTags, the success of the voters is reversed: highest is all learners equal; next, Learners A and D doubled; last, Learner A doubled. As it turns out, our worst SuperTags result is still slightly higher than our best plain result, so we will simply take the middle numbers from both to compare against the baseline, as shown in Figure 8-G. By doing so, we must accept a result for the word "escape" that is slightly below the baseline. This point would be above the baseline if we were to use the *all learners equal* schema for SuperTags.

Figure 8-G Plain, SuperTag, and Baseline Comparison



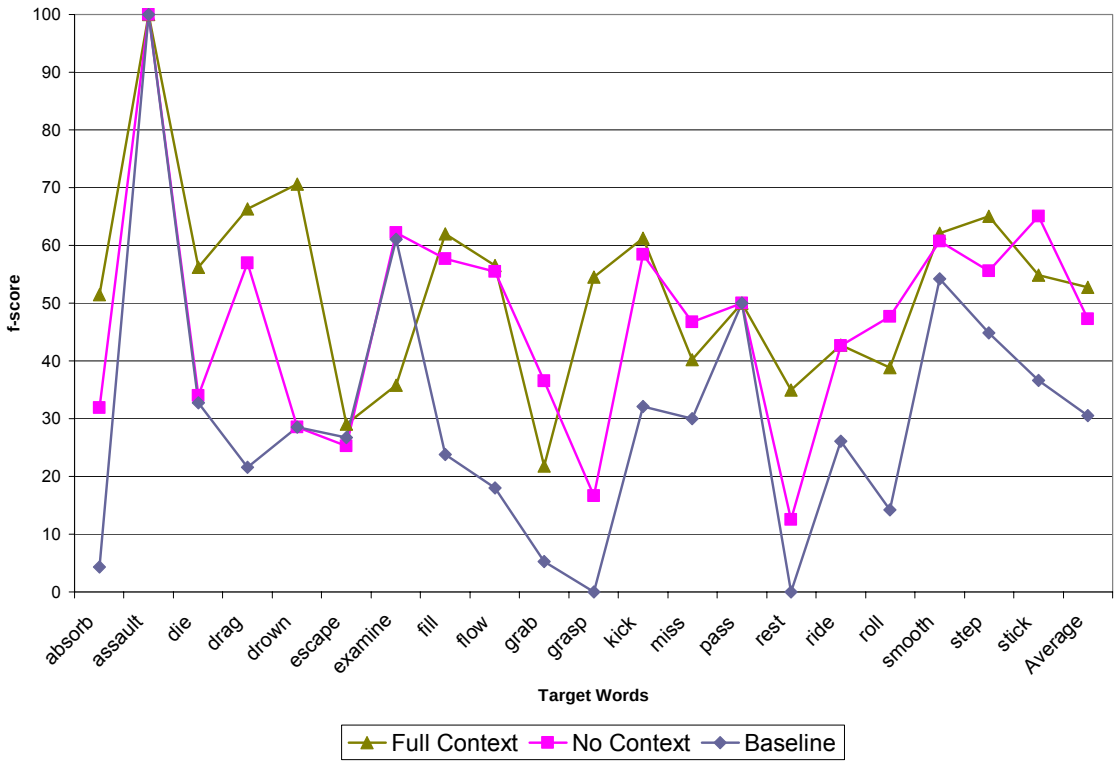
Although we cannot make a definitive decision about the superiority of SuperTags based on these results, we will use them in our remaining experiments. We place such great import on phrasal and expression verbs for scrubbing that it makes sense to incorporate these features into other parts of the algorithm as well.

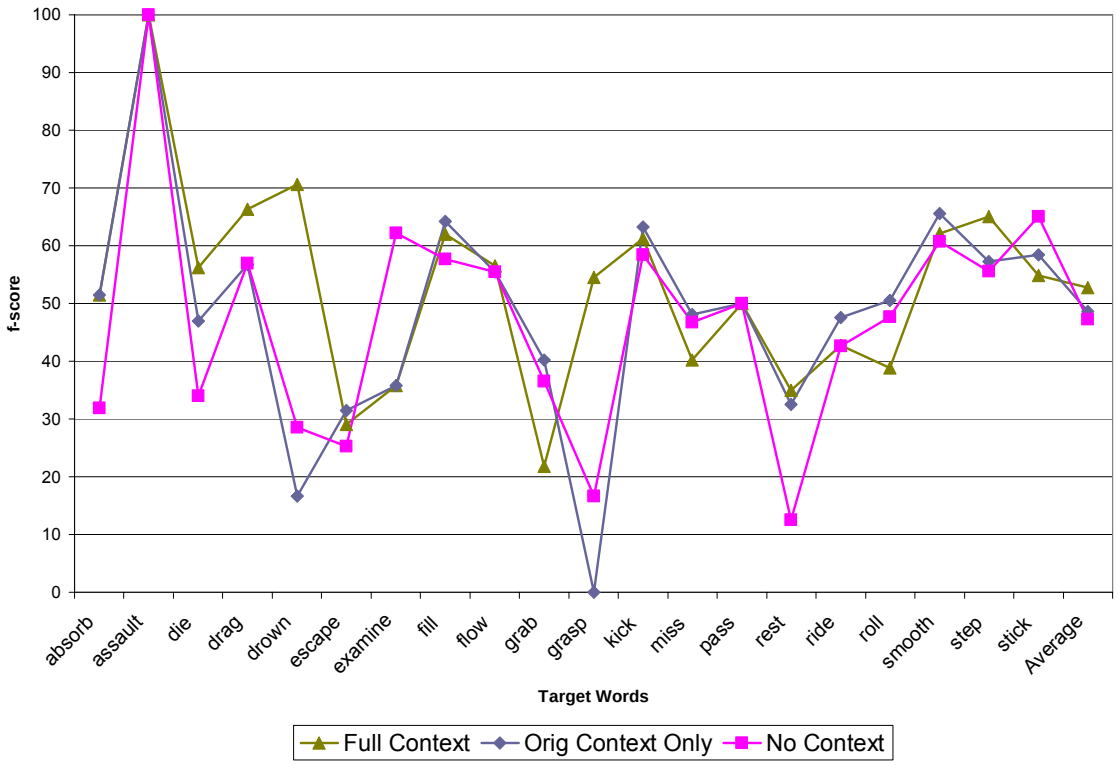
8.7 Experiment 5: Effects of Increasing Context

In Sections 6.2.2.3 and 7.1.2.4, we discuss the fact that the relevant context for clustering a given original sentence is not always in the sentence itself. Often it is in an adjacent sentence. In this section we experiment with expanding the feature sets to include the sentences immediately preceding and immediately following the sentence containing the target word. We conduct two main experiments regarding context: one where we increase the context of both the original and the feedback set sentences, and one where we only increase the context of the original sentences.

The motivation for the first experiment is obvious. The motivation for the second is processing time. TroFi currently uses an $O(n^2)$ implementation with a large coefficient, where n is the size of the union of all the unique words in the feature lists. This means that multiplying the number of features by two, or even three, does not scale well in the current implementation. We therefore additionally examine the results obtained by adding context to the original sentences only. We wish to see if we can obtain similar results without sacrificing scalability. For five of our 20 target words – "fix", "knock", "lend", "strike", and "touch" – processing time becomes prohibitive in the first experiment, forcing us to exclude them. The results are shown in Figure 8-H. Recall that we are using the Learners A and D doubled voting schema with SuperTags.

Figure 8-H Context Comparisons

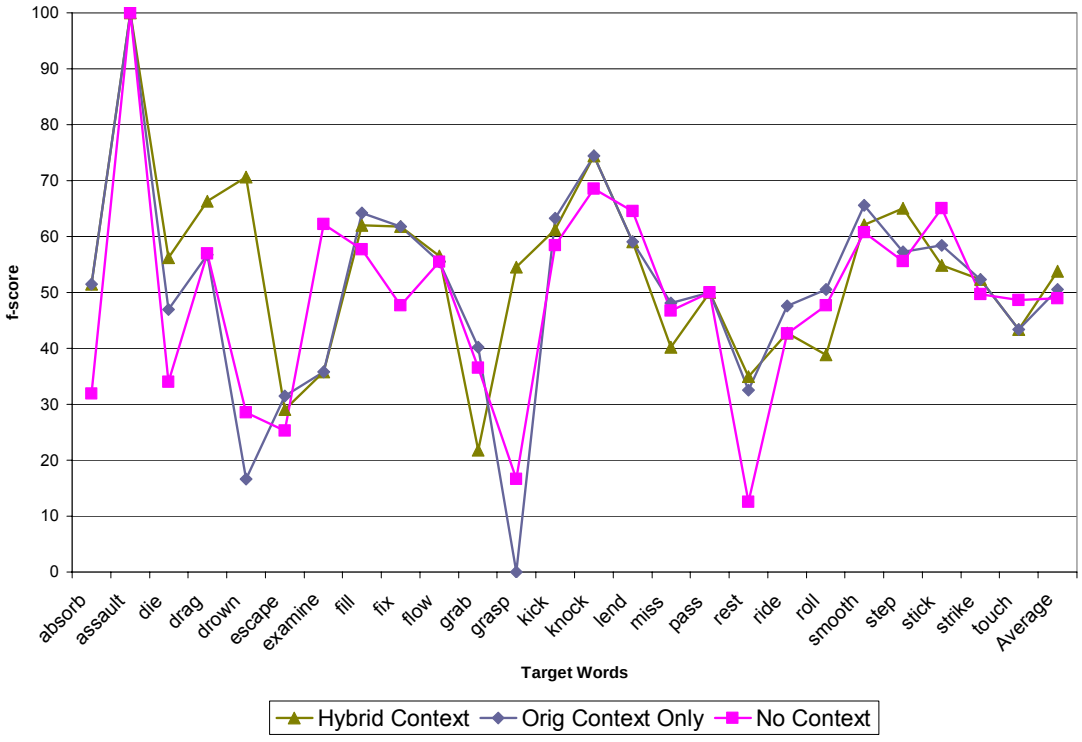




Looking at the first graph in Figure 8-H, we find that, as usual, our enhancement improves the performance on some words and decreases it on others. Worth noting is that the two target words exhibiting the most significant improvement, "drown" and "grasp", have some of the smallest original and feedback set feature sets, supporting the theory that lack of cogent features may be a cause of poor performance. Note also that the addition of context results in a comparatively large increase in accuracy – 5.5% over the *no context* model.

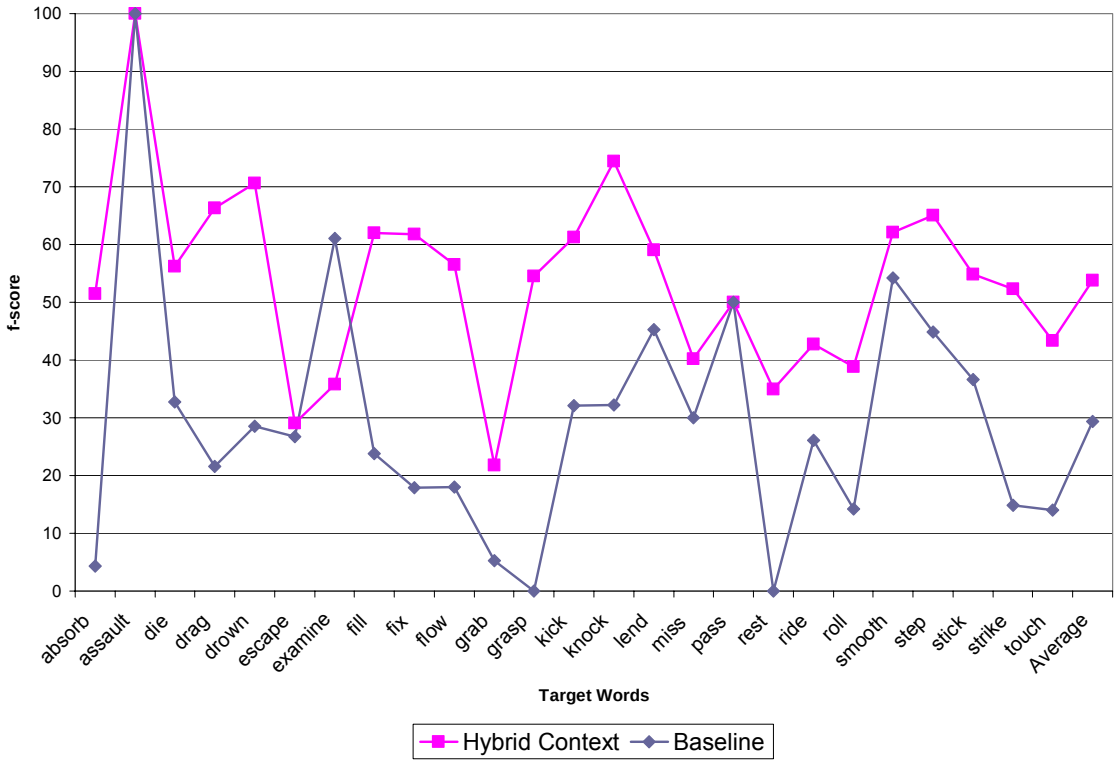
We ran the second experiment hoping to find a similar increase in accuracy with a lesser hit on processing time. Sadly, adding context only to the original set sentences decreases our gain to just 1.4%. Still, it may be worthwhile to use the *original context only* approach for those target words for which the *full context* approach proves too unwieldy. We run a third experiment using *hybrid context* – full context whenever possible and original context only for the rest. The results are shown in Figure 8-I.

Figure 8-I Original Context Only and Hybrid Context Comparison



As always, we gain a little and we lose a little with our new innovation. However, since the hybrid context approach allows us to handle all our target words while still producing a 4.7% accuracy improvement over the baseline, we accept it as a good compromise. We compare our new state-of-the-art to the baseline in Figure 8-J. Except for the word "examine", where adding context has caused something of a train wreck, we are moving steadily away from the baseline.

Figure 8-J Baseline and Hybrid Context Comparison



8.8 Summary

At the beginning of this chapter, we started with a baseline for which the average accuracy was 29.4%. The core TroFi algorithm brought us up to 36.9%. After we added all our enhancements, TroFi produced an average accuracy of 53.8% – 16.9% above the core and 24.4% above the baseline. The biggest gains came from the sum of similarities approach and the use of additional context.

An important observation arising from the experiments performed in this chapter is the variation in the behaviour of different target words. Looking only at average accuracy, one would be inclined to think that many of the enhancements had hardly any effect. However, if we look at individual target words, we often see drastic changes between models. What exactly causes these differences in behaviour is difficult to define. Some factors may be:

- different original set sizes – for some target words there are 100 examples, for some there are only three.
- literal/nonliteral imbalance – for some target words there are many more literal than nonliteral usages and vice versa; for some, one of the categories is completely empty.
- looseness of selectional restrictions – it may be easier to cluster verbs with tight selectional restrictions like "eat" – for example, "she ate her lunch" vs. "she ate her words" – than verbs with loose selectional restrictions like "examine" – for example, "the detective examined the report" vs. "the committee examined the proposal".
- frequency – some verbs, like "examine", are so frequent that they are apt to occur in sentences where there is a nonliteral usage that has nothing to do with the target word – for example, "He *examined* her closely as she sat devouring her book". If such a sentence is an example in our collection of known metaphors, idioms, and expressions, it will show up in the nonliteral feedback set of "examine" regardless.
- state of fossilization – as metaphors are absorbed into common usage, they become fossilized. It may be more difficult to cluster nonliteral language in the process of fossilization – consider "his anger is rising" vs. "the prices are rising".
- general noise – some target words may simply have noisier original sets and feedback sets than others.
- lack of good examples – for some target words there are excellent sets of seeds and examples on which to build feedback sets. For others there are not. For some target words the nonliteral feedback sets end up completely empty, making it difficult to attract things to them.

We might have avoided much of this problematic variance had we chosen our example sets for the experiments more carefully. However, that would have defeated one of the primary

goals of this thesis: applying TroFi *not* to a handful of carefully crafted examples, but rather to real-world data, in all its messy, ugly glory.

Given the massive fluctuations in individual target word accuracy from enhancement to enhancement, it is conceivable that optimal results could be obtained by tuning up a separate configuration for each word. This would require manually annotating a held-out set for each one. This could be justified only for very large projects where the held-out set would make up a tiny fraction of all the sentences to be clustered. For smaller sets, the general optimal model arrived upon in this chapter is sufficient. To review, this model uses sum of similarities, the Learner A doubled voting schema, SuperTags, and two sentences of context for target words where the union of all feature sets is a reasonable size. Again, this model produces an average accuracy of 53.8% compared to a baseline of 29.4%.

The question is: what if this is not good enough? What if we need greater accuracy and someone is willing to work for it? For those cases, we introduce *active learning* (see Section 7.2). Experiments and results for TroFi's active learning component are discussed in Chapter 9.

9 ACTIVE LEARNING EXPERIMENTS & RESULTS

In this chapter we discuss various experiments for evaluating TroFi's active learning component. As we have seen in Chapter 8, the purely unsupervised TroFi algorithm does fairly well on its own, but it may be even more effective to use TroFi as a tool for *helping* a human with a literal/nonliteral clustering task.

With active learning TroFi is able to send sentences it is unsure about to the human. We allow different settings for what percentage of the original set the human is willing to evaluate manually. The benefit of this approach is that only some of the sentences – hopefully those most in need of help – are sent to the human. Risks are that sentences showing a strong attraction to the incorrect feedback set will never make it to the human and that the human will be sent some sentences that TroFi's default decision process would have classified correctly without any help.

Another approach would be to ask the human to manually process a randomly selected subset up front. We use this as a test case for evaluating the appropriateness of TroFi's choices later in this chapter. Risks of random selection are that the results could vary greatly from one run to the next.

Other options for human involvement – which lie outside the scope of this thesis – might be to allow TroFi to classify all the sentences and to then send the classified sentences to the human for verification. Careful tests with human subjects would have to be run to determine whether this would save time and effort as compared to having the human draw conclusions about the literal/nonliteral distinctions independently. Much of the success or failure of this kind of approach is likely to be UI-dependent.

The purpose of our first active learning experiment is to set the parameters for *how many* sentences are sent to the human; the second, to determine *when* the sentences are sent. We then compare the results of our selected model to those obtained by sending a randomly selected set to the human. Finally, we examine the value gained and the effort saved by using active learning.

9.1 Experiment 1: Setting the Parameters

In this experiment we attempt to determine *how many* sentences should ideally be sent to the human. We do this by finding appropriate settings for two parameters: the similarity threshold below which sentences are sent to the human; and, the limit on the percentage of sentences the human is willing to look at.

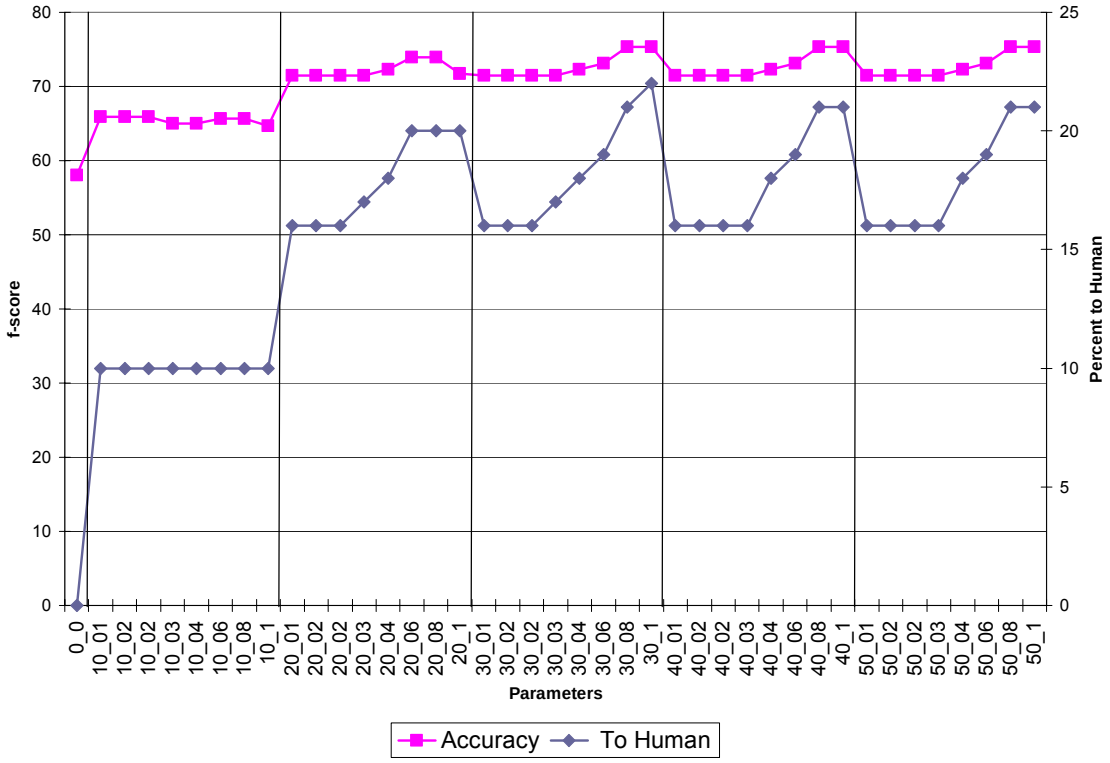
To set the parameters, we ran some experiments on a held-out set of the target word "fill". All experiments were run using sum of similarities, voting with Learners A and D doubled, no SuperTags, and no context. The chosen sentences were sent to the human after the last iteration to avoid potential irregularities introduced by learning from human input at earlier iterations. In addition to the baseline of (0,0), the combinations of similarity thresholds and human percentage limits shown in Table 9-A were tested.

Table 9-A Active Learning Parameter Combinations

	10%	20%	30%	40%	50%
0.15	x	x	x	x	x
0.2	x	x	x	x	x
0.25	x	x	x	x	x
0.3	x	x	x	x	x
0.4	x	x	x	x	x
0.6	x	x	x	x	x
0.8	x	x	x	x	x
1	x	x	x	x	x

The results are shown in Figure 9-A.

Figure 9-A Parameter Evaluation on Held-Out Set of "fill"



The graph in Figure 9-A demonstrates the power of both the similarity threshold and the human percentage limit. We see that the similarity threshold can limit the number of sentences to be sent to the human because TroFi is only able to approach the human percentage limit as we loosen the similarity threshold. It is also obvious that TroFi does not want to send more than 22 sentences to the human at any point for this particular target word. Note that the accuracy also levels off. At first glance it may seem odd that the accuracy line on the graph does not parallel exactly the line of the number of sentences sent to the human. The reason for this is that TroFi occasionally sends sentences to the human that its default decision process would have handled correctly in the end, making the human evaluation redundant.

The experiment indicates that, at least for the held-out set of "fill", we cannot improve on sending to the human everything that TroFi wants to send – i.e. a similarity threshold of 1.0 – up to a human percentage limit of 30%. For the word "fill", there would be little point in setting the

human percentage limit any higher. Although there might be target words for which it would be beneficial to send more sentences, the human really should not be made to deal with more than 30% in any case. Based on these results, the remaining experiments in this chapter and in Chapter 10 are run with a 1.0 similarity threshold and a 30% human percentage limit.

9.2 Experiment 2: Timing

In this section we look at four different options for *when* to send the chosen sentences to the human. As discussed in Section 7.2, when the sentences are sent determines how much TroFi is able to learn – or mislearn – from the human helper.

In the first model, as in the experiment for determining the parameters in Section 9.1, we send everything to the human after the first iteration, hoping to correct some potential errors and produce some augmented feedback set benefits early on.

In the second model, we send everything to the human after the third iteration. Through an informal investigation of how the similarity values fluctuate over iterations, we determined that the order of certainty[1] with which original sentences are attracted to a feedback set generally stops changing after the third iteration. The individual certainty values continue to change after this point, but not enough to affect the certainty order. What this indicated to us is that the selection of sentences to be sent to the human would not change much after this point. In addition, by the third iteration there should be few certainty values of 0 remaining. Only those sentences truly dissimilar to anything else would still be at 0.

In the third model, we distribute the sentences to be sent to the human across a number of iterations in order to get a bootstrapping effect. We divide the human percentage limit by six – an estimate by inspection of the number of iterations in an average run. If we want to send 30% to the human, this means that we will be sending 5% per iteration. The idea is that if we move some

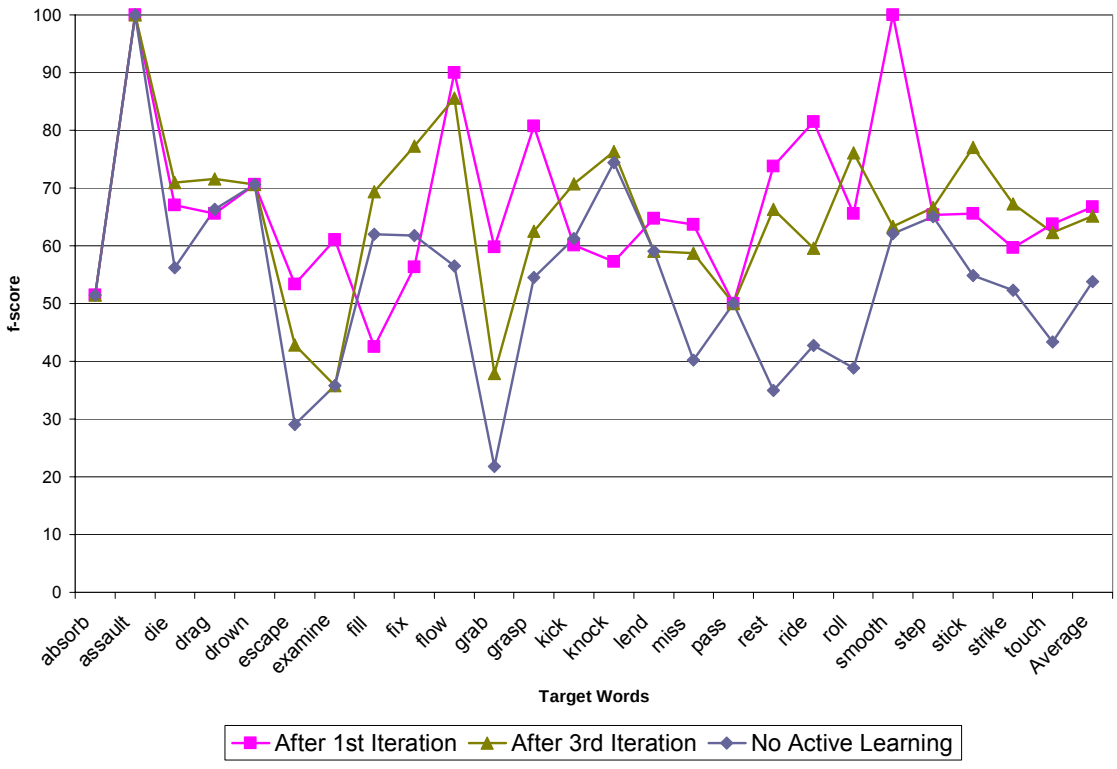
[1] *Certainty* values are snapshots of the similarity values taken at each iteration right before clustering.

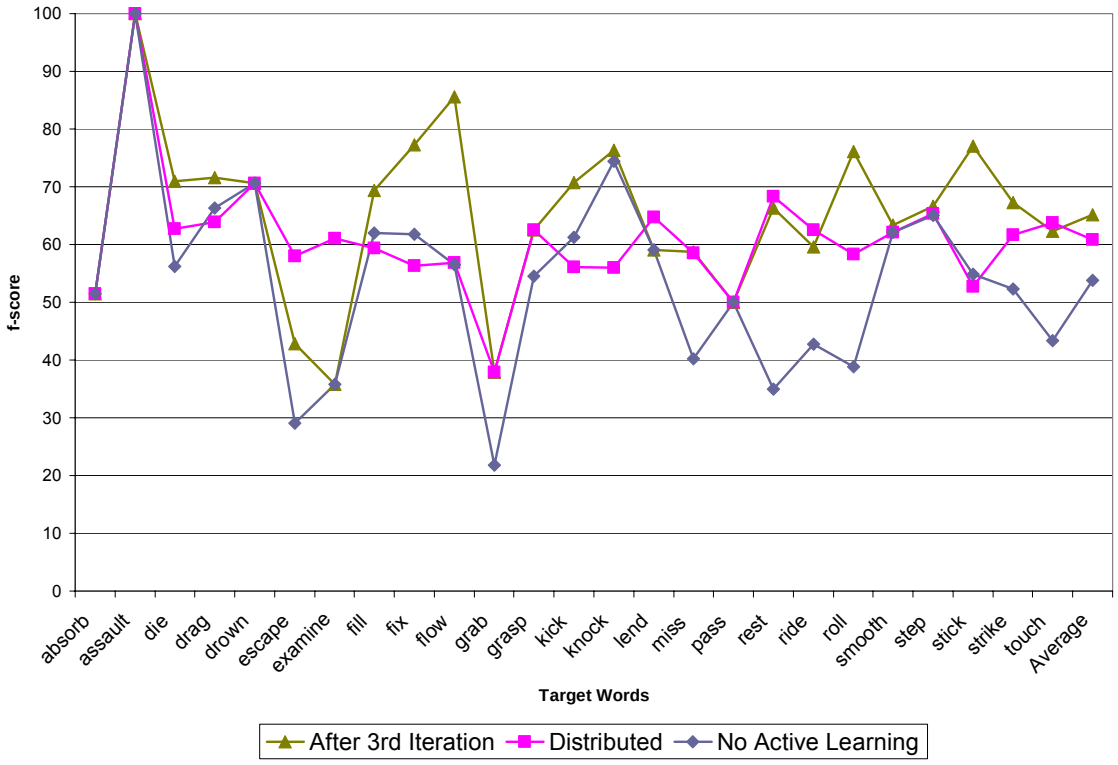
sentences known to be correct into the feedback sets at a particular iteration, perhaps these will attract some other sentences that would otherwise have been sent to the human. This would help refine the list of sentences to be sent to the human in the next iteration.

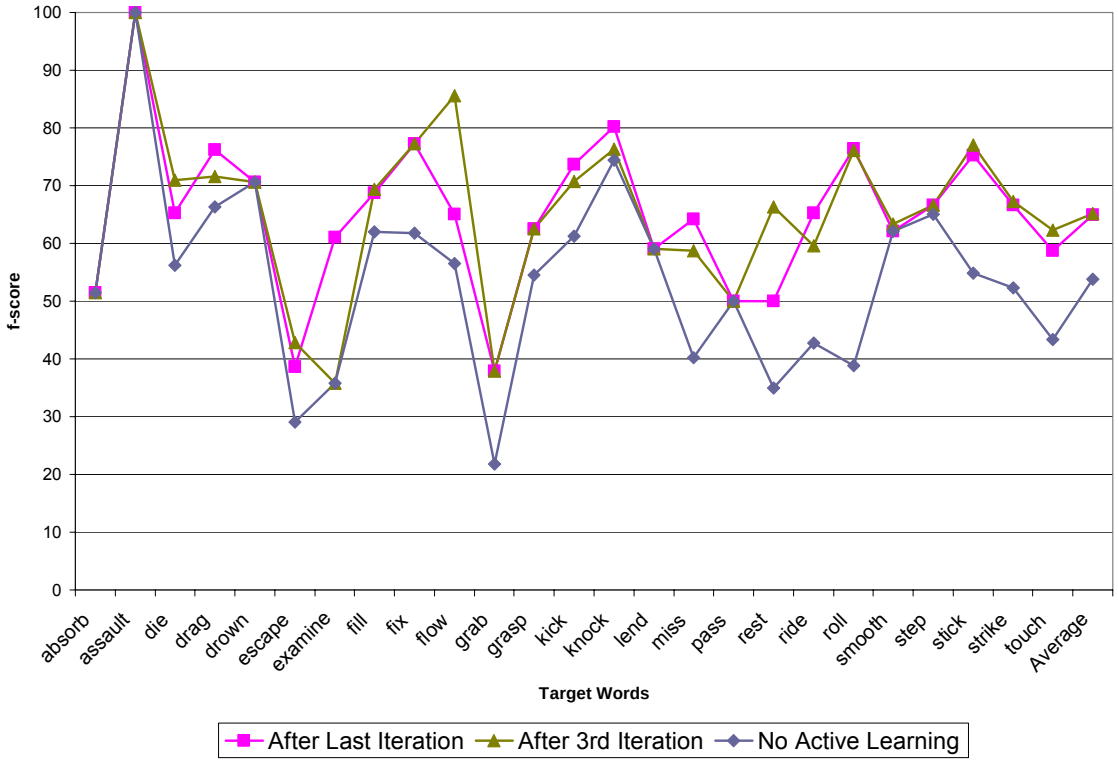
In the fourth and final model, we simply send all the sentences at the end, after TroFi has gone about as far as it can go. This has the benefit of providing more predictable results, but unfortunately it limits what TroFi can learn from the human choices in a particular run. However, there is still a chance to learn from these choices during *iterative augmentation* (see Section 7.3).

In order to show the results as clearly as possible, we will not plot all four models on the same graph. Instead we compare various subsets. As a baseline we use the hybrid context results from Section 8.7.

Figure 9-B Comparison of Models for *When* to Send Sentences to Human




f-score
100
90
80
70
60
50
40
30
20
10
0
absorb
assault
die
drag
drown
escape
examine
fill
fix
flow
grab
grasp
kick
knock
lend
miss
pass
rest
ride
roll
smooth
step
stick
strike
touch
Average
Target Words
After 3rd Iteration
Distributed
No Active Learning


f-score
100
90
80
70
60
50
40
30
20
10
0
absorb
assault
die
drag
drown
escape
examine
fill
fix
flow
grab
grasp
kick
knock
lend
miss
pass
rest
ride
roll
smooth
step
stick
strike
touch
Average
Target Words
After Last Iteration
After 3rd Iteration
No Active Learning

The first graph in Figure 9-B indicates that – as expected – we get a certain amount of stability after the third iteration. The first iteration results are a little erratic, exhibiting extraordinary luck on words like "ride" and "smooth" and catastrophic misfortune on words like "fill" and "knock". This can happen if we move an original sentence to a given feedback set (see Section 7.2) and its features end up causing false attraction. We conclude that although sending the sentences after the first iteration gives us slightly higher results on average – about 1.6% – the difference is not particularly significant and does not compensate for the risk we are taking: we cannot have the human input causing the results to fall *below* the no-active-learning baseline!

We must also ask why human help makes absolutely no difference to some words. This can happen for any of the following reasons: the way the similarity threshold has been recalculated to work with sum of similarities is too low to allow for the full 30% of sentences to be sent to the human; there are not enough sentences in the original set to allow for 30% of them to be sent to the human (for example, the "pass" original set contains only one sentence); the decisions made by the human are the same as those that TroFi would have made using its default decision process; TroFi is so confident in its choices that none of the similarity values fall below the threshold, so nothing is sent to the human. The last of these explains the lack of accuracy change for the target word "absorb", for example.

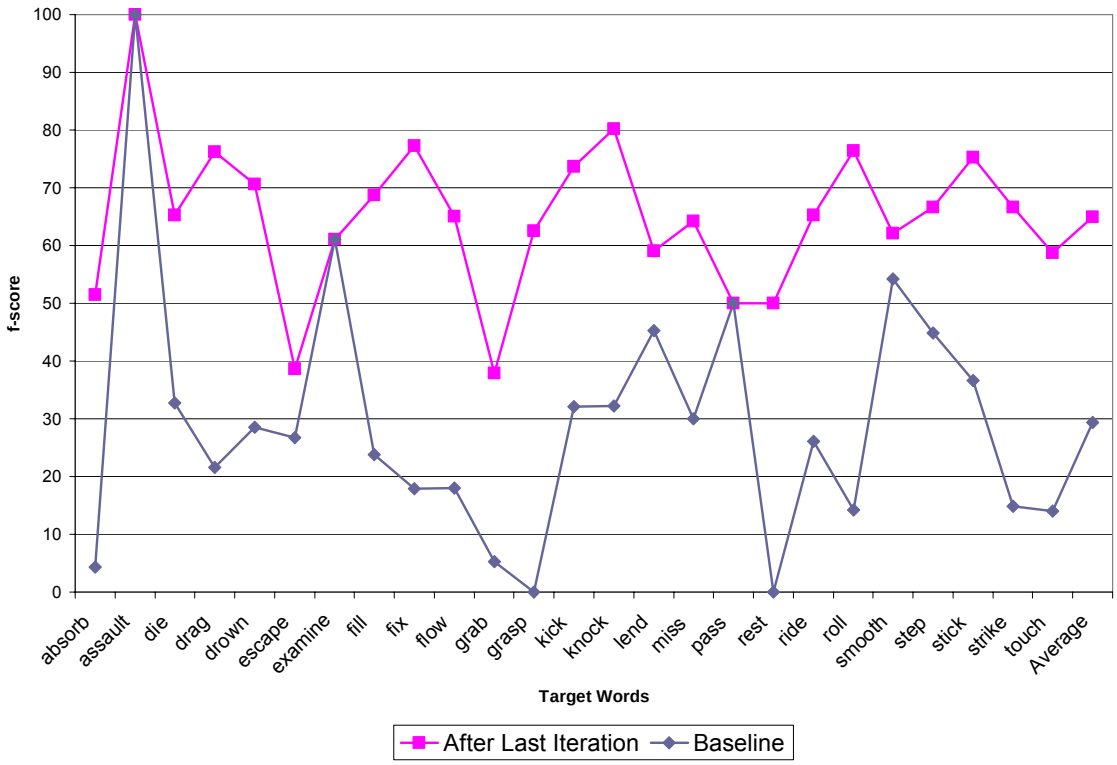
In the second graph in Figure 9-B, we compare sending everything after the third iteration to sending the sentences in a distributed manner – a few percent per iteration. Sending sentences to the human a bit at a time seems to cause fairly flat results, which, unfortunately, means that there is not much gain and there are also a number of words that end up performing below the baseline.

Finally, we compare sending everything after the third iteration to sending everything after the last iteration. As we can see, the differences are minimal, mainly because we already have a fair amount of stability after the third iteration. We do get a few lucky – and some not so

lucky – breaks by sending the sentences after the third iteration. However, since this does not make much difference to the average (0.3%), we select the most stable and predictable option – sending everything after the last iteration – as our optimal active learning model.

In Figure 9-C we show the results of the optimal TroFi-with-active-learning model. An average accuracy of 64.9% puts us 35.5% above our original baseline.

Figure 9-C Optimal Active Learning and Baseline Comparison



9.3 Experiment 3: Random Comparison

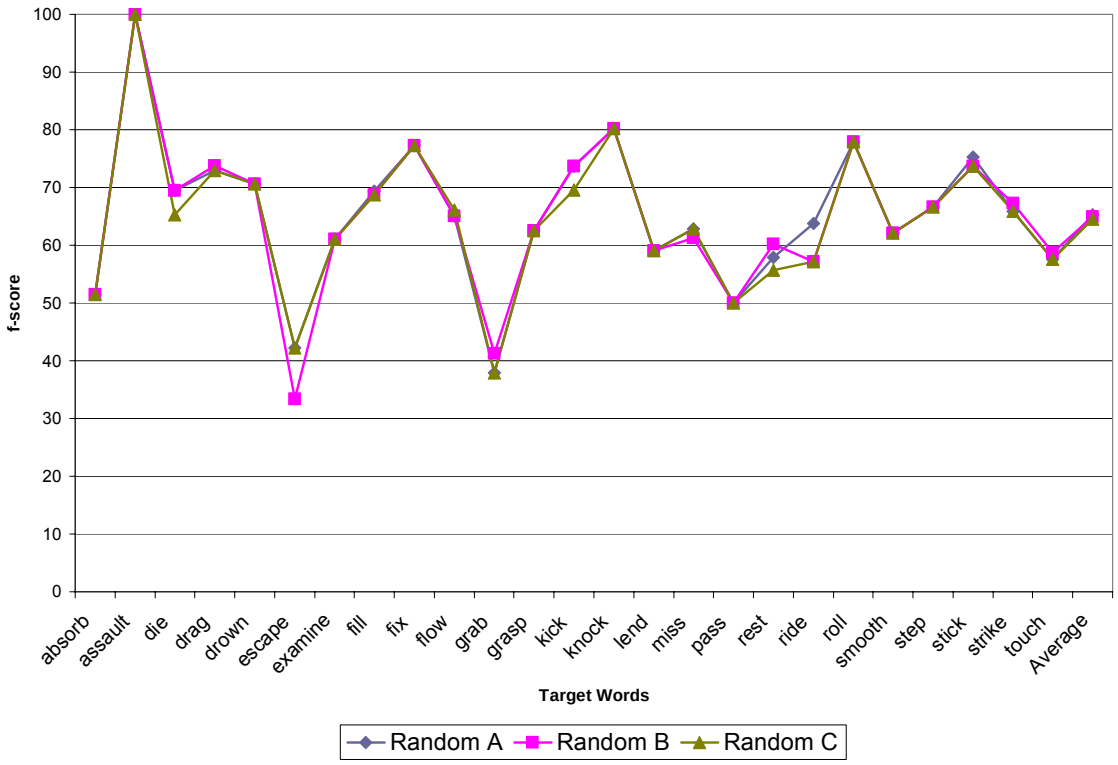
In our chosen active learning model, TroFi makes as many decisions as it can on its own and then sends the rest to the human. We suggested at the beginning of this chapter that another approach would be to select a percentage of the original sentences randomly and send those instead. In this section we experiment with this approach by randomly selecting a certain

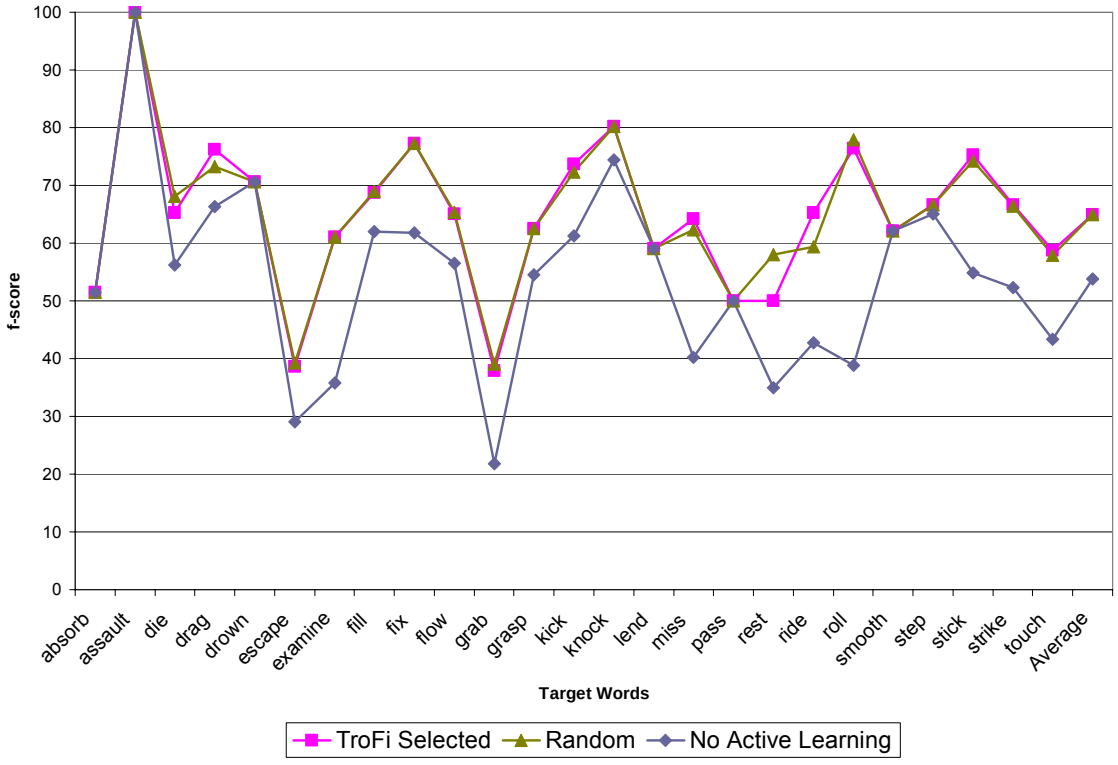
percentage of sentences at the beginning, running TroFi, and then sending the randomly chosen sentences, rather than sentences chosen by TroFi, to the human at the end.

Conceivably, choosing sentences randomly could be either beneficial or detrimental. Results could improve if the randomly selected sentences were ones that TroFi on its own would have clustered incorrectly. Deterioration could result if the selected sentences were ones TroFi would have clustered correctly regardless: we may not be sending the neediest sentences to the human. Also, by using random selection, we introduce an element of unpredictability, and the outcome becomes non-deterministic.

To compare the two approaches we use the optimal active learning model from Section 9.2. For the random model, we select randomly the same number of sentences that were sent to the human in that non-random run. This is to avoid giving the random model an unfair advantage by allowing it to send more sentences to the human than the non-random model. Note that the randomly selected sentences are still part of the clustering process. Holding them out completely would make for incomparable results. We take the average of three random runs to reduce the likelihood of "dumb luck". The three random runs and the comparison of the random results to the TroFi-selected, non-random results are shown in Figure 9-D.

Figure 9-D Random and TroFi-Selected Comparison





There does not appear to be much difference between the results of the random model and the results of the non-random model. To see why this might be, we must examine the possible sources of a randomly chosen sentence. It could be a sentence that TroFi would also have picked for human evaluation, a sentence that TroFi would have clustered correctly in the main part of the algorithm, a sentence that TroFi would have clustered correctly using its default decision process, or a sentence that TroFi would have clustered incorrectly. As suggested earlier in this section, for the random model to outperform the non-random one it would have to select its sentences from the last of these sets; to do worse it would have to select only sentences that TroFi could have handled on its own. Since the likelihood of the random choices coming exclusively from these two sets is low, it makes sense that the results are similar to the non-random ones.

In Section 9.2, we decided that TroFi should send its selected sentences to the human after the last iteration. To make the random and non-random results comparable, we had to do the same in the random case. One might ask, however, if results would not be improved by having the human evaluate the randomly chosen sentences at the beginning for addition to the appropriate feedback sets. We ran some informal experiments on sending the randomly selected sentences to the human after the first iteration. Although the results were impressive for some target words, on the whole the model proved to be unstable and unpredictable.

9.4 Benefits of TroFi with Active Learning over Manual Clustering

In Section 9.2 we found that TroFi with active learning attained an accuracy of 64.9%. Although this is 35.5% above the baseline, it is only an 11.1% improvement on TroFi without active learning. The question is, is the 11.1% gain worth the effort the human must contribute? We examine this issue below.

Let us imagine a manual clustering task where we want to send only 30% of the sentences – chosen randomly – to the human. We assume that we are using the TroFi evaluation

standards – i.e. *precision*, *recall*, and *f-score* defined as in Section 8.1, and *unknown* sentences being sent to the cluster opposite their manual testing label. We look at three potential scenarios with 100 original sentences each. The first is a perfectly balanced scenario: 50 literal; 50 nonliteral. Of the 30 sentences sent to the human, half are literal and half are nonliteral. In the second and third scenarios we have an imbalance: 96 literal; 4 nonliteral. In an imbalanced situation like this, the outcome depends on what is randomly chosen for the human. In the second scenario, we assume that all four nonliteral sentences are chosen; in the third scenario, we assume that none are. We provide the results of our hypothetical experiment in Table 9-B.

Table 9-B Results of Manually Clustering 30% of the Original Sentences

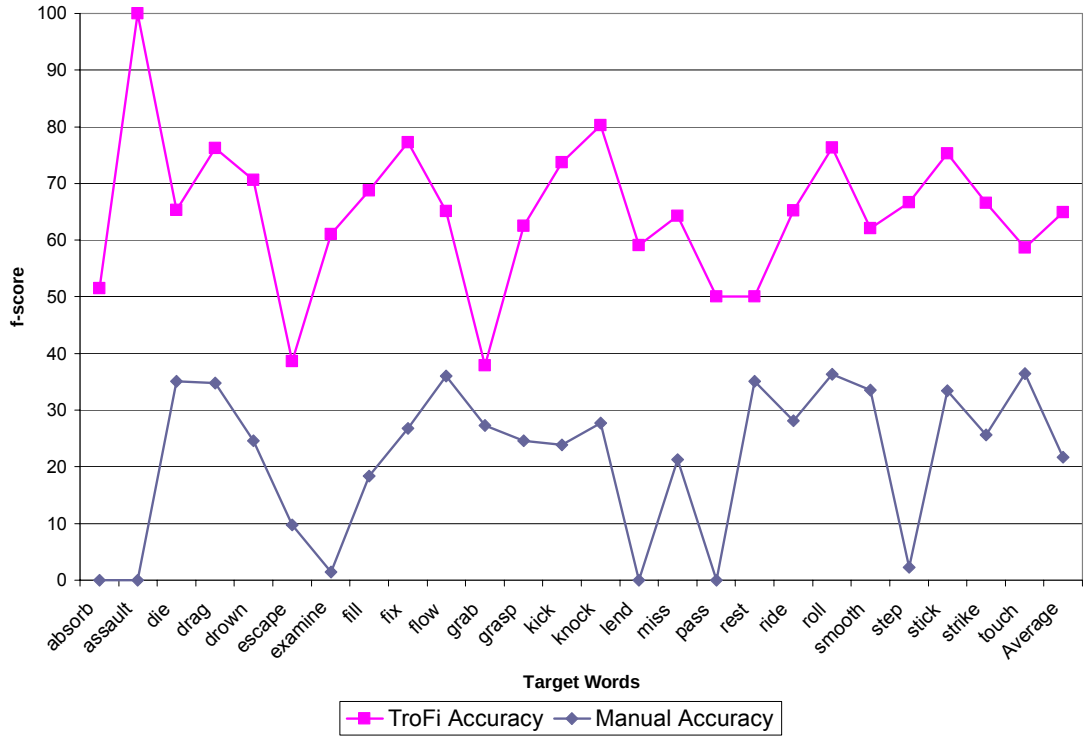
	Balanced		Unbalanced -- All Nonliterals Randomly Selected		Unbalanced -- No Nonliterals Randomly Selected		Average Expected f-score
Nonliterals							
Precision	30		5.405405405		0		
Recall	30		100		0		
f-score	30		10.25641026		0		
Literals							
Precision	30		100		88.23529412		
Recall	30		27.08333333		31.25		
f-score	30		42.62295082		46.15384615		
Average							
Precision	30		52.7027027		44.11764706		
Recall	30		63.54166667		15.625		
f-score	30		57.61685638		23.07692308		**36.897926**
Total for Human Decision							
To Human	30		30		30		
Counts							
T	100		100		100		
NT	50		4		4		
LT	50		96		96		

We can see that in a balanced scenario we get an average f-score of 30%, as would be expected. In the second scenario, where all four nonliteral sentences are sent to the human, we get a much higher average f-score, primarily due to the perfect recall score for the nonliteral cluster and the perfect precision score for the literal cluster. In the third scenario, where none of

the four nonliteral sentences is sent to the human, we get disastrous results for the nonliteral cluster. This drags down even the high literal precision score. Since we cannot predict how the random sentence selector will behave, we average the three experimental results to give us an average score we could be expected to obtain by randomly selecting 30% of the sentences. This score comes out to nearly 36.9%.

In Figure 9-E and Figure 9-F, we explore the benefits gained by using TroFi. Note that often TroFi does not end up sending the whole 30% to the human. In these cases we estimate the accuracy one could be expected to attain giving the human only the number of sentences TroFi would have sent. We calculate the expected manual process accuracy by taking the number of sentences sent to the human and multiplying by 1.23. This factor is derived from our hypothetical situation above, where $30 \cdot 1.23 = 36.9$. Based on these premises, we produce a results that could be expected from a manual process and compare them to the TroFi results in Figure 9-E.

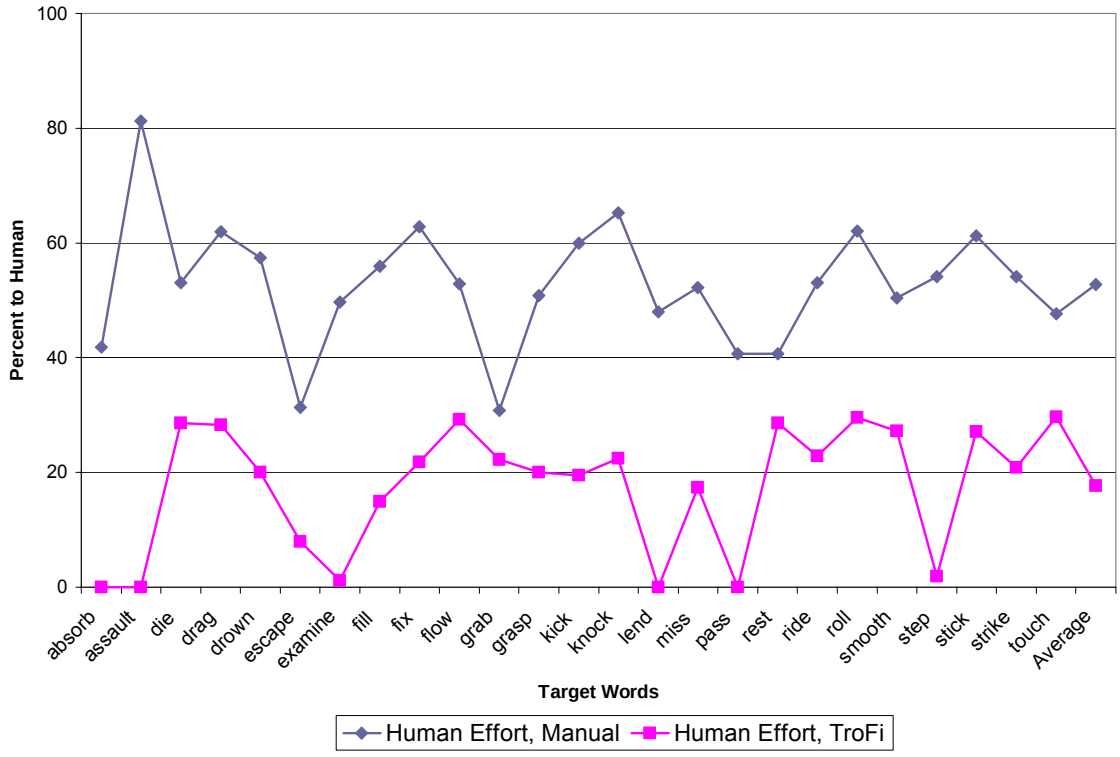
Figure 9-E TroFi and Manual Accuracy for Same Human Effort



Sending the same number of sentences to the human in both the TroFi and the manual case gives us an average accuracy of 64.9% for TroFi and 21.7% for the manual process. For the same effort, TroFi gives us an almost threefold improvement in accuracy.

We can also estimate the amount of effort that can be saved by using TroFi. We divide the TroFi accuracy scores by 1.23 to give us the percentage of sentences the human would have to cluster manually in order to obtain the same results. Figure 9-F compares the amount of human effort required by TroFi to the amount of human effort required by the manual process to reach an average accuracy of 64.9%.

Figure 9-F Human Effort Required to Attain 64.9% Accuracy



We can see that TroFi allows us to attain the same results as a manual process with about 35% less effort. With TroFi we obtain a 64.9% average accuracy with only 17.7% human effort!

We must admit that it would be difficult to get 100% accuracy using TroFi, whereas if one sends 100% of the sentences to the human, one should get 100% accuracy – in theory. Also, we must remember that for the purposes of our experiment we placed unknown sentences in the cluster opposite to where they belong. If we were to ignore such sentences, the manual process would necessarily give us cleaner clusters. However, it is worth keeping in mind that literal/nonliteral judgments are often extremely difficult for humans, and inconsistencies tend to occur. One could claim, therefore, that a purely manual process cannot attain 100% accuracy either. We do not pretend to adequately support this claim. The human subject studies required lie outside the scope of this thesis.

To conclude, we claim that TroFi with active learning is a helpful tool for a literal/nonliteral clustering project. It can save the human significant effort while still producing reasonable results.

9.5 Summary

In this chapter we presented experiments for determining the optimal configuration of TroFi with active learning and for comparing the performance of the resultant model to the optimal *core* model. We further presented an informal analysis of the significant human effort saved by using TroFi in a literal/nonliteral clustering project rather than depending on a purely manual process.

In Chapter 10, we combine the optimal active learning model with iterative augmentation to construct the long-awaited TroFi Example Base.

10 BUILDING THE TROFI EXAMPLE BASE

In this chapter we discuss the TroFi Example Base and its construction. First, we briefly revisit the *iterative augmentation* process described in Section 7.3. Then we discuss the structure and contents of the example base itself, together with the potential for expansion.

After an initial run for a particular target word, not only the cluster results, but also a record of the feedback sets augmented with the newly clustered original sentences is produced. For simplicity's sake, we will call these *classifiers*. As explained in Section 7.3, each feedback set sentence is saved with a *classifier weight*: its highest similarity to any of the original sentences plus ε. The newly clustered original sentences, which are added to the feedback sets, are assigned a weight of 1.0.

Subsequent runs may be carried out to augment the initial clusters. For these runs, we use the classifiers from our initial run as feedback sets. New sentences for clustering are treated like a regular original set. TroFi then proceeds normally, with or without active learning. At the end, TroFi produces new clusters and re-weighted classifiers augmented with the newly clustered sentences. There can be as many runs as desired; hence the term *iterative augmentation.*[1]

We used the iterative augmentation process to build a small example base consisting of the target word list described in Section 6.1.5, as well as another 25 words. These additional target words are discussed in more detail below, followed by an outline of the process used to augment the clusters from the initial run to produce the TroFi Example Base.

[1] If active learning is used, the selected sentences do not have to be checked right away, but can be saved for checking and incorporating into the classifier at a later time.

The additional 25 target words were drawn from the experiments and examples of scholars whose work was reviewed in Chapter 3. Table 10-A presents the original set counts for each word for the initial run as well as the name of the scholar from whose work the word stems[2].

Table 10-A Target Words Selected from the Literature

Dolan			
	flourish	**plant**	**wither**
Literal	4	39	6
Nonliteral	46	11	31
Total	50	50	37

Fass						
	dance	**drink**	**fly**	**play**	**plow**	**sleep**
Literal	39	49	87	6	12	41
Nonliteral	7	1	12	12	38	8
Total	46	50	99	18	50	49

Martin		
	eat	**kill (also Fass)**
Literal	39	84
Nonliteral	14	15
Total	53	99

Mason						
	attack	**besiege**	**cool**	**destroy**	**dissolve**	
Literal	35	3	11	40	14	
Nonliteral	69	18	39	32	26	
Total	104	21	50	72	40	
	evaporate	**melt**	**pour**	**pump**	**target**	**vaporize**
Literal	7	24	13	15	4	1
Nonliteral	41	19	37	35	60	6
Total	48	43	50	50	64	7

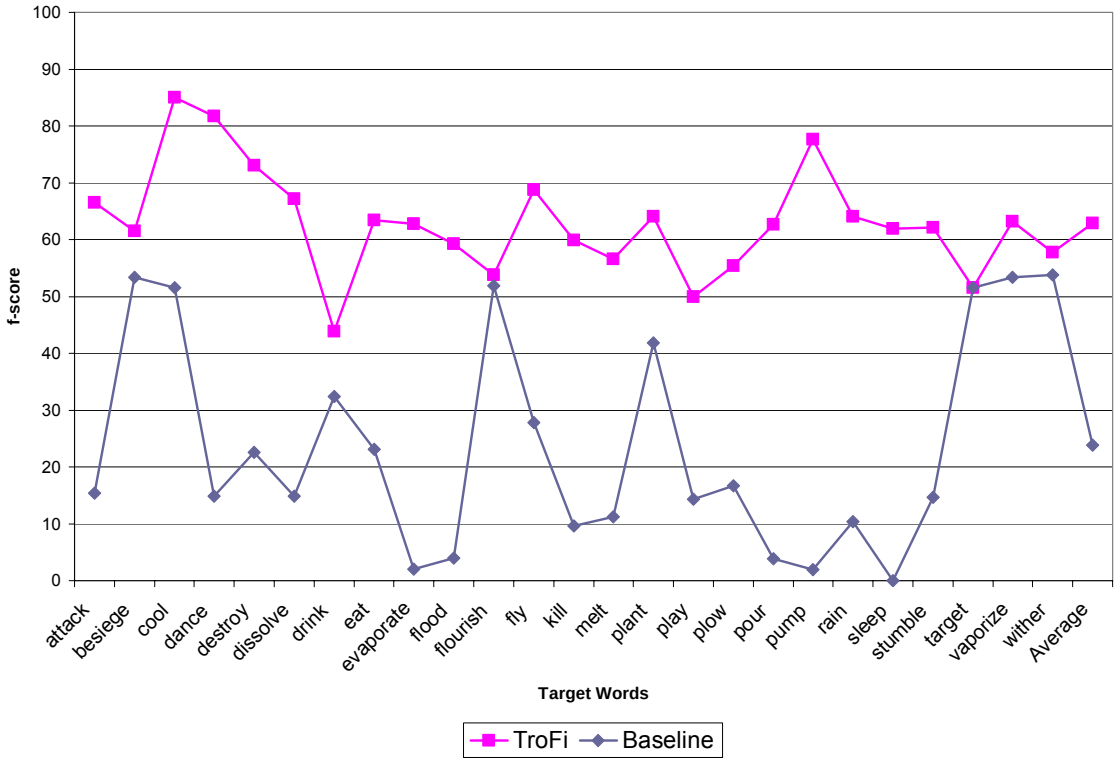
Narayanan	
	stumble
Literal	6
Nonliteral	41
Total	47

Russell		
	rain	**flood**
Literal	32	3
Nonliteral	11	46
Total	43	49

We ran TroFi on this set of words using our optimal models from Chapters 8 and 9 – i.e. sum of similarities, voting with Learners A and D doubled, SuperTags, additional context, and active learning with a similarity threshold of 1.0 and a human percentage limit of 30%. The results, set off against the baseline, are shown in Figure 10-A.

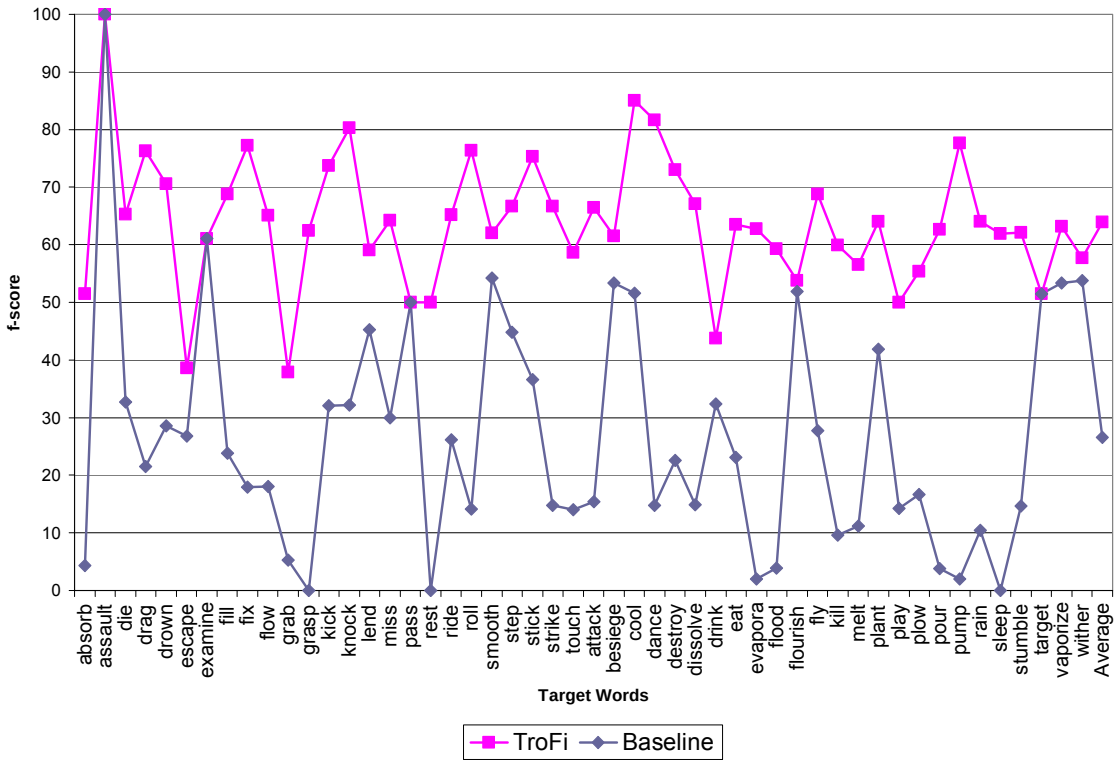
[2] Other words under consideration were Martin's "give" and "take", and Zernik & Dyer's "throw". All three had to be dropped because their occurrences were too numerous to be handled within the constraints of this thesis.

Figure 10-A Words from Literature using Optimal TroFi Model



We can see that we have managed to attain an average accuracy 39.1% above the baseline, 3.6% more than the improvement achieved on our first set of 25 words. For a final look, we show all 50 words in Figure 10-B.

Figure 10-B All 50 Target Words Results



As explained earlier, at the end of the initial run, TroFi produces *classifiers* for future iterative augmentation. The reader will recall that the original sets for the initial run were collected from a 10% subset of the Wall Street Journal Corpus. We pulled additional sentences for the iterative augmentation phase out of the remaining 90%. Where possible, we randomly selected a set of 100 additional original sentences per target word. Where this proved impossible due to insufficient remaining examples, we simply took what was left. Unfortunately, in a few cases – namely "besiege", "play", "vaporize", and "wither" – all we had left was a great big goose egg. The reason for this is that in order to fill out the initial run, if there were too few examples for a given target word, we attempted to round out the set by stealing from the 90% set.

We ran each additional original set through the iterative augmentation version of TroFi, which uses as feedback sets the weighted classifiers created in the initial run. After sending up to 30% of the sentences to the human, TroFi produced the new clusters and augmented classifiers.

The combined old and new clusters of each of the 50 target words make up the TroFi Example Base. The TroFi Example Base is publicly available at http://www.cs.sfu.ca/~anoop/students/jbirke/.

The current example base should be seen as a starting point. It is conceivable that additional iterative augmentation will increase not only the size, but also the quality of the clusters, particularly if active learning is used. Since clustered sentences are always added to the feedback sets, the ability of the feedback sets to attract appropriate sentences should increase over time. In addition to augmenting existing clusters, new target words could be added to the example base. Interested parties wishing to expand the TroFi Example Base should contact the author of this thesis for further information.[3]

It is expected that the TroFi Example Base will be useful not only for future research in the field of nonliteral language processing, but also as training data for other statistical processes or as a general resource for the NLP community.

[3] See http://natlang.cs.sfu.ca/people.php for contact information.

11 CONCLUSION

It has long been a dream of mankind to be able to interact naturally with machines, and, slowly, science fiction is starting to become reality. In certain contexts, we expect machines to understand us and provide appropriate responses, preferably in fluent English. In other situations, we want the machine to translate input from one language into another. And sometimes, we just want to chat. In those cases, we even expect appropriate responses in terms of facial expressions! As suggested in the Introduction, it is surprising then that so many of the sophisticated systems available can still be derailed by something as fundamental and pervasive as nonliteral language.

Part of the difficulty of nonliteral language is that it is so wide-spread and productive. This thesis, for example, is full of it. Humans are somehow, through world-knowledge and experience, able to follow the connections between domains and come up with appropriate interpretations of what a nonliteral utterance might mean. It would be extremely optimistic to expect the same from a machine, or to expect a human to enter all the necessary knowledge into a machine. Luckily, with ever-increasing hardware capabilities, another road has been rapidly opening up in the form of statistical and example-based techniques. Our ultimate hope for TroFi is that it may point nonliteral language processing down that road as well, thus avoiding much of the labour and complexity currently associated with nonliteral language processing systems.

In the remainder of this chapter we look at some suggestions for future work with TroFi, including an expansive section on possible applications of TroFi. We also make suggestions for extending TroFi from the realm of nonliteral language recognition to that of nonliteral language interpretation. We follow this up with a brief summary of the work completed and the contributions to the field made as part of this thesis.

11.1 Future Work

In this section we look at some ideas for further improving and/or experimenting with TroFi. We examine four different areas:

1. Core Algorithms
2. Active Learning
3. TroFi Example Base
4. Applications

We begin our discussion by looking at some possible improvements to the core algorithms.

11.1.1 Core Algorithms

There are a number of ways in which the core algorithms could be tweaked in an attempt to improve accuracy. Areas worth revisiting are the scrubbing procedures, the composition of the feature sets, and the efficiency of the TroFi algorithm and/or its implementation.

The scrubbing algorithm lends itself to further exploration. One area that we began to investigate was *scrubbability scores*. To produce such scores, every trigger for scrubbing – for example, each phrasal verb in a synonym list – could be given a value. Summing over these values would result in a score. If this score was above a certain threshold, the synset or feature set in question would be scrubbed. In this thesis we concentrated on phrasal/expression verbs and overlapping words as scrubbing triggers. There could well be others.

The expansion of the trigger set for scrubbing could result in an increase in the number of learners. There is nothing to prevent the voting algorithm from being tuned to handle more learners. Also with regards to the voting system, further exploring voting schemata other than *majority rules* may still be a worthwhile exercise.

Another potential for further investigation also lies in the feature sets. The feature sets currently contain only stemmed nouns and verbs, and, optionally, SuperTag trigrams. It may be interesting to study the effects of adding some other types of features. Adjectives would be a good place to start. One can often tell a great deal about a domain from adjectives. Adverbs, although also interesting, might prove to be too common and create too many similarities.

The composition of the SuperTag *n*-grams could also be studied further. Our research suggests that trigrams containing the SuperTag of the target word and the following two tags with the addition of actual verbs, nouns, and adverbs/prepositions/particles strike a good balance between being too general and too specific. However, our average accuracy improvements using SuperTag trigrams were not overwhelming, suggesting further experimentation in this area could prove beneficial.

Also in the realm of features, more could be done with weighting. One may get interesting results by giving certain features more weight than others – for example, one might want to weight nouns more heavily than verbs, or one could create weights based on some frequency measure of the word in the corpus.

Finally, further experimentation could be done with the contents of the extended context. For this thesis we only looked at adding the two immediately adjacent sentences as additional context. Research done by Martin (1994), however, indicates that much of the source domain of a metaphor may be exposed in the sentences preceding the metaphor. It would be interesting to look at differences resulting from choosing different contexts – for example, choosing only the preceding two sentences, or choosing only the following two sentences. Furthermore, there is no reason why one should restrict oneself to two sentences, except performance, which brings us to our next point.

The TroFi algorithm – or rather, its current implementation – must be optimized. At the moment, it is $O(n^2)$ and consequently does not scale particularly well in terms of processing time and memory. Until this is improved, dramatically increasing the amount of extra context might prove difficult.

11.1.2 Active Learning

Future work on the TroFi active learning component should focus on research involving live human subjects. Possible experiments include: testing the difficulty of manually annotating literal/nonliteral distinctions; and, testing the effort level and final accuracy difference between manually annotating everything from scratch and manually correcting TroFi output.

It would also be interesting to research the most effective UI for human input to TroFi: a UI that minimizes the perceived effort.

11.1.3 TroFi Example Base

One of the contributions of this thesis was to build the TroFi Example Base. It currently contains fairly small literal and nonliteral clusters for 50 verbs. In order to further improve the value of the TroFi Example Base, a couple of things must happen: it must be launched on the Internet with an interface that allows for its expansion; and, the program must be tuned to allow for the addition of nouns and adjectives as well as verbs.

The second of these items is comparatively trivial. There is nothing inherent in TroFi that limits it to verbs, although some minor changes to the implementation may be required.

The first item is less trivial. Although it is not difficult to put static collections of sentences on the Web as an example base, the benefit of TroFi would be so much greater if anyone could add to the Example Base, either by iteratively augmenting existing sets or by adding new sets. Although possible, this would require: a. the optimization of the TroFi

implementation; b. software engineering and UI development to make the application usable by the general public; c. security measures.

11.1.4 Applications

One of the most important areas of future work for TroFi is studying its possible integration with various NLP applications. On a basic level, we suggested that the TroFi Example Base could be used to train other statistical algorithms – categorizers and the like – for recognizing nonliteral usages in incoming text.

Recognizing a usage as nonliteral, even without any interpretation or translation into literal text, is useful because it allows a warning to be sent to the rest of the system that a given input phrase cannot be handled in the standard[1] way. For example, a dialogue system could respond, "I'm not sure what you mean; could you please rephrase that?" as opposed to interpreting the input literally and giving the user some silly response. In a more sophisticated scenario, the system might include a database of common nonliteral expressions with their literal translations – perhaps limited to a particular domain – and the recognition of nonliteral input could trigger a search of this database. This would be more efficient than searching the database for every single input sentence.

A bigger project would be to use TroFi to automatically generate a kind of nonliteral to literal example-based machine translation (EBMT) system. A high-level overview of such a system is provided in the following section.

11.1.4.1 TroFi Example-based Nonliteral/Literal Translation System

In this thesis we have concentrated on developing a system for clustering literal vs. nonliteral usages – i.e. nonliteral language *recognition*. In this section we further explore the

[1] What is meant by "standard" would depend on the particular application.

possibility of developing a system for *interpreting* nonliteral language, or translating between nonliteral and literal language, using TroFi.

The motivation for building a nonliteral interpretation system modelled on example-based machine translation (EBMT) systems stems from a belief that translating from a source language into a target language is much like translating from nonliteral language into literal language. Looking carefully at the dictionary-based metaphor processing systems discussed in Section 3.2, we find that Dolan (1995) refers to his lexical knowledge base as an example-based system, and we note the similarity of the Zernik & Dyer phrasal lexicon (Zernik & Dyer, 1986) to an example-based system. This begs the question, what exactly is an example-based system?

Simply put, an example-based system is a system that depends on a database filled with examples. The general idea of EBMT is to break the source sentence into input fragments, match the input fragments to fragments in the database, retrieve the translation, and recombine the retrieved fragments into fluent output in the target language. In the sections below we examine what would be required to build such a system using TroFi.

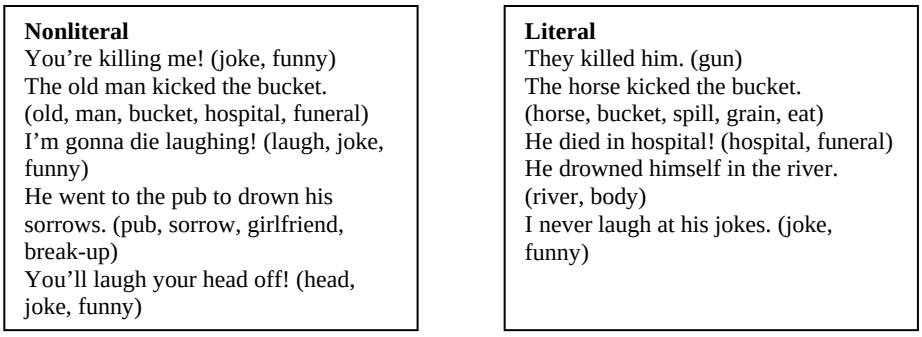
We begin by outlining the basic steps and then describe each step in detail:

1. Separate literal and nonliteral usages of target words into two clusters
2. Recluster to find nonliteral and literal phrases with similar meanings
3. Store phrases in example base
4. Provide method for processing metaphors using the example base

11.1.4.1.1 Separating Literal and Nonliteral Usages

Step 1 was the focus of this thesis so the only thing we will say about it is that it would be possible to produce output in the form of Figure 11-A, including the original set sentence for each example, as well as the feature set with extended context. For the sake of illustration, we have also included adjectives in the feature lists.

Figure 11-A Separating Literals and Nonliterals

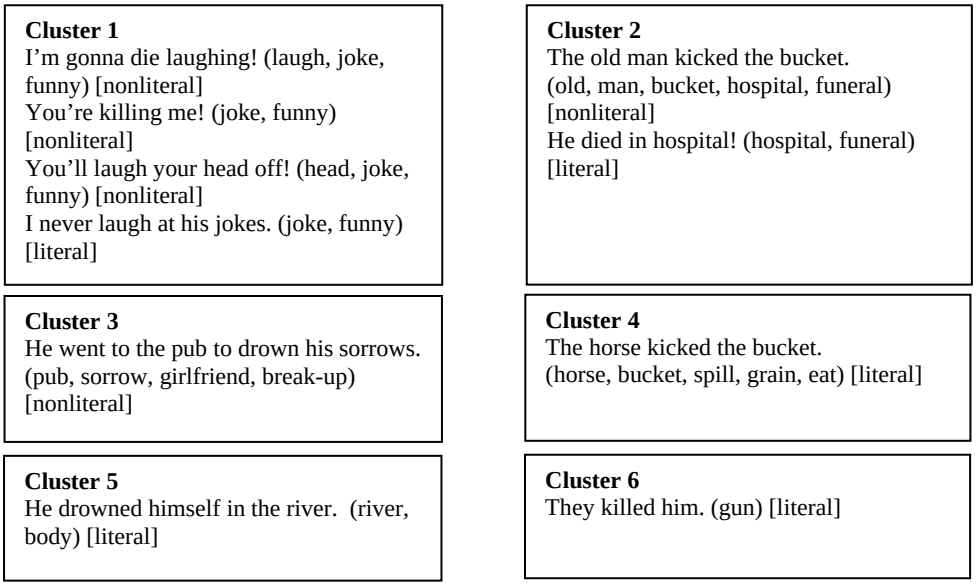


11.1.4.1.2 Reclustering According to Meaning

After the first step, we have literal/nonliteral clusters for each target word. Unfortunately this gives us no hint as to the meaning of the nonliteral sentences.

So far in this thesis we have not concerned ourselves much with *meaning*, and we do not want to change that now. Therefore we approach the finding of nonliteral/literal translations the same way we approached literal/nonliteral clustering: we use the context. The idea is that words that can be used in similar contexts have similar meanings. Going on this principle, we can proceed by throwing all the literal and nonliteral sets for all the words together and reclustering to find literal and nonliteral sentences displaying a similar context. An adapted version of TroFi could be used for this by simply having one sentence similarity matrix with the nonliteral sentences along one axis and the literal sentences along the other. Since we are just looking for similarities of the sentences to each other, no feedback sets would be needed. The clusters produced would depend on the similarities found in the matrix. Careful work would be required to determine cluster boundaries. The kinds of clusters we might expect are shown in Figure 11-B.

Figure 11-B Reclustering across the Literal/Nonliteral Divide



Note that due to the small size of our sample set, we do not get translation clusters for all the sentences. However, we do get two usable translation clusters for taking us to the next step.

11.1.4.1.3 Storing Phrases in the Example Base

The phrases extracted in the previous step are stored in the example base in their translation clusters, with the set of nonliteral phrases for each aligned with the corresponding set of literal phrases. There are numerous ways to store examples, but we will not go into that here.

Since the way the examples from the TroFi Example Base are used will depend largely on the NLP system into which the database is integrated, we must choose the most flexible solution possible. One possibility, since we have access to the SuperTags anyway, is to store the target word and the surrounding words required by its SuperTag. Complications that will arise are cases where we are dealing with something like an idiom which expands beyond the SuperTag of the target word. Furthermore there will be issues with deciding which part of the phrase-structure tree is variable and which is a necessary part of the nonliteral expression.

Figure 11-C shows an extremely simplified version – no trees, no argument structure – of what this might look like. Note that we still keep a link to the context for recognition purposes.

Figure 11-C Storing Phrases in the Example Base

NonLiteral	Literal
die laughing (laugh, joke, funny) kill me (joke, funny) laugh /PRP$ head off (head, joke, funny)	laugh at (joke, funny)
kick the bucket (old, man, bucket, hospital, funeral)	die (hospital, funeral)

11.1.4.1.4 Processing Metaphor Using the Example Base

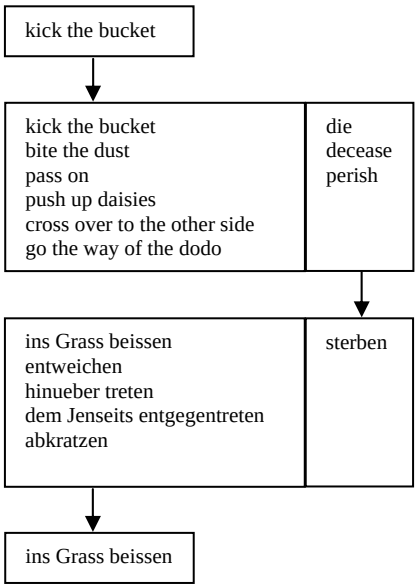
The purpose of building the example base would be to use it in NLP systems for processing nonliteral language. Access should be general enough to allow the example base to be easily incorporated into a variety of different systems. In this section we provide a hypothetical example of how the system might work integrated with another application.

Each input sentence determined to be nonliteral by a literal/nonliteral classifier trained on the basic TroFi Example Base (literal/nonliteral clusters) would be compared to the entries in the translation database to find the closest nonliteral match. The nonliteral phrase showing the greatest similarity to the input sentence would be selected together with the literal phrases of that translation cluster. Naturally, each translation cluster would contain numerous literal phrases. The most appropriate of these would have to be chosen. This literal phrase could then – with some syntactic and morphological massaging – replace the nonliteral phrase in the input sentence.

The above discussion assumes the most complicated integration scenario, where the desired output is a literal paraphrase of a nonliteral sentence. In many systems, however, it may be sufficient just to return the literal phrase from the example base. In an information retrieval system, for example, it may be possible to replace (or even just augment) nonliterals in the text with their corresponding literal with no regard to the consequent grammaticality of the sentence. In this way literal search criteria would produce results, and incorrect matches would be reduced.

Another integration scenario might involve example bases in several languages used in combination to get nonliteral-to-nonliteral translations. This would involve finding the literal in the source language, translating it into the target language, looking the literal up in the target language example base, and spitting out an appropriate nonliteral, as illustrated in Figure 11-D.

Figure 11-D Example Base in a Machine Translation System



In this section we have provided some informal sketches of possible applications of the TroFi Example Base. Although none of these have been explored in any detail, and much more work would be required to actually implement them, they do give us a view to the possibilities.

11.2 Summary and Contributions

In this thesis we presented TroFi, a system for separating literal and nonliteral usages of verbs through statistical word-sense disambiguation and clustering techniques. We motivated the usefulness of literal/nonliteral clustering for NLP applications – e.g. dialogue systems, paraphrasing and summarization, machine translation, information extraction – and for the science of lexical semantics.

We positioned TroFi in the research area of metaphor and metonymy recognition, provided an overview of some past work in this area, and explained that we do not actually claim to solve the problem of either metonymy or metaphor recognition in the detailed sense seen in the literature. Rather we claimed to provide a rough, real-world, scalable approach dependent not on selectional constrain violations and paths in semantic hierarchies, but rather on simple sentential context. We further suggested that TroFi is applicable to all sorts of *nonliteral* language. In order to be able to make this claim, we provided an in-depth explanation of what we mean by *literal* and *nonliteral* in the context of TroFi and this thesis. By our definition, *nonliteral* is a blanket term for any language that is "not literal", including different types of metaphor, metonymy, idioms, and even phrasal verbs.

We adapted an existing unsupervised word-sense disambiguation algorithm to the task of literal/nonliteral clustering through the redefinition of literal and nonliteral as word senses, alterations to the core algorithm such as changing the nature of the similarity scores used, and enhancements such as the addition of learners and a voting schema, SuperTags, and additional context. We further introduced an active learning component and the notion of iterative augmentation.

For all our models and algorithms, we carried out detailed experiments on hand-annotated data both to fully evaluate the system and to arrive at an optimal configuration. Our experiments show that our models outperform the baseline on a number of levels. Through our

enhancements we were able to produce results that are, on average, 16.9% higher than the core algorithm and 24.4% higher than the baseline. We further found that by using the optional active learning component, we were able to improve on those results by another 11 or 12%, giving us a model that outperforms the baseline by just over 35%.

Finally, we used our optimal configuration of TroFi, together with active learning and iterative augmentation, to build the TroFi Example Base, a publicly available, expandable resource of literal/nonliteral usage clusters for use by the NLP community.

APPENDICES

Appendix A

This appendix contains pseudo-code, by section, for most of the algorithms described throughout this thesis.

6.2.2.1 Learners

```
&selectMet

for each WM (Wayne Magnuson) entry containing the target word
       for each verb or noun in the definition
              unless it is the target word or it is in freqhash[1]
                     add word to metSeeds
                     add word to metScrubber
for each WM example sentence
       convert into feature set (stemmed, non-freqhash, non-target
          nouns and verbs only)
       add each word in feature set to metScrubber
       if xtags is on
              &findXtag
              add xtag trigram to feature set
              add xtag-augmented feature set to metEGs
       else
              add plain feature set to metEGs
for each CM (Conceptual Metaphor) example sentence
       convert into feature set
       add each word in feature set to metScrubber
       if xtags is on
              &findXtag
              add xtag trigram to feature set
              add xtag-augmented feature set to metEGs
       else
              add plain feature set to metEGs
for each word in metSeeds
       &selectSentences (selects sentences from WSJ)
```

[1] *freqhash* is a hash of the 332 most frequent words from the British National Corpus plus some additional items such as numbers up to ten and all individual letters.

&selectLit

```
for each WordNet synset containing the target word
       if any of the synonyms are in metScrubber
          or are phrasal/expressions verbs
          AND learner is A or B
              add synset to metSynset list
       else
              add synset to litSynset list
for each synset in the litSynset list
       add synonyms to litSeeds
if learner is A
       for each synset in the metSynset list
              add synonyms to metSeeds
for each WordNet example sentence
       convert into feature set
       if xtags is on
              &findXtag
              add xtag trigram to feature set
              if synset is in litSynset list
                     add xtag-augmented feature set to litEGs
              else if synset is in metSynset list AND learner is A
                     add xtag-augmented feature set to metEGs
       else
              if synset is in litSynset list
                     add plain feature set to litEGs
              else if synset is in metSynset list AND learner is A
                     add plain feature set to metEGs
for each synset definition (gloss; before examples)
       convert into feature set
              if synset is in litSynset list
                     add feature set to litEGs
              else if synset is in metSynset list AND learner is A
                     add feature set to metEGs
for each feature set in litEGs
       add all words to litScrubber
for each word in litSeeds
       &selectSentences
```

&selectSentences

```
for each search phrase
       break phrase into searchword and particle (if exists)
       for each sentence in WSJ containing searchword
          OR containing searchword followed by particle as
          next or next-next word
              convert into feature set
              if xtags is on
                     &findXtag
                     add xtag trigram to feature set
                     add xtag-augmented feature set to feedback set
              else
                     add plain feature set to feedback set
```

&findXtag

(See Section 6.2.2.2)

&removeOverlap

```
if sentence is in both the literal and nonliteral feedback sets
       remove sentence from both sets
```

6.2.2.1.1 Learner A

```
&scrubMovePhrasal

&selectMet
&selectLit
if xtags is on
       &augmentLitNoScrubPlus
else
       &augmentLitNoScrub
&augmentMetNoScrub
&removeOverlap

&augmentLitNoScrubPlus

for each potential literal feedback set feature set
       if xtag feature contains tags for adverbs, particles,
          or prepositions
              if learner is A
                     add feature set to metEGs
              else if learner is B
                     ignore feature set
       else
              add feature set to literal feedback set

&augmentLitNoScrub

for each potential literal feedback set feature set
       add feature set to literal feedback set

&augmentMetNoScrub

for each word in metSeeds
       &selectSentences
for each potential nonliteral feedback set feature set
       add feature set to nonliteral feedback set
```

6.2.2.1.2 Learner B

```
&scrubRemovePhrasal

&selectMet
&selectLit
if xtags is on
       &augmentLitNoScrubPlus
else
       &augmentLitNoScrub
&augmentMetNoScrub
&removeOverlap

NOTE – See Section 6.2.2.1.1 for the following procedures:
       &augmentLitNoScrubPlus
       &augmentLitNoScrub
       &augmentMetNoScrub
```

6.2.2.1.3 Learner C^1

```
&scrubRemoveOverlapC1

&selectMet
&selectLit
&scrubLit
&scrubMet
&removeOverlap

&scrubLit

for each potential literal feedback set feature set
       for each word
              if word is in metScrubber
                     add word to litScrubber
              else
                     keep word in feature set

&scrubMet

for each word in metSeeds
       &selectSentences
for each potential nonliteral feedback set feature set
       for each word
              unless word is in litScrubber
                     keep word in feature set
```

6.2.2.1.4 Learner C^2

```
&scrubOverlapC2

&selectMet
&selectLit
&scrubLitExtra
&scrubMetExtra
&removeOverlap

&scrubLitExtra

for each potential literal feedback set feature set
       for each word
              if word is in metScrubber
                     add word to litScrubber
                     mark feature set for scrubbing
unless feature set marked for scrubbing
       add feature set to literal feedback set

&scrubMetExtra

for each word in metSeeds
       &selectSentences
for each potential nonliteral feedback set feature set
       for each word
              if word is in litScrubber
                     mark feature set for scrubbing
unless feature set marked for scrubbing
       add feature set to nonliteral feedback set
```

6.2.2.1.5 Learner D

&scrubNeither

```
&selectMet
&selectLit
&augmentLitNoScrub
&augmentMetNoScrub
&removeOverlap

NOTE – See Section 6.2.2.1.1 for the following procedures:
       &augmentLitNoScrub
       &augmentMetNoScrub
```

6.2.2.2 SuperTags

&findXtag

```
select the tag of the target word and also the following two tags
   (one, if sentence final)
if the trigram contains any of the following: nouns, adverbs,
   particles, or prepositions
       if these tags occur in the second and third tags
              concat all three words and their tags as xtag trigram
       else if these tags occur in the second tag only
              concat first and second words and their tags as xtag trigram
       else if these tags occur in the third tag only
              concat first and third words and their tags as xtag trigram
       else
              do not return xtag trigram
else
       do not return xtag trigram
```

7.1.1.2 The TroFi Algorithm

&TroFi

```
for each target word
       &setup
       update Original SSM
       update WSM from Original SSM
while highest diff in similarities < diff threshold
       &update
       &clusterer
if active learning is on
       &activeLearning
else if active learning is off
   OR there are remaining undecided sentences after active learning
       &noActiveLearning
&buildClassifier
```

&setup

```
make WSM with similarity of each word to self set to 1
make Original SSM
make Literal Feedback SSM
make Nonliteral Feedback SSM
```

&update

```
update Nonliteral Feedback SSM
collect highest similarity scores for each original sentence
   to nonliteral feedback set
update Literal Feedback SSM
collect highest similarity scores for each original sentence
   to literal feedback set
update WSM from Nonliteral Feedback SSM
update WSM from Literal Feedback SSM
```

&clusterer

```
for each original sentence
       if highest similarity[2] to both literal and nonliteral
          feedback sets is below given threshold, or absolute
          difference between similarity to literal and nonliteral
          feedback sets is below given threshold
              add original sentence to undecided cluster
       else if highest similarity to literal feedback set is
          higher than to nonliteral feedback set
              add original sentence to literal cluster
       else
              add original sentence to nonliteral cluster
```

&activeLearning

(See Section 7.2)

&noActiveLearning

```
for each undecided sentence
       if highest similarity to literal feedback set is greater
          than highest similarity to nonliteral feedback set
              add original sentence to literal cluster
       else
              add original sentence to nonliteral cluster
```

&buildClassifier

(See Section 7.3)

[2] Note that we are using *similarities* here to refer to either single similarities or sums of similarities.

7.1.2.2.2 Voting System

&3of4Voter

```
for each original sentence
       if nonliteral for 3 of 4 learners
              add to nonliteral cluster
       else if literal for 3 of 4 learners
              add to literal cluster
       else
              add to undecided cluster
```

7.2 Active Learning

&activeLearning

```
sort undecided sentences from lowest to highest certainty
for each undecided sentence, beginning with lowest certainty
       if percentage of sentences already sent to human plus current
          < human threshold
              if NOT using distributed algorithm
                 OR if using distributed and percent sent this iter
                 < human threshold/6
                 OR if this is last iter
                     get human decision
                     for each learner
                            add sentence to appropriate feedback set
                            add sentence to appropriate SSM
                               and set similarity to self to 1
```

7.3 Iterative Augmentation

&buildClassifier

```
for each learner
       for each feedback set
              for each feedback set sentence
                     find highest similarity to any original sentence
                     set weight equal to highest similarity + ε
                     unless exists
                            add feedback sentence to appropriate
                               classifier
                     else if weight > current classifier weight
                            update current classifier weight
for each original sentence in literal cluster
       set weight equal to 1.0
       unless exists
              add to literal classifier
       else if weight > current classifier weight
              update current classifier weight
for each original sentence in nonliteral cluster
       set weight equal to 1.0
       unless exists
              add to nonliteral classifier
       else if weight > current classifier weight
              update current classifier weight
```

8.2 Baseline

&highBaseline

```
for each target word
       &setup
       for each original sentence
              for each original word
                     for each literal feedback set sentence
                            add 1 to total for that feedback sentence
                               for each occurrence of original word
                     for each nonliteral feedback set sentence
                            add 1 to total for that feedback sentence
                               for each occurrence of original word
              set literal total to highest total for any literal
                 feedback set sentence
              set nonliteral total to highest total for any nonliteral
                 feedback set sentence
              if literal total > nonliteral total
                     add sentence to literal cluster
              else if nonliteral total > literal total
                     add sentence to nonliteral cluster
              else
                     if manual testing label is “literal”
                            add sentence to nonliteral cluster
                     else if manual testing label is “nonliteral”
                            add sentence to literal cluster
```

BIBLIOGRAPHY

Abeillé, A. and Schabes, Y. 1989. Parsing idioms in lexicalized TAGs. In *Proceedings of the Fourth Conference on European Chapter of the Association For Computational Linguistics* (Manchester, England, April 10 - 12, 1989). European Chapter Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 1-9.

Abeillé, A., Schabes, Y., and Joshi, A. K. 1990. Using lexicalized tags for machine translation. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 3* (Helsinki, Finland, August 20 - 25, 1990). H. Karlgren, Ed. International Conference On Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 1-6.

Bangalore, S. and Joshi, A. K. 1999. Supertagging: an approach to almost parsing. *Comput. Linguist.* 25, 2 (Jun. 1999), 237-265.

Barzilay, R. and Lee, L. 2003. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In *Proceedings of HLT/NAACL 2003*, (Edmonton, Canada, May-June, 2003), 16-23.

Beeferman, D., Berger, A., and Lafferty, J. 1997. A model of lexical attraction and repulsion. In *Proceedings of the Eighth Conference on European Chapter of the Association For Computational Linguistics* (Madrid, Spain, July 07 - 12, 1997). European Chapter Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 373-380.

Daelemans, W. 1993. Memory-based lexical acquisition and processing. *EAMT Workshop 1993*, 85-98.

Dagan, I., Pereira, F., and Lee, L. 1994. Similarity-based estimation of word cooccurrence probabilities. In *Proceedings of the 32nd Annual Meeting on Association For Computational Linguistics* (Las Cruces, New Mexico, June 27 - 30, 1994). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 272-278.

Dagan, I., Lee, L., and Pereira, F. 1997. Similarity-based methods for word sense disambiguation. In *Proceedings of the 35th Annual Meeting on Association For Computational Linguistics* (Madrid, Spain, July 07 - 12, 1997). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 56-63.

Dolan, W. B. 1995. Metaphor as an emergent property of machine-readable dictionaries. In *Proceedings of Representation and Acquisition of Lexical Knowledge: Polysemy, Ambiguity, and Generativity* (March 1995, Stanford University, CA). AAAI 1995 Spring Symposium Series, 27-29. (Technical Report MSR-TR-95-11), Redmond, WA: Microsoft Corporation.

Fass, D. 1997. *Processing metonymy and metaphor*. Greenwich, CT: Ablex Publishing Corporation.

Hahn, U. and Markert, K. 1999. On the Formal Distinction between Literal and Figurative Language. In *Proceedings of the 9th Portuguese Conference on Artificial intelligence: Progress in Artificial intelligence* (September 21 - 24, 1999). P. Barahona and J. J. Alferes, Eds. Lecture Notes In Computer Science, vol. 1695. Springer-Verlag, London, 133-147.

Karov, Y. and Edelman, S. 1998. Similarity-based word sense disambiguation. *Comput. Linguist.* 24, 1 (Mar. 1998), 41-59. (Technical Report CS96-05), Rehovot, Israel: Weizmann Institute Of Science: Mathematics & Computer Science.

Kullback, S. and Leibler, R. A. 1951. On information and sufficiency. *Annals of Mathematical Statistics*, 22, 79-86.

Lakoff, G. and Johnson, M. 1980. *Metaphors we live by*. Chicago, IL: University of Chicago Press.

Lee, L. and Pereira, F. 1999. Distributional similarity models: clustering *vs*. nearest neighbors. In *Proceedings of the 37th Annual Meeting of the Association For Computational Linguistics on Computational Linguistics* (College Park, Maryland, June 20 - 26, 1999). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 33-40.

Manning, C. D. and Schuetze, H. 1999. *Foundations of statistical natural language processing*. Cambridge, MA: The MIT Press.

Markert, K. and Nissim, M. 2002. Towards a corpus annotated for metonymies: the case of location names. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)* (Canary Islands, Spain, 27 May-2 June 2002). 1385-1392.

Martin, J. H. 1990. *A computational model of metaphor interpretation*. Toronto, ON: Academic Press, Inc.

Martin, J. H. 1992. Computer understanding of conventional metaphoric language. *Cognitive Science* 16, 2 (1992), 233-270.

Martin, J. H. 1994. A corpus-based analysis of context effects on metaphor comprehension. (Tech. Rep. No. CU-CS-738-94), Boulder: University of Colorado: Computer Science Department.

Mason, Z. J. 2004. CorMet: a computational, corpus-based conventional metaphor extraction system. *Comput. Linguist.* 30, 1 (Mar. 2004), 23-44.

Murata, M., Ma, Q., Yamamoto, A. and Isahara, H. 2000. Metonymy interpretation using x no y examples. In *Proceedings of SNLP2000* (Chiang Mai, Thailand, 10 May 2000).

Narayanan, S. 1999. Moving right along: a computational model of metaphoric reasoning about events. In *Proceedings of the Sixteenth National Conference on Artificial intelligence and the Eleventh innovative Applications of Artificial intelligence Conference innovative Applications of Artificial intelligence* (Orlando, Florida, United States, July 18 - 22, 1999). American Association for Artificial Intelligence, Menlo Park, CA, 121-127.

Newmark, P. 1980. The translation of metaphor. In Wolf Paprotté & René Dirven (Eds.), *The ubiquity of metaphor* (pp. 295-326). Philadelphia, PA: John Benjamins Publishing Company.

Nissim, M. and Markert, K. 2003. Syntactic features and word similarity for supervised metonymy resolution. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)* (Sapporo, Japan, 2003). 56-63.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program*. 14(3), 130-137.

Resnik, P. 1997. Selectional preference and sense disambiguation. In *Proceedings of the ANLP Workshop "Tagging Text with Lexical Semantics: Why What and How?"* (Washington, DC, April 4-5, 1997).

Russell, S. W. 1976. Computer understanding of metaphorically used verbs. *American Journal of Computational Linguistics*, Microfiche 44.

Schank, R. 1973. The fourteen primitive actions and their inferences. *AIM – 183*, Computer Science Department, Stanford University, Stanford, California.

Semino, E. and Steen, G. 2001. A method for the annotation of metaphors in corpora. *Interdisciplinary Workshop on Corpus-Based & Processing Approaches to Figurative Language* (Lancaster University, 27 March 2001).

Shieber, S. M. and Schabes, Y. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 3* (Helsinki, Finland, August 20 - 25, 1990). H. Karlgren, Ed. International Conference On Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 253-258.

Somers, H. 1999. Review article: Example-based machine translation. *Machine Translation* 14, 113-157.

Yarowsky, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association For Computational Linguistics* (Cambridge, Massachusetts, June 26 - 30, 1995). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 189-196.

Zernik, U. and Dyer, M. G. 1986. Disambiguation and language acquisition through the phrasal lexicon. In *Proceedings of the 11th Coference on Computational Linguistics* (Bonn, Germany, August 25 - 29, 1986). International Conference On Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 247-252.

idiom. (n.d.). *Wikipedia*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/trope

irony. (n.d.). *The Columbia Electronic Encyclopedia, Sixth Edition*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/metaphor

literal. (n.d.). *The American Heritage® Dictionary of the English Language, Fourth Edition*. Retrieved June 25, 2005, from Answers.com Web site: http://www.answers.com/topic/literal

metaphor. (n.d.). *The Columbia Electronic Encyclopedia, Sixth Edition*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/metaphor

metonymy. (n.d.). *The Columbia Electronic Encyclopedia, Sixth Edition*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/metaphor

nonliteral. (n.d.). *WordWeb Online*. Retrieved June 25, 2005, from WordWeb Online Web site: http://www.wordwebonline.com/search.pl?w=nonliteral

phrasal verb. (n.d.). *Wikipedia*. Retrieved June 25, 2005, from Answers.com Web site: http://www.answers.com/topic/phrasal-verb

synecdoche. (n.d.). *The Columbia Electronic Encyclopedia, Sixth Edition*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/metaphor

trope. (n.d.). *Poetry Glossary*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/trope

trope. (n.d.). *Wikipedia*. Retrieved June April 29, from Answers.com Web site: http://www.answers.com/topic/trope

trope. (n.d.). *WordNet 1.7.1*. Retrieved April 29, 2005, from Answers.com Web site: http://www.answers.com/topic/trope