BUILDING A LEXICAL KNOWLEDGE-BASE OF NEAR-SYNONYM DIFFERENCES

by

Diana Inkpen

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Building a Lexical Knowledge-Base of Near-Synonym Differences

Diana Inkpen

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2004

Current natural language generation or machine translation systems cannot distinguish among near-synonyms—words that share the same core meaning but vary in their lexical nuances. This is due to a lack of knowledge about differences between near-synonyms in existing computational lexical resources.

The goal of this thesis is to automatically acquire a lexical knowledge-base of near-synonym differences (LKB of NS) from multiple sources, and to show how it can be used in a practical natural language processing system.

I designed a method to automatically acquire knowledge from dictionaries of near-synonym discrimination written for human readers. An unsupervised decision-list algorithm learns patterns and words for classes of distinctions. The patterns are learned automatically, followed by a manual validation step. The extraction of distinctions between near-synonyms is entirely automatic. The main types of distinctions are: stylistic (for example, *inebriated* is more formal than *drunk*), attitudinal (for example, *skinny* is more pejorative than *slim*), and denotational (for example, *blunder* implies *accident* and *ignorance*, while *error* does not).

I enriched the initial LKB of NS with information extracted from other sources. First, information about the senses of the near-synonym was added (WordNet senses). Second, knowledge about the collocational behaviour of the near-synonyms was acquired from free text. Collocations between a word and the near-synonyms in a dictionary entry were classified into: preferred collocations, less-preferred collocations, and anti-collocations. Third, knowledge

about distinctions between near-synonyms was acquired from machine-readable dictionaries (the *General Inquirer* and the *Macquarie Dictionary*). These distinctions were merged with the initial LKB of NS, and inconsistencies were resolved.

The generic LKB of NS needs to be customized in order to be used in a natural language processing system. The parts that need customization are the core denotations and the strings that describe peripheral concepts in the denotational distinctions. To show how the LKB of NS can be used in practice, I present Xenon, a natural language generation system system that chooses the near-synonym that best matches a set of input preferences. I implemented Xenon by adding a near-synonym choice module and a near-synonym collocation module to an existing general-purpose surface realizer.

Dedication

To my beloved husband Tim, who has faith in me.

Acknowledgements

I wish to thank the following people:

- Graeme Hirst, my academic adviser, for all his guidance, for helping with many things, from research to job hunting, and for making me feel that I am part of a family.
- Suzanne Stevenson for helping when I got stuck with my research.
- Gerald Penn for his tough questions and for encouraging me.
- John Mylopoulos for being in my thesis committee.
- Irene Langkilde-Geary for letting me use HALogen and the input construction tool, and for her suggestions about how to integrate my modules with HALogen.
- Ol'ga Feiguina for being an enthusiastic research assistant, and for implementing the programs from Chapter 5.
- Philip Edmonds for his earlier work on near-synonyms and for the I-Saurus code.
- My officemate Cosmin Munteanu for his support and for helping with proofreading.
- Ramona and Cosmin Truta for their moral support.
- Greg Kondrak and Iluju Kiringa for discussions about research and job hunting.
- Vivian Tsang, Afsaneh Fazli, and Afra Alishahi for their support and for tolerating my weekly visits to their office.
- Faye Baron for her support and affection.
- Melanie Baljko and Natalia Modjeska for their sharing the struggle of finishing a PhD.
- Eric Joanis, Jane Morris, Alex Budanitsky, and Tristan Miller for helping in the judging tasks; to Jane and Eric also for interesting discussions.
- Christopher Collins, Neil Graham, Yun Niu, Jane Li, Amber Wilcox-O'Hearn, Kenneth Hoetmer, Preetam Maloor, and Robert Swier for their collegial support.
- My husband, Tim Inkpen for listening to my whining about the thesis and for his love and patience.
- Tudor Muresan for teaching me Prolog and for pushing me to apply for a PhD.
- My parents, Mariana and Ilie Zaiu, for supporting me morally and financially when I was an undergraduate student in Romania.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Near-synonyms

Near-synonyms are words that are almost synonyms, but not quite. They are not fully inter-substitutable, but rather vary in their shades of denotation or connotation, or in the components of meaning they emphasize; they may also vary in grammatical or collocational constraints. Examples of near-synonymic variations are given in Table 1.1 [Hirst, 1995]. Most of these examples are self-explanatory, except maybe the last two: *foe* emphasizes active warfare more than *enemy* does [Gove, 1984]; the distinction between *forest* and *woods* is a complex combination of size, proximity to civilization, and wildness (as determined by the type of animals and plants therein) [Room, 1981].

There are very few absolute synonyms, if they exist at all. The so-called "dictionaries of synonyms" actually contain near-synonyms. This is made clear by dictionaries such as *Webster's New Dictionary of Synonyms* [Gove, 1984] and *Choose the Right Word* (hereafter CTRW) [Hayakawa, 1994], which list clusters of similar words and explicate the differences between the words in each cluster. A page from CTRW is presented in Figure 1.1. These dictionaries are in effect dictionaries of near-synonym discrimination. Writers often turn to such resources when confronted with a choice between near-synonyms, because choosing the wrong word can

Type of variation	Example
Collocational	*task* : *job* w.r.t. *daunting*
Stylistic, formality	*pissed* : *drunk* : *inebriated*
Stylistic, force	*ruin* : *annihilate*
Expressed attitude	*skinny* : *thin* : *slim*
Emotive	*daddy* : *dad* : *father*
Continuousness	*seep* : *drip*
Emphasis on different aspects of meaning	*enemy* : *foe*
Fuzzy boundary	*woods* : *forest*

Table 1.1: Examples of near-synonymic variations.

be imprecise or awkward, or convey unwanted implications. These dictionaries are made for human consumption and they are available only on paper, not in electronic format.

Understanding the differences between near-synonyms is important for fine-grained distinctions in machine translation. For example, when translating the French word *erreur* to English, one of the near-synonyms *mistake, blooper, blunder, boner, contretemps, error, faux pas, goof, slip, solecism* could be chosen, depending on the context and on the nuances that need to be conveyed. Another application where knowledge of near-synonyms is vital is lexical choice in natural language generation systems. Such a system takes a non-linguistic input (semantic representation) and generates text. When more than one word can be used, the choice should be based on some explicit preferences. A third application is an intelligent thesaurus, which assists writers not only with lists of possible synonyms, but also with the nuances they carry.

1.2 Distinctions among near-synonyms

Near-synonyms can vary in many ways. DiMarco et al. [1993] analyzed the type of differences adduced in dictionaries of near-synonym discrimination. They found that a limited number of types occurred frequently, but an unlimited number were used [DiMarco and Hirst, 1993]. A detailed analysis of the types of variation is given by Edmonds [1999]. Some of the most

abjure. Do not confuse the verb *abjure* (renounce under oath) with the verb *adjure* (urge solemnly).

abrogate. Do not confuse the verb *abrogate* (cancel or repeal) with the verb *arrogate* (claim a power, privilege, etc., unduly).

absorb, assimilate, digest, imbibe, incorporate, ingest

These verbs, all relatively formal, indicate the taking in of one thing by another. **Absorb** is slightly more informal than the others and has, perhaps, the widest range of uses. In its most restricted sense it suggests the taking in or soaking up specifically of liquids: the liquid *absorbed* by the sponge. In more general uses *absorb* may imply the thoroughness of the action: not merely to read the chapter, but to *absorb* its meaning. Or it may stress the complete disappearance of the thing taken in within the encompassing medium: once-lovely countryside soon *absorbed* by urban sprawl. **Ingest** refers literally to the action of taking into the mouth, as food or drugs, for later absorption by the body. Figuratively, it designates any taking in and suggests the receptivity necessary for such a process: too tired to *ingest* even one more idea from the complicated philosophical essay she was reading. To **digest** is to alter food chemically in the digestive tract so that it can be *absorbed* into the bloodstream. In other uses, *digest* is like *absorb* in stressing thoroughness, but is even more emphatic. [You may completely *absorb* a stirring play in one evening, but you will be months *digesting* it.]

Assimilate is even more emphatic about the thoroughness of the taking in than either *absorb* or *digest*–in both its specific physiological and general uses. Physiologically, food is first *digested*, then *absorbed* by the bloodstream, and then *assimilated* bit by bit in each cell the blood passes. In more general uses, *assimilate*, unlike the previous verbs, often implies a third agent beside the absorber and the absorbed–an agent that directs this process: architects who *assimilate* their buildings to the environment. The process, furthermore, often implies the complete transformation of the absorbed into the absorbing medium. *Assimilate* also suggests a much slower process than *digest* and certainly than *absorb*, which can be nearly instantaneous: It would take the city generations to *assimilate* the newcomers into the patterns of a strange life.

Incorporate is the only verb here that does not have a specific use pertaining to the taking in of liquids or of food, meaning literally embody. It resembles the aspect of *assimilate* that stresses the loss of separate identity for the absorbed quantity: *incorporating* your proposals into a new system that will satisfy everyone. It is unlike *assimilate* in lacking that verb's suggestion of necessarily careful, time-consuming thoroughness.

Imbibe, while capable of uses comparable to those for *assimilate*, is mainly rooted still to its specific use for the taking in of liquids. Even this use, and certainly any others, now sound slightly archaic and excessively formal: Do you *imbibe* alcoholic beverages? See EAT. **Antonyms**: *disgorge*, *disperse*, *dissipate*, *eject*, *emit*, *exude*.

abstain, forbear, refrain

The verb **abstain** means withhold oneself from an action or self-indulgence. [There were six votes in favor, two against, and two *abstaining*; She *abstained* from drinking.] **Refrain** has to do with withholding an action temporarily, or checking a momentary desire: He *refrained* from scolding his child until the company left. To **forbear**, in its intransitive sense, is to exercise self-control, often out of motives of patience or charity. [Though impatient, the customer *forbore* to upbraid the harried sales clerk; The teacher *forbore* to report Johnnie's misbehavior to his parents.] See FORGO, FORSWEAR. **Antonyms**: *BEGIN*, *PERMIT*.

Figure 1.1: A page from *Choose the Right Word* (CTRW) by S.I. Hayakawa. Copyright ©1987. Reprinted by arrangement with HarperCollins Publishers, Inc.

relevant types of distinctions, with examples from CTRW, are presented below.

Denotational Distinctions Near-synonyms can differ in the frequency with which they express a component of their meaning (e.g., *Occasionally, **invasion** suggests a large-scale but unplanned **incursion***), in the latency (or indirectness) of the expression of the component (e.g., ***Test** strongly implies an actual application of these means*), and in fine-grained variations of the idea itself (e.g., ***Paternalistic** may suggest either benevolent rule or a style of government determined to keep the governed helpless and dependent*). The frequency is signaled by words such as *always*, *usually*, *sometimes*, *seldom*, *never*. The latency is signaled by many words in CTRW, including the obvious words *suggests*, *denotes*, *implies*, and *connotes*. The strength of a distinction is signaled by words such as *strongly* and *weakly*.

Attitudinal Distinctions Near-synonyms can convey different attitudes of the speaker towards an entity of the situation. Attitudes can be `Pejorative`, `Neutral`, or `Favourable`. Examples of sentences in CTRW expressing attitudes are: ***Blurb** is also used pejoratively to denote the extravagant and insincere praise common in such writing* and ***Placid** may have an unfavorable connotation in suggesting an unimaginative, bovine dullness of personality*. Both contain information about the pejorative attitude, though they also contain information about denotational distinctions.

Stylistic Distinctions Stylistic variations of near-synonyms concern their level of `Formality`, `Concreteness`, `Force`, `Floridity`, and `Familiarity` [Hovy, 1990]. Only the first three occur in CTRW. Examples of sentences in CTRW expressing stylistic distinctions are: ***Assistant** and **helper** are nearly identical except for the latter's greater informality* and ***Crime** may be used as an abstract noun to suggest all illegal activity*. Words that signal the degree of `Formality` include *formal*, *informal*, *formality*, and *slang*. The degree of `Concreteness` is signaled by words such as *abstract*, *concrete*, and *concretely*. `Force` can be signaled by words such as *emphatic*, and *intensification*.

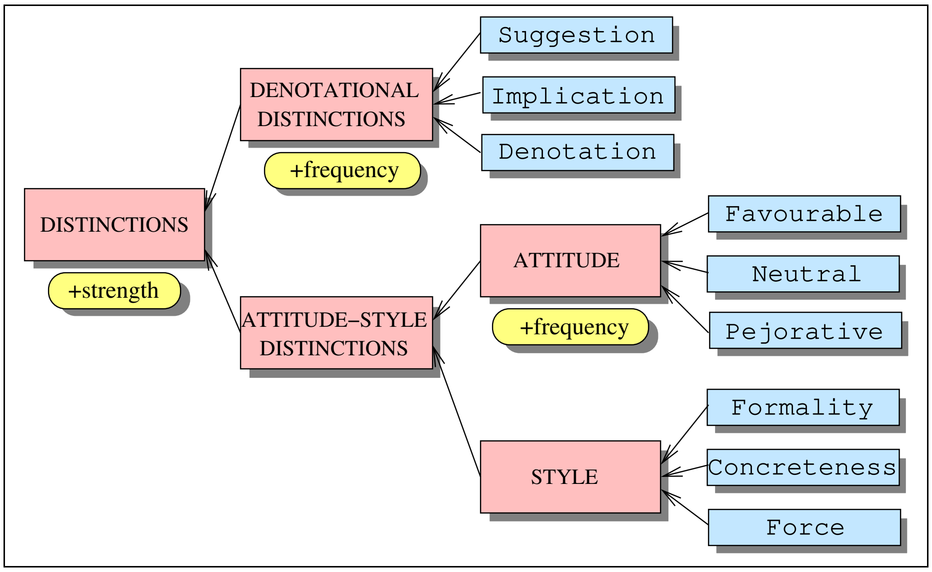

Figure 1.2: The class hierarchy of distinctions; rectangles represent classes, ovals represent attributes that a class and its descendants have.

1.2.1 The class hierarchy of distinctions

Following Edmonds's analysis of the distinctions among near-synonyms, I derived the class hierarchy of distinctions presented in Figure 1.2. The top-level class DISTINCTIONS consists of DENOTATIONAL DISTINCTIONS, ATTITUDE, and STYLE. The last two are grouped together in a class ATTITUDE-STYLE DISTINCTIONS because they are expressed by similar syntactic constructions in the text of CTRW. Therefore the algorithm described in Section 2.2 treats them together.

The leaf classes of DENOTATIONAL DISTINCTIONS are `Suggestion`, `Implication`, and `Denotation`; those of ATTITUDE are `Favourable`, `Neutral`, and `Pejorative`; those of STYLE are `Formality`, `Concreteness`, and `Force`. All these leaf nodes have the attribute strength, which takes the values `low`, `medium`, and `high`. All the leaf nodes except those in the class STYLE have the attribute frequency, which takes the values `always`, `usually`, `sometimes`, `seldom`, and `never`. Besides what is it shown in the figure, all the classes have as the first attribute the near-synonym to which the distinction belongs. The DENOTATIONAL DISTINCTIONS have an additional attribute for the peripheral concept is suggested / implied / denoted.

1.3 The clustered model of lexical knowledge

Hirst [1995] and Edmonds and Hirst [2002] show that current models of lexical knowledge used in computational systems cannot account well for the properties of near-synonyms.

The conventional view is that the denotation of a lexical item is represented as a concept or a structure of concepts (i.e., a word sense is linked to the concept it lexicalizes), which are themselves organized into an ontology. The ontology is often language-independent, or at least language-neutral, so that it can be used in multilingual applications. Words that are nearly synonymous have to be linked to their own slightly different concepts. For example, the fragment of a hierarchical model shown in Figure 1.3 needs to make many fine-grained distinctions to accommodate the near-synonyms of the word *error*. It needs to branch according to multiple criteria, such as blameworthiness, significance, stupidity, etc. Hirst [1995] showed that such a model entails an awkward taxonomic proliferation of language-specific concepts at the fringes, thereby defeating the purpose of a language-independent ontology. For example, if French near-synonyms need to be added into the model, the word *erreur* can be easily attached, but the word *bavure* (which means an unfortunate error made by the police) cannot be added without additional branching. Moreover, some words need to cut across the hierarchy, for example if a word denotes both a social error and a funny error. The hierarchical model defines words in terms of necessary and sufficient truth-conditions; therefore it cannot account for indirect expressions of meaning or for fuzzy differences between near-synonyms.

Edmonds [1999] modified this model to account for near-synonymy. The meaning of each word arises out of a context-dependent combination of a context-independent denotation and a set of explicit differences from its near-synonyms. Thus the meaning of a word consists of both necessary and sufficient conditions that allow the word to be selected by a lexical choice process and a set of nuances of indirect meaning that may be conveyed with different strengths. In this model, a conventional ontology is cut off at a coarse grain and the near-synonyms are clustered under a shared concept, rather than linking each word to a separate concept. The result is a *clustered model of lexical knowledge*.

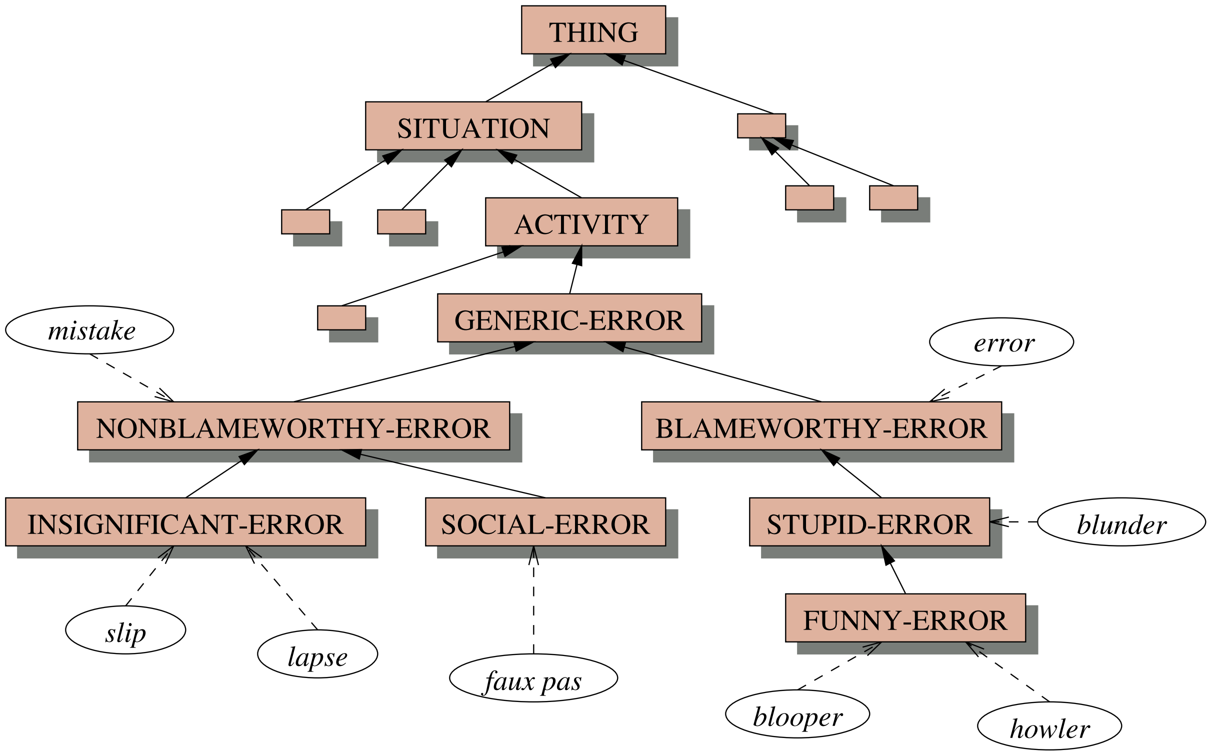

Figure 1.3: The hierarchical model of lexical knowledge. The solid lines connect concepts in a IS-A hierarchy, while the dashed lines connect words to concepts they instantiate.

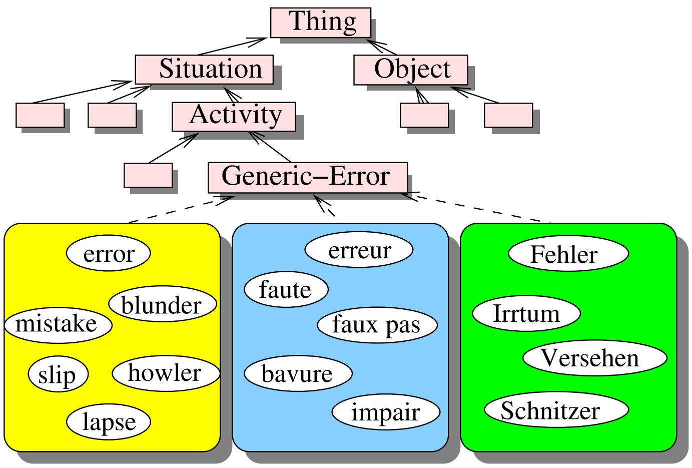

Figure 1.4: The clustered model of lexical knowledge. From [Edmonds, 1999]. The solid lines connect concepts in a IS-A hierarchy, and the dashed lines connect clusters of near-synonyms to generic concepts. The representation of the differences inside each cluster is not shown in this figure.

Each cluster has a core denotation that represents the essential shared denotational meaning of its near-synonyms. The internal structure of each cluster is complex, representing semantic (or denotational), stylistic, and expressive (or attitudinal) differences between near-synonyms. The differences or lexical nuances are expressed by means of peripheral concepts (for denotational nuances) or attributes (for nuances of style and attitude).

A clustered model has the advantage that it keeps the ontology language-neutral by representing language-specific distinctions inside the cluster of near-synonyms. For example, in Figure 1.4 there is one concept called "Generic-Error", and there is a cluster of near-synonym differences for each language. Fine-grained distinctions can be easily added to each cluster. The clusters of near-synonyms for each language do not need to be separate clusters. They can be part of one bigger cluster, but the separation is good for modularity.

Inside each cluster of near-synonyms, denotational distinctions are captured by using peripheral concepts. For example, the structure for the near-synonyms of the word *error*, built by hand by Edmonds [1999], is shown in Figure 1.5. In this model, a cluster includes the following fields:

- `syns` – a list of near-synonyms in the cluster;
- `core` – the core denotation, or essential shared meaning of the near-synonyms in the cluster, represented as a configuration of concepts;
- `periph` – a set of peripheral concepts that extend the core denotation, and pertain to the differentiation of the near-synonyms;
- `distinctions` – the actual distinctions between near-synonyms.

The *distinctions* field in Figure 1.5 contains a list of distinctions (instances of a distinction class). Each distinction has a type (the class name could be Suggestion, Implication, Denotation, Favourable, Neutral, Pejorative, Formality, Force, or Concreteness), a near-synonym to which the distinction refers, and a strength with which the distinction is expressed. Depending on the class of distinction, there are additional attributes: the frequency of expression for denotational and attitudinal distinctions, and the peripheral concept for denotational distinctions.

Building such representations by hand is difficult and time-consuming, and Edmonds completed only nine of them. My goal is to automatically extract the content of all the entries in a dictionary of near-synonym discrimination. In order to build a practical lexical knowledge-base, I use a simplified form of Edmonds's representation for the content of a cluster.

1.4 Related work

I discuss here related work on building lexical resources, and related work involving near-synonyms. A comparison of my word-sense disambiguation experiments to related work is presented in Section 3.5. A comparison of my collocation extraction experiments to related work is presented in Section 4.5.

```
(defcluster error_C
 :syns (error mistake blunder slip lapse howler)
 :core (ROOT Generic-Error)
 :periph ((P1 Stupidity) (P2 Blameworthiness)
          (P3 Criticism (ATTRIBUTE (P3-1 Severity)))
          (P4 Misconception) (P5 Accident) (P6 Inattention))
 :distinctions
 ((blunder usually medium Implication P1)
   (mistake sometimes medium Implication (P2 (DEGREE 'medium)))
   (blunder sometimes medium Implication (P2 (DEGREE 'high)))
   (mistake always medium Implication (P3-1 (DEGREE 'low)))
   (error always medium Implication (P3-1 (DEGREE 'medium)))
   (blunder always medium Implication (P3-1 (DEGREE 'high)))
   (mistake always medium Implication P4)
   (slip always medium Implication P5)
   (mistake always low Implication P5)
   (lapse always low Implication P5)
   (lapse always medium Implication P6)
   (blunder always medium Pejorative)
   (blunder high Concreteness)
   (error low Concreteness)
   (mistake low Concreteness)
   (howler low Formality)))
```

Figure 1.5: Edmonds's representation for the cluster *error*, *mistake*, *blunder*, *slip*, *lapse*, *howler*.

1.4.1 Building lexical resources

Lexical resources are vital for developing natural language processing (NLP) applications. They can be acquired by various means, depending on the type of resource.

One important resource is the lexicon. Lexicographers have developed lexicons for NLP applications manually, semi-automatically, and automatically. The ACQUILEX[1] Project explored the utility of constructing a multilingual lexical knowledge-base (LKB) from machine-readable versions of conventional dictionaries. Ide and Véronis [1994] argue that it is not possible to build a lexical knowledge-base from a machine-readable dictionary (MRD), because the information it contains may be incomplete, or it may contain circularities. But it is possible to combine information from multiple MRDs, or to enhance an existing LKB. According to them, human supervision may be needed.

Other types of knowledge can enhance existing lexicons. Information extracted from corpora augments existing lexicons with frequency information, or with knowledge about collocations (the COBUILD[2] project). Other corpus-based research concerns learning subcategorization frames and selectional restrictions [Korhonen, 2002], inducing morphology for new languages from parallel corpora [Yarowsky and Wicentowski, 2000], and lexical tuning (learning new word senses from corpora) [Wilks and Catizone, 2002].

Automatically extracting world knowledge from MRDs was attempted by projects such as MindNet at Microsoft Research [Richardson et al., 1998], and Barièrre and Popowich's project [Barrière and Popowich, 1996] that learns from children's dictionaries. IS-A hierarchies were learned automatically from MRDs [Hearst, 1992] and from corpora ([Caraballo, 1999] among others).

Work on merging information from various lexical resources is related to my work in the sense that the consistency issues to be resolved are similar. One example is the construction of

[1] http://www.cl.cam.ac.uk/Research/NL/acquilex/acqhome.html
[2] http://titania.cobuild.collins.co.uk/

UMLS (Unified Medical Language System)[3] [Lindberg et al., 1993], in the medical domain. UMLS takes a wide range of lexical and ontological resources and puts them together as a single resource. Most of this work is done manually, at the moment.

The lexical resource I acquire in this thesis contains a new type of knowledge: lexical nuances of near-synonyms. It is meant to be used together with some of the resources mentioned above. The method I use is different. As explained in Chapter 2, my algorithm for automatically acquiring knowledge from the dictionary of near-synonym differences combines ideas from Collins and Singer [1999], who classify proper names, and from Riloff and Jones [1999], who acquire domain-specific lexicons. In their turn, these researchers were inspired by Yarowsky [1995], who presents an unsupervised word sense disambiguation algorithm.

1.4.2 Work involving near-synonyms

My work is based on that of Edmonds and Hirst [2002] and Hirst [1995], in particular the model for representing the meaning of the near-synonyms presented in Section 1.3, and the preference satisfaction mechanism used in Chapter 7.

Other related research involving differences between near-synonyms has a linguistic or lexicographic flavour, rather than computational.

Apresjan built a bilingual dictionary of synonyms, more specifically a dictionary of English synonyms explained in Russian [Apresjan et al., 1980]. It contains 400 entries selected from the approximately 2500 entries from *Webster's New Dictionary of Synonyms*, but reorganized by splitting or merging clusters of synonyms, guided by lexicographic principles described by Apresjan [2000]. An entry consists of: headwords, explication (equivalent to the core meaning in my work), translation, meaning (semantic similarities and differences), notes, syntax, co-occurrence constraints, and illustrations. From these, the part called meaning includes the following types of differences: semantic, evaluative, associative and connotational, and differences in emphasis or logical stress. These differences are similar to the ones used in my

[3] http://www.nml.nih.gov/research/umls/

work. The semantic differences refer to: the subject of the action, property, or state; the object or content; the addressee; the instrument or operative part; the cause and effect; the purpose; the motivation; time; manner, nature; degree; properties (states) or their external manifestations; permanent or temporary state; active or passive mode; static or dynamic mode; mind or emotion; physical or spiritual aspects; intentional or unintentional action. This list is open-ended; therefore we could include the elements of this list in a computational representation of near-synonym differences, but we still need to allow for additional types of differences.

Gao [2001] studied the distinctions between near-synonym verbs, more specifically Chinese physical action verbs such as verbs of: cutting, putting, throwing, touching, and lying. Her dissertation presents an analysis of the types of semantic distinctions relevant to these verbs, and how they can be arranged into hierarchies on the basis of their semantic closeness.

Arppe [2002] studied the relation between Finnish morphology and near-synonymy. He showed that synonymous words can have different behaviour, depending on their inflectional form, in a language with a very rich morphology.

There is related work on representing the distinctions between near-synonyms, concerned not with a general model, but rather with the linguistic analysis of particular words, in more than one language. Wierzbicka [1997] presents a detailed analysis of the words *freedom* and *homeland* in a couple of languages, with emphasis on differences that reflect cultural differences. Mel'čuk and Wanner [2001] present a detailed semantic representation, in the style of the Text Meaning Theory, for a class of nouns. They exemplify their representation on near-synonyms of the word *appeal* in German and Russian. Their main goal is to facilitate lexical choice in transfer-based machine translation. They suggest a computational implementation using unification of feature structures. Their complex representations have to be build manually by a lexicographer. Their mechanism for choosing the best near-synonym has some similarity to my method in Chapter 7, especially the idea to use collocations of near-synonyms in the choice process.

There is related work that investigates the question of which words are considered near-synonyms, without interest in their nuances of meaning. In my work, the words that are considered near-synonyms are taken from CTRW. A different dictionary of synonyms may present slightly different views. For example a cluster may contain some extra words, some missing words, or sometimes the clustering could be done in a different way. Ploux and Ji [2003] merge clusters of near-synonyms from several dictionaries and represent them in a geometric space. They experiment with French and English near-synonyms, looking at clusters in each language and at bilingual clusters. A different approach is to automatically acquire near-synonyms from free text. Lin et al. [2003] acquire words that are related by contextual similarity, and then filter out the antonyms. They detect antonyms by using a small set of manually determined patterns (such as "either X or Y") to construct Web queries for pairs of candidate words. The problem of this approach is that it still includes words that are in relations other than near-synonymy.

Compared to lexical resources such as WordNet [Miller, 1995] (where the words in synsets are considered "absolute" synonyms, ignoring any differences between them) or thesauri (Roget's [Roget, 1852] and Macquarie [Bernard, 1987] – which contain hierarchical groups of similar words), the lexical knowledge-base of near-synonym differences I acquired includes, in addition to the words that are near-synonyms, explicit explanations of differences between these words.

1.5 Overview of contributions of thesis

In this thesis I show that it is possible to automatically acquire knowledge about the differences between near-synonyms. Denotational, attitudinal, and stylistic differences are extracted from a special dictionary of synonyms and from machine-readable dictionaries. Knowledge about the collocational behaviour of the near-synonyms is acquired from free text. The resulting lexical knowledge-base of near-synonym differences is useful in many natural language processing applications. I show how a natural language generation system can use it in order to choose the best near-synonym that matches a set of input preferences. If the preferences are lexical nuances extracted by the analysis component of a machine translation system, the translation quality would be higher: it would preserve not only the meaning of the text, but also its nuances of meaning.

To accomplish these goals, I develop a method for automatically acquiring knowledge of differences between near-synonyms, from multiple sources. I build a lexical knowledge-base of differences between English near-synonyms that can be used in NLP applications such as: natural language generation, machine translation, and writing assistance programs. I show how the LKB of NS can be used for lexical choice in Xenon, an NLG system that extends an existing sentence realization program with a near-synonym choice module and a near-synonym collocation module. The contributions of this research are the following:

Extraction patterns and knowledge acquisition I develop a method for extracting knowledge from special dictionaries of near-synonym discrimination. An unsupervised decision-list algorithm learns extraction patterns for classes of distinctions among near-synonyms. Then, from each sentence of the dictionary, denotational, attitudinal, and stylistic distinctions are extracted. The method can potentially be applied to any dictionary of near-synonym discrimination, for any language for which preprocessing tools, such as part-of-speech taggers and parsers, are available. I build a new lexical resource, an LKB of differences between English near-synonyms, by applying the extraction method to *Choose the Right Word*. The method and the actual acquisition of distinctions between near-synonyms is described in Chapter 2.

Sense disambiguation I apply a combination of existing word sense disambiguation techniques to a new task: disambiguating the senses of the near-synonyms in a dictionary entry. The dictionary entry provides a strong context for disambiguation. These experiments are presented in Chapter 3.

Acquisition of collocational knowledge I automatically acquire knowledge of collocational behaviour of near-synonyms from free text. This knowledge is useful in a lexical choice process to ensure that it chooses near-synonyms that generate preferred collocations, and avoids generating anti-collocations. I classify collocations of near-synonyms in three classes (preferred collocations, less-preferred collocations, and anti-collocations) on the basis of a differential t-test computed by using the Web as a corpus. Chapter 4 describes the method for the automatic learning of differences in the collocational behaviour of the near-synonyms.

Knowledge extraction from MRDs I show that knowledge of differences between near-synonyms can also be extracted from MRDs. In Chapter 5, I enrich the initial LKB of NS with attitudinal and stylistic distinctions extracted from special MRDs that mark words for attitude and style. I extract denotational distinctions from definitions in MRDs, by considering only the definitions that contain another near-synonym from the same cluster.

Customization of the lexical knowledge-base of near-synonym differences I show how the generic LKB of NS can be customized for use in a particular NLP system. The only parts that may require customization are the core denotations and the peripheral concepts, to ensure that they are expressed in terms of concepts understood by the system. Chapter 6 discusses the customization needs.

Utility of the lexical knowledge-base of near-synonym differences I present Xenon, an NLG system that uses the LKB of NS to choose the near-synonym that best matches a set of input preferences. Xenon extends an existing NLG system with two new modules. The near-synonym choice module matches input preferences against distinctions. The near-synonym collocation module uses the knowledge of collocational behaviour to ensure that only near-synonyms that generate preferred collocations are chosen. Chapter 7 describes the implementation and evaluation of Xenon.

Evaluation I design suitable evaluation experiments for each of the steps presented above. The evaluation of the module that extracts knowledge from CTRW is presented in Section 2.4.

The evaluation of the word sense disambiguation experiments is described in Section 3.4. The evaluation of the module that learns collocational behaviour of the near-synonyms is presented in Section 4.4. The evaluation of the module that adapts the LKB of NS to Xenon is presented in Section 6.2.3. The evaluation of Xenon is done on a subset of French and English near-synonyms. Xenon's near-synonym choice module is presented in Section 7.7.1, and Xenon's near-synonym collocations module is presented in 7.7.2.

Chapter 2

Building the Lexical Knowledge-Base of Near-Synonym Differences

The goal of this chapter is to automatically acquire a lexical knowledge-base of near-synonyms (LKB of NS) from a dictionary of near-synonym discrimination. Each entry in the dictionary enumerates a set of near-synonyms and describes the differences among them. I use the term *cluster* in a broad sense to denote both the near-synonyms from an entry and their differences. My goal is not only to automatically extract knowledge from one dictionary of synonym discrimination, but also to discover a general method which can be applied to any such dictionary with minimal adaptation.

The task can be divided into two phases, treated by two consecutive modules. The first module, called the extraction module in Figure 2.1, is described in this chapter. The generic clusters produced by this module contain the concepts that near-synonyms may involve (the peripheral concepts) as simple strings. This generic LKB of NS can be adapted for use in any NLP application. The second module customizes the LKB of NS, so that it satisfies the requirements of the particular system that employs it. This customization module deals with the core denotations and the peripheral concepts. It transforms the strings from the generic clusters into concepts in the particular ontology. An example of a customization module is

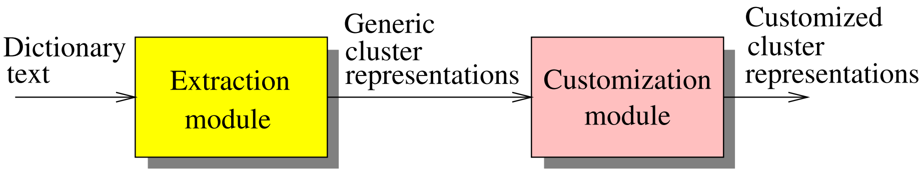

Figure 2.1: The two modules of the task.

described in Chapter 6.

In order to automatically derive a lexical knowledge-base of near-synonyms from a dictionary of near-synonym discrimination, I rely on the hypothesis that the language of the entries contains enough regularities to allow automatic extraction of knowledge from them. The dictionary of near-synonym differences that I use is *Choose the Right Word* [Hayakawa, 1994] (CTRW).[1] A page from this dictionary is presented in Figure 1.1.

CTRW contains 909 clusters, with a total of 14,138 sentences (excluding examples of usage), from which I derive the lexical knowledge-base. An example of results of this phase, corresponding to the second, third, and fourth sentence for the *absorb* cluster in Figure 1.1 (repeated here without the examples of usage: "**Absorb** is slightly more informal than the others and has, perhaps, the widest range of uses. In its most restricted sense it suggests the taking in or soaking up specifically of liquids. In more general uses *absorb* may imply the thoroughness of the action."), is presented in Figure 2.2.

The next sections describe the extraction module, whose architecture is presented in Figure 2.3. It has two main parts. First, it learns extraction patterns; then it applies the patterns to extract differences between near-synonyms. Earlier versions of this work were previously published in [Inkpen and Hirst, 2001a] and [Inkpen and Hirst, 2001b].

[1] I am grateful to HarperCollins Publishers, Inc. for permission to use CTRW in this project.

```
Cluster: absorb, assimilate, digest, imbibe, incorporate, ingest

subj: absorb
freq: usually
strength: low
class: Formality

subj: absorb
freq: usually
strength: medium
class: Suggestion
periph: the taking in of liquids

subj: absorb
freq: sometimes
strength: medium
class: Implication
periph: the thoroughness of the action
...
```

Figure 2.2: Example of distinctions extracted from *CTRW*.

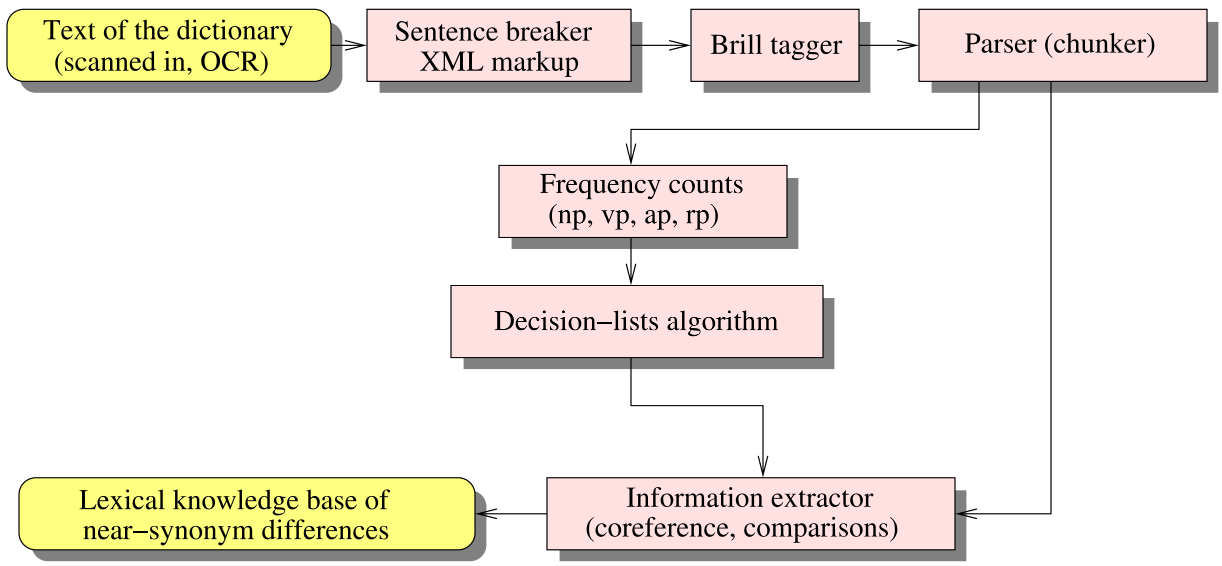

Figure 2.3: The architecture of the extraction module.

```
<s punctuation="."> These verbs, all relatively formal, indicate the taking in of one
thing by another </s>
<s punctuation="."> <near_syn><b>Absorb</b></near_syn> is slightly more informal than
the others and has, perhaps, the widest range of uses </s>
<s punctuation=":"> In its most restricted sense it suggests the taking in or soaking up
specifically of liquids </s>
<eg type=":" punctuation="."> the liquid <near_syn><i>absorbed</i></near_syn> by the
sponge </eg>
```

Figure 2.4: Example of text from CTRW with XML markup.

2.1 Preprocessing the dictionary

After OCR scanning of CTRW and error correction, I used XML markup to segment the text of the dictionary into: cluster name, cluster identifier, members (the near-synonyms in the cluster), entry (the textual description of the meaning of the near-synonyms and of the differences among them), cluster's part-of-speech, cross-references to other clusters, and antonyms list.

Sentence boundaries are detected using general heuristics, plus specific heuristics for this particular dictionary; e.g., examples are in square brackets or after a colon. The tag <s> marks sentences that describe the nuances of meaning, and the tag <eg> marks examples using the near-synonyms. Each occurrence of a near-synonym is marked with the tag <near_syn>. An example of segmented content from Figure 1.1 is given in Figure 2.1.

2.2 The decision-list learning algorithm

Before the system can extract differences between near-synonyms, it needs to learn extraction patterns. For each leaf class in the hierarchy (Figure 1.2 on page 5) the goal is to learn a set of words and expressions from CTRW (extraction patterns) that characterizes descriptions of the class. Then, during the extraction phase, for each sentence in CTRW (or fragment of a sentence) the program will decide which leaf class is expressed, with what strength and what frequency.

I use a decision-list algorithm to learn sets of words and extraction patterns for the classes DENOTATIONAL DISTINCTIONS and ATTITUDE-STYLE DISTINCTIONS. These are further split down for each leaf class, as explained in section 2.3.

The algorithm I implemented is inspired by the work of Yarowsky [1995] on word sense disambiguation. He classified the senses of a word on the basis of other words that the given word co-occurs with. Collins and Singer [1999] classified proper names as `Person`, `Organization`, or `Location` using contextual rules (other words appearing in the context of the proper names) and spelling rules (words in proper names). Starting with a few spelling rules (some proper-name features) in the decision list, their algorithm learns new contextual rules; using these rules then it learns more spelling rules, and so on, in a process of mutual bootstrapping. Riloff and Jones [1999] learned domain-specific lexicons and extraction patterns (such as *shot in* $\langle x \rangle$ for the terrorism domain). They used a mutual bootstrapping technique to alternately select the best extraction pattern for a category and add its extractions to the semantic lexicon; the newly added entries in the lexicon help in the selection of the next best extraction pattern.

My decision-list (DL) algorithm (Figure 2.5) is tailored for extraction from CTRW. It learns two different types of rules. Main rules are for words that are significant for distinction classes. Auxiliary rules are for frequency words, strength words, and comparison words. Mutual bootstrapping in the algorithm alternates between the two types.

The idea behind the algorithm is that starting with a few main rules (seed words), the program selects examples containing them and learns a few auxiliary rules. Using these, it selects more examples and learns new main rules. It keeps iterating until no more new rules are learned.

My program learns two kinds of rules, similarly to Collins and Singer's algorithm. Moreover, my program also extracts patterns and relevant words for the classes DENOTATIONAL DISTINCTIONS and ATTITUDE-STYLE DISTINCTIONS, similar to the domain-specific lexicon extracted by Riloff and Jones.

Input: Set E of training examples, class, main seed words for class, part-of-speech (pos) for words that are to be in mainDL, and pos for words that are to be in auxDL.

Output: Two decision lists for the given class: main decision list (mainDL) and auxiliary decision list (auxDL), plus list E' of patterns for the class. (Each decision list contains rules of the form $x \rightarrow h(x)$, meaning that the word x is significant for that class with confidence $h(x)$ computed by Equation 2.1.)

1. Set $N = 10$, the maximum number of rules to be induced at each step.
2. Initialization: Set the mainDL to the set of main seed words (with confidence 0.99). Set E' to empty set.
3. Add to mainDL those words in chunks from E that have the same stem as any words already in mainDL. (For example, if *suggest* is in mainDL, add *suggests*, *suggesting*, *suggested*, *suggestion*.)
4. Select examples (chunks) from $E - E'$ that contain words in mainDL, and add them to E'.
5. Use E' to compute more auxiliary rules. For each word x not in any DL, compute the confidence $h(x)$ using Equation 2.1. Take the N highest values and add them to auxDL.
6. Select more examples from $E - E'$ using auxDL, and add them to E'. Stop if E' is unchanged.
7. Using the new E', compute more main rules. For each word x not in any DL, compute the confidence $h(x)$. Take the N highest values and add them to mainDL.
8. Go to step 3 unless E' is unchanged.

Figure 2.5: The decision-list learning algorithm.

In order to obtain input data, I replace all the near-synonyms in the text of the dictionary with the term `near_syn`; then I chunk the text with Abney's chunker [Abney, 1996]. The training set E is composed of all the verb phrases, noun phrases, adjectival phrases, and adverbial phrases (denoted `vx`, `nx`, `ax`, `rx`, respectively) that occur more than t times in the text of the dictionary (where $t = 3$ in my experiments). Phrases that occur very few times are not likely to be significant patterns and eliminating them makes the algorithm faster (fewer iterations are needed).

The program learns rules of the form: word x is significant for the given class with confidence $h(x)$. All the rules $x \to h(x)$ for that class form a decision list that allows us to compute the confidence with which new patterns are significant for the class.

The confidence of a word x is computed with the formula:

$$h(x) = \frac{count(x, E') + \alpha}{count(x, E) + k\alpha} \tag{2.1}$$

where E' is the set of patterns selected for the class, and E is the set of all input data. Following Collins and Singer [1999], $k = 2$, because there are two partitions (relevant and irrelevant for the class). $\alpha = 0.1$ is a smoothing parameter. So, I count how many times x is in the patterns selected for the class versus the total number of occurrences in the training data.

I apply the DL algorithm for each of the classes DENOTATIONAL DISTINCTIONS and ATTITUDE-STYLE DISTINCTIONS. The input to the algorithm is: the set E of all chunks, the main seed words, and the restrictions on the part-of-speech (pos) of the words in main and auxiliary rules. For the class DENOTATIONAL DISTINCTIONS the main seed words are: *suggest*, *imply*, *denote*, *mean*, *designate*, *connote*; the words in main rules are verbs and nouns, and the words in auxiliary rules are adverbs and modals. For the class ATTITUDE-STYLE DISTINCTIONS the main seed words are: *formal*, *informal*, *pejorative*, *disapproval*, *favorable*, *abstract*, *concrete*; the words in main rules are adjectives and nouns, and the words in auxiliary rules are adverbs.

For example, for the class DENOTATIONAL DISTINCTIONS, starting with the rule *suggest* $\to$ 0.99, the program selects examples such as these (where the numbers give the frequency in the training data):

```
[vx [md can] [vb suggest]]--150
[vx [rb sometimes] [vb suggest]]--12
```

Auxiliary rules are learned for the words *sometimes* and *can* with confidence factors given by the count of these words in the current set of selected examples compared with the count in the rest of the set of examples. Using the new auxiliary rules for the words *sometimes* and *can*, the program selects more examples such as these:

```
[vx [md can] [vb refer]]--268
[vx [md may] [rb sometimes] [vb imply]]--3
```

From these, new main rules are learned, for the words *refer* and *imply*. Using new main rules more auxiliary rules are selected – for the word *may*, and so on.

The ATTITUDE and STYLE classes had to be considered together because both of them use adjectival comparisons. Examples of ATTITUDE-STYLE DISTINCTIONS class are these:

```
[ax [rbs most] [jj formal]]--54
[ax [rb much] [more more] [jj formal]]--9
[ax [rbs most] [jj concrete]]--5
```

For this example, main rules contain the words *formal* and *concrete*, and auxiliary rules *much*, *more*, and *most*.

2.3 Classification and extraction

After I run the DL algorithm for the class DENOTATIONAL DISTINCTIONS, I manually split the words in the list mainDL into three classes: `Suggestion`, `Implication`, and `Denotation`. This sub-classification is manual for lack of a better procedure. Furthermore, some words can be insignificant for any class (e.g., the word *also*) or for the given class; during the sub-classification I mark them as OTHER. I repeat the same procedure for frequencies and strengths with the words in auxDL. The words marked as OTHER and the patterns which do not contain any word from mainDL are automatically ignored in the next processing steps.

After I run the algorithm for the class ATTITUDE-STYLE DISTINCTIONS, I split the words in the list mainDL into two classes: ATTITUDE and STYLE. I split ATTITUDE into `Favourable`, `Neutral`, `Pejorative`. and STYLE into `Formality`, `Concreteness`, `Force`. Frequencies will be computed from the auxDL list, and strengths will be computed by a module which resolves comparisons.

Once I obtained the words and patterns for all the classes, I implemented an automatic knowledge-extraction component. This program takes each sentence in CTRW and tries to extract one or more pieces of knowledge from it.

The information extracted for denotational distinctions has the fields: subject (which near-synonym), frequency, strength, class, and peripheral concept. The class takes the value `Suggestion`, `Denotation`, or `Implication`. The peripheral concept is a string extracted from the sentence. Strength takes the value `low`, `medium`, or `high`. Frequency takes the value `always`, `usually`, `sometimes`, `seldom`, or `never`. Default values (`usually` and `medium`) are used when the strength and the frequency are not specified in the sentence.

The information extracted for attitudinal distinctions has the fields: subject, frequency, strength, and class, where strength and frequency have the same values and significance as in the case of denotational distinctions, and class can be `Favourable`, `Neutral`, or `Pejorative`.

The information extracted for stylistic distinctions has the fields: subject, strength, and class, where the class can be `Formality`, `Force`, or `Concreteness`. Strength has the value `low`, `medium`, or `high`, indicating the level of the stylistic attribute.

The extraction program considers what near-synonyms each sentence fragment is about (most often expressed as the subject of the sentence), what the expressed distinction is, and with what frequency and relative strength. If it is a denotational distinction, then the peripheral concept involved has to also be extracted (from the object position in the sentence). Therefore, my program extracts the subject of the sentence (the first noun phrase before the main verb) and the object of the sentence (the first noun phrase after the main verb). This heuristic for extracting information works for sentences that present one piece of information. But there are many sentences that present two ore more pieces of information. In such cases, my program splits a sentence into coordinated clauses (or coordinated verb phrases), by using a parser [Collins, 1996] to distinguish when a coordinating conjunction (*and*, *but*, *whereas*) is conjoining two main clauses or two parts of a complex VP. For example:

```
near_syn(acumen) has to do with keenness of intellect and
implies an uncommon quickness and discrimination of mind

NP has to do with NP and implies NP
```

From 60 randomly selected sentences, 52 were correctly dealt with (41 needed no split, 11 were correctly split). The 8 mistakes included 3 sentences that were split but shouldn't have been, and 5 that needed splitting and they were not. The mistakes were mainly due to wrong parse trees. Therefore, the accuracy was 86.6%.

When no information is extracted in this way, a few patterns are applied. An example of extraction pattern is: `To NS1 is to NS2 ...`. There are also heuristics to retrieve compound-subjects of the form *near_syn and near_syn* or *near_syn, near_syn, and near_syn*. Once the class is determined to be either DENOTATIONAL DISTINCTIONS or ATTITUDE-STYLE DISTINCTIONS, the target class (one of the leaves in the class hierarchy in Figure 1.2 on page 5) is determined by using the manual partitions of the rules in the mainDL of the two classes.

Coreferences and Comparisons Sometimes the subject of a sentence refers to a group of near-synonyms. For example, if the subject is *the remaining words*, my program needs to assert information about the near-synonyms from the same cluster not mentioned yet in the text.

In order to implement coreference resolution, I applied the same DL algorithm to retrieve expressions used to refer to near-synonyms or groups of near-synonyms. When running the algorithm with the seeds *noun, word, term, verb, adverb, adjective*, the expressions retrieved look like these:

```
[nx [dtp these] [nns adjectives]]--397
[nx [dtp these] [nns nouns]]--372
```

```
[nx [dtp these] [nns verbs]]--330
[nx [dt the] [jj other] [nns adjectives]]--43
[nx [dt the] [vbg remaining] [nns nouns]]--28
```

The auxiliary words include: *the, three, both, preceding, previous, remaining, other*. Using these auxiliary words, more coreferences are resolved. Any time the subject is one of the main words (*noun, word, term, verb, adverb, adjective, preposition, nouns, words, terms, verbs, adverbs, adjectives, pair*), if there is an auxiliary word, the meaning is modified accordingly. This is done by manually encoding it into the program. For example, the expression *the remaining verbs* will cause the program to compute the set of near-synonyms of that entry not processed yet to that point.

Another case my extraction program needs to deal with is when stylistic or attitudinal distinctions are expressed relative to other near-synonyms in the cluster. Such comparisons are resolved in a simple way, by considering only three absolute values: (`low`, `medium`, `high`). I explicitly tell the system which words represent what absolute values of the corresponding distinction (e.g., *abstract* is at the low end of `Concreteness`), and how the comparison terms increase or decrease the absolute value (e.g., *less abstract* could mean a `medium` value of `Concreteness`).

2.4 Evaluation

CTRW contains 909 clusters that group 5452 near-synonyms (more precisely near-synonym senses, because a word can be in more than one cluster). The explanatory text of all the entries consists of 14,138 sentences (without counting examples of use). My program is able to extract 12,365 distinctions from 7450 of the sentences. The rest of the sentences usually do not contain directly expressed distinctions. Here is an example of such a sentence: *A* **terror-stricken** *person who is drowning may in panic resist the efforts of someone who is trying to save him.*

In order to evaluate the final results, I randomly selected 25 clusters as a development set, and another 25 clusters as a test set. The development set was used to tune the program by adding new patterns if they helped improve the results. The test set was used exclusively for testing. I built by hand a standard solution for each set. The results of my algorithm on the development and test set need to be compared with the results of a baseline algorithm. The baseline algorithm chooses the default values whenever it is possible; it is not possible for peripheral concepts (the direct object in the sentence) and for the near-synonyms the sentence is about (the subject in the sentence). In this case, the baseline algorithm relies only on tuples extracted by the chunker to extract the subjects and the objects.

The measures I use for evaluating each piece of information extracted from a sentence fragment are *precision* and *recall*. The results to be evaluated have four constituents for ATTITUDE-STYLE DISTINCTIONS and five constituents for DENOTATIONAL DISTINCTIONS. There could be missing constituents (except strength and frequency, which take default values). Precision is the total number of correct constituents found (summed over all the sentences in the test set) divided by the total number of constituents found. Recall is the total number of correct constituents found divided by the number of constituents in the standard solution.

For example, for the sentence *Sometimes, however, **profit** can refer to gains outside the context of moneymaking*, the program obtains:

```
subj: profit
freq: usually
strength: medium
class: Denotation
periph: gains outside the context of moneymaking
```

while the solution is:

```
subj: profit
freq: sometimes
strength: medium
class: Denotation
periph: gains outside the context of moneymaking
```

The precision is .80 (4 correct out of 5 found), and the recall is .80 (4 correct out of 5 in the standard solution). for simplicity, all the mistakes are equally penalized. Some mistakes could be more heavily penalized than others. For example, if the extracted frequency is `sometimes` and the solution is `never`, this mistake is bigger than in the case when the solution is `usually`.

	Baseline algorithm		My system (dev. set)		My system (test set)	
	Precision	Recall	Precision	Recall	Precision	Recall
All constituents	.40	.23	.76	.69	**.83**	**.73**
Class only	.49	.28	.79	.70	.82	.71

Table 2.1: Labeled precision and labeled recall of the baseline and of my algorithm.

Table 2.1 presents the results of the evaluation. The first row of the table presents the results as a whole (all the constituents of the extracted lexical knowledge-base). My system (on the development set) increases precision by 36% and recall by 46% over the baseline. The recall and precision on the test set are similar to the ones on the development set. They are slightly higher on the test set; this shows that the patterns added during the development stage were general.

The second row of the table gives the evaluation results only for the class of the distinction expressed, ignoring the strengths, frequencies, and peripheral concepts. This allows for a more direct evaluation of the acquired extraction patterns. In this case, the baseline algorithm attains higher precision than in the case when all the constituents are considered, because the default class `Denotation` is the most frequent in CTRW. My algorithm attains slightly higher precision and recall on the development set, probably due to a few cases in which the frequency and strength were incorrectly extracted, and slightly lower on the test set, probably due to some cases in which the frequency and strength were easy to extract correctly.

2.5 Semi-automatic acquisition

For some intended uses of the automatically acquired LKB of NS, its quality might not be enough. If the LKB of NS needs to be 100% correct, then a human lexicographer can participate in the acquistion process, in a semi-automated manner. The lexicographer would validate or change each distinction asserted in the LKB. For the acquisition from CTRW, the evaluation experiment showed estimated precision and recall in the range of 70 - 80%. This means that the lexicographer will have to validate the automatically extracted knowledge in 70 - 80% of the cases, and to manually correct or extract in the rest of the cases.

Future work could investigate the possibility that the extraction algorithm computes confidence factors for each extracted distinction; in this case the lexicographer needs to inspect only the distinctions for which the confidence factor is low.

Knowledge extracted from additional sources, such as the ones used in Chapter 5, can confirm or bring into doubt asserted knowledge. This is done implicitly for stylistic and attitudinal distinctions by the conflict resolution method in Chapter 5. If the human lexicographer is part of the system, she can skip inspecting knowledge confirmed by multiple sources.

2.6 Summary

This chapter presented a general method for extracting knowledge from a dictionary of near-synonym differences. The main idea was to learn words and patterns for the classes of interest (in this case denotational distinctions and attitude-style distinctions) by using an unsupervised decision-list algorithm. Then the words and patterns were used for extracting information from the sentences of the dictionary. The extracted information is: what near-synonym(s) the part of the sentence is talking about, what class of distinction is expressed, with what frequency and strength. In the case of denotational distinctions, a peripheral string that shows what is implied, connoted, or denoted, was also extracted. The method was applied to the dictionary of near-synonym differences *Choose the Right Word*. Evaluation of the results was done on a sample of randomly chosen clusters of near-synonyms. A development set was used for adding a few patterns; a separate test set was used for evaluation.

The result of this chapter is a generic lexical knowledge-base of near-synonym differences. It will be later enriched with knowledge from other sources: information about the senses of the near-synonyms is added in Chapter 3; information about the collocational behaviour of the near-synonyms is added in Chapter 4; and more distinctions acquired from machine-readable dictionaries are added in Chapter 5. To be used in a particular NLP system, the generic LKB of NS needs to be customized (Chapter 6). Chapter 7 shows how the customized LKB of NS can actually be used in NLG.

Chapter 3

Disambiguating the Senses of the Near-Synonyms in a Dictionary Entry

In this chapter, the lexical knowledge-base of near-synonym differences is extended with information about which senses of the near-synonyms are relevant for each cluster (for each entry in CTRW). This information can be useful if the LKB of NS is integrated into an NLP system based on WordNet. The sense inventory used in this chapter is WordNet1.7.

Each entry in CTRW explains the differences between the near-synonyms in that cluster. But some of the near-synonyms are ambiguous words; therefore some particular senses of a near-synonym are distinguished from some senses of the other near-synonyms. For example, if the words *bank* and *trust* are in a cluster of near-synonyms, the financial sense of *bank* (and not the river-bank sense) is distinguished from the financial sense of *trust* (and not the sense denoting confidence). The problem of ambiguity is further complicated by polysemy (related senses grouped together).

The task of automatically disambiguating the meaning of near-synonyms is easier than the general task of word sense disambiguation (WSD), which consists of selecting one or more senses in which a word is being used in a particular sentence. But disambiguating the meaning of the near-synonyms is not a simple task. As will be shown in section 3.3, it is not easy even for humans.

I implemented a program that decides for each sense of a near-synonym whether it is relevant for the entry or not. These experiments were previously published in [Inkpen and Hirst, 2003]. For example, the near-synonym *acumen* from the cluster *acumen, acuity, insight, perception* has two WordNet senses: `acumen#n#1` glossed as "a tapering point", and `acumen#n#2` glossed as "shrewdness shown by keen insight". The decision to be made is that the second one is relevant for the entry, and the first one is not. More than one sense of a near-synonym can be relevant for an entry, so the problem is one of binary decisions: for each sense, decide if is is relevant for the context or not. To disambiguate each sense, a series of indicators are computed and combined in order to decide if the sense is relevant.

In this task, the context is richer than in the general case of word sense disambiguation: the full text of each entry (including the cross-references). For each entry in CTRW, all senses of each near-synonym are considered. The average polysemy for CTRW is 3.18 (for 5,419 near-synonyms there are 17,267 WordNet senses).

3.1 Indicators of word sense relevance

Intersection of text and gloss A main indicator of word sense relevance is the size of the intersection of the text of the entry with the WordNet gloss of the sense, both regarded as bags of words. This is a Lesk-style approach. This method [Lesk, 1986] determines the correct word sense in a context by counting overlaps between the context and the dictionary definitions of the possible senses of the word. In the intersection of the text and the gloss, the stopwords and the word to be disambiguated are ignored. (I experimented with stemming the words, but it did not improve the results.) The other near-synonyms occur in the text of the entry; if they happen to occur in the gloss, this is a good indication that the sense is relevant.

Sometimes the intersection contains only very common words that do not reflect a real overlapping of meaning. In order to avoid such cases, each word in the intersection is weighted by its *tf·idf* score. The weight of each word i in the entry j is $tf{\cdot}idf_{i,j} = n_{i,j} \log \frac{n_i}{N}$, where $n_{i,j}$ is the number of occurrences of the word i in the entry j, n_i is the number of entries that contain the word i, and N is the total number of entries. Then, the weights of the words in the intersection are summed to produce a score for the intersection. If the score of the intersection is lower than a threshold, the sense is not relevant. Instead of choosing one such threshold, I trained a decision tree to choose a series of thresholds (see section 3.2).

As another indicator of sense relevance, I computed the intersection of the text of the entry with the glosses of words that have a direct WordNet relation with the sense under consideration. The hyponym/hypernym glosses can be expected to work well because some of the near-synonyms in CTRW are in a hypernymy/hyponymy relation with each other.

Other words in synsets being near-synonyms Another indicator of the relevance of a sense is the other words in each synset. They reliably indicate a sense being relevant for the entry because the near-synonyms in the entry help disambiguate each other. For example, if the cluster is: *afraid, aghast, alarmed, anxious, apprehensive, fearful, frightened, scared*, when examining the senses of *anxious*, the sense corresponding to the synset `anxious#a#1`, `apprehensive#a#2` is relevant because the other word in the synset is *apprehensive*, which is one of the near-synonyms. If there is such overlap in more than one synset, all the sysnset with overlap are classified as relevant. For each synset, the value of the indicator is 1 if other words in synsets are near-synonyms, and zero otherwise.

I also used the words in synsets of related words, where by related words we mean words connected by a direct WordNet relation. If any of the words in the synsets of the words related to the sense under consideration happens to be a near-synonym in the same cluster, the sense can be judged as relevant.

Antonyms The antonyms in the entry are intersected with the antonyms of the sense under consideration. Figure 1.1 on page 3 shows examples of antonyms for a cluster. If two words share an antonym, they are likely to be synonyms. By extension, if the examined sense has antonyms that intersect the antonyms of the cluster, then the sense is relevant for the cluster. For each synset, the value of the indicator is 1 if there are antonyms in common, and zero otherwise.

Context vectors Sometimes, when the intersection of text and gloss is empty, it still could be the case that they are semantically close. For example, for the sense `reserved#a#2` with the WordNet gloss "marked by self-restraint and reticence", the intersection with the text of the CTRW entry *aloof, detached, reserved* is empty. The text of the entry happens to not use any of the words in the WordNet gloss, but the entry contains semantically close words such as *reluctant* and *distant*. By considering second-order co-occurrences (words that co-occur in the BNC with the words of the text or of the gloss) the chance of detecting such similarity increases [Schütze, 1998]. One problem with this approach is that false positives can be also introduced.

I collected frequencies from the 100-million-word British National Corpus[1] (BNC). I chose the 2,000 most frequent words as dimensions, and the 20,000 most frequent words as "features". By counting how many times each "feature" word co-occurs with a dimension word in the BNC, the "feature" words can be represented in the vector space of the dimensions. Then, the vectors of all "feature" words in an entry (except the near-synonym to be disambiguated) are summed to compute the context vector for the entry. The vectors of all words in a gloss are summed to get the context vector for the gloss. The cosine between the two vectors measures how close the two vectors are. The context vector for the entry will be the sum of many vectors, and it may be a longer vector than the context vector for the gloss, but this does not matter because I measure only the angle between the two vectors. Figure 3.1 presents a simplified example of context vectors for the second sense in of *acumen*. For simplicity, only two dimensions are represented: *plant* and *mental*. Also, from the four content words in the gloss, three happen to be "feature" words, and only *keen* and *insight* are presented in the figure. The

[1] http://www.hcu.ox.ac.uk/BNC/

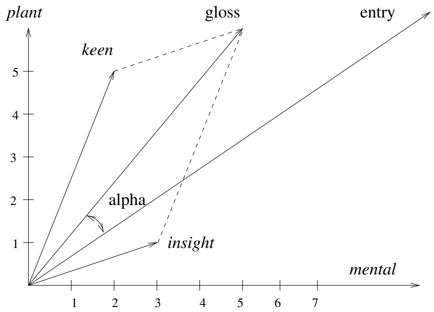

Figure 3.1: Context vectors in a 2D space for the words *keen* and *insight*, for the WordNet gloss of the second sense of *acumen*, and for the CTRW entry for the cluster *acumen, acuity, insight, perception*.

context vector of the gloss is sum of these two (or more) vectors. In a similar manner the context vector for the entry is obtained, and the cosine of the angle α between the two context vectors is used as an indicator for the relevance of the sense. Here, the cosine is 0.909, while the cosine between the context vector for the entry and the context vector for the first sense of *acumen* is 0.539. Probably it would be better to choose as "feature" words, instead of the 2000 most-frequent words in the BNC, but 2000 frequent words that occur in both the BNC and the WordNet glosses. This may reduce the number of cases when very few "feature" words occur in the gloss of a synset, making the context vectors of the synsets more reliable. But the problem may still remain. For example, Niwa and Nitta [1994] show that for their word-sense disambiguation task, the performance was higher when using co-occurrence vectors from the 1987 Wall Street Journal (20 million words) than when using distance vectors from the Collins English Dictionary (60,000 head-words and 1.6 million words in definitions).

3.2 Using a decision tree to combine indicators

I use decision tree learning to determine the best combination of indicators. I use C4.5[2] on my 904 data points (all the senses of all the near-synonyms), using six attributes (features):

[2]http://www.cse.unsw.edu.au/~quinlan/

```
intersection_text_gloss > 4.41774 : Y (406.0/59.4)
intersection_text_gloss <= 4.41774 :
| intersection_text_gloss_related_words > 23.7239 : Y (28.0/1.4)
| intersection_text_gloss_related_words <= 23.7239 :
| | words_in_related_synsets = 0:
| | | words_in_synset = 0:
| | | | intersection_text_gloss_related_words <= 4.61842 : N (367.0/62.5)
| | | | intersection_text_gloss_related_words > 4.61842 :
| | | | | intersection_text_gloss_related_words <= 4.94367 : Y (4.0/1.2)
| | | | | intersection_text_gloss_related_words > 4.94367 : N (42.0/14.6)
| | | words_in_synset = 1:
| | | | intersection_text_gloss <= 1.19887 : N (16.0/8.9)
| | | | intersection_text_gloss > 1.19887 : Y (3.0/1.1)
| | words_in_related_synsets = 1:
| | | intersection_text_gloss <= 0.89000 : Y (24.0/4.9)
| | | intersection_text_gloss > 0.89000 :
| | | | cosine <= 0.856407 : Y (3.0/2.1)
| | | | cosine > 0.856407 : N (5.0/1.2)
```

Figure 3.2: Simplified decision tree for the combination of indicators.

intersection text and gloss (numerical value), intersection of text and gloss of related words (numerical value), words in synset (0 or 1), words in synsets of related words (0 or 1), antonyms (0 or 1), and the cosine between context vectors (numerical value). The classification is binary: Y/N, meaning relevant or not relevant for the entry. See Figure 3.2 for a simplified decision tree that combines indicators.

I experimented with manual combinations, but using a decision tree is better because this learning mechanism has the ability to decide which indicators have more influence on the classification, and it can completely ignore indicators with low influence. Another advantage of the decision tree is that it determines the best values for the thresholds for weighted intersection and for cosine. I use the standard solution built in section 3.3 as training and test data in the decision-tree learning process. Instead of splitting this data into a training set and a test set, I did a 10-fold cross-validation, as a better method to estimate the error rate.

3.3 Building a standard solution

The goal of using human judges in my work was twofold: to get a measure of how difficult the task is for humans, and to build a standard solution for use in evaluation. The standard solution also serves as training and test data for the decision tree used in section 3.2.

There were $k = 6$ judges (native or near-native speakers of English, five of them with computational linguistics background) doing the same job as the WSD program. I randomly selected 50 of the 909 clusters, containing 282 near-synonyms with 904 senses in total. The judges were presented with the text of the entry for each cluster, including antonyms and cross-references. For each near-synonym, all the WordNet senses (with their glosses and all the words in the synset) were listed, and the judges had to decide whether the sense is relevant for the cluster or not. The judges had no information about hypernyms, hyponyms, or antonyms.

There were 904 decisions the judges had to make. If we consider the decisions as votes, for 584 decisions, the judges voted 6–0 (or 0–6), for 156 decisions 5–1, and for 108 decisions 4–2. There were 56 ties (3–3).

The percent agreement among our judges was 85%. To get a more accurate measure of the agreement among the k judges, I used the well-known kappa statistic [Siegel and Castellan, 1988] [Carletta, 1996], that factors in the probability of agreement by chance. For the task at hand, the chance agreement is 50.2%. Therefore the kappa coefficient is $\kappa = 0.699$. The figures of agreement between pairs of two judges vary from 90% ($\kappa = 0.80$) to 78.8% ($\kappa = 0.57$).

The judges met to discuss the ties. The discussion had a very small influence on the agreement figures (because the number of cases discussed was small), but it helped clarify the sources of disagreement. Senses which are "instances" or "proper names" (e.g. the sense "*the United States*" for the near-synonym *union*) were rejected by some judges as too specific, even if they were mentioned in the text of the entry. There was disagreement about intransitive senses of some transitive verbs (or the other way around). Another problem was posed by mentions of extended senses (literal or figurative senses) in the text. For example, the CTRW entry for *bombastic, orotund, purple, turgid* mentions that "these adjectives are used to describe

Method or combination of methods	Accuracy
Baseline (select all senses as relevant for the entry)	53.5%
Antonyms	47.0%
Cosine (decision tree)	52.7%
Words in synsets of hypernyms and hyponyms	56.4%
Intersection text & gloss of hypernyms and hyponyms (*tf·idf*)	61.0%
Words in synsets of related words	61.3%
Words in synset	67.1%
Intersection text & gloss of related words (*tf·idf*) (decision tree)	70.6%
Intersection text & gloss (no *tf·idf*)	76.8%
Intersection text & gloss (*tf·idf*) (decision tree)	77.6%
Best combination (no decision tree)	79.3%
Best combination (decision tree)	**82.5%**
Best combination (decision tree – Resnik's coefficient included)	**83.0%**

Table 3.1: Accuracy of disambiguation for different combinations of indicators.

styles of speaking or writing"; and later on: "turgid literally means swollen or distended". The question the judges had to ask themselves is whether this literal sense is included in the entry or not. In this particular case maybe the answer is negative. But it is not always clear whether the extended sense is mentioned by the lexicographer who designed the entry because the extended sense is very close and should be included in the meaning of the cluster, or whether it is mentioned so that the reader will be able to distinguish it. Some judges decided to include more often than exclude, while the other judges excluded the senses when they thought appropriate.

If we omit one of the judges who expressed singular opinions during the discussion, the agreement is higher: 86.8% ($\kappa = 0.73$).

In the standard solution, I decided to correct a few of the 56 cases of ties, to correct the apparent bias of some judges. I decided to include senses that are too specific or are instances, but to exclude verbs with wrong transitivity. I produced two solutions: a more inclusive one (when a sense is mentioned in the entry it was included) and a more exclusive solution (when a sense is mentioned, it was included only if the judges included it). The more inclusive solution was used in my experiments, but the results would change very little with the more exclusive one, because they differ only in 16 cases out of 904.

Method	All	Nouns	Verbs	Adjectives
All indicators except Resnik's coefficient	82.6%	81.8%	83.2%	78.3%
All indicators including Resnik's coefficient	83.0%	84.9%	84.8%	78.3%
Only Resnik's coefficient	71.9%	84.0%	77.7%	–

Table 3.2: Accuracy of disambiguation per part-of-speech.

3.4 Results and evaluation

Table 3.1 presents the results of using each indicator alone and in combinations with other indicators[3]. The accuracy of each method is computed by comparison to the standard solution (Section 3.3 explains how the standard solution was produced).

For the indicators using *tf·idf* and for the cosine between context vectors I use a decision tree to avoid manually choosing a threshold; therefore the figures in the table are the results of the cross-validation. By manually combining indicators, the best accuracy obtained was 79.3% for the attributes: intersection text and gloss (with a fixed threshold), words in synsets, and antonyms.

I found the best combination of indicators by training a decision tree as described in section 3.2. The best accuracy is 82.5%, computed by 10-fold cross-validation. The indicators that contribute the most to improving the accuracy are the ones in the upper-part of the decision tree (Figure 3.2): the intersection of the text with the gloss, the intersection of the text with the glosses of the related words, the words in the synset, and the words in the synsets of the related words. The ones in the lower part (CoreLex, and the cosine between context vectors) have little influence on the results. Their contribution is likely to be included in the contribution of the other indicators.

If the evaluation is done for each part-of-speech separately (see the first row in Table 3.2), it can be observed that the accuracy for nouns and verbs is higher than for adjectives. In our

[3]The standard error was 0.1% in most of the experiments involving cross-validation in decision trees. The standard error in this case shows how much the result of each test in the cross-validation experiment deviated from the reported mean.

data set of 50 randomly selected near-synonym clusters, there are 276 noun senses, 310 verb senses, and 318 adjective senses. There were no adverbs in the test set, because there are only a few adverbs in CTRW.

Another indicator that I implemented after the previous experiments were done is Resnik's coefficient, which measures how strongly a word sense correlates with the words in the same grouping (in the case when we have groups of similar nouns). The algorithm for computing this coefficient was originally proposed by Resnik [1999b] in a paper that presented a method for disambiguating noun groupings, using the intuition that when two polysemous words are similar, their most informative subsumer provides information about which sense of which word is the relevant one. The method exploits the WordNet noun hierarchy, and uses Resnik's similarity measure based on information content ([Resnik, 1999b], but see [Budanitsky and Hirst, 2001] for a critique of the similarity measure). I also implemented the same algorithm for verbs, using the WordNet verb hierarchy.

When Resnik's coefficient is added as a feature in the decision tree, the total accuracy (after cross-validation) increases slightly (but not statistically significant), to 83%. If Resnik's coefficient is included, the accuracy is improved for nouns and verbs (84.9% for nouns and 84.8% for verbs). The accuracy for adjectives is the same, because Resnik's coefficient is not defined for adjectives. If the only feature in the decision tree is Resnik's coefficient, the accuracy is high for nouns, as expected, and lower for verbs and for all parts of speech considered together.

In conclusion, the disambiguation method presented here does well for nouns and verbs, but it needs improvement for adjectives.

3.5 Comparison with related work

The Senseval competition [4] had the goal of evaluating WSD systems. I will refer here only to the experiments for the English language, in Senseval2, which used WordNet1.7 senses. I

[4] http://www.itri.brighton.ac.uk/events/senseval/

cannot compare the Senseval2 results with my results, because the words and texts are different, due to the nature of my task. Senseval2 had two tasks. One task was to disambiguate all the content words in the test set. The other task was to disambiguate only selected words (this is closer to my task). The precision and recall reported by the participating systems were all below 65% (40% for the best unspervised algorithm and 64% for the best supervised algorithm), while the Lesk-baseline algorithm was 23%.

My WSD program attains 82.5% accuracy, compared to a baseline of 53.5%. My task is relatively easier than the general WSD task because the text of the dictionary entry and the other near-synonyms in the same cluster provide a strong context for disambiguation. I report accuracy figures, but this is equivalent to reporting precision and recall in Senseval. This is because I disambiguate all the near-synonyms. The accuracy figures I reported are results of cross-validation experiments, while the Senseval evaluation used a special set test. The value for inter-annotator agreement (85%) is comparable to that of Senseval2 (85.5% for the English lexical sample task, according to the Senseval2 webpage).

Combining classifiers for WSD is not a new idea, but it is usually done manually, not on the basis of a small amount of annotated data. Stevenson and Wilks [2001], among others, combine classifiers (knowledge-sources) by using a weighted scheme.

An adapted Lesk-style algorithm for WSD that uses WordNet, but in a different manner, is presented by Pedersen and Banerjee [2002]. They intersected glosses of all words in the context of a target word. The intersection is done on pairs of words, considering the intersections between the gloss of the first word and the words related to the second word (by WordNet relations). They achieve an accuracy of 32%. Unlike Pedersen and Banerjee, I focus only on the target word (I do not use glosses of words in context), when I use the gloss of a near-synonym I include examples in the gloss.

Schütze [1998] uses context vectors to cluster together all the contexts in which a word is used in the same sense. In this way it is possible to distinguish among word senses without using a sense inventory from a lexical resource. I use the context vectors as a measure of the semantic relatedness between the text of an entry and the gloss of a synset. Earlier work on context vectors for disambiguating word sense in text is [Niwa and Nitta, 1994] and [Wilks, 1993].

3.6 Summary

This chapter discussed the problem of disambiguating the senses of the near-synonyms in a dictionary entry. The sense inventory used in this chapter was WordNet1.7. The word-sense disambiguation was based on a Lesk-style algorithm: intersecting the gloss of the sense with the text of the dictionary entry (which acts as a context for all the near-synonyms in the cluster). In addition to this, other indicators were considered, including other words (and their glosses) in WordNet that are in direct relation with the sense under consideration. The final decision whether a sense is relevant or not to a dictionary entry was made by a decision tree. The evaluations experiments (using cross-validation) on a sample of the data showed 82.5% accuracy.

The data annotated by human judges showed that usually several WordNet senses are relevant to a dictionary entry. If a near-synonym is part of more than one entry in CTRW, in fact different subsets of its senses are members of the respective entries. The information about near-synonym senses is needed when the LKB of NS is used in an NLP system, unless the system permits the use of near-synonyms without requiring their disambiguation.

Chapter 4

Adding Collocational Knowledge from Free Text

In this chapter the lexical knowledge-base of near-synonym differences is enriched with knowledge about the collocational behaviour of the near-synonyms. Collocational behaviour can help in the process of choosing between near-synonyms, because one must not choose a near-synonym that does not collocate well with the other word choices for the sentence. For example *daunting task* is a preferred collocation, while *daunting job* is less preferred (it should not be used in lexical choice unless there is no better alternative), and *daunting duty* is an anti-collocation[1] (it must not be used in lexical choice). For the purpose of this work, collocations consist of consecutive words that appear together much more often than by chance. I also include words separated by a few non-content words (short-distance co-occurrence in the same sentence).

The focus of this chapter is on differences between collocations, employing a differential t-test. In order to do this, several steps are necessary. First, I automatically acquire collocates of all near-synonyms in CTRW, from free text. Then, I acquire knowledge about less-preferred collocations and anti-collocations. Most of this chapter was previously published in [Inkpen

[1]This term was introduced by Pearce [2001].

and Hirst, 2002].

I am interested in how collocations can be used in lexical choice among open-class words. Therefore I need to extract collocations between open-class words (content words), which do not contain closed-class words (function words). For example, *defeat the enemy* is a collocation that contains the function word *the*, while *defeat enemy* is a collocation of interest. For now, only two-word collocations are considered.

In the experiment described in sections 4.1 and 4.2 (with results in section 4.3, and evaluation in section 4.4), knowledge about the collocational behaviour of the near-synonyms is acquired. In step 1 (section 4.1), potential collocations from the British National Corpus (BNC)[2] are acquired, combining several measures. In section 4.2, the following steps are presented: (step 2) collocations for the near-synonyms in CTRW are selected; (step 3) the selected collocations are filtered using mutual information on the Web; (step 4) for each cluster, new collocations are obtained by combining the collocate of one near-synonym with another near-synonym, and the differential t-test is used to classify them into preferred collocations, less-preferred collocations, and anti-collocations.

4.1 Extracting collocations from free text

Before being able to look at differences between collocations, I need to automatically acquire collocations of near-synonyms, from a corpus. I experimented with 100 million words from the *Wall Street Journal* (WSJ). Some of the near-synonyms appear very few times (10.64% appear fewer than 5 times) and 6.87% of them do not appear at all in the WSJ (due to its focus on business and news). Therefore I needed a more general corpus. I used the 100-million-word BNC. The BNC is a good choice of corpus for us because it has been tagged (automatically by the CLAWS tagger). Only 2.61% of the near-synonyms do not occur in the BNC; and only 2.63% occur between 1 and 5 times.

[2]http://www.hcu.ox.ac.uk/BNC/

Many of the near-synonyms appear in more than one cluster, with different parts-of-speech. I experimented on extracting collocations from raw text, but I decided to use a part-of-speech tagged corpus because I need to extract only collocations relevant for each cluster of near-synonyms.

I preprocessed the BNC by removing all words tagged as closed-class. To reduce computation time, words that are not useful for the purpose of this work are removed. For example, proper names (tagged NP0) are unlikely to help with lexical choice among open-class words.

There are many statistical methods that can be used to identify collocations. Four general methods are presented by Manning and Schütze [1999]. The first one is based on frequency of co-occurrence, does not consider the length of the corpus, and uses part-of-speech filtering to obtain useful collocations. The second method considers the mean and variance of the distance between two words, and can compute non-rigid collocations [Smadja, 1993] (collocations that can be interrupted by other words). The third method is hypothesis testing, which uses statistical tests to decide if the words occur together with probability higher than chance (it tests whether the null hypothesis that the two words occurred together by chance can be rejected). The fourth method is (pointwise) mutual information, an information-theoretical measure.

To acquire collocations from the BNC, I used the Ngram Statistics Package[3] [Pedersen and Banerjee, 2003]. NSP is a suite of programs to aid in analyzing *N*-grams in a corpus. I used it for bigrams only. The package computes bigram frequencies in a corpus and various statistics to measure the degree of association between two words: pointwise mutual information (MI), Dice, chi-square (χ^2), log-likelihood (LL), and Fisher's exact test.

I briefly describe the methods I used in my experiments, for the two-word case. I use these methods because they were readily available in NSP. The theoretical description of these methods, taken from Manning and Schütze [1999], is briefly presented below.

Each bigram *xy* can be viewed as having two features represented by the binary variables

[3] http://www.d.umn.edu/~tpederse/nsp.html
In fact I used an earlier version (0.4) of NSP, known as BSP (Bigram Statistics Package).

	y	$\neg y$	
x	$n_{11} = 66$	$n_{10} = 54$	$n_{1+} = 120$
$\neg x$	$n_{01} = 4628$	$n_{00} = 15808937$	$n_{0+} = 15813565$
	$n_{+1} = 4694$	$n_{+0} = 15808991$	$n_{++} = 15813685$

Table 4.1: Contingency table for *daunting task*: x = *daunting*, y = *task*).

X and Y. The joint frequency distribution of X and Y is described in a contingency table. Table 4.1 shows an example for the bigram *daunting task*. n_{11} is the number of times the bigram xy occurs; n_{10} is the number of times x occurs in bigrams at the left of words other than y; n_{01} is the number of times y occurs in bigrams after words other than x; and n_{00} is the number of bigrams containing neither x nor y. In this notation, the first index is for x and the second for y; 1 indicates presence and 0 indicates absence. In Table 4.1, the variable X denotes the presence or absence of *daunting* in the first position of a bigram, and Y denotes the presence or absence of *task* in the second position of a bigram. The marginal distributions of X and Y are the row and column totals obtained by summing the joint frequencies: $n_{+1} = n_{11} + n_{01}$, $n_{1+} = n_{11} + n_{10}$, and n_{++} is the total number of bigrams.

The NSP tool counts for each bigram in a corpus how many times it occurs, how many times the first word occurs at the left of any bigram (n_{1+}), and how many times the second word occurs at the right of any bigram (n_{+1}).

The five measures that I used from NSP are described below. I combine them in order to select potential collocations, as a necessary step towards the final goal of acquiring differences in the collocational behaviour of the near-synonyms.

Pointwise mutual information, $I(x;y)$, compares the probability of observing words x and y together (the joint probability) with the probabilities of observing x and y independently (the probability of occurring together by chance) [Church and Hanks, 1991].

$$I(x;y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

The probabilities can be approximated by: $P(x) = n_{1+}/n_{++}$, $P(y) = n_{+1}/n_{++}$, $P(x,y) =$

n_{11}/n_{++}. Therefore:

$$I(x;y) = \log_2 \frac{n_{++}n_{11}}{n_{+1}n_{1+}}$$

The motivation behind the mutual information measure comes from the theory of information: it roughly measures how much one word tells us about the other. It presents problems when used of low-frequency words. For example, in Figure 4.2 low frequency words that occur only in particular collocations are ranked first. The range of values the measure took in my experiments was from zero to 21.91. Adding a multiplicative factor, $P(x,y)$, improves the measure by ranking higher more frequent collocations.

The **Dice** coefficient is closely related to mutual information and therefore suffers from the same drawbacks. It is calculated as:

$$Dice(x,y) = \frac{2P(x,y)}{P(x)+P(y)} = \frac{2n_{11}}{n_{+1}+n_{1+}}$$

The range of values for this measure are in the interval $(0,1]$.

The next methods fall under hypothesis testing methods. **Pearson's Chi-square** and **Log-likelihood ratios** measure the divergence of observed (n_{ij}) and expected (m_{ij}) sample counts ($i = 1,0$, $j = 1,0$). The expected values are for the model that assumes independence (assumes that the null hypothesis is true). For each cell in the contingency table, the expected counts are: $m_{ij} = \frac{n_{i+}+n_{+j}}{n_{++}}$. The measures are calculated as [Pedersen, 1996]:

$$\chi^2 = \Sigma_{i,j} \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$$

$$LL = 2\ \Sigma_{i,j} \frac{\log_2 n_{ij}^2}{m_{ij}}$$

Log-likelihood ratios [Dunning, 1993] are more appropriate for sparse data than chi-square. Log-likelihood ratio seems to produce the best collocations (see Table 4.4), with scores ranging from zero to 123,548. The chi-square scores range from zero to 15,813,684. Log-likelihood ratio is easier to interpret than chi-square; the former is a number that tells us how much more likely one hypothesis is then the other.

Fisher's exact test is a significance test that is considered to be more appropriate for sparse and skewed samples of data than statistics such as the log-likelihood ratio or Pearson's Chi-Square test [Pedersen, 1996]. Fisher's exact test is computed by fixing the marginal totals of a contingency table and then determining the probability of each of the possible tables that could result in those marginal totals. Therefore it is computationally expensive. The formula is:

$$P = \frac{n_{1+}!n_{0+}!n_{+1}!n_{+0}!}{n_{++}!n_{11}!n_{10}!n_{01}!n_{00}!}$$

The scores for Fisher's exact test are in the interval $(0,1]$.

Because these five measures rank collocations in different ways (as the results in Tables 4.2–4.6 show), and have different advantages and drawbacks, I decided to combine them in choosing collocations. I choose as potential collocations for each near-synonym a collocation that is selected by at least two of the measures. For each measure I need to choose a value T, and consider as selected collocations only the T highest-ranked bigrams (where T can differ for each measure). By choosing lower values for T, I increase the precision (reduce the chance of accepting wrong collocations). By choosing higher values for T, I increase the recall. If I opt for lower recall, I may not get many collocations for some of the near-synonyms. Because there is no principled way of choosing these values, I prefer to include more collocations (the first 200,000 collocations selected by each measure, except Fisher's measure for which I take all 435,000 collocations of rank one) and to filter out later (in step 2) the bigrams that are not true collocations, using mutual information on the Web. Also, step 2 looks only at collocations for which one of words participating in the collocation is a near-synonym in CTRW.

The first 10 collocations selected by each measure are presented in tables 4.2–4.6. Note that some of the measures rank many collocations equally at rank 1: MI 358 collocations; LL one collocation; χ^2 828 collocations; Dice 828 collocations; and Fisher 435,000 collocations (when the measure is computed with a precision of 10 digits — higher precision is recommended, but the computation time becomes a problem). The columns in these tables are: the collocation, the rank assigned by the measure, the value of the measure, the frequency of the collocation in

Collocation (*xy*)	Rank	Score	Freq *xy*	Freq *x*+	Freq +*y*
source-level/A debugger/N	1	21.91	4	4	4
prosciutto/N crudo/N	1	21.91	4	4	4
rumpy/A pumpy/A	1	21.91	4	4	4
thrushes/N blackbirds/N	1	21.91	4	4	4
clickity/N clickity/N	1	21.91	4	4	4
bldsc/N microfilming/V	1	21.91	4	4	4
chi-square/A variate/N	1	21.91	4	4	4
long-period/A comets/N	1	21.91	4	4	4
tranquillizers/N sedatives/N	1	21.91	4	4	4
one-page/A synopsis/N	1	21.91	4	4	4

Table 4.2: Some of the collocations ranked 1 by MI.

Collocation (*xy*)	Rank	Score	Freq *xy*	Freq *x*+	Freq +*y*
clarinets/N bassoons/N	1	1.00	5	5	5
email/N footy/N	1	1.00	4	4	4
tweet/V tweet/V	1	1.00	5	5	5
garage/parking/N vehicular/A	1	1.00	4	4	4
growing/N coca/N	1	1.00	5	5	5
movers/N seconders/N	1	1.00	5	5	5
elliptic/A integrals/N	1	1.00	8	8	8
viscose/N rayon/N	1	1.00	15	15	15
cause-effect/A inversions/N	1	1.00	5	5	5
first-come/A first-served/A	1	1.00	6	6	6

Table 4.3: Some of the collocations ranked 1 by Dice.

the BNC, the frequency of the first word in the first position in bigrams, and the frequency of the second word in the second position in bigrams.

4.2 Differential collocations

For each cluster of near-synonyms, I now have the words (collocates) that occur in preferred collocations with near-synonyms. In this section the focus is on less-preferred collocations

Collocation (*xy*)	Rank	Score	Freq *xy*	Freq *x*+	Freq +*y*
prime/A minister/N	1	123548	9464	11223	18825
see/V p./N	2	83195	8693	78213	10640
read/V studio/N	3	67537	5020	14172	5895
ref/N no/N	4	62486	3630	3651	4806
video-taped/A report/N	5	52952	3765	3765	15886
secretary/N state/N	6	51277	5016	10187	25912
date/N award/N	7	48794	3627	8826	5614
hon./A friend/N	8	47821	4094	10345	10566
soviet/A union/N	9	44797	3894	8876	12538
report/N follows/V	10	44785	3776	16463	6056

Table 4.4: First 10 collocations selected by LL.

Collocation (*xy*)	Rank	Score	Freq *xy*	Freq *x*+	Freq +*y*
lymphokine/V activated/A	1	15813684	5	5	5
config/N sys/N	1	15813684	4	4	4
levator/N depressor/N	1	15813684	5	5	5
nobile/N officium/N	1	15813684	11	11	11
line-printer/N dot-matrix/A	1	15813684	4	4	4
dermatitis/N herpetiformis/N	1	15813684	9	9	9
self-induced/A vomiting/N	1	15813684	5	5	5
horoscopic/A astrology/N	1	15813684	5	5	5
mumbo/N jumbo/N	1	15813684	12	12	12
long-period/A comets/N	1	15813684	4	4	4

Table 4.5: Some of the collocations ranked 1 by χ^2.

Collocation (xy)	Rank	Score	Freq xy	Freq x+	Freq +y
roman/A artefacts/N	1	1.00	4	3148	108
qualitative/A identity/N	1	1.00	16	336	1932
literacy/N education/N	1	1.00	9	252	20350
disability/N pension/N	1	1.00	6	470	2555
units/N transfused/V	1	1.00	5	2452	12
extension/N exceed/V	1	1.00	9	1177	212
smashed/V smithereens/N	1	1.00	5	194	9
climbing/N frames/N	1	1.00	5	171	275
inclination/N go/V	1	1.00	10	53	51663
trading/N connections/N	1	1.00	6	2162	736

Table 4.6: Some of the collocations ranked 1 by Fisher.

and anti-collocations in order to acquire knowledge that will discriminate between the near-synonyms in the cluster. I need to check how the collocates combine with the other near-synonyms in the same cluster. For example, if *daunting task* is a preferred collocation, I check whether *daunting* collocates well or not with the other near-synonyms of *task*.

I use the Web as a corpus for differential collocations. I don't use the BNC corpus to rank less-preferred and anti-collocations, because their absence in the BNC may be due to chance. I can assume that the Web (the portion indexed by search engines) is big enough that a negative result can be trusted.

I use an interface to the AltaVista search engine to count how often a collocation is found. (See Table 4.7 for an example search done on 13 March 2002.) A low number of co-occurrences indicates a less-preferred collocation. But I also need to consider how frequent the two words in the collocation are. I use the differential t-test to find collocations that best distinguish between two near-synonyms [Church et al., 1991], but I use the Web as a corpus. The collocations were acquired from the BNC with the right part-of-speech for the near-synonym, but on the Web there are no part-of-speech tags; therefore a few wrong instances may be included in the counts. I approximate the number of occurrences of a word on the Web with the number of documents containing the word.

The t**-test** can also be used in the hypothesis-testing method to rank collocations. It looks at the mean and variance of a sample of measurements, where the null hypothesis is that the sample was drawn from a normal distribution with mean μ (the distribution we work with can be approximated by a normal one if the amount of data is large enough). It measures the difference between observed ($\bar{x}$) and expected means, scaled by the variance of the data (s^2), which in turn is scaled by the sample size (N).

$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}}$$

The **differential** t**-test** can be used for hypothesis testing of differences. It compares the means of two normal populations:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{N} + \frac{s_2^2}{N}}}$$

Here the null hypothesis is that the average difference is $\mu = 0$.Therefore $\bar{x} - \mu = \bar{x} = \bar{x}_1 - \bar{x}_2$. In the denominator, the variances of the two populations is added.

If the collocations of interest are xw and yw (or similarly wx and wy), then we have the approximations $\bar{x}_1 = s_1^2 = P(x,w)$ and $\bar{x}_2 = s_2^2 = P(y,w)$; therefore:

$$t = \frac{P(x,w) - P(y,w)}{\sqrt{\frac{P(x,w)+P(y,w)}{n_{++}}}} = \frac{n_{xw} - n_{yw}}{\sqrt{n_{xw} + n_{yw}}}$$

If w is a word that collocates with one of the near-synonyms in a cluster, and x is one of the near-synonyms, the mutual information relative to w can be approximated:

$$\frac{P(w,x)}{P(x)} = \frac{n_{wx}}{n_x}$$

where $P(w)$ was dropped because it is the same for various x (the computation cannot be done if I keep it, because the total number of bigrams on the Web is not known).

I use this measure to eliminate collocations inappropriately selected in step 1. I eliminate those with mutual information lower than a threshold. I describe the way I chose this threshold

(T_{mi}) in section 4.4.

Collocations of a near-synonym with a wrong part-of-speech are not considered (the collocations are tagged). But there is also the case when a near-synonym has more than one major sense. In this case, collocations for senses other than the one required in the cluster could be retrieved. For example, for the cluster *job, task, duty*, etc., the collocation *import/N duty/N* is likely to be for a different sense of *duty* (the customs sense). Therefore the sense of the near-synonym in a collocation needs to be disambiguated (assuming one sense per collocation). I experimented with a simple Lesk-style method [Lesk, 1986]. For each collocation, instances from the corpus are retrieved, and the content words surrounding the collocations are collected. This set of words is then intersected with the entry for the near-synonym in CTRW. If the intersection is not empty, it is likely that the collocation and the entry use the near-synonym in the same sense. If the intersection is empty, the collocation is not retained.

In step 3, the collocations of each near-synonym with a given collocate are grouped in three classes, depending on the t-test values of pairwise collocations. The t-test between each collocation and the collocation with maximum frequency is computed, and so is the t-test between each collocation and the collocation with minimum frequency. Table 4.7 presents an example. The second column shows the number of hits for the collocation *daunting* x, where x is one of the near-synonyms in the first column. The third column shows the mutual information, the fourth column, the differential t-test between the collocation with maximum frequency (*daunting task*) and *daunting* x, and the last column, the t-test between *daunting* x and the collocation with minimum frequency (*daunting hitch*). After the t-test scores are computed, a set of thresholds is determined to classify the collocations in the three groups: preferred collocations, less preferred collocations, and anti-collocations. The procedure used in this step is detailed in section 4.4.

x	**Hits**	**MI**	***t* max**	***t* min**
task	63573	0.011662	-	252.07
job	485	0.000022	249.19	22.02
assignment	297	0.000120	250.30	17.23
chore	96	0.151899	251.50	9.80
duty	23	0.000022	251.93	4.80
stint	0	0	252.07	-
hitch	0	0	252.07	-

Table 4.7: Example of counts, mutual information scores, and t-test scores for the collocate *daunting* with near-synonyms of *task*.

4.3 Results

I obtained 15,813,685 bigrams. Of these, 1,350,398 were distinct and occurred at least 4 times. As mentioned, some of the top-ranked collocations for each measure are presented in Tables 4.2–4.6. I present the rank given by each measure (1 is the highest), the value of the measure, the frequency of the collocation, and the frequencies of the words in the collocation.

I selected collocations for all 909 clusters in CTRW (5419 near-synonyms in total). An example of collocations extracted for the near-synonym *task* is presented in Table 4.8, where the columns are, in order, the name of the measure, the rank given by the measure, and the value of the measure.

I filtered out the collocations using MI on the Web (step 2), and then I applied the differential t-test (step 3). Table 4.9 presents an example of results for differential collocations, where $\surd$ marks preferred collocations, ? marks less-preferred collocations, and $*$ marks anti-collocations.

Before proceeding with step 3, the collocations in which the near-synonym is used in a different sense could be filtered out, using the Lesk method explained above. For example, *suspended/V duty/N* is kept while *customs/N duty/N* and *import/N duty/N* are rejected. The disambiguation algorithm was run only for a subset of CTRW, because hand-annotated data is needed to evaluate how well it works and because it it very time-consuming (due to the need to

Collocation	Measure	Rank	Score
daunting/A task/N	MI	24887	10.85
	LL	5998	907.96
	χ^2	16341	122196.82
	Dice	2766	0.02
repetitive/A task/N	MI	64110	6.77
	χ^2	330563	430.40

Table 4.8: Example of collocations extracted for the near-synonym *task*. The first collocation was selected (ranked in the set of first T collocations) by four measures; the second collocation was selected by two measures.

	Collocates		
Near-synonyms	*daunting*	*particular*	*tough*
task	$\surd$	$\surd$	$\surd$
job	?	$\surd$	$\surd$
assignment	$*$	$\surd$	$\surd$
chore	$*$	?	$*$
duty	$*$	$\surd$	$*$
stint	$*$	$*$	$*$
hitch	$*$	$*$	$*$

Table 4.9: Example of results for collocations of near-synonyms ($\surd$ marks preferred collocations, ? marks less-preferred collocations, and $*$ marks anti-collocations).

retrieve corpus instances for each collocation). It is easier to just skip the disambiguation step, because the wrong senses in the final lexical knowledge-base of near-synonym collocations will not hurt. For example, if the collocation *customs/N duty/N* is associated with the cluster *job, duty, etc.*, it does not matter because an interlingual input containing the collocation will replace the near-synonym *duty* with a different meta-concept than the one for this cluster (if *duty* is also a member in a different cluster).

The differential collocation algorithms were run on the whole CTRW to produce a lexical knowledge-base of near-synonym collocational behaviour. The next section describes evaluation experiments.

4.4 Evaluation

The evaluation has two purposes: to get a quantitative measure of the quality of the results, and to choose thresholds in a principled way.

As described in the previous sections, in step 1, I selected potential collocations from the BNC (the ones selected by at least two of the five measures). Then, I selected collocations for each of the near-synonyms in CTRW (step 2). I need to evaluate the MI filter (step 3), which filters out the bigrams that are not true collocations, on the basis of their mutual information computed on the Web. I also need to evaluate step 4, the three-way classification based on the differential t-test on the Web.

For evaluation purposes I selected three clusters from CTRW, with a total of 24 near-synonyms. For these, I obtained 916 collocations from the BNC according to the method described in section 4.1.

Two human judges (computational linguistics students, native speakers of English) reviewed these collocations. They were instructed to mark which of them are true collocations and which are not. I presented the collocations to the judges in random order, and each collocation was presented twice. The first judge was consistent (judged a collocation in the same way both times it appeared) in 90.4% of the cases. The second judge was consistent in 88% of the cases. The agreement between the two judges was 67.5% (computed in a strict way, that is I considered agreement only when the two judges had the same opinion including the cases when they were not consistent). The value of the kappa statistic coefficient is $\kappa = 0.35$. The consistency and agreement figures show how difficult the task is for humans.

I used the data annotated by the two judges to build a standard solution, so I can evaluate the results of the MI filter. In the standard solution a bigram was considered a true collocation if both judges considered it so. I used the standard solution to evaluate the results of the filtering, for various values of the threshold T_{mi}. That is, if a bigram had the value of MI on the Web lower than a threshold T_{mi}, it was filtered out. I choose the value of T_{mi} so that the accuracy of the filtering program is the highest. By accuracy I mean the number of true collocations (as given by the standard solution) identified by the program over the total number of bigrams I used in the evaluation. The best accuracy was 70.7% for T_{mi} = 0.0017. I used this value of the threshold when running the programs for all CTRW. In order to evaluate the MI filter, I also run 10-fold cross-validation experiments, using a decision tree to choose a series of thresholds. In this case the accuracy on the test set was 68.3%.

As a result of this first part of the evaluation, I can say that after filtering collocations depending on their MI on the Web, approximately 68.3% of the remaining bigrams are true collocations. This value is not absolute, because I used a sample of the data for three clusters of near-synonyms for the evaluation. The 68.3% accuracy is better than the baseline (approximately 50% for random choice). Table 4.10 summarizes the evaluation results.

Next, I proceeded with evaluating the differential t-test three-way classifier. For each cluster, for each collocation, new collocations were formed from the collocate and all the near-synonyms in the cluster. In order to learn the classifier, and to evaluate its results, the two judges manually classified a sample of data into preferred collocations, less-preferred collocations, and anti-collocations. The data presented to the judges were 2838 potential collocations obtained for the same three clusters of near-synonyms, from 401 collocations (out of the initial 916) that remained after filtering. The judges were instructed to mark as preferred collocations all the potential collocations that they consider good idiomatic use of language, as anti-collocations the ones that they won't normally use, and as less-preferred collocations the ones that they are not comfortable classifying in either of the other two classes. I built a standard solution for this task, based on the classifications of both judges. When the judges agreed, the class was clear. When they did not agree, I designed simple rules, such as: when one judge chose the class preferred collocation, and the other judge chose the class anti-collocation, the class in the solution was less-preferred collocation; when one judge chose the class preferred collocation, and the other judge chose the class less-preferred collocation, the class in the solution was preferred collocation; when one judge chose the class anti-collocation, and the other judge chose the class less-preferred collocation, the class in the solution was anti-collocation.

Step	Baseline	My method
Filter (MI on the Web)	50%	**68.3%**
Dif. t-test classifier	71.4%	**84.1%**

Table 4.10: Accuracy of the main steps.

The agreement between judges was 80% ($\kappa = 0.54$). I used this standard solution as training data to learn a decision tree[4] for the three-way classifier. The features in the decision tree are the t-test between each collocation and the collocation from the same group that has maximum frequency on the Web, and the t-test between the current collocation and the collocation that has minimum frequency (as presented in Table 4.7). I could have set aside a part of the training data as a test set. Instead, I did 10-fold cross-validation to estimate the accuracy on unseen data. The average accuracy was 84.1%, with a standard error of 0.5%. The accuracy is higher than the baseline of 71.4% that always chooses the most frequent class, anti-collocations. I also experimented with including MI as a feature in the decision tree, and with manually choosing thresholds (without a decision tree) for the three-way classification, but the accuracy was lower than 84.1%. The three-way classifier can fix some of the mistakes of the MI filter. If a wrong collocation remains after the MI filter, the classifier can classify it in the anti-collocations class. I conclude that the acquired collocational knowledge has acceptable quality.

4.5 Comparison with related work

There has been a lot of work done in extracting collocations for different applications.

Like Church et al. [1991], I use the t-test and mutual information, but unlike them I use the Web as a corpus for this task (and a modified form of mutual information), and I distinguish three types of collocations (preferred, less-preferred, and anti-collocations).

I extract collocations for use in lexical choice. There is a lot of work on using collocations

[4] I used C4.5, http://www.cse.unsw.edu.au/~quinlan

in NLG (but not in the lexical choice sub-component). There are two typical approaches: the use of phrasal templates in the form of canned phrases, and the use of automatically extracted collocations for unification-based generation [McKeown and Radev, 2000].

Statistical NLG systems (such as Nitrogen [Langkilde and Knight, 1998]) make good use of the most frequent words and their collocations. But such a system cannot choose a less-frequent synonym that may be more appropriate for conveying desired nuances of meaning, if the synonym is not a frequent word.

Finally, there is work related to ours from the point of view of the synonymy relation.

Turney [2001] used mutual information to detect the best answer to questions about synonyms from Test of English as a Foreign Language (TOEFL) and English as a Second Language (ESL). Given a problem word (with or without context), and four alternative words, the question is to choose the alternative most similar in meaning to the problem word (the problem here is to detect similarities, while in my work differences are detected). His work is based on the assumption that two synonyms are likely to occur in the same document (on the Web). This can be true if the author needs to avoid repeating the same word, but not true when the synonym is of secondary importance in a text. The alternative that has the highest PMI-IR (pointwise mutual information for information retrieval) with the problem word is selected as the answer. I used the same measure in section 4.2 — the mutual information between a collocation and a collocate that has the potential to discriminate between near-synonyms. Both works use the Web as a corpus, and a search engine to estimate the mutual information scores.

Pearce [2001] improves the quality of retrieved collocations by using synonyms from WordNet [Pearce, 2001]. A pair of words is considered a collocation if one of the words significantly prefers only one (or several) of the synonyms of the other word. For example, *emotional baggage* is a good collocation because *baggage* and *luggage* are in the same synset and $*$*emotional luggage* is not a collocation. As in my work, three types of collocations are distinguished: words that collocate well; words that tend to not occur together, but if they do the reading is acceptable; and words that must not be used together because the reading will be unnatural (anti-collocations). In a manner similar to Pearce [2001], in section 4.2, I don't record collocations in the lexical knowledge-base if they don't help discriminate between near-synonyms. A difference is that I use more than frequency counts to classify collocations (I use a combination of t-test and MI).

My evaluation method was partly inspired by Evert and Krenn [2001]. They collect collocations of the form noun-adjective and verb-prepositional phrase. They build a solution using two human judges, and use the solution to decide what the best threshold is for taking the N highest-ranked pairs as true collocations. They compare the performance of various measures, and conclude that LL is the best in most (but not all) of their experimental settings. In my work, I combine the various measures. I do not explicitly compare them, but I could say, by inspecting the top-ranked collocations, that LL behaves best. My evaluation has a different focus: I evaluate the MI filter on the Web, and the differential t-test for judging pairs of collocations.

4.6 Summary

This chapter described the acquisition of a lexical knowledge-base of near-synonym collocational behaviour. This knowledge can be added to the LKB of NS acquired in Chapter 2, but it contains a different type of knowledge for the same clusters of near-synonyms.

Collocational behaviour is important in an NLG system: the lexical choice process should choose near-synonyms that collocate well with the other words to be used in the sentence, and it should not choose near-synonyms that would create anti-collocations in the generated sentence.

The acquired knowledge is about differences between collocations of the near-synonyms from the same cluster. The acquisition method first collected collocations from the BNC, keeping only the ones for the near-synonyms of interest. In the next step they were filtered by using mutual information on the Web. The results of the filter were good collocations for some near-synonyms. Then, each near-synonym was replaced with each of the near-synonyms in the same cluster, to see how the near-synonyms behave together with the collocate word. The potential collocations generated in this way were classified into: preferred collocations, less-preferred collocations, and anti-collocations, by a decision tree that uses as its main attribute the differential t-test on the Web. A standard solution was built from data annotated by two human judges. For the MI filter, the solution was used to choose the MI threshold. For the 3-way classification task using the differential t-test on the Web, the solution was used to choose thresholds between the 3 classes. The evaluation experiments (cross-validation) estimate that the acquired collocational knowledge has acceptable quality.

Chapter 5

Adding Knowledge from Machine-Readable Dictionaries

Other types of dictionaries besides those explicitly on near-synonyms may contain information about near-synonym differences. I explored several machine-readable dictionaries (MRDs) and special-purpose dictionaries. The lexical knowledge-base of near-synonym differences built in Chapter 2 is extended by adding information automatically extracted from such dictionaries.

This chapter presents the acquisition of additional knowledge from several sources, the procedure for merging all the knowledge in a final LKB of NS, and the procedure for ensuring its consistency.[1]

5.1 Adding knowledge from the General Inquirer

The *General Inquirer*[2] [Stone et al., 1966] is a computational lexicon compiled from several sources, including the Harvard IV-4 dictionary and the Lasswell value dictionary. It contains markers that classify each word according to an extendable number of categories. There are markers for words of pleasure, pain, virtue, and vice; markers for words indicating overstate-

[1] I am grateful to our research assistant Olga Feiguina for helping with most of the programs in this chapter.

[2] http://www.wjh.harvard.edu/~inquirer/

CORRECT#1	H4Lvd Positiv Pstv Virtue Ovrst POSAFF Modif 21% adj: Accurate, proper
CORRECT#2	H4Lvd Positiv Pstv Strng Work IAV TRNGAIN SUPV 54% verb: To make right, improve; to point out error (0)
CORRECT#3	H4Lvd Positiv Pstv Virtue Ovrst POSAFF Modif 25% adv: "Correctly" – properly, accurately
CORRECT#4	H4Lvd Virtue TRNGAIN Modif 0% adj: "Corrected" – made right

Table 5.1: Example of entries in the *General Inquirer* for the word *correct*.

ment and understatement; markers for places and locations; etc. The definitions of each word are very brief. Examples of entries in this dictionary are presented in Table 5.1.

The category of interest to my work is Positiv/Negativ. The abbreviations Pstv/Ngtv are earlier versions of Positiv/Negativ. There are 1,915 words marked as Positiv (not including words for *yes*, which has been made a separate category of 20 entries). There are 2,291 words marked as Negativ (not including the separate category *no* in the sense of refusal). For each near-synonym in CTRW, if it is marked as Positiv or Negativ, an attitudinal distinction is asserted. A positive word corresponds to a favourable attitude; a negative one corresponds to a pejorative attitude. Before the extraction from the *General Inquirer*, disambiguation needs to be done. First, only the desired parts-of-speech are chosen. Second, if there is more than one entry (several senses) for the same word, the knowledge is asserted only if the majority of its senses have the same marker (either Positiv or Negativ). It is not possible to actually disambiguate which senses are the relevant ones for each near-synonym because the definitions of the sense in the *General Inquirer* are very short, and a Lesk-style disambiguation algorithm will not succeed. Therefore all the senses with the right part-of-speech are taken into account. An example of distinctions extracted in this step is presented in Figure 5.1. The number of attitudinal distinctions acquired from the *General Inquirer* was 5358.

```
Cluster: accurate, correct, exact, precise, right, true

subj: accurate
freq: usually
strength: medium
class: Favourable

subj: correct
freq: usually
strength: medium
class: Favourable

subj: exact
freq: usually
strength: medium
class: Favourable
....
```

Figure 5.1: Example of distinctions extracted from the *General Inquirer*.

5.2 Adding knowledge from the *Macquarie Dictionary*

The most common machine-readable dictionaries contain definitions for each word in isolation, unlike the CTRW, which explains groups of near-synonyms. Nonetheless, MRDs may contain useful information about near-synonyms.

I experimented with the *Macquarie Dictionary*[3] [Delbridge et al., 1987], because this dictionary contains all the near-synonyms of interest and it was available for my work. Other machine-readable dictionaries could have been used. There is a large body of work that exploits the content of LDOCE (Longman Dictionary of Contemporary English)[4]. It proved useful in word sense disambiguation and other tasks [Wilks et al., 1996]. It has a controlled vocabulary for the definitions, and contains information such as Activator (about 1000 categories) and subject codes (approximately 200 subject codes). LDOCE was not available for my research. For the task of extracting information from definitions, LDOCE would have been

[3] http://www.macquariedictionary.com.au/

[4] http://www.longman.com/dictionaries/research/resnlapp.html

useful only if at least one near-synonyms from each cluster is part of the controlled language.

From the SGML-marked text of the *Macquarie Dictionary*, the definitions of the near-synonyms in CTRW are extracted. A fragment of an SGML-marked entry for the word *burlesque* is presented in Figure 5.2. The definitions for the expected part-of-speech are extracted. Only the definitions that contain another near-synonym from the same cluster are retained, because these are definitions of the desired senses and express a distinction relative to another near-synonym. For example, for the cluster: *caricature, burlesque, mimicry, parody, takeoff, travesty*, one definition extracted for the near-synonym *burlesque* is: *any ludicrous take-off or debasing caricature*. Another relevant definition, for the near-synonym *parody*, is: *a burlesque imitation of a musical composition*. The result of the extraction is presented in Figure 5.3. The other near-synonym will be later deleted from the peripheral string, during the customization phase (Section 6.2.2). The last distinction in the figure contains only another near-synonym, not new information; therefore it will not be useful for further processing.

The number of new denotational distinctions acquired by this method was 5731. They have the same format as the denotational distinctions acquired from CTRW in the initial LKB of NS (which contained 10,789 denotational distinctions).

5.3 Adding knowledge from WordNet

In WordNet some word senses are marked as informal, in a parenthesis at the beginning of their glosses. For example, the gloss for `wee#a#1` (the first sense of the adjective *wee*) is `(used informally) very small; "a wee tot"`. I extracted this information for the near-synonyms of interest. Word sense disambiguation is necessary to keep only the right senses of the near-synonyms, therefore I used the WSD module described in Chapter 3.

The knowledge about informal near-synonyms acquired in this way is useful. Unfortunately, it turned out that this kind of information is not very frequently expressed in WordNet. Only 11 new distinctions for near-synonyms were acquired. Future work will investigate other

```
<RECORD id=000010095>
 <HEAD>burlesque</HEAD>
 <SORTKEY>BURLESQUE0990010000</SORTKEY> <FLAGS>BIGM N</FLAGS>
 <PRON><PRN>b3'l8sk</PRN><PRN TYPE=SAY>ber'lesk</PRN></PRON>
 <BODY>
   <CHUNK><POS>noun</POS>
    <DEF id=899> <DTEXT>an artistic composition, especially literary or
     dramatic, which, for the sake of laughter, vulgarises lofty material
     or treats ordinary material with mock dignity.</DTEXT>
     <THES>326.02.10</THES><THES>228.03.40</THES></DEF>
    <DEF id=955><DTEXT>any ludicrous take-off or debasing caricature.</DTEXT>
     <THES>326.02.10</THES></DEF>
    <DEF id=890><DTEXT>a theatrical or cabaret entertainment featuring
     coarse, crude, often vulgar comedy and dancing.</DTEXT>
     <THES>326.02.10</THES></DEF>
   </CHUNK>
   <CHUNK><POS>adjective</POS>
    <DEF id=535><DTEXT>involving ludicrous or debasing treatment of a
     serious subject.</DTEXT>
     <THES>326.04.20</THES><THES>228.14.20</THES></DEF>
    <DEF id=014><DTEXT>of or relating to risqu&eacute; burlesque.</DTEXT>
     <THES>326.04.20</THES></DEF>
   </CHUNK>
  ...
 </BODY>
 <ETY><LANG>French</LANG>, from <LANG>Italian</LANG> <I>burlesco</I>,
  from <I>burla</I> jest, mockery</ETY>
 <RUNON><RON>burlesquer</RON><POS>noun</POS></RUNON>
 <STERM TYPE=IINF LEMMA=HWD POS=V TENSE=PAST>burlesqued</STERM>
 <STERM TYPE=IINF LEMMA=HWD POS=V TENSE=PPR>burlesquing</STERM>
 <STERM TYPE=IINF LEMMA=HWD POS=V TENSE=PS>burlesques</STERM>
 <STERM TYPE=RINF LEMMA=HWD NUMBER=PL POS=N>burlesques</STERM>
 <STERM TYPE=RINF LEMMA=RUNON NUMBER=PL POS=N>burlesquers</STERM>
</RECORD>
```

Figure 5.2: Example of entry in the *Macquarie Dictionary* for the word *burlesque*.

```
Cluster: caricature, burlesque, mimicry, parody, takeoff, travesty

subj: burlesque
freq: usually
strength: medium
class: Denotation
periph: any/DT ludicrous/JJ take-off/NN or/CC debasing/VBG caricature/NN

subj: parody
freq: usually
strength: medium
class: Denotation
periph: a/DT burlesque/JJ imitation/NN of/IN a/DT musical/JJ composition/NN

subj: parody
freq: usually
strength: medium
class: Denotation
periph: a/DT travesty/NN
```

Figure 5.3: Example of distinctions extracted from the *Macquarie Dictionary*.

MRDs that might specify which words are formal / informal.

5.4 Consistency checking

After extracting knowledge from multiple sources, the next step is to merge all the knowledge together. The merging program merges knowledge acquired from two sources at a time. For each near-synonym cluster in the first LKB, it copies in the resulting LKB the name of the cluster and the distinctions, and if there are distinctions for the same cluster in the second LKB, it copies them too. I applied the merging program first on the LKB of NS acquired from CTRW and the LKB acquired from one of the other sources, let's say the *Macquarie Dictionary*. Then, I merged the resulting LKB with the LKB extracted form the *General Inquirer*. Then the small LKB extracted from WordNet (about informal near-synonyms) is added.

During the merging process, each piece of knowledge (each distinction) is preserved to-

```
Cluster: animal, beast, brute, creature
...
In this use, near_syn(beast) is more pejorative than near_syn(brute), but
both are more negative than near_syn(animal) when it is used in this way.

subj: beast
freq: usually
strength: medium
class: Pejorative

subj: beast
freq: usually
strength: medium
class: Pejorative
...

From GI

subj: beast
freq: usually
strength: medium
class: Favourable
...
```

Figure 5.4: Example of conflict in the merged lexical knowledge-base.

gether with a string indicating its source. For the distinctions extracted from CTRW, the sentence they were extracted from is provided. For the other sources, a string indicating the name of the source is added. Figure 5.4 contains an example of a merged LKB with indicators of the sources. The first two distinctions originate from the CTRW sentence; the source of the third distinction is the *General Inquirer* (abbreviated GI).

Figure 5.4 also shows that conflicting information can arise. This can be due to different sources having a different opinion about a word, or there could be cases in which there is a conflict coming from one source only (there are a couple of such cases in CTRW). Sometimes the information is not contradictory, but duplicated. For the example the first two distinctions in Figure 5.4 are expressing the same distinction, extracted from the two parts of the CTRW sentence.

The consistency-checking program detects conflicts and resolves them. The algorithm for resolving conflicts is a kind of voting scheme. It is rather ad-hoc, based on the intuition that neutral votes should have less weight than votes for the two extremes. If the conflict is over a stylistic distinction (`Formality`, `Force`, or `Concreteness`) than the values for the strength features (`low`, `medium`, or `high`) are collected for the two or more conflicting distinctions. Let N_{low}, N_{medium}, and N_{high} be the number of times that each strength feature occurs. If one of the three numbers is higher than the other two, that feature is the clear winner. If there are ties, then the algorithm is as follows:

if $N_{low} = N_{high}$ then winner = `medium`,

if $N_{low} = N_{medium}$ and $N_{low} > N_{high}$ then winner = `low` else winner = `high`,

if $N_{high} = N_{medium}$ and $N_{high} > N_{low}$ then winner = `high` else winner = `low`.

Conflicts for attitudinal distinctions arise when the same near-synonym has distinctions for more than one of the classes `Pejorative`, `Neutral`, and `Favourable` (as in Figure 5.4). In this case, the counts are stored in N_{pejo}, $N_{neutral}$, and N_{favo}. The conflict resolution strategy is similar: when one of the numbers is higher than the other two, the corresponding class wins. If there are ties, then the algorithm is as follows:

if $N_{pejo} = N_{favo}$ then winner = `Neutral`,

if $N_{pejo} = N_{neutral}$ and $N_{pejo} > N_{favo}$ then winner = `Pejorative` else winner = `Favourable`,

if $N_{favo} = N_{neutral}$ and $N_{favo} > N_{pejo}$ then winner = `Favourable` else winner = `Pejorative`.

The strength feature for the winning class is decided by applying the algorithm presented above for stylistic distinctions, for all the distinctions that have the winning attitude. For example, in Figure 5.4 the winning class is `Pejorative` and there are two pejorative distinctions with strength `medium`; therefore the winning strength is `medium`.

For denotational distinctions there is no conflict detection. This is left for future research. Semantic similarity measures could help to detect when two concepts are opposites, but more processing is needed (such as detecting negation, antonyms, etc.). If the same information is asserted multiple times, duplicates can be eliminated. But if very similar information is asserted multiple times, it is not clear what value for the similarity should be considered high enough to decide that the distinction was already asserted.

If the same attitudinal or stylistic distinction is asserted more than once, duplicates are eliminated. This is done implicitly by the conflict resolution algorithm: when the same information is repeated, two of the three numbers computed by the algorithm are zero; therefore, the repeated distinction is the winner and it is be asserted to replace the repetitions.

The number of conflicts detected by the consistency-checking program was 302 for the merged LKB of 23,469 distinctions. After conflict resolution, 22,932 distinctions remained.

The consistency-checking program produces a new LKB of NS without conflicts or duplicate knowledge. It also provides a list of the conflicts together with the proposed solution. This list can be easily inspected by a human who can change the solution of the conflict in the final LKB of NS, if desired.

5.5 Summary

This chapter acquired knowledge about distinctions between near-synonyms, from machine-readable dictionaries. Attitudinal distinctions were extracted from the *General Inquirer*. Denotational distinctions were acquired from the *Macquarie Dictionary*. In order to select only the right senses of the near-synonyms and to make sure a distinction is expressed, only the definitions that contain another near-synonym from the same cluster were retained. A few stylistic distinctions (about informal words) were extracted from WordNet.

This knowledge was easily added to the initial LKB of NS acquired in Chapter 2, because it has exactly the same format. Inconsistencies that appear in the merged LKB were resolved (only for attitudinal and stylistic distinctions).

Future work includes applying the extraction programs presented in section 2.3, without modification, to the usage notes from an MRD such as *Merriam-Webster Online Dictionary*[5].

[5] http://www.m-w.com/cgi-bin/dictionary

The distinctions expressed in these usage notes are similar to the explanations from CTRW.

Chapter 6

Using the Lexical Knowledge-Base of Near-Synonym Differences

The lexical knowledge-base of near-synonym differences, acquired initially from CTRW, enriched from machine-readable dictionaries, and extended with knowledge of collocational behaviour of the near-synonyms, can be used in several NLP applications.

One possible application is an intelligent thesaurus that offers a writer a set of alternative words and can also explain what are the differences between them. Automatic choice of the best alternative can be envisioned.

Another possible application, the one I will focus on, is lexical choice in natural language generation and machine translation. The LKB of NS is generic. Depending on the NLP system that uses it, customization of some aspects of the LKB of NS may be needed.

6.1 Lexical choice

The clustered model of lexical knowledge is applicable to both the lexical analysis and lexical choice phases of a machine translation system. Figure 6.1 shows that during the analysis phase, the lexical knowledge-base of near-synonym differences (LKB of NS) in the source

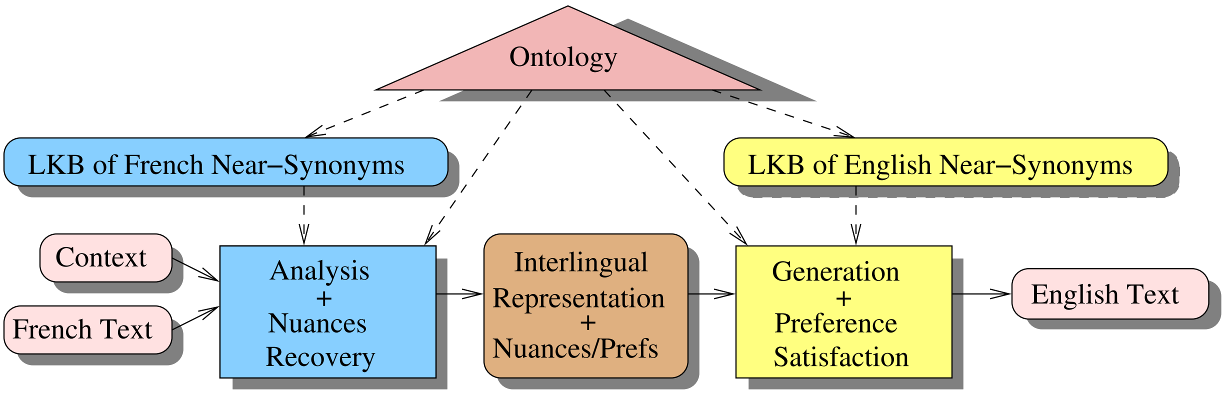

Figure 6.1: Lexical analysis and choice in MT. Adapted from [Edmonds and Hirst, 2002]. The solid lines show the flow of data: input, intermediate representations, and output; the dashed lines show the flow of knowledge from the knowledge sources to the analysis and the generation module. The rectangles denote the main processing modules; the rest of the boxes denote data or knowledge sources.

language is accessed, together with the context, to determine a set of nuances expressed by the source language text. During the generation phase, these nuances become *preferences* for the lexical choice process. The target language text has to express the same meaning as the source language text (necessary condition). In addition, the choice of words for the target language text should try to satisfy the *preferences* as much as possible. In this section, I focus on the generation phase of an interlingual machine translation (MT) system, specifically the lexical choice process.

Two examples of systems that use knowledge of near-synonym differences are I-Saurus and Xenon.

I-Saurus is a prototype NLG system that uses a hand-crafted ontology, just enough to allow the integration of a small, manually developed LKB of NS (for 9 clusters of NS). I-Saurus was implemented by Edmonds [1999]. It is based on MOOSE [Stede, 1998] (a system designed for paraphrasing verb alternations) and it uses the surface realizer named Penman [Penman Natural Language Group, 1989] for actually generating text. The output varies with the input preferences. An example of input and output to I-Saurus is presented in Figure 6.2.

I-Saurus implements a two-tiered lexical choice process. First, it chooses a set of clus-

```
Input:
(order1 (SAYER john1)
        (SAYEE alcoholic1)
        (SAYING (perform1 (ACTOR alcoholic1)
                          (ACTEE (tell1 (SAYER alchoholic1)
                                        (SAYING lie1))))))
Input preferences:
   (high formality)
   (favour alcoholic1)
   (imply (contradict2 (ACTOR lie1) (ATTRIBUTE catergorical2)))

Output: "John enjoins a tippler to tell a prevarication."
```

Figure 6.2: Example of input and output to I-Saurus.

ters that cover the meaning expressed by the semantic representation. Then for each possible cluster option, it chooses the near-synonym that maximizes the degree of satisfaction of the preferences.

Xenon is a a general-purpose NLG system that exploits my LKB of NS. To implement Xenon, I modified the lexical choice component of a pre-existing NLG system (HALogen [Langkilde, 2000], [Langkilde and Knight, 1998]) so that it handles the knowledge about the near-synonym differences. Xenon will be described in detail in Chapter 7.

The difference between I-Saurus and Xenon is that I-Saurus works for a small hand-coded LKB of NS (9 clusters) and a small hand-crafted ontology, while Xenon uses the automatically-built LKB of NS (909 clusters), and a large-scale ontology. There is, in principle, no restriction on the kind of text Xenon can produce.

6.2 Customizing the LKB of NS

The initial LKB of NS built in the Chapter 2 (and enriched in Chapter 5) is a general one, and it can be used in any NLP system. Most likely it will need some adaptation in order to be integrated with the other components of the NLP system. In particular, it has to be integrated with the ontology employed by the NLP system.

There are two aspects of the LKB of NS that might need adaptation: the core denotations, and the peripheral concepts. They might or might not need to be put in correspondence with concepts in the main ontology of the system, depending on the requirements of the system. The NLP system might use a general-purpose ontology, such as Cyc [Lenat, 1995] and WordNet, or an ontology specially built for the systems (such as Mikrokosmos [Mahesh and Nirenburg, 1995] or domain-specific ontologies).

In the next sections I discuss the customization task in general, and after that specific solutions adopted in particular systems.

6.2.1 Customizing the core denotations

In most NLP systems, the core denotation of each cluster of near-synonyms needs to be explicit. The clustered model of lexical knowledge views the core denotation as a generic concept. In I-Saurus, because of its hand-crafted ontology, the core denotations are concepts manually declared in the Loom knowledge representation system[1]. In Xenon, the core denotation of a cluster is an OR of all the concepts that could correspond to the near-synonyms in a cluster. For example, if the cluster contains the near-synonyms: *mistake, blooper, blunder, boner, contretemps, error, faux pas, goof, slip, solecism*, the WordNet senses that remain after disambiguation are:

```
mistake#n#1, blooper#n#1, blunder#n#1, boner#n#1, error#n#1,
faux pas#n#1, slip#n#1, slip#n#2, solecism#n#1
```

Xenon uses Sensus [Knight and Luk, 1994] as its ontology. Sensus was built by combining WordNet1.5 with the Penman Upper Model (a top-level ontology used in Penman) and with information from *Longman Dictionary of Contemporary English* (LDOCE)[2]. Because my disambiguation experiments (see Chapter 3) use the WordNet 1.7 sense inventory, I implemented a mapping from WordNet1.7 to Sensus that puts a WordNet sense (synset) in correspondence

[1] http://www.isi.edu/isd/LOOM/LOOM-HOME.html
[2] http://www.longman.com/ldoce/

with a Sensus concept. My mapping uses the inverse of the mapping from WordNet 1.5 to 1.6 to 1.7 developed by Rigau et al. [1998] to map senses from WordNet 1.7 to 1.6 to 1.5. Then, I map WordNet1.5 senses to Sensus concepts on the basis of the Sensus source files. After mapping to Sensus concepts, the core denotation from the previous example looks as follows:

```
:core (ROOT generic_mistake_n)
    (OR |fault,error| |boner| |gaffe| |slipup|)
```

Sensus concepts correspond to WordNet1.5 synsets; in addition, they have distinct names, consisting on one or more words, delimited by vertical bars. For example, `|fault,error|` is the name of a Sensus concept.

In fact, in the actual implementation of Xenon, the core denotations are represented as meta-concepts that are later expanded in an OR of all the near-synonyms of the cluster. The name of a meta-concept is formed by the prefix “generic”, followed by the name of the first near-synonym in the cluster, followed by the part-of-speech. For example, if the cluster is *lie, falsehood, fib, prevarication, rationalization, untruth*, the name of the cluster is “generic_lie_n”. The names have to be distinct. For example, the part-of-speech helps to distinguish between *lie* as a noun and *lie* as a verb, if they both happen to name a meta-concept. If there are cases where there could still be conflicts after differentiating by part-of-speech, the name of the second near-synonym should help. For example, “stop” is the name of two verb clusters, therefore the two clusters are renamed: “generic_stop_arrest_v” and “generic_stop_cease_v”.

6.2.2 Customizing the peripheral concepts

A peripheral concept is a concept from the ontology (or a configuration of concepts) that is implied, suggested, or denoted by the near-synonyms. The peripheral concepts are initially expressed as strings. The adaptation module can include parsing the strings, and then mapping the syntactic representation into a semantic representation.

In I-Saurus a small number of peripheral concepts were implemented manually in Loom. Examples of peripheral concepts for the near-synonyms of *lie* are shown in Figure 6.3.

```
((P1 Contradict (ACTOR ROOT)) ;; contradiction of truth
 (P2 Intend (ACTOR V1)  ;; intends to mislead
           (ACTEE (P2-1 Mislead)))
 (P3 Aware (ACTOR V1) (ACTEE V2))
 (P4 Criticism (ACTEE V1)
              (ATTRIBUTE (P4-1 Severity)))
 (P5 FaceSave (ACTOR V1))
 (P6 Significance (ATTRIBUTE-OF ROOT))
 (P7 Misconception (ACTOR V1) (CAUSE-OF ROOT))
)
```

Figure 6.3: Examples of peripheral concepts in I-Saurus.

In Xenon, I implemented a set of simple rules that extract the actual peripheral concepts from the initial peripheral strings. The peripheral concepts are represented in the same representation language used to describe inputs to Xenon (the ISI interlingua). A transformation rule takes a string of words annotated with part-of-speech information and extracts a main word, several roles, and fillers for the roles. The fillers can be words or recursive structures. In Xenon the words used in these representation are not sense disambiguated.

Here are two examples of input strings and extracted peripheral concepts:

```
"an embarrassing breach of etiquette"
=> (C / breach :GPI etiquette :MOD embarrassing)

"to an embarrassing or awkward occurrence"
=> (C / occurrence :MOD (OR embarrassing awkward))
```

The roles used in these examples are part of the set of roles of the ISI interlingua: MOD (modifier) and GPI (generalized possession inverse).

The rules that were used for these two examples are these:

```
Adjective Noun1 of Noun2 => (C / Noun1 :GPI Noun2 :MOD Adjective)

Adjective1 or Adjective2 Noun => (C / Noun :MOD (OR Adjective1 Adjective2))
```

Algorithm	**Coverage**	**Correctness**
Baseline	100%	16%
Rules on development set	79%	68%
Rules on test set	**75%**	**55%**

Table 6.1: Coverage and correctness of the customization rules.

6.2.3 Evaluation of the Xenon customization module

To evaluate the customization for Xenon, I built by hand a standard solution for a set of peripheral strings: 139 strings to be used as a test set, and 97 strings to be used as a development set. I used the development set to figure out what rules should be added. I implemented about 22 transformation rules (more rules can be easily added). They cover 79% of the strings in the development set, with a correctness of 68%. The correctness is defined as the percent of times the extracted peripheral concept is the one expected in the solution. Sometimes a rule may extract only a fragment of the expected configuration of concepts and still provide useful knowledge in the final LKB of NS. Such cases are not considered correct for the evaluation done in this section. The evaluation is strict: for simplicity reasons, it does not allow credit for partial correctness. For example, if the near-synonym *command* denotes `the/TD stated/VB demand/NN of/IN a/TD superior/JJ`, the expected peripheral concept is: `(C1 / demand :GPI superior :MOD stated)`. If the program extracts only `(C1 / demand :GPI superior)`, the result is not considered correct, but the information may still help in a system that uses the LKB of NS.

The test set was used for evaluation to make sure the rules work in general, not only for the development set. The results of the evaluation are presented in Table 6.1. The coverage for the test set is 75% and the correctness 55%. To have a base for comparison, if the rules are extremely simple, taking the first word in each string as the peripheral concept, we can cover all the strings, but with a correctness of 16%.

6.2.4 The final LKB of NS customized for Xenon

I customized the generic LKB of NS for use in Xenon. The core denotations and the peripheral concepts are obtained as described in the previous sections. Figure 6.4 presents a fragment of the knowledge extracted for the near-synonyms of *error* in the generic LKB of NS. The knowledge extracted for all the near-synonyms in this cluster can be viewed in Appendix B. The customization programs produce the final representation of the cluster in a form that is understood by Xenon (see Figure 6.5 for the full customized representation for the near-synonyms of *error* and Appendix C for more examples of customized clusters). The peripheral concepts are factored out, and the list of distinctions contains pointers to them. This allows peripheral concepts to be shared by two or more near-synonyms.

6.3 Summary

This chapter discussed the customization of the LKB of NS acquired in Chapter 2 and enriched in the subsequent chapters. The main parts that need to be customized in order to be used in a particular NLP system are the core denotations of each cluster of near-synonyms, and the peripheral strings from the denotational distinctions. The peripheral strings need to be transformed into concepts or configurations of concepts that are understood by the NLP system.

The problem was discussed first in general, with examples from the prototype NLG system I-Saurus. Then, the customization for my NLG system, Xenon (presented in Chapter 7) was explained. The core denotations were meta-concepts consisting of an OR of all the near-synonyms in a cluster. The peripheral strings were transformed into configurations of concepts by using a set of rules. The correctness and coverage of the rules were evaluated on a sample of the data. A development set was used to figure out what rules need to be added, and a separate set was used for testing.

Cluster: mistake=blooper=blunder=boner=contretemps=error=faux pas=goof=slip=solecism=
...
a near_syn(blunder) is a blatant near_syn(error), usually one involving behavior or judgment, and implying an ignorant or uninformed assessment of a situation

subj: blunder
freq: usually
strength: medium
class: Implication
periph: an/DT ignorant/JJ or/CC uninformed/JJ assessment/NN of/IN a/DT situation/NN

near_syn(slip) emphasizes the accidental rather than ignorant character of a near_syn(mistake) and is often used to mean the careless divulging of secret or private information

subj: slip
freq: usually
strength: high
class: Denotation
periph: the/DT accidental/JJ rather/RB than/IN ignorant/JJ character/NN of/IN a/DT NS_mistake/NN

subj: slip
freq: usually
strength: medium
class: Denotation
periph: the/DT careless/JJ divulging/VBG of/IN secret/JJ or/CC private/JJ information/NN

near_syn(blooper), an informal term, is usually applied to particularly infelicitous mix-ups of speech, such as ”rit of fellus jage” for ”fit of jealous rage”

subj: blooper
freq: usually
strength: medium
class: Denotation
periph: to/TO particularly/RB infelicitous/JJ mix-ups/NNS of/IN speech/NN ,/, such/JJ as/IN rit/NN of/IN fellus/JJ jage/NN for/IN fit/NN of/IN jealous/JJ rage/NN

subj: blooper
freq: usually
strength: low
class: Formality
...

Figure 6.4: Fragment of the initial representation of the *error* cluster.

```
(defcluster generic_mistake_n
    :syns (mistake blooper blunder boner contretemps error faux_pas goof
           slip solecism )
    :senses (mistake#n#1 blooper#n#1 blunder#n#1 boner#n#1 error#n#1
             faux_pas#n#1 slip#n#1 slip#n#2 solecism#n#1)
    :core (ROOT GENERIC_MISTAKE (OR |fault,error| |boner| |gaffe| |slipup|))
    :periph (
       (P1  (C1 / deviation))
       (P2  (C1 / sin))
       (P3  (C1 / assessment :MOD (*OR* ignorant uninformed)))
       (P4  (C1 / accidental))
       (P5  (C1 / careless))
       (P6  (C1 / indefensible))
       (P7  (C1 / occurrence :MOD (*OR* embarrassing awkward)))
       (P8  (C1 / (*OR* action opinion judgment)))
       (P9  (C1 / (*OR* gross stupid)))
       (P10  (C1 / belief))
       (P11  (C1 / manners))
     )
    :distinctions
    ((error usually medium Implication P1)
     (error usually medium Denotation P2)
     (blunder usually medium Implication P3)
     (slip usually high Denotation P4)
     (slip usually medium Denotation P5)
     (blooper low Formality)
     (goof usually medium Denotation P6)
     (goof medium Formality)
     (contretemps usually medium Denotation P7)
     (mistake usually medium Denotation P8)
     (blunder usually medium Denotation P9)
     (error usually medium Denotation P10)
     (faux_pas usually medium Denotation P11)
     (mistake usually medium Favourable :agent)
     (blunder usually medium Favourable :agent)
     (boner usually medium Pejorative :agent)
     (contretemps usually medium Favourable :agent)
     (error usually medium Favourable :agent)
     (faux_pas usually medium Pejorative :agent)
     (goof usually medium Favourable :agent)
     (slip usually medium Favourable :agent)
     (solecism usually medium Favourable :agent)))
```

Figure 6.5: The final representation of the *error* cluster.

Chapter 7

Xenon: An NLG System that Uses Knowledge of Near-Synonym Differences

This chapter presents Xenon, a large-scale NLG system that uses the lexical knowledge-base of near-synonyms customized in Chapter 6. Xenon integrates a new near-synonym choice module with the sentence realization system named HALogen[1] [Langkilde, 2000], [Langkilde and Knight, 1998]. HALogen is a broad-coverage general-purpose natural language sentence generation system that combines symbolic rules with linguistic information gathered statistically from large text corpora. The internal architecture of HALogen is presented in Figure 7.1. A former version of the system, called Nitrogen, builds a lattice of all possible sentences (combinations of words), and then ranks all the sentences in the lattice according to an N-gram language model, in order to choose the most likely sentence as output. HALogen replaces the lattice with a forest of trees, where the shared parts of the lattice are not duplicated. It implements a faster and more reliable ranking algorithm.

Figure 7.2 presents Xenon's architecture. The input is a semantic representation and a set of preferences to be satisfied. The final output is a set of sentences and their scores. A concrete example of input and output is shown in Figure 7.3. Note that HALogen may generate some

[1] http://www.isi.edu/licensed-sw/halogen/

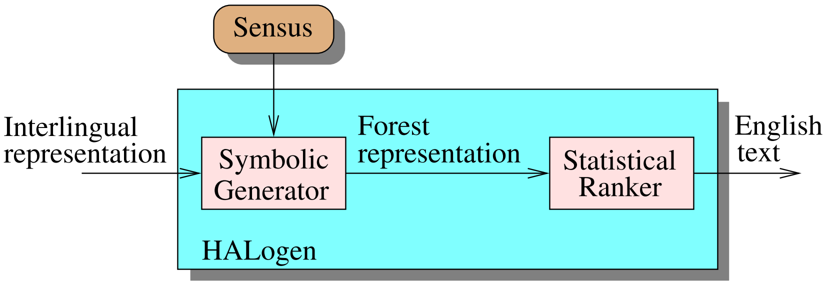

Figure 7.1: The architecture of HALogen.

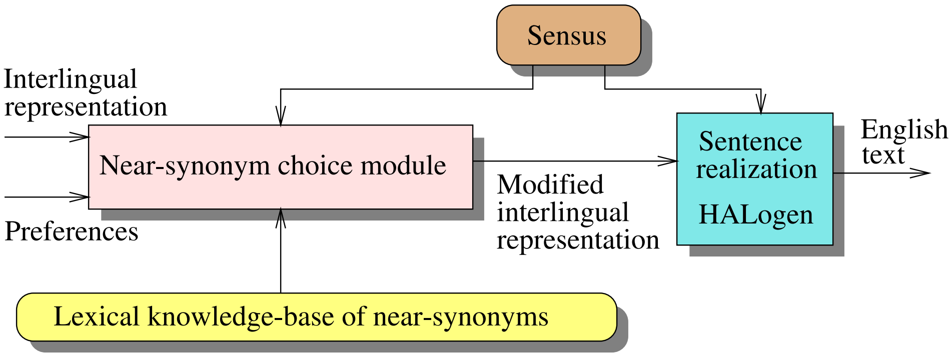

Figure 7.2: The architecture of Xenon.

ungrammatical constructs, but they are assigned lower scores. The first sentence (the highest-ranked) is considered to be the solution.

7.1 Meta-concepts

The semantic representation that is one of Xenon's inputs is represented, like the input to HALogen, in an interlingua developed at ISI.[2] As described by Langkilde-Geary [2002b], this language contains a specified set of 40 roles, whose fillers can be words, concepts from Sensus [Knight and Luk, 1994], or complex interlingual representations. The interlingual representations could be underspecified: if some information needed by HALogen is not present, the generator will make choices using the knowledge accumulated from the corpora. Figures 7.4

[2] http://www.isi.edu/licensed-sw/halogen/interlingua.html

```
Input:
(A9 / tell
   :agent (V9 / boy)
   :object (O9 / generic_lie_n))

Input preferences
 ((DISFAVOUR :AGENT) (LOW FORMALITY) (DENOTE (C1 / TRIVIAL)))

Output sentences in file: /h/34/dianaz/my_nlg/xxx0.sen
The boy told fibs.    -40.8177
Boy told fibs.    -42.3818
Boys told fibs.    -42.7857
A boy told fibs.    -43.0738
Some boys told fibs.    -46.4388
Any boy told fibs.    -50.0306
An boy told fibs.    -50.15
Were told fibs by the boy.    -55.3801
Was told fibs by the boy.    -55.4211
Are told fibs by the boy.    -56.2969
Is told fibs by the boy.    -56.6791
Am told fibs by the boy.    -56.8718
```

Figure 7.3: Example of input and output of Xenon.

```
(e1 / eat
    :agent (d1 / dog)
    :patient (b1 / bone
                :premod (m1 / meaty)))
```

Figure 7.4: Interlingual representation for the sentence *"The dog eats a meaty bone"*.

```
(n1 / |necessary<inevitable|
    :polarity -
    :domain (g1 / go
               :agent (b1 / boy)))
```

Figure 7.5: Interlingual representation for the sentence *"The boy need not go"*.

and 7.5 present examples of semantic representations using the ISI interlingua. The example in Figure 7.5 contains the Sensus concept `|necessary<inevitable|`, which expresses an inevitable action.

Xenon extends this representation language by adding meta-concepts corresponding to the core denotation of the clusters of near-synonyms. For example, in Figure 7.3, the meta-concept is "generic_lie_n". As explained in Section 6.2.1, they may be seen as an OR of all the senses of the near-synonyms of the cluster.

7.2 Near-synonym choice

The near-synonym choice module has to choose the best near-synonym from each cluster. The choice of clusters is already made: it is given in the input, in the semantic representation.

Near-synonym choice involves two steps: expanding the meta-concepts, and choosing the best near-synonym for each cluster according to the preferences. I implemented this in a straightforward way: the near-synonym choice module computes a satisfaction score for each of the near-synonyms in a cluster; then satisfaction scores become weights; in the end, HALogen makes the final choice, by combining the weights with the probabilities from its language model. For the example in Figure 7.3, the expanded representation is presented in Figure 7.6. The near-synonym choice module gives higher weight to *fib* because it satisfies the preferences better than the other near-synonyms in the cluster, *lie, falsehood, fib, prevarication, rationalization,* and *untruth*.

Section 7.3 will explain the algorithm for computing the weights. HALogen simply adds these weights to N-gram probabilities from the language model (more precisely, the negative logarithms of these values) before choosing the highest-ranked sentence.

```
(A9 / tell
   :agent (V9 / boy)
   :object
    (OR (e1 / (:CAT NN :LEX "lie") :WEIGHT 1.0e-30)
        (e2 / (:CAT NN :LEX "falsehood") :WEIGHT 6.937703e-8)
        (e3 / (:CAT NN :LEX "fib") :WEIGHT 1.0)
        (e4 / (:CAT NN :LEX "prevarication") :WEIGHT 1.0e-30)
        (e5 / (:CAT NN :LEX "rationalization") :WEIGHT 1.0e-30)
        (e6 / (:CAT NN :LEX "untruth") :WEIGHT 1.3875405e-7))
```

Figure 7.6: The interlingual representation from Figure 7.3 after expansion by the near-synonym choice module.

7.3 Preferences

The preferences that are input to Xenon could be given by the user, or they could come from an analysis module if Xenon is used in a machine translation system (corresponding to nuances of near-synonyms in a different language, see Figure 6.1). The formalism for expressing preferences and the preference satisfaction mechanism is taken from I-Saurus [Edmonds, 1999].

The preferences, as well as the distinctions expressed in the LKB of NS, are of three types: attitudinal (express a favourable, neutral, or pejorative attitude), stylistic (express a certain formality, force, or abstractness level), and denotational (connote a particular concept or configuration of concepts).

- Denotational preferences have the form: (`method` `peripheral-concept`), where `method` takes the values `suggest`, `imply`, `denote`.
- Attitudinal preferences have the form: (`stance` `entity`), where `stance` takes the values `favour`, `neutral`, `disfavour`.
- Stylistic preferences have the form: (`strength` `stylistic-feature`).

All the preferences can be preceded by an importance factor, which is taken into consideration during the near-synonym choice process. The importance factors are numbers between 0 and 1, and show that some preferences should be weighted heavier than others. By default all the preferences are 1 and do not need to be specified. Examples of preferences are:

```
(low formality)
(disfavour :agent)
(imply (C / assessment :MOD ( M / (*OR* ignorant uninformed)).
```

As mentioned in Section 1.2, the form of the distinctions in the LKB of NS is the following:

- Denotational distinctions

  (`near-synonym frequency strength latency peripheral-concept`).

- Attitudinal distinctions

  (`near-synonym frequency strength attitude entity`).

- Stylistic distinctions

  (`near-synonym strength stylistic-feature`).

In Xenon, preferences are transformed internally into pseudo-distinctions that have the same form as the corresponding type of distinctions. In this way, the preferences can be directly compared with the distinctions.

The pseudo-distinctions corresponding to the previous examples of preferences are:

```
(- low Formality)
(- always high Pejorative :agent)
(- always medium Implication
    (C/assessment :MOD (M/(OR ignorant uninformed)).
```

The algorithm that computes the degree of satisfaction of preferences for a given sentence plan differs from the algorithm implemented in I-Saurus, because of the different nature of the generators used for surface realization. A sentence plan corresponds in my system to the input semantic representation in which lexical choice has been partly made (near-synonyms were assigned weights according to their degree of satisfaction of preferences). For each near-synonym *NS* in a cluster, a weight is computed by summing the degree to which the near-synonym satisfies each preference from the set *P* of input preferences:

$$Weight(NS,P) = \sum_{p \in P} \frac{Importance(p)}{|P|} Sat(p,NS). \tag{7.1}$$

The weights are transformed through an exponential function:

$$f(x) = \frac{e^{x^k}}{e-1} \tag{7.2}$$

The main reason this function is exponential is that differences between final weights of near-synonyms from a cluster need to be numbers that are comparable with the differences of probabilities from HALogen's language model. More explanation about the form of this function and the method for choosing the optimal value of k are presented in Section 7.7.1, page 110.

For a given preference $p \in P$, the degree to which it is satisfied by NS is reduced to computing the similarity between each of NS's distinctions and a pseudo-distinction $d(p)$ generated from p. The maximum value over i is taken (where $d_i(w)$ is the i-th distinction of NS):

$$Sat(p, NS) = max_i \; Sim(d(p), d_i(NS)). \tag{7.3}$$

The computation of *Sim* is explained in the next section.

7.4 Similarity of distinctions

The similarity of two distinctions, or of a distinction and a preference (transformed into a distinction), is computed using the formulas from I-Saurus:

$$Sim(d_1, d_2) = \begin{cases} Sim_{den}(d_1, d_2) & \text{if } d_1 \text{ and } d_2 \text{ are denotational distinctions} \\ Sim_{att}(d_1, d_2) & \text{if } d_1 \text{ and } d_2 \text{ are attitudinal distinctions} \\ Sim_{sty}(d_1, d_2) & \text{if } d_1 \text{ and } d_2 \text{ are stylistic distinctions} \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

Distinctions are formed out of several components, represented as symbolic values on certain dimensions. In order to compute a numeric score, each symbolic value has to be mapped into a numeric one. The numeric values and their possible ranges (see Table 7.1) are not as important as their relative difference.The weights are in the interval [0,1]. They are proportionally transformed so that the maximum weight becomes 1.0 by dividing all of them with the highest value. This step does not change the relative differences.

Frequency		Latency		Attitude		Strength		Style	
never	0.00	Suggestion	2	Pejorative	-2	low	-1	low	0.0
seldom	0.25	Implication	5	Neutral	0	medium	0	medium	0.5
sometimes	0.50	Denotation	8	Favourable	2	high	1	high	1.0
usually	0.75								
always	1.00								

Table 7.1: The functions that map symbolic values to numeric values.

If the two distinctions are incommensurate (they are not the same type of distinction), their difference is zero. For stylistic distinctions, the degree of similarity is one minus the absolute value of the difference between the style values.

$$Sim_{sty}(d_1, d_2) = 1.0 - |Style(d_2) - Style(d_1)| \tag{7.5}$$

For attitudinal distinctions, similarity depends on the frequencies and on the attitudes. The similarity of two frequencies is one minus their absolute differences. For the attitudes, the attitudes are combined with their strengths and the result is normalized by the length of the range of values – 6, in this case.

$$\begin{aligned}
Sim_{att}(d_1, d_2) &= S_{freq}(d_1, d_2) \times S_{att}(d_1, d_2) && (7.6)\\
S_{freq}(d_1, d_2) &= 1.0 - |Frequency(d_2) - Frequency(d_1)| && (7.7)\\
S_{att}(d_1, d_2) &= 1.0 - |Att(d_2) - Att(d_1)| \;/\; 6 && (7.8)\\
Att(d) &= Attitude(d) + \mathrm{sign}(Attitude(d)) \times Strength(d) && (7.9)
\end{aligned}$$

The similarity of two denotational distinctions is the product of the similarities of their three components: frequencies, latencies, and conceptual configurations. The first score is calculated as for the attitudinal distinctions. The second score combines the latency (indirectness) with the strength and the result is normalized by 8, the length of the range of values. The computation of conceptual similarity (S_{con}) is discussed in the next section.

$$
\begin{array}{lrcl}
\text{if} & d_1 & = & (lex_1 \text{ low Formality}) \\
 & d_2 & = & (lex_2 \text{ medium Formality}) \\
\text{then} & Sim(d_1, d_2) & = & 1 - |0.5 - 0| \\
 & & = & 0.5
\end{array}
$$

$$
\begin{array}{lrcl}
\text{if} & d_1 & = & (lex_1 \text{ always high Favourable :agent}) \\
 & d_2 & = & (lex_2 \text{ usually medium Pejorative :agent}) \\
\text{then} & Sim(d_1, d_2) & = & S_{freq}(d_1, d_2) \times S_{att}(d_1, d_2) \\
 & & = & (1 - |0.75 - 1|) \times (1 - |(-2 - 0) - (2 + 1)|) \ / \ 6 \\
 & & = & 0.125
\end{array}
$$

$$
\begin{array}{lrcl}
\text{if} & d_1 & = & (lex_1 \text{ always medium Implication } P_1) \\
 & d_2 & = & (lex_2 \text{ seldom medium Suggestion } P_1) \\
\text{then} & Sim(d_1, d_2) & = & S_{freq}(d_1, d_2) \times S_{lat}(d_1, d_2) \times S_{con}(d_1, d_2) \\
 & & = & (1 - |0.25 - 1|) \times (1 - (2 + 0 + 5 + 0) \ / \ 8) \times 1 \\
 & & = & 0.031
\end{array}
$$

Figure 7.7: Examples of computing the similarity of lexical distinctions.

$$Sim_{den}(d_1, d_2) = S_{freq}(d_1, d_2) \times S_{lat}(d_1, d_2) \times S_{con}(d_1, d_2) \tag{7.10}$$

$$S_{freq}(d_1, d_2) = 1.0 - |Frequency(d_2) - Frequency(d_1)| \tag{7.11}$$

$$S_{lat}(d_1, d_2) = 1.0 - |Lat(d_2) - Lat(d_1)| \ / \ 8 \tag{7.12}$$

$$Lat(d) = Latency(d) + Strength(d) \tag{7.13}$$

Examples of computing the similarity between distinctions are presented in Figure 7.7.

7.5 Similarity of conceptual configurations

Peripheral concepts are complex configurations of concepts. In Xenon, they are full-fledged interlingual representations, containing concepts (or words) and roles that are filled by complex concepts. The conceptual similarity function S_{con} is in fact the similarity between two interlingual representations t_1 and t_2. Examples of computing the similarity of conceptual configurations are presented in Figure 7.8. The formula for computing the similarity of conceptual configurations from I-Saurus was adapted to work for this type of representation. The formula for S_{con} is taken from I-Saurus, while the formula for S is new.

$$S_{con}(t_1,t_2) = \begin{cases} S(\text{concept}(t_1), \text{concept}(t_2)) & \text{if } N(t_1,t_2) = 0 \\ \alpha\, S(\text{concept}(t_1), \text{concept}(t_2)) + \beta \frac{1}{N(t_1,t_2)} \sum_{s_1,s_2} S_{con}(s_1,s_2) & \text{otherwise} \end{cases} \tag{7.14}$$

In equation 7.14, concept(C), where C is a interlingual representation, is the main concept (or word) in the representation. For example, for the interlingual representation for Figures 7.4 and 7.5 on page 89, the main concepts are `eat` and `|necessary<inevitable|`, respectively.

Equation 7.14 computes similarity by simultaneously traversing the two representations. The first line corresponds to the situation when there are only main concepts, no roles. The second line deals with the case when there are roles. There could be some roles shared by both representations, and there could be roles appearing only in one of them. $N(t_1,t_2)$ is the sum of the number of shared roles and the number of roles unique to each of the representations (at the given level in the interlingua). s_1 and s_2 are the values of any shared role. α and β are weighting factors. If $\alpha = \beta = 0.5$, the sub-structure corresponding to the roles is weighted equally to the main concepts.

The similarity function S is different from the one implemented in I-Saurus. It deals with the case in which the main concepts are atomic (words or basic concepts) and when they are an OR or AND of complex concepts. It is assumed that the order of the constituents in disjunctions and conjunctions does not matter. For example, (OR embarrassing awkward) is equivalent to (OR awkward embarrassing). This assumption is needed in cases when the same knowledge come from different sources, where lexicographers used slightly different language to express the same distinction. If we have two disjunctions, $C_1 = (OR \;\; C_{11} \cdots C_{1n})$, and

C_1 = (C1 / departure :MOD (M / physical) :PRE-MOD unusual)
C_2 = (C2 / departure :MOD (M / physical))
$S_{con}(C_1, C_2) = 0.5 \times 1 + 0.5 \times 1/2 \times (0.5 \times 1) = 0.625$

C_1 = (C1 / person :AGENT_OF (A / drinks :MOD frequently))
C_2 = (C2 / person :AGENT_OF (A / drinks))
$S_{con}(C_1, C_2) = 0.5 \times 1 + 0.5 \times 1/1 \times (0.5 + 0.5 \times 1/2 \times 1) = 0.875$

C_1 = (C1 / occurrence :MOD (M / (OR embarrassing awkward)))
C_2 = (C2 / occurrence :MOD (M / awkward))
$S_{con}(C_1, C_2) = 0.5 \times 1 + 0.5 \times 1/1 \times 1 = 1.0$

C_1 = (C1 / (AND spirit purpose) :MOD (M / hostile))
C_2 = (C2 / purpose :MOD (M / hostile))
$S_{con}(C_1, C_2) = 0.5 \times (1 + 0)/2 + 0.5 \times 1 = 0.75$

Figure 7.8: Examples of computing the similarity of conceptual configurations.

$C_2 = (OR\ \ C_{21} \cdots C_{2m})$, then

$$S(C_1, C_2) = max_{i,j}\ S_{con}(C_{1i}, C_{2j}).$$

The components could be atomic or they could be complex concepts; that's why the S_{con} function is called recursively. If one of them is atomic, it can be viewed as a disjunction with one element, so that the previous formula can be used. If both are conjunctions, then the formula computes the maximum of all possible sums of pairwise combinations. If $C_1 = (AND\ \ C_{11}\ C_{12} \cdots C_{1n})$, and $C_2 = (AND\ \ C_{21}\ C_{22} \cdots C_{2m})$, then the longest conjunction is taken. Let's say $n \geq m$ (if not the procedure is similar). All the permutations of the components of C_1 are considered, and paired with components of C_2. If some components of C_1 remain without pair, they are paired with null (and the similarity between an atom and null is zero). Then the similarity of all pairs in a permutation is summed and divided by the number of pairs, and the maximum (from all permutations) is the resulting score:

$$S(C_1, C_2) = max_{p \in permutations}\ \frac{1}{n} \sum_{k=1}^{m} S_{con}(C_{1p_k}, C_{2k}) + \sum_{k=m+1}^{n} S_{con}(C_{1k}, \text{null})$$

Here is a simple example to illustrate this procedure.

$$S_{con}((AND\ \ a\ b\ c)\ (AND\ \ b\ a)) = \frac{1}{3}\ max(S_{con}(a,b) + S_{con}(b,a) + S_{con}(c, \text{null}), S_{con}(a,a) +$$

$$+ S_{con}(b,b) + S_{con}(c,\text{null})) = \frac{1}{3}\, max(0+0+0, 1+1+0) = 0.66$$

The similarity of two words or two atomic concepts is computed from their positions in the ontology of the system. A simple approach would be this: the similarity is 1 if they are identical, 0 otherwise. But we have to factor in the similarity of two words or concepts that are not identical but closely related in the ontology. There are off-the-shelf semantic similarity packages, such as the one provided by Patwardhan and Pedersen[3] or Alexander Budanitsky [Budanitsky and Hirst, 2001], implement various similarity measures. Most of the measures work only for nouns, except the Lesk measure and the Hirst and St-Onge measure, which tend to run rather slow. Similarity measures based on thesauri [Jarmasz and Szpakowicz, 2003] might work well for all parts of speech. But it would be time-consuming to call an external package from Xenon.

Therefore, I implemented a measure of similarity for all the words, using the Sensus ontology. Two concepts are similar if there is a link of length one or two between them in Sensus. The degree of similarity is discounted by the length of the link.

$$S(C_1, C_2) = \begin{cases} 1.0 & \text{if } C_1 \text{ and } C_2 \text{ are the same concept} \\ 0.75 & \text{if there is a link of length 1 between } C_1 \text{ and } C_2 \\ 0.50 & \text{if there is a link of length 2 between } C_1 \text{ and } C_2 \end{cases} \tag{7.15}$$

The similarity between a word and a concept is given by the maximum of the similarities between all the concepts (senses) of the word and the given concept. The similarity of two words is given by the maximum similarity between pairs of concepts corresponding to the words. Before looking at the concepts associated with the words, the Porter stemmer[4] is used to see if the two words share the same stem, in which case the similarity is 1. Similarity across part-of-speech is therefore possible due to the stem checking and to the fact that the lexicon that Sensus comes with has some derivational information incorporated from LDOCE.

[3] http://search.cpan.org/dist/WordNet-Similarity/

[4] http://www.tartarus.org/~martin/PorterStemmer/index.html

7.6 Integrating the knowledge of collocational behaviour

In Chapter 4, knowledge about the collocational behaviour of the near-synonyms was acquired. For each cluster of near-synonyms, there are preferred collocations, less-preferred collocations, and anti-collocations. Integrating this information into the near-synonym choice module is important because the system should not choose a near-synonym that causes an anti-collocation to appear in the generated sentence, and it should give priority to a near-synonym that produces a preferred collocation. A fragment of the lexical knowledge-base of near-synonym collocations is presented in Figure 7.9. The preferred collocations have no markup, the less-preferred collocations are marked by `?`, and the anti-collocations are marked by `*`.

Knowledge of collocational behaviour is not usually present in NLG systems; adding it increases the quality of the generated text, making it more idiomatic. Unlike other NLG systems, Xenon already incorporates some collocational knowledge encoded in HALogen's language model (bigrams or trigrams). But this includes mainly collocations between content words and function words. Therefore, the knowledge acquired in Chapter 4 is useful, since it includes collocations between near-synonyms and other content words (not function words). This means that sometimes the near-synonym and the collocate word can be separated by several function words (in practice one or two). Also, it is important whether the near-synonym is on the left-side or on the right-side of the collocation. If the near-synonym can be on either side, both collocations are in the knowledge-base.

There are several ways to approach the integration of the near-synonym collocation module into Xenon. The first idea is to modify the initial weights associated with the near-synonyms before the extended interlingual representation is fed into HALogen, in a similar manner to the treatment of preferences / lexical nuances in the near-synonym choice module. But this does not work, because the lexical choice is not made yet and therefore it would be hard to detect possible collocations and anti-collocations. The interlingual representation may contain Sensus concepts; and even if it contains words, they might not be yet in their inflected forms. What makes the detection of possible collocations and anti-collocations even harder is the fact

```
Cluster: benefit, advantage, favor, gain, profit

    financial gain
    financial benefit
    financial advantage
   ?financial profit
   *financial favor

    profit forecasts
   *benefit forecasts
   *gain forecasts
   *advantage forecasts
   *favor forecasts
```

Figure 7.9: Fragment of the lexical knowledge-base of near-synonym collocations.

that the interlingual representation does not necessarily encode the order of the constituents. If a word forms a preferred collocation or an anti-collocation with a near-synonym, most often the collocation is formed when the word is on the left or on the right of the near-synonym, but not both.

The second idea is to implement a collocational filter that takes the sentences produced by HALogen, and removes the ones that contain anti-collocations. This approach would not increase the rank of sentences that contain preferred collocations. By increasing the scores of such sentences, ungrammatical constructions might be promoted. This is the case because HALogen overgenerates: some of the generated sentences may be ungrammatical; but they are not usually ranked first by the statistical ranker. The main reason that such a filter would not work is that HALogen does not generate all the possible choices (because it would be too computationally expensive); therefore the preferred collocations may have been already eliminated by the time HALogen finishes generation. HALogen encodes all the possible sentences in a packed representation called a forest. Then it has a statistical ranker which traverses the structure to find the optimal path. But in order to make the computation feasible, only the paths that could potentially maximize the final score of the sentence are considered, while the paths that guarantee lower scores are pruned.

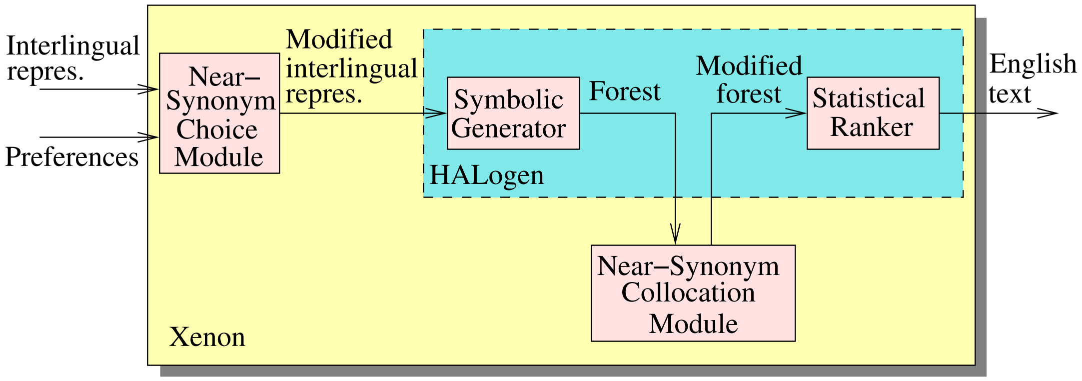

Figure 7.10: The architecture of Xenon extended with the Near-Synonym Collocation module. In this figure the knowledge sources are not shown.

The third alternative, which I implemented, is to integrate the near-synonym collocation module inside HALogen (see Figure 7.1 for its internal architecture), after lexical choice is made by the symbolic generator and encoded in the forest representation, but before the forest representation is ranked by the statistical ranker to produce the ranked sentences. The architecture of Xenon extended with the near-synonym collocation module is presented in Figure 7.10.

HALogen's symbolic generator applies grammar rules, lexical rules, and other transformation rules to produce the forest structure. It encodes all the possible sentences in a tree in which duplicate sub-trees are not allowed. Instead, each subtree is represented the first time it occurs and subsequent occurrences are labeled with the name of the first sub-tree. The forest structure is an AND-OR tree, where AND nodes represent a sequence of constituents, and OR nodes represent a choice between mutually exclusive alternatives. Figure 7.11 shows an example of forest representation that encodes four sentences:

The dogs eat bones.

Dogs eat bones.

Bones are eaten by the dogs.

Bones are eaten by dogs.

The nodes are labeled with symbols consisting of an arbitrary alphanumeric sequence (most of

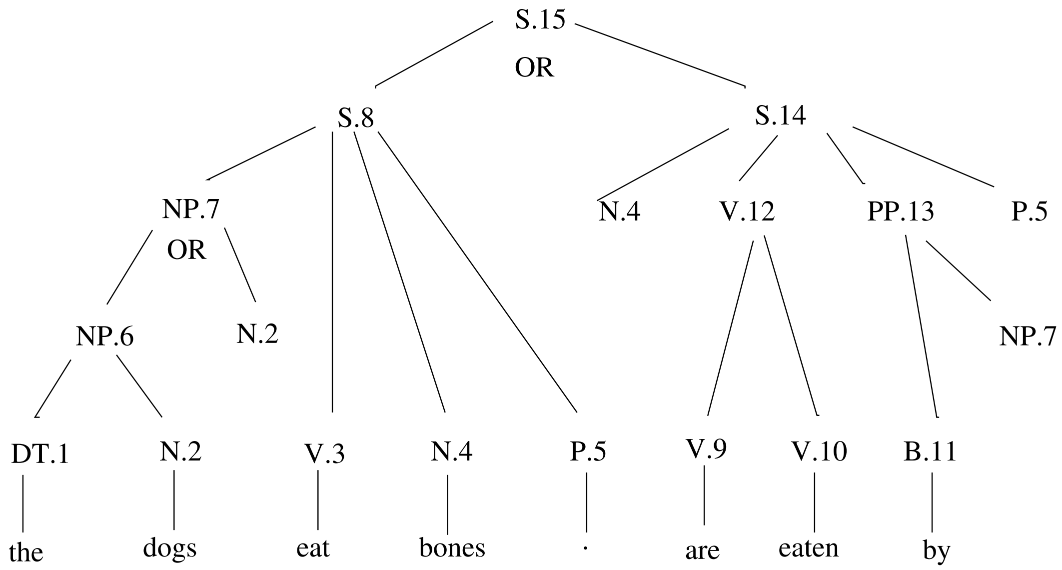

Figure 7.11: An example of forest representation from [Langkilde-Geary, 2002b].

them look like category labels), then a period, and a number. The forest is represented textually as a non-recursive context-free grammar. The nodes labeled OR$\rightarrow$ in Figure 7.12 are a short notation for two or more rules with the same left-hand side. For example the rule NP.7 OR$\rightarrow$ NP.6 N.2 corresponds to two rules: NP.7 $\rightarrow$ NP.6 and NP.7 $\rightarrow$ N.2.

The rules have weights with default value 1. These weights are added to the scores assigned by HALogen's statistical ranker. In fact these weights have values other than 1 (for rules that correspond to lexical items) only if some words or concepts in the interlingual representation come with values for the feature :WEIGHT (these values are copied unchanged in the forest structure). The synonym-choice module uses the weights in the interlingual representation to rank near-synonyms by their degree of satisfying preferences.

The near-synonym collocation module intercepts the forest structure, modifies it, and then forwards it to the statistical ranking module. The modification is only in the weights of the rules for near-synonyms and words that collocate with them, if possible preferred collocations or anti-collocations are detected.

The first step is to detect a cluster of near-synonyms in the forest structure by looking for

1.0 TOP $\rightarrow$ S.15
1.0 S.15 OR$\rightarrow$ S.8 S.14
1.0 S.8 $\rightarrow$ NP.7 V.3 N.4 P.5
1.0 NP.7 OR$\rightarrow$ NP.6 N.2
1.0 NP.6 $\rightarrow$ DT.1 N.2
1.0 DT.1 $\rightarrow$ "the"
1.0 N.2 $\rightarrow$ "dog"
1.0 V.3 $\rightarrow$ "eat"
1.0 N.4 $\rightarrow$ "bones"
1.0 P.5 $\rightarrow$ "."
1.0 S.14 $\rightarrow$ N.4 VP.12 PP.13 P.5
1.0 V.12 $\rightarrow$V.9 V.10
1.0 V.9 $\rightarrow$ "are"
1.0 V.10 $\rightarrow$ "eaten"
1.0 PP.13 $\rightarrow$ B.11 NP.7
1.0 B.11 $\rightarrow$ "by"

Figure 7.12: The textual representation of the forest from Figure 7.11.

consecutive rules in the forest structure that contain all the near-synonyms in a cluster from the LKB of NS. The left-hand sides of these rules appear in the right-hand side of an OR rule, because of the way the forest was generated from an OR of near-synonyms in the interlingual representation.

The next step is to detect possible preferred collocations or anti-collocations in the forest structure. This is done by performing operations on trees. In particular, a word is a possible collocate if it is a close neighbour of the near-synonyms, at the left or at the right.

Figure 7.13 presents the recursive algorithms for tree operations that retrieve the direct left and right neighbours of a node in the AND-OR tree, where, by neighbours I mean words on the leaves of the tree.

To find the neighbours of the meta-concept it is sufficient to find the neighbours of one of the near-synonyms, by calling the function `left_neighbours` or `right_neighbours` having as argument the node corresponding to the near-synonym. The `left_neighbours` function is used if the near-synonym is at the right side in the collocation, and the `right_neighbours` is used if the near-synonym is at the left side in the collocation.

```
function left_neighbours (Node R)
 { Collect all rules Ri that contain R in their right-hand side;
   foreach Ri
     if Ri --> ... Rj R ... and not an OR rule
       result = result + rightmost_daughters(Rj)
     else (no rule at the left or an OR rule)
       result = result + left_neighbours(Rj)
   return result
 }

function right_neighbours (Node R)
 { Collect all rules Ri that contain R in their right-hand side;
   foreach Ri
     if Ri --> ... R Rj... and not an OR rule
       result = result + leftmost_daughters(Rj)
     else (no rule at the right or an OR rule)
       result = result + right_neighbours(Rj)
  return result
 }

function leftmost_daughters (Node R)
 { if R --> "word" is a rule
     result = result + word
   if R --> Rj ...
     result = leftmost_daughters(Rj)
   if R OR--> R1 ... Rn
     foreach Ri in  R1 ... Rn
        result = result + leftmost_daughters(Ri)
  return result
 }

function rightmost_daughters (Node R)
 { if R --> "word" is a rule
     result = result + word
   if R --> ... Rj
     result = rightmost_daughters(Rj)
   if R OR--> R1 ... Rn
     foreach Ri in  R1 ... Rn
        result = result + rightmost_daughters(Ri)
  return result
 }
```

Figure 7.13: Algorithm for left and right neighbours in the AND-OR tree.

To detect the possible collocations or anti-collocations for the meta-concept, the possible collocate words are extracted form the knowledge-base. If none of these collocates are in the forest structure, there are no possible collocations. If some of them are in the forest structure, then the one that is in the set of the neighbours of the meta-concept is the detected collocate. Because the collocations are between content words, it is possible to have one or two function words between the two words in the collocation. Therefore the neighbours that are separated by one or two words are computed when needed, by using the same functions on the neighbours and on the neighbours of the neighbours. The module detects one collocate for one cluster of near-synonyms, but it could be extended to detect all the near-synonym clusters in a sentence (if more than one is present) and all the possible collocates.

After detection, the weights for the near-synonyms are changed in the forest representation. The knowledge-base of near-synonym collocations is consulted to see how the collocate combines with each of the near-synonyms. If it is a preferred collocation, the weight of the near-synonym is unchanged (usually it is 1 unless it was changed by the near-synonym choice module on the basis of preferences). If it is an anti-collocation, the weight of the near-synonym is discounted by W_{anti_colloc}. If it is a less-preferred collocation, the weight of the near-synonym is discounted by $W_{less\_pref\_colloc}$. If the collocate is not the only alternative, the other alternatives should be discounted, unless they also form a preferred collocation. Section 7.7.2 explains how the discount weights are chosen.

7.7 Evaluation of Xenon

The main components of Xenon are the near-synonym choice module, the near-synonym collocation module, and the sentence-realization module, HALogen.

An evaluation of HALogen was already presented by Langkilde-Geary [2002a]. HALogen is evaluated for coverage and correctness using a section of the PennTreebank as test set. HALogen was able to produce output for 80% of a set of 2400 inputs (automatically derived

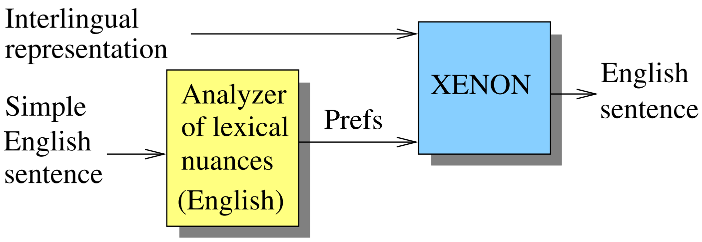

Figure 7.14: The architecture of DevSimple and TestSimple.

from the test sentences by an input construction tool). The output was 94% correct when the input representation was fully specified, and between 94% and 55% for various other experimental settings. The accuracy was measured using the IBM BLEU score [Papineni et al., 2001] and the string edit distance, by comparing the generated sentences with the original sentences. This evaluation can be considered as English-to-English translation via meaning representation.

Therefore, I only need to evaluate the near-synonym choice module, and the near-synonym collocation module, each of them in interaction with HALogen. I evaluate each module separately, and then I evaluate both modules working together.

7.7.1 Evaluation of the near-synonym choice module

For the evaluation of the near-synonym choice module I conducted two kinds of experiments. The near-synonym collocation module was disabled for these experiments.

The first type of experiment (DevSimple and TestSimple) feeds Xenon with a suite of inputs: for each test case, an interlingual representation and a set of nuances. The set of nuances correspond to a given near-synonym. A graphic depiction of these two tests is shown in Figure 7.14. The sentence generated by Xenon is considered correct if the expected near-synonym was chosen. The sentences used in this first experiment are very simple; therefore, the interlingual representations were easily built by hand. In the interlingual representation, the near-synonym was replaced with the corresponding meta-concept.

```
(run_en '(competitor) '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(adversary)  '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(opponent)   '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(antagonist) '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(enemy)      '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(foe)        '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
(run_en '(rival)      '(C1 / is  :agent he
                             :object (A1 / generic_opponent_n :MOD my)))
```

Figure 7.15: Examples of test cases from the test set DevSimple.

The analyzer of lexical nuances for English simply extracts the distinctions associated with a near-synonym in the LKB of NS. If a near-synonym is ambiguous, i.e., if it is a member of more than one cluster of near-synonyms, the analyzer needs to disambiguate. This is not an easy task because the sentence that includes the near-synonym may not provide enough context for a reliable disambiguation. For the purpose of the evaluation experiments, the disambiguation is simplified by giving a list of all the clusters that participate in the experiments. Therefore, if a near-synonym is in more than one cluster, the one from the list is chosen (and no near-synonym is in more than one cluster from the list).

The analyzer of lexical nuances takes a sentence as input. It simply extracts the nuances of each near-synonym in a sentence and puts them together. The complex interaction between the nuances of different words is not modeled, because it is a difficult problem and it is not needed for the purpose of these experiments. In fact, I limit the evaluation to sentences containing only one near-synonym.

An example of a set of test cases is presented in Figure 7.15. The first argument of the Lisp function is the near-synonym whose nuances are the input preferences, and the second argument is a simple interlingual representation. The output produced by Xenon for the first

```
NUANCES
((DISFAVOUR :AGENT) (DENOTE (C1 / PERSON :AGENT_OF SEEKS)))

(OR (/ (:CAT NN :LEX "opponent") :WEIGHT 1.0e-30)
    (/ (:CAT NN :LEX "adversary") :WEIGHT 9.4352764e-6)
    (/ (:CAT NN :LEX "antagonist") :WEIGHT 9.019014e-7)
    (/ (:CAT NN :LEX "competitor") :WEIGHT 1.0)
    (/ (:CAT NN :LEX "enemy") :WEIGHT 9.019014e-7)
    (/ (:CAT NN :LEX "foe") :WEIGHT 9.019014e-7)
    (/ (:CAT NN :LEX "rival") :WEIGHT 9.4352764e-6))
Output sentences in file: /h/34/dianaz/my_nlg/xxx0.sen
He is my competitor.    -23.9105
```

Figure 7.16: Example of output for the first test case from Figure 7.15.

English: benefit, advantage, favor, gain, profit
POS: noun

English:flow, gush, pour, run, spout, spurt, squirt, stream
POS: verb

English: deficient, inadequate, poor, unsatisfactory
POS: adj

English: afraid, aghast, alarmed, anxious, apprehensive, fearful, frightened, scared, terror-stricken
POS: adj

English: disapproval, animadversion, aspersion, blame, criticism, reprehension
POS: noun

Figure 7.17: Near-synonyms used in the evaluation of Xenon (DevSimple).

English: mistake, blooper, blunder, boner, contretemps, error, faux pas, goof, slip, solecism
French: erreur, égarement, illusion, aberration, malentendu, mécompte, bévue, bêtise, blague, gaffe, boulette, brioche, maldonne, sophisme, lapsus, méprise, bourde
POS: noun

English: alcoholic, boozer, drunk, drunkard, lush, sot
French: ivrogne, alcoolique, intempérant, dipsomane, poivrot, pochard, sac à vin, soûlard, soûlographe, éthylique, boitout, imbriaque
POS: noun

English: leave, abandon, desert, forsake
French: abandonner, délaisser, déserter, lâcher, laisser tomber, planter là, plaquer, livrer, céder
POS: verb

English: opponent, adversary, antagonist, competitor, enemy, foe, rival
French: ennemi, adversaire, antagoniste, opposant, détracteur
POS: noun

English: thin, lean, scrawny, skinny, slender, slim, spare, svelte, willowy, wiry
French: mince, élancé, svelte, flandrin, grêle, fluet, effilé, fuselé, pincé
POS: adj

English: lie, falsehood, fib, prevarication, rationalization, untruth
French: mensonge, menterie, contrevérité, hâblerie, vanterie, fanfaronnade, craque, bourrage de crâne
POS: noun

Figure 7.18: Near-synonyms used in the evaluation of Xenon (TestSimple, TestSample, and TestAll).

test case from Figure 7.15, where the input nuances are for the word *competitor*, is shown in Figure 7.16. The nuances of *competitor* are extracted, the meta-concept `generic_opponent_n` is expanded into an OR of the near-synonyms in the cluster and the degree of satisfaction of preferences is computed for each near-synonym. The near-synonym *competitor* obtained the highest weight. In the end, a number of sentences are generated; the last line of the example shows the sentences that scored best. The expected near-synonym was chosen in this case.

In DevSimple, I used 32 near-synonyms that are members of the 5 clusters presented in Figure 7.17. The set DevSimple was used as a development set in order to choose the exponent k of the function that translated the scale of the weights (equation 7.2 on page 93). As the value of k increased (staring at 1) the accuracy on the development set increased. The final value chosen for k was 15. In TestSimple, I used 43 near-synonyms selected from 6 other clusters, namely the English near-synonyms from Figure 7.18. TestSimple was used only for testing, not for development.

The shape of the function $f(x)$ (equation 7.2 on page 93) was decided by looking at one example of differences between weights. A linear function $f(x) = x \cdot 10^{-k}$ was tried, but it did not produce good results on the development set because it kept the differences between weights as small as they were before applying the function. Several exponential functions could be tried. Only one function was tried ($f(x) = \frac{e^{x^k}}{e-1}$) and kept because it performed well on the development set. The example that shows why the differences between weights need to be increased was the near-synonyms *lie, falsehood, fib, prevarication, rationalization, untruth*. The nuances of *untruth* were fed as input preferences. But the near-synonym chosen in Xenon's output was *lie*:

```
The boy told a lie.    -28.1873
The boy told a falsehood.    -30.3267
The boy told a rationalization.    -31.2279
The boy told a fib.    -41.9603
The boy told an untruth.    -41.9603
The boy told a prevarication.    -41.9603
```

This happened because *lie* has the highest probability in the language model. The loga-

```
lnP( lie ) = -11.0292                   lnP( lie | a ) = -9.50365
lnP( falsehood ) = -14.7825             lnP( falsehood | a ) = -12.7358
lnP( fib ) = -19.7797                   lnP( fib | a ) = -22.5867
lnP( prevarication ) = -19.7797         lnP( prevarication | a ) = -22.5867
lnP( rationalization ) = -14.024        lnP( rationalization | a ) = -12.2762
lnP( untruth ) = -19.7797               lnP( untruth | an ) = -22.5867
```

Table 7.2: Example of values for logarithms of probabilities in the language model (unigrams and bigrams).

rithms of the unigram and bigram probabilities are presented in Table 7.2.

The weights assigned by the preference satisfaction formulas are:

```
(A9 / tell
   :agent (V9 / boy)
   :object (O9 / (*OR* (e1 / lie :WEIGHT 0.2857143)
      (e2 / falsehood :WEIGHT 0.2857143)
      (e3 / fib :WEIGHT 0.2857143)
      (e4 / prevarication :WEIGHT 0.5714286)
      (e5 / rationalization :WEIGHT 0.2857143)
      (e6 / untruth :WEIGHT 1.0))))
```

In contrast with the log-probabilities from the language model, the scale difference between these weights is much smaller:

```
ln( 0.2857143) = -1.2528
ln( 0.5714286) = -0.5596
ln( 1.0)       = 0
```

HALogen's statistical ranker adds the language model log-probabilities and the logarithms of the weights together; therefore the differences between the logarithms of the weights need to be between -2 and -14 in this case before they would make any difference. In other words, the difference between the raw weights needs to be roughly between e^{-2} and e^{-14}, or between 0.1353 and 0.00000083. The exact value of the exponent k is not chosen to work well for this example; it is chosen on the basis of Xenon's performance on the whole development set.

In order to measure the baseline (the performance that can be achieved without using the LKB of NS), I ran Xenon on all the test cases, but this time with no input preferences. Some

Experiment	Number of cases	Correct by default	Correct	Ties	Baseline %	Accuracy (no ties) %	Accuracy %
DevSimple Simple sentences (development set)	32	5	27	5(4)	15.6	84.3	**96.4**
TestSimple Simple sentences (test set)	43	6	35	10(5)	13.9	81.3	**92.1**

Table 7.3: Xenon evaluation experiments for simple input.

of the choices could be correct solely because the expected near-synonym happens to be the default one (the one with the highest probability in the language model).

The results of the first type of evaluation experiment are presented in Table 7.3. For each test, the second column shows the number of test cases. The column named "Correct" shows the number of answers considered correct. The column named "Ties" shows the number of cases of ties, that is, cases when the expected near-synonym had weight 1.0, but there were other near-synonyms that also had weight 1.0, because they happen to have the same nuances in the LKB of NS. The same column shows in brackets how many of this ties caused an incorrect near-synonym choice. In such cases, Xenon cannot be expected to make the correct choice. More precisely, the other choices are equally correct, at least as far as Xenon's LKB is concerned. Therefore, the accuracies computed without considering these cases (the seventh column) are underestimates of the real accuracy of Xenon. The last column presents accuracies while taking the ties into account, defined as the number of correct answers divided by the difference between the total number of cases and the number of incorrectly resolved ties.

The second type of experiment (TestSample and TestAll) is based on machine translation. These experiments measure how successful the translation of near-synonyms from French into English and from English into English is. The machine translation experiments were done on French and English sentences that are translations of each other, extracted from parallel text (text aligned at the sentence level). An example of such sentences is shown in Figure 7.19.

⟨en⟩ Even granting that those things happened and without admitting anything for the sake of argument, let me say with respect to **mistakes** that two wrongs do not make a right. ⟨/en⟩
⟨fr⟩ En admettant que ces choses se soient produites et sans rien ajouter pour les seules fins de l'argument, je tiens à dire, au sujet des **erreurs**, qu'on ne guérit pas le mal par le mal. ⟨/fr⟩

⟨en⟩ Canada must send a clear message to international governments to **abandon** such atrocities if they wish to carry on co-operative relations with our country. ⟨/en⟩
⟨fr⟩ Le Canada doit faire savoir clairement aux gouvernements étrangers qu'ils doivent **abandonner** de telles pratiques atroces s'ils veulent entretenir des relations de coopération avec notre pays. ⟨/fr⟩

⟨en⟩ Populism is the natural **enemy** of representative democracy. ⟨/en⟩
⟨fr⟩ Le populisme est l'**ennemi** naturel de la démocratie représentative. ⟨/fr⟩

Figure 7.19: Examples of parallel sentences used in TestAll, extracted from the Canadian Hansard.

Xenon should generate an English sentence that contains an English near-synonym that best matches the nuances of the French near-synonym used in the corresponding French sentence. If Xenon chooses exactly the English near-synonym used in the parallel text, this means that Xenon's behaviour was correct. This is a conservative evaluation measure, because there are cases in which more than one translation is correct.

As illustrated in Figure 6.1 from page 78 in Section 6.1, an analysis module is needed. For the evaluation experiments, a simplified analysis module is sufficient. The French–English translation experiments take French sentences (that contain near-synonyms of interest) and their equivalent English translations. Because the French and English sentences are translations of each other, I can assume that the interlingual representation is the same for both. For the purpose of these experiments, it is acceptable to use the interlingual representation for the English sentence to approximate the interlingual representation for the French sentence. This is a simplification because there may be some sentences for which the interlingual representation of the French sentence is different, due to translation divergencies between languages [Dorr, 1993]. For the sentences in my test cases a quick manual inspection shows that this happens very rarely or not at all. The advantage of this simplification is that it eliminates the

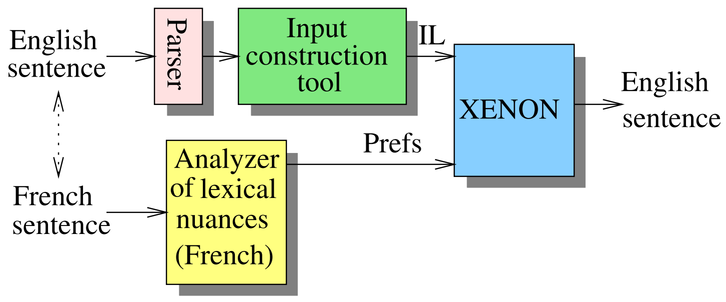

Figure 7.20: The architecture of TestSampleFr and TestAllFr (French-to-English).

need to parse the French sentence and the need to build a tool to extract its semantics. As depicted in Figure 7.20, the interlingual representation is produced using an existing tool that was previously used by Langkilde-Geary [2002a] in HALogen's evaluation experiments[5]. In order to use this input construction tool, I parsed the English sentences with Charniak's parser[6] [Charniak, 2000]. The tool was designed to work on parse trees from the PennTreebank, that have some extra annotations; it works on parse trees produced by Charniak's parser, but it fails on some parse trees probably more often than it did in HALogen's evaluation experiments. I replaced the near-synonym with the meta-concept that is the core meaning of its cluster. The interlingual representation for the English sentence is semantically shallow; it does not reflect the meaning of the French sentence perfectly, but in these experiments I am interested only in the near-synonyms from the test set; therefore, the other words in the French sentence are not important.

The analyzer of French nuances from Figure 7.20 needs to extract nuances from an LKB of French synonyms. I created by hand an LKB for six clusters of French near-synonyms (those from Figure 7.18), from two paper dictionaries of French synonyms, [Bénac, 1956] and [Bailly, 1973]. For each peripheral string, in French, an equivalent concept is found in Sensus by looking for English translations of the words and then finding Sensus concepts for the appropriate senses of the English words. Figure 7.21 presents a fragment of a cluster of French

[5]Thanks to Irene Langkilde-Geary for making the input construction tool available.
[6]ftp://ftp.cs.brown.edu/pub/nlparser/

```
(defcluster generic_erreur_n
    :syns (erreur egarement illusion aberration malentendu mecompte bevue
           betise blague gaffe boulette brioche maldonne sophisme lapsus
           meprise bourde)
:periph ((P1 (c1 / |take amiss| :object thing))
          ...
         (P7 (c7 / |glaring,gross|))
         (P8 (c8 / |view<belief| :mod |false>untrue|))
          ... )
:distinctions ( ...
               (meprise usually medium Denotation P1)
               ; "prend une chose pour une autre"
               (gaffe usually medium Denotation P7)
               ; "bevue grossiere"
               (erreur usually medium Denotation P8)
               ; "fausse opinion"
              )
)
```

Figure 7.21: Fragment of a cluster of French near-synonyms.

near-synonyms (see Appendix D for all the details). For example, if we are told that *erreur* denotes *fausse opinion*, the equivalent peripheral concept is `(P8 (c8 / |view<belief| :mod |false>untrue|))`. If we are told that *gaffe* denotes *bêvue grossiere*, then the equivalent peripheral concept is `(P7 (c7 / |glaring,gross|))`.

A similar experiment, translating not from French into English but from English into English, is useful for evaluation purposes. An English sentence containing a near-synonym is processed to obtain its semantic representation (where the near-synonym is replaced with a meta-concept), and the lexical nuances of the near-synonym are fed as preferences to Xenon. Ideally, the same near-synonym as in the original sentence would be chosen by Xenon. The percentage of times this happens is an evaluation measure. The architecture of these experiments is presented in Figure 7.22.

TestSample and TestAll consist of real English and French sentences, translations of each

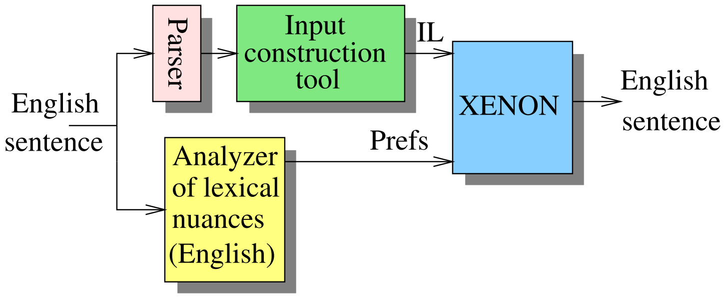

Figure 7.22: The architecture of TestSampleEn and TestAllEn (English-to-English).

other, extracted from the Canadian Hansard[7] (approximately one million pairs of sentences). Other sources of parallel text, such as parallel translations of the bible[8] [Resnik, 1999a], and a collection of webpages [Resnik et al., 1997], happened to contain very few occurrences of the near-synonyms of interest.

TestSample contains pairs of sentences, selected such that there are at most two sentences for each pair of a French and English near-synonym that are translations of each other. The sentences selected for each pair are the first two for which the input construction tool succeeded to produce a valid interlingual representation. It happens that not all the near-synonyms in the test set are found in the Hansard. In fact, only 14 distinct pairs of French and English near-synonyms from the test set occur as translations of each other. Some of these pairs are very frequent, and some are very rare. TestAll is similar to TestSample, but instead of having two sentences for a near-synonym, it contains all the sentences in a given fragment of Hansard (about half of it) in which the near-synonyms of interest occurred. Therefore, TestAll has the advantage of containing naturally occurring text, not artificially selected sentences. It has the disadvantage of lacking those near-synonyms in the test set that are less frequent. Initially there were 564 pairs of sentences, but the input construction tool worked successfully only on 298 English sentences.

The English and French near-synonyms used in TestSample and TestAll are the ones pre-

[7]http://www.isi.edu/natural-language/download/hansard/
[8]http://benjamin.umd.edu/parallel/

```
(run_en '(It will not correct mistakes and rectify the problems)
'(H18 :MOOD INDICATIVE
     :TENSE MODAL
     :LOGICAL-SUBJECT (H2 :CAT NN
                          :CAT1 PRONOUN
                          :NUMBER SINGULAR
                          :LEX "it")
     :MODAL WILL
     :POLARITY (H4 :CAT RB
                   :CAT1 NEGATIVE
                   :LEX "not")
     / (H15 / (H8 :VOICE ACTIVE
                  / (H5 :CAT VV
                        :CAT1 INFINITIVE
                        :LEX "correct")
                  :LOGICAL-OBJECT (H7 / generic_mistake_n :DETERMINER NONE
                                      :CAT NN
                                      :CAT1 COMMON
                                      :NUMBER PLURAL))
            :CONJ (H9 :CAT CC
                      :LEX "and")
            / (H14 :VOICE ACTIVE
                   / (H10 :CAT VV
                          :CAT1 INFINITIVE
                          :LEX "rectify")
                   :LOGICAL-OBJECT (H13 :DETERMINER (H11 :CAT DT
                                                         :LEX "the")
                                        / (H12 :DETERMINER NONE
                                               :CAT NN
                                               :CAT1 COMMON
                                               :NUMBER PLURAL
                                               :LEX "problem"))))
     :PUNC PERIOD)
)
```

Figure 7.23: Example of test case from TestAll (English input).

Experiment	Number of cases	Correct by default	Correct	Ties	Baseline %	Accuracy (no ties) %	Accuracy %
TestSampleFr French-to-English (test set)	26	10	13	5(3)	38.4	50.0	**56.5**
TestSampleEn English-to-English (test set)	26	10	26	2(0)	38.4	100	**100.0**
TestAllFr French-to-English (test set)	298	214	217	7(1)	71.8	72.8	**73.0**
TestAllEn English-to-English (test set)	298	214	296	2(0)	71.8	99.3	**99.3**

Table 7.4: Xenon's machine translation evaluation experiments and their results.

sented in Figure 7.18 from page 109. Figure 7.23 presents an example of a test case used in TestAll, for the English sentence *It will not correct mistakes and rectify the problems*. The second argument of the function is the interlingual representation produced by the input construction tool, where the word *mistake* was replaced with the meta-concept `generic_mistake_n`. The interlingual representations used in TestSample and TestAll are quite complex; the one in Figure 7.23 was chosen as an example because it was shorter than others.

The results of the machine translation evaluation experiments are presented in Table 7.4. Similarly with the Table 7.3 on page 112, the second column shows the number of test cases; the third column shows the number of answers considered correct; the fourth column shows the number of cases of ties (cases when more than one near-synonym had weight 1.0), and in the brackets it shows how many of this ties caused incorrect choice; the fifth column shows the baseline accuracy (how many test cases can be solved by HALogen only, without any input nuances to Xenon); the sixth column shows the accuracy computed as number of correct choice dived by the total number of cases; the seventh column presents accuracies while not penalizing incorrectly solved ties (defined as the number of correct answers divided by the difference between the total number of cases and the number of incorrectly resolved ties).

Table 7.4 also includes the results for the baseline experiments. In general, baseline accuracies are much lower than Xenon's performance. For example, for DevSimple, Xenon achieves 96.8% accuracy, while the baseline is 15.6%. An exception is the baseline for TestAll, which is high (71.8%), due to the way the test data was collected: it contains sentences with frequent near-synonyms, which happen to be the ones Xenon chooses by default in the absence of input preferences. For TestSample and TestAll the baseline is the same for the French and English experiments because no nuances were used as input for the baseline experiments.

The experiments summarized in Table 7.4 use Xenon with the automatically produced LKB of NS. If the LKB of NS were perfect, Xenon would be expected to perform better. I ran the same experiments on Xenon with a slightly better LKB of NS, produced by using hand-built solutions for the extraction phase (Section 2.3) but still using the automatic programs for the customization phase (Section 6.2.2). The results improved only slightly or not at all.

Analysis of the near-synonym choice evaluation results

Tables 7.3 on page 112 and 7.4 on page 118 show that Xenon's performance is better than the baseline (no input nuances, HALogen's defaults caused by the language model). The baselines are low, with the exception of the baseline for TestAll, which is high (71.8%), due to the way the test data was collected: it contains sentences with frequent near-synonyms, which happen to be the ones Xenon chooses by default in the absence of input preferences. For TestSample and TestAll the baseline is the same for the French and English experiments because no nuances were used as input for the baseline experiments.

In general, Xenon's accuracy is much higher that the baseline accuracy (except for TestAllFr). For DevSimple, Xenon's accuracy is 80.8 percentage points higher than the baseline. For TestSimple the improvement is 78.2 percentage points. Therefore the performance on the test sets is comparable with the performance on the development set. The improvement in accuracy is higher in the test cases with sample sentences than in the cases with all sentences from a fragment of text. This is true for both the English-to-English and the French-to-English experiments. If we look at the English-to-English cases first, for TestSampleEn the improvement over the baseline is 61.6 percentage points, while for TestAllEn the improvement is 27.5 percentage points. This happens because the tests with all sentences in a fragment tend to contain only frequent near-synonyms, most of which can be chosen by default in HALogen. The sample sentences also include less frequent near-synonyms that cannot be chosen by HALogen. Xenon successfully choses them on the basis of their nuances.

There are two reasons to expect Xenon's accuracy to be less then 100% in the English-to-English experiments, even if the input nuances are the nuances of a particular English near-synonym. The first reason is that there are cases in which two or more near-synonyms get an equal, maximal score because they do not have nuances that differentiate them (either they are perfectly interchangeable, or the LKB of NS does not contain enough knowledge) and the one chosen is not the expected one. The second reason is that sometimes Xenon does not choose the expected near-synonym even if it is the only one with maximal weight. This may happen because HALogen makes the final choice by combining the weight received from the near-synonym choice module with the probabilities from the language model that is part of HALogen. Frequent words may have high probabilities in the language model. If the expected near-synonym is very rare, or maybe was not seen at all by the language model, its probability is very low. When combining the weights with the probabilities, a frequent near-synonym may win even if it has a lower weight assigned by the near-synonym choice module. In such cases, the default near-synonym (the most frequent of the cluster) wins. Sometimes such behaviour is justified, because there may be other constraints that influence HALogen's choice.

In the French-to-English experiments the performance of Xenon is lower than in the English-to-English experiments. There are two explanations. First, there is some overlap between the nuances of the French and the English near-synonyms, but less than one would expect. For example, the English adjective *alcoholic* is close in nuances to the French adjective *alcoolique*, but they have no nuance in common in Xenon's knowledge-bases. In this case, the lack of overlap is due to the incompleteness of the explanations given by lexicographers in the dictionary

Experiment	Number of cases without correct defaults and ties	Correct non-defaults	Accuracy for non-defaults (%)
DevSimple	32 − 5 − 4 = 23	27 − 5 = 22	**95.6**
TestSimple	43 − 6 − 5 = 32	35 − 6 = 27	**84.3**
TestSampleFr	26 − 10 − 3 = 13	13 − 10 = 3	**23.0**
TestSampleEn	26 − 10 − 0 = 16	26 − 10 = 16	**100.0**
TestAllFr	298 − 214 − 1 = 83	217 − 214 = 3	**3.6**
TestAllEn	298 − 214 − 0 = 84	296 − 214 = 82	**97.6**

Table 7.5: Xenon's accuracy for "non-default" cases.

entries. There could also be cases when there are similar nuances, but they were not correctly extracted when the LKB of English near-synonyms was acquired.

The second explanation is related to what is considered the "correct" choice of near-synonym. Sometimes more than one translation of a French near-synonym could be correct, but in this conservative evaluation, the solution is the near-synonym used in the equivalent English sentence. Therefore, test cases that would be considered correct by a human judge are harshly penalized. Moreover, the near-synonym choice module always chooses the same translation for a near-synonym, even if the near-synonym is translated in Hansard in more than one way, because the context of the near-synonym in the sentence is not considered. (The context is taken into account only when the collocation module is enabled and a preferred collocation is detected in the sentences.) For example, the French noun *erreur* is translated in Hansard sometimes with *error*, sometimes with *mistake*. Both have nuances in common with *erreur*, but *mistake* happened to have higher overlap with *erreur* than *error*; as a result, the near-synonym choice module always chooses *mistake* (except the case when the collocation module is enabled and finds a preferred collocation such as *administrative error*). All the cases when *error* was used as translation of *erreur* in Hansard are penalized as incorrect in the evaluation of the near-synonym choice module. A few of these cases could be indeed incorrect, but probably some of them would be considered correct by a human judge.

Another way to look at the performance of Xenon is to measure how many times it makes appropriate choices that cannot be made by HALogen, that is, cases that make good use of the nuances from the LKB of NS. This excludes the test cases with default near-synonyms, in other words the cases when Xenon makes the right choice due to the language model. It also excludes the cases of ties when Xenon cannot make the expected choice. Table 7.5 shows accuracies for these "non-default" cases. For the experiments with only English near-synonyms, Xenon is performing very well, managing to make correct choices that cannot be made by default. Accuracies vary from 84.3% to 100%. For the experiments involving both French and English experiments, Xenon makes only a few correct choices that cannot be made by default. This is due to the fact that most of the overlap in nuances between French and English synonyms happens for the near-synonyms that are defaults.

7.7.2 Evaluation of the near-synonym collocation module

For the evaluation of the near-synonym collocation module, I collected sentences from the BNC that contain preferred collocations from the knowledge-base of near-synonym collocational behavior. The BNC was preferred over the Hansard for these evaluation experiments because it is a balanced corpus and contains the collocations of interests, while the Hansard does not contain some of the collocations and near-synonyms of interest. The sentences were actually collected from the first half of the BNC (50 million words). The test sets CDevSample and CDevAll contain collocations for the near-synonyms in Figure 7.17. CTestSample and CTestAll contain collocations for the English near-synonyms in Figure 7.18. CDevSample and CTestSample include at most two sentences per collocation (the first two sentence from the corpus, except the cases when the input construction tool failed to produce valid interlingual representations), while CDevAll and CTestAll include all the sentences with collocations as they occurred in the fragment of the corpus (frequent collocations can occur more than once), except the sentences for which the input construction tool failed. For example, for CDevAll there were initially 527 sentences, and the input construction tool succeeded on 297 of them. CDevSample was used for development, and CDevAll, CTestSample, and CTestAll only for

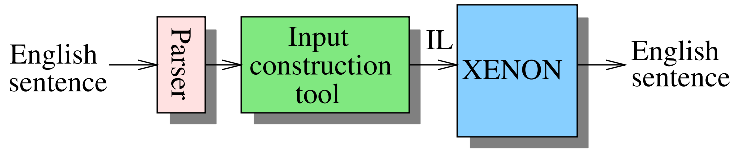

Figure 7.24: The architecture of tests for evaluating the near-synonym collocation module.

testing (CDevAll was not used for development, but its name reflects the fact that it contains collocations for the same near-synonyms as the development set CDevSample).

The sentences were parsed and the input construction tool was used to produce interlingual representations. The near-synonyms in the interlingual representations were replaced with meta-concepts. Then the interlingual representations were fed into Xenon. No input preferences were used in this experiment, therefore disabling the near-synonym choice module. The architecture of these tests in presented in Figure 7.24.

I mention that the test sets may contain collocations for the wrong senses of some near-synonyms, because, as explained in Section 4.3, the near-synonym collocations knowledge-base may contain, for a cluster, collocations for a different sense. For example, the collocation *trains run* appears in the cluster *flow, gush, pour, run, spout, spurt, squirt, stream*, when it should appear only in another cluster. In this case the near-synonym *run* should not be replaced with the meta-concept "generic_flow_v" because it corresponds to a different meta-concept. The sentences should be eliminated from the test set, but this would involve disambiguation or manual elimination. Therefore such sentences are still in the test sets. They do not affect the evaluation results because they are unlikely to produce anti-collocations. This is because *trains run* is a frequent bigram, while *trains flow* is not; Xenon will make the correct choice by default.

The development set was used to choose the best values of the discount weight W_{anti_colloc} and $W_{less\_pref\_colloc}$. In fact $W_{less\_pref\_colloc}$ could be approximated by W_{anti_colloc}, treating less-preferred collocations as anti-collocations, because the number of less-preferred collocations is very small in the knowledge-base. Also, less-preferred collocations were almost nonexistent

		HALogen only			HALogen + collocs module		
Experiment	No. of cases	Correct NS choice baseline	Preferred collocs baseline	Anti-collocs baseline	Correct NS choice	Preferred collocs	Anti-collocs
CDevSample (dev. set)	105	66 (62%)	92 (88%)	13 (12%)	74 (70%)	105 (100%)	0 (**0%**)
CDevAll (test set)	297	247 (83%)	271 (91%)	24 (9%)	260 (87%)	293 (99%)	4 (**1%**)
CTestSample (test set)	44	26 (59%)	35 (80%)	9 (20%)	33 (75%)	44 (100%)	0 (**0%**)
CTestAll (test set)	185	108 (58%)	126 (68%)	59 (32%)	160 (86%)	183 (99%)	2 (**1%**)

Table 7.6: The results of the evaluation of Xenon's collocations module (the lexical nuances module is disabled for this experiment).

in the test cases. There was only two less-preferred collocation that occurred in CDevAll, and they disappeared when increasing the discount weight $W_{less\_pref\_colloc}$ to 0.5. As the value of the discount weight W_{anti_colloc} increased (between 0.0 and 1.0), the number of anti-collocations generated decreased; there were no anti-collocations left for $W_{anti\_colloc} = 0.995$.

Table 7.6 presents the results of the evaluation experiments. These results refer to the evaluation of Xenon with the near-synonym collocations module enabled and the near-synonym choice module disabled (lexical nuances are ignored in this experiment). The baseline used for comparison is obtained by running HALogen only without any extension modules (no knowledge of collocations). For each test, the first four columns contain: the number of test cases, the number of near-synonyms correctly chosen by the baseline system (HALogen only), the number of preferred collocations, and the number of anti-collocations produced by the baseline system. The rest of the columns present results obtained by running Xenon with only the near-synonym collocations module enabled (that is HALogen and the collocations module): the number of near-synonyms correctly chosen, the number of preferred collocations produced, and number of anti-collocations produced. The number of anti-collocations was successfully reduced to zero for all the tests (except for CDevAll and CTestAll where 1% of the anti-collocations remained). The sixth column (correct choices or accuracy) differ from the seventh column (preferred collocations) in the following way: the correct choice is the near-synonym used in the original BNC sentence; sometimes the generated sentence can choose a different near-synonym that is not the expected one but that participate in a preferred collocation (this happens when more than one near-synonym from the same cluster collocates well with the collocate word). For example, both *serious mistake* and *serious blunder* are preferred collocations, while only one of the near-synonyms *mistake* and *blunder* is the correct choice in a particular context. The number of correct choices is relevant in this experiment only to show that the collocations module does not have a negative effect on correctness; it even increases the accuracy[9].

Analysis of the collocation module evaluation results

The results in Table 7.6 show that the anti-collocations are almost entirely eliminated. The number of preferred collocations is 100% minus the number of remaining anti-collocations. There could be a few less-preferred collocations, but they were almost nonexistent in the test sets. The test sets consist of input sentences that contained preferred collocations before the near-synonyms wewre replaced by meta-concpets. Without the collocation module, Xenon manages to re-generate sentences that contain only from 9% to 30% anti-collocations (the columns called anti-collocations baseline in the table). This happens because HALogen's language model helps choosing near-synonyms that produce preferred collocations in the rest of the cases. CTestAll and CDevALL produce more anti-collocations (in the baseline case) than CTestSample and CDevSample because the former test sets tend to contain frequent collocates, and if HALogen generates anti-collocations for some of them, this may happen several times. In other words, the same mistake may be penalized multiple times. This does not happen in the sample test cases, because the same anti-collocation could appear a maximum of two times.

The test set CDevSample was the only set used for development. The reduction in anti-collocation is the same on the rest of the test sets. CDevAll was used for testing not develop-

[9]The accuracy without ties was used here; therefore the numbers are conservative.

ment, but it contains the same set of near-synonyms as CDevSample, while CTestSample and CTestAll contains a different set of near-synonyms.

The collocations module ensures that anti-collocations are not generated. The near-synonym choice module tries to choose the best near-synonyms that matches the input preferences. By enabling both modules, Xenon chooses the best near-synonyms on the basis of both their lexical nuances and their collocational properties.

7.7.3 Evaluation of the two modules in interaction

The interaction between the near-synonym collocations module and the synonym choice module (called here nuances module) does not have a negative effect on Xenon's performance; it even increases it. To prove this, I repeated the experiments from the previous section, but this time with input preferences (the nuances of the near-synonym from the original sentence). The architecture of this test is same with the one for the English-to-English experiments from Section 7.7.1, depicted in Figure 7.22. Table 7.7 shows the number of correct near-synonym choices (and the percent accuracy in brackets) for the baseline case (no nuances, no collocation module; that is HALogen by itself), for the collocations module alone (that is HALogen and the collocations module only; this column is also part of Table 7.6), for the synonym choice module alone (that is HALogen and the nuances module only), and for Xenon with both modules enabled. When both modules are enabled there is a slight increase in accuracy for CDevAll, CTestSample, and CTestAll; the accuracy on CDevSample is the same as using the near-synonyms module only.

All the evaluation experiments presented in this chapter were run by using HALogen with the trigram language model in its statistical ranker. The same experiments were repeated for the bigram model, and the results were almost identical.

		HALogen	Xenon		
Experiment	Number of cases	Correct NS baseline	Correct NS collocations module	Correct NS nuances module	Correct NS nuances module + collocs module
CDevSample	105	66 (62%)	74 (70%)	98 (93%)	98 (**93%**)
CDevAll	297	247 (83%)	260 (87%)	282 (95%)	283 (**95%**)
CTestSample	44	26 (59%)	33 (75%)	41 (93%)	42 (**95%**)
CTestAll	185	108 (58%)	160 (86%)	169 (91%)	177 (**95%**)

Table 7.7: Correct near-synonym choices for the baseline system (HALogen only), for HALogen with each module of Xenon separately, and for HALogen with both modules of Xenon.

7.8 Summary

This chapter presented Xenon, an NLG system capable of choosing the best near-synonym that satisfies a set of input preferences (lexical nuances). The input preferences could come from an analysis module, for a different language; in this case the translation into English would preserve not only the meaning but also nuances of meaning.

Xenon uses HALogen for surface realization. A near-synonym choice module computes, for each near-synonym, a weight that reflects how well it satisfies the input preferences. HALogen makes the final choice, taking into account these weights as well as other generation constraints. The algorithm for computing preference satisfaction scores is adapted from I-Saurus. The similarity of conceptual configurations is new, because the representation language is different.

A module that uses the knowledge of collocation behaviour of the near-synonyms was integrated into Xenon. It was inserted inside HALogen: after the symbolic generator produces a forest representation that encodes all the possible sentences, the collocations module changes the forest by discounting the weight of possible anti-collocations (detected using tree-like operations); then the forest is passed to HALogen's statistical ranker that produces the sentences with highest scores.

Extensive evaluation experiments were described. Test sentences were collected, parsed, and analyzed with an input construction tool (for English). In the interlingual representations produced in this way, a near-synonym was replaced with its meta-concept.

In order to evaluate the near-synonym choice module, a small LKB of French near-synonyms was manually produced. Then the nuances of the French near-synonyms were used as input preferences. The evaluation was done on French and English sentences that are translations of each other. This had two advantages: the interlingual representation could be extracted from the English sentence instead of the French sentence; and the expected solution is known, the English near-synonym that was used in the English translation. This evaluation is sometimes too strict: it penalizes other possibly correct choices. Experimental results for English-to-English translation are also reported.

Evaluation experiments also targeted the near-synonym collocation module. In this case the test sets contained English sentences with preferred collocations. The near-synonyms were again replaced with meta-concepts in the semantic representations. The results were considered correct if no anti-collocations were generated. A development set was used to determine how much to discount near-synonyms that could generate anti-collocations, and a test set was used to make sure the discount weight works in general.

The evaluation of Xenon's two new modules shows that they behave well, independently and in interaction.

Future work can investigate the possibility to iteratively adapt the weights assigned by each module in order to maximize the benefit of their interaction. Another possibility for future work is to not treat all the preferred collocations as equally good, but to have a finer-grained classification of collocations. For example, if *drink alcohol* and *open a book* are both preferred collocations, the first one could be considered stronger if *drink* associates with fewer types of nouns than *open* (selectional restrictions).

Chapter 8

Conclusion

In the research that this dissertation presents, I have shown that it is possible to automatically acquire knowledge of the differences between near-synonyms. I built a lexical knowledge-base of differences between English near-synonyms;this new lexical resource can be used in natural language processing applications such as machine translation, natural language generation, and writing assistance programs. I have shown how it can be used to choose the best near-synonyms in a natural language generation system.

8.1 Summary of contributions

The contributions of this research have been presented in Section 1.5. I summarize them here, adding emphasis on evaluation results.

Extraction patterns and knowledge acquisition I developed a method for extracting knowledge from special dictionaries of near-synonym discrimination. The method can potentially be applied to any dictionary of near-synonym discrimination, for any language for which preprocessing tools, such as part-of-speech taggers and parsers, are available.

I built a new lexical resource, an LKB of differences between English near-synonyms, by applying the extraction method to *Choose the Right Word*. The precision and recall of the extracted knowledge was estimated to be in the range 70–80%. If higher precision and recall are needed for particular applications, a human could validate each extraction step.

Sense disambiguation I applied a combination of existing word sense disambiguation techniques to a new task: disambiguating the senses of the near-synonyms in a dictionary entry. The results of the evaluation were good. The problem is simpler that the general task of word sense disambiguation because the dictionary entry provides a strong context for disambiguation. To produce a standard solution for the evaluation, I collected data annotated by human judges.

Acquisition of collocational knowledge I automatically acquired knowledge of collocational behaviour of near-synonyms from free text. To estimate the correctness of the near-synonym collocation classifier (the three classes being preferred collocations, less-preferred collocations, and anti-collocations), I built a standard solution from a sample of data annotated by human judges. This knowledge is used in Xenon's lexical choice process to ensure that it chooses near-synonyms that generate preferred collocations, and avoids generating anti-collocations.

Knowledge extraction from MRDs I showed that knowledge of differences between near-synonyms can also be extracted from MRDs. I enriched the initial LKB of NS with distinctions extracted from MRDs.

Customization of the lexical knowledge-base of near-synonym differences I showed how the generic LKB of NS can be customized for use in a particular NLP system. The customization of peripheral concepts for Xenon was evaluated on a subset of hand-annotated data.

Utility of the lexical knowledge-base of near-synonym differences I presented Xenon, an NLG system that uses the LKB of NS to choose the near-synonym that best matches a set of input preferences. Xenon extends an existing NLG system with two new modules: a module that chooses near-synonyms on the basis of their lexical nuances, and a module that chooses near-synonyms on the basis of their collocations.

To evaluate Xenon, I manually built a small LKB of French near-synonyms. The test set consisted of English and French sentences that are translations of each other. An interlingual representation (where a near-synonym is replaced with the core denotation of its cluster) is fed into Xenon, together with the nuances of the near-synonym from the French sentence. The generated sentence is considered correct if the chosen English near-synonym is the one from the original English sentence. This evaluation is conservative, because it penalizes other possibly-correct translations. I also evaluated the near-synonym collocation module, and the interaction of the two modules.

8.2 Future work

8.2.1 Extensions

Short-term future work includes overcoming some limitations or extending the current work.

- Investigate the possibility that the extraction algorithm presented in Chapter 2 could compute confidence factors for each extracted distinction.

- Extend the near-synonym representation with other types of distinctions such as: information about more general or more specific words, and information about special meanings some word may have in particular contexts (or domains) (e.g. in legal context).

- Apply the extraction programs presented in Section 2.3, without modification, to the usage notes from an MRD such as *Merriam-Webster Online Dictionary*. The distinctions expressed in these usage notes are similar to the explanations from CTRW. These usage notes appear after the definitions of some words, to explain how they differ from their near-synonyms.

- Extend the consistency checking model (Section 5.4) to include denotational distinctions.
- Extend the transformation rules used in Section 6.2.2 for customizing the LKB of NS for Xenon, to achieve better coverage and correctness.
- Extend the near-synonym collocation module in Section 7.6 to detect all the potential collocations in a forest representation.
- Design a finer-grained classification of collocations, that does not treat all the preferred collocations as equally good, but distinguishes them on the basis of selectional restrictions and other factors.
- Investigate the possibility of iteratively adapting the weights assigned by each of the two new modules of Xenon (Chapter 7) in order to maximize the benefit of their interaction.
- Improve the lexical similarity measure implemented in Xenon.

8.2.2 Directions for future work

I envision several major directions for future work.

Analysis of lexical nuances

A fully-fledged analysis module can be developed. Sense disambiguation needs to be done when a near-synonym is a member of more then one cluster of near-synonyms, that is it could be a member of more than one CTRW entry. This problem is not too difficult, because the text of the CTRW entries provide a strong context for disambiguation. The paragraph or sentence to be analyzed can be intersected with the text of the candidate dictionary entries. Extensions of this Lesk-style approach, using *tf·idf* or semantic relatedness measures, could lead to accurate disambiguation. More difficult is to model the influence of the context (when a nuance is actually expressed in a context) and the complex interaction of the lexical nuances. The analysis module could be used in an MT system that preserves lexical nuances. The analysis module could be used to determine nuances of text for different purposes. For example, a system could decide if a text is positive, neutral, or negative in its semantic orientation. Then Xenon can be used to generate a new text, that has the same meaning as the original text, but a different semantic orientation. This could be useful, for example, in an application that sends letters to customers: if the initial text is too negative, it could transform it into a positive one before sending it to a customer.

Lexical and conceptual associations

The method presented in Chapter 4 can be extended to acquire lexical associations (longer-distance collocations) of near-synonyms. Words that strongly associate with the near-synonyms can be useful, especially words that associate with only one of the near-synonyms in the cluster. These strong associations could possibly provide knowledge about nuances of near-synonyms. An experiment similar to the one presented in Chapter 4 could look for words that co-occur in a window of size K, to acquire lexical associations, which would include the collocations extracted in Section 4.1. The method I used in Section 4.2 needs to be modified so that the query asked to AltaVista is: x NEAR y (where x and y are the words of interest).

Church et al. [1991] presented associations for the near-synonyms *ship* and *boat*. They suggest that a lexicographer looking at these associations can infer that a *boat* is generally smaller than a *ship*, because they are found in *rivers* and *lakes*, while the *ships* are found in *seas*. Also, *boats* are used for small jobs (e.g., *fishing*, *police*, *pleasure*), whereas *ships* are used for serious business (e.g., *cargo*, *war*). It could potentially be possible to automatically infer this kind of knowledge or to validate already acquired knowledge.

Words that do not associate with a near-synonym but associate with all the other near-synonyms in a cluster can tell us something about its nuances of meaning. For example *terrible* *slip* is an anti-association, while *terrible* associates with *mistake, blunder, error*. This is an indication that *slip* is a minor error. By further generalization, the associations could become conceptual associations. This may allow the automatic learning of denotational distinctions between near-synonyms from free text. The concepts that are common to all the near-synonyms in a cluster can be considered part of the core denotation, while the concepts that associate only with one near-synonym may be peripheral concepts in a denotational distinction.

Cross-lingual lexical nuances

The method presented in Chapter 2 can be used to automatically build a lexical knowledge-base of near-synonym differences for another language, for example for French. Dictionaries of synonyms discriminations are available (in paper form). Another resources, such as part-of-speech taggers and parsers are available. In order to use the French and the English knowledge-bases in the same system, a study the cross-lingual lexical nuances is needed.

Analysis of types of peripheral nuances

Linguists and lexicographers have looked at differences between particular types of near-synonyms. For example, Gao [2001] studied the semantic distinctions between Chinese physical action verbs; one type of distinctive peripheral nuance is the manner in which the movement is done for each verb. These kind of studies could help to develop a list of the main types of peripheral nuances (peripheral concepts). In my work, the form that the peripheral nuances can take is not restricted. This is necessary because the list of peripheral nuances is open-ended. It may be possible to keep the form unrestricted but add restrictions for the most important type of peripheral nuances.

Intelligent thesaurus

The acquired lexical knowledge-base of near-synonym differences can be used to develop an intelligent thesaurus, that assists a writer not only with a list of words that are similar to a given word, but also with explanations about the differences in nuances of meaning between the possible choices. The intelligent thesaurus could order the choices to suit a particular writing context. The knowledge about the collocational behaviour of near-synonyms can be used in determining the order: near-synonyms that produce anti-collocations would be ranked lower than near-synonyms that produce preferred collocations.

Automatic acquisition of near-synonyms

This work considered as near-synonyms the words given by the lexicographers in CTRW. Other dictionaries of synonym discrimination may have slightly different views. Merging clusters from different dictionaries is possible. Also, near-synonym clusters can be acquired from free text. This involves distinguishing near-synonyms from the pool of related words. As mentioned in Section 1.4.2, Lin et al. [2003] acquired words that are related by contextual similarity, and then filtered out the antonyms. Words that are related by relations other then near-synonymy could also be filtered out. One way to do this could be to collect signatures for each potential near-synonym, composed of words that associate with it in many contexts. For two candidate words, if one signature is contained in the other, the words are probably in a IS-A relation. If the signatures overlap totally, it is a true near-synonymy relation. If the signatures overlap partially, it is a different kind of relation. The acquisition of more near-synonyms, followed by the acquisition of more distinctions, is needed to increase the coverage of our lexical knowledge-base of near-synonym differences.

Appendix A

List of Abbreviations

CTRW	*Choose the Right Word* [Hayakawa, 1994].
DL	Decision List.
LKB	Lexical Knowledge-Base.
LKB of NS	Lexical Knowledge-Base of Near-Synonym Differences.
MRD	Machine-Readable Dictionary.
MT	Machine Translation.
NLG	Natural Language Generation.
NLP	Natural Language Processing.
WSD	Word Sense Disambiguation.

Appendix B

Example of Generic LKB of NS for the Near-Synonyms of *error*

Cluster: mistake=blooper=blunder=boner=contretemps=error=faux pas=goof=slip=solecism=

these nouns denote something done, said, or believed incorrectly or improperly

subj: these nouns
freq: usually
strength: medium
class: Denotation
periph: something/NN done/VBN ,/, said/VBD ,/, or/CC believed/VBD incorrectly/RB or/CC improperly/RB

near_syn(mistake) and near_syn(error) are the most common and general of this group

in many contexts they are interchangeable, but near_syn(error) often implies deviation from a standard or model, whereas near_syn(mistake) is preferred in the common situations of everyday life

subj: error
freq: usually
strength: medium
class: Implication
periph: deviation/NN from/IN a/DT standard/NN or/CC model/NN whereas/IN NS_mistake/NN is/VBZ preferred/VBN in/IN the/DT common/JJ situations/NNS of/IN everyday/JJ life/NN

near_syn(error) is also used in a theological sense to mean sin, since sin is perceived as deviation from the moral standards or theological truths established by religion

subj: error
freq: usually
strength: medium
class: Denotation
periph: sin/NN ,/, since/IN sin/NN is/VBZ perceived/VBN as/IN deviation/NN from/IN the/DT moral/JJ standards/NNS or/CC theological/JJ truths/NNS established/VBN by/IN religion/NN

a near_syn(blunder) is a blatant near_syn(error), usually one involving behavior or judgment, and implying an ignorant or uninformed assessment of a situation

subj: blunder
freq: usually
strength: medium
class: Implication
periph: an/DT ignorant/JJ or/CC uninformed/JJ assessment/NN of/IN a/DT situation/NN

near_syn(slip) and near_syn(faux pas) (literally, false step) are minor near_syn(mistakes)

near_syn(slip) emphasizes the accidental rather than ignorant character of a near_syn(mistake) and is often used to mean the careless divulging of secret or private information

subj: slip
freq: usually
strength: high
class: Denotation
periph: the/DT accidental/JJ rather/RB than/IN ignorant/JJ character/NN of/IN a/DT NS_mistake/NN

subj: slip
freq: usually
strength: medium
class: Denotation
periph: the/DT careless/JJ divulging/VBG of/IN secret/JJ or/CC private/JJ information/NN

a near_syn(faux pas) is an embarrassing breach of etiquette

a near_syn(solecism) denotes any near_syn(error), but especially a breach of good manners, grammar, or usage

subj: solecism
freq: usually
strength: medium
class: Denotation
periph: any/DT NS_error/NN ,/, but/CC especially/RB a/DT breach/NN of/IN good/JJ manners/NNS ,/, grammar/NN ,/, or/CC usage/NN

near_syn(blooper), near_syn(boner), and near_syn(goof) all have a somewhat humorous tone and a distinctively american flavor

americans apparently feel that it a near_syn(mistake) is outrageous enough it is funny, and this feeling is reflected in the variety of nouns available to describe such near_syn(mistakes)

near_syn(blooper), an informal term, is usually applied to particularly infelicitous mix-ups of speech, such as ”rit of fellus jage” for ”fit of jealous rage”

subj: blooper
freq: usually
strength: medium
class: Denotation
periph: to/TO particularly/RB infelicitous/JJ mix-ups/NNS of/IN speech/NN ,/, such/JJ as/IN rit/NN of/IN fellus/JJ jage/NN for/IN fit/NN of/IN jealous/JJ rage/NN

subj: blooper
freq: usually
strength: low
class: Formality

a near_syn(boner) is any egregious and mindless near_syn(mistake)

the slang noun near_syn(goof) denotes an indefensible near_syn(error) honestly admitted, which contains in its admission a covert plea that the near_syn(goof) be regarded with indulgent forgiveness

subj: goof
freq: usually
strength: medium
class: Denotation
periph: an/DT indefensible/JJ NS_error/NN honestly/RB admitted/VBN ,/, which/WDT contains/VBZ in/IN its/PRP$ admission/NN a/DT covert/JJ plea/NN that/IN the/DT NS_goof/NN be/VB regarded/VBN with/IN indulgent/JJ forgiveness/NN

subj: goof
freq: usually
strength: medium
class: Formality

a near_syn(contretemps), literally ”counter to or against the time,” that is, at the wrong time, refers to an embarrassing or awkward occurrence

subj: contretemps
freq: usually
strength: medium

class: Denotation
periph: an/DT embarrassing/JJ or/CC awkward/JJ occurrence/NN

the young woman who puts off an admirer by saying she is quite ill and then sees him that very night at a party learns the meaning of near_syn(contretemps) in a way she is not likely to forget

From MQ

subj: mistake
freq: usually
strength: medium
class: Denotation
periph: an/dt error/nn in/in action/nn opinion/nn or/cc judgment/nn

subj: blooper
freq: usually
strength: medium
class: Denotation
periph: a/dt slip/nn of/in the/dt tongue/nn especially/rb of/in a/dt broadcaster/nn resulting/vbg in/in a/dt humorous/jj or/cc indecorous/jj misreading/nn

subj: blunder
freq: usually
strength: medium
class: Denotation
periph: a/dt gross/jj or/cc stupid/jj mistake/nn

subj: boner
freq: usually
strength: medium
class: Denotation
periph: a/dt mistake/nn

subj: error
freq: usually
strength: medium
class: Denotation
periph: a/dt mistake/nn as/in in/in action/nn speech/nn etc/fw

subj: error
freq: usually
strength: medium
class: Denotation
periph: belief/nn in/in something/nn untrue/jj the/dt holding/nn of/in mistaken/vbn opinions/nns

subj: faux pas
freq: usually
strength: medium
class: Denotation
periph: a/dt slip/nn in/in manners/nns

From GI

subj: mistake
freq: usually
strength: medium
class: Favourable

subj: blunder
freq: usually
strength: medium
class: Favourable

subj: boner
freq: usually
strength: medium
class: Pejorative

subj: contretemps
freq: usually
strength: medium
class: Favourable

subj: error
freq: usually
strength: medium
class: Favourable

subj: faux pas
freq: usually
strength: medium
class: Pejorative

subj: goof
freq: usually
strength: medium
class: Favourable

subj: slip
freq: usually

strength: medium
class: Favourable

subj: solecism
freq: usually
strength: medium
class: Favourable

Appendix C

Example of Customized English LKB Entry for the Near-Synonyms of *error*

```
(defcluster generic_mistake_n
    :syns (mistake blooper blunder boner contretemps error faux_pas goof
           slip solecism )
    :senses (mistake#n#1 blooper#n#1 blunder#n#1 boner#n#1 error#n#1
             faux_pas#n#1 slip#n#1 slip#n#2 solecism#n#1)
    :core (ROOT GENERIC_MISTAKE (OR |fault,error| |boner| |gaffe| |slipup|))
    :periph (
       (P1  (C1 / deviation))
       (P2  (C1 / sin))
       (P3  (C1 / assessment :MOD (*OR* ignorant uninformed)))
       (P4  (C1 / accidental))
       (P5  (C1 / careless))
       (P6  (C1 / indefensible))
       (P7  (C1 / occurrence :MOD (*OR* embarrassing awkward)))
       (P8  (C1 / (*OR* action opinion judgment)))
       (P9  (C1 / (*OR* gross stupid)))
       (P10  (C1 / belief))
       (P11  (C1 / manners))
     )
    :distinctions
    ((error usually medium Implication P1)
     (error usually medium Denotation P2)
     (blunder usually medium Implication P3)
     (slip usually high Denotation P4)
     (slip usually medium Denotation P5)
     (blooper low Formality)
     (goof usually medium Denotation P6)
```

```
     (goof medium Formality)
     (contretemps usually medium Denotation P7)
     (mistake usually medium Denotation P8)
     (blunder usually medium Denotation P9)
     (error usually medium Denotation P10)
     (faux_pas usually medium Denotation P11)
     (mistake usually medium Favourable :agent)
     (blunder usually medium Favourable :agent)
     (boner usually medium Pejorative :agent)
     (contretemps usually medium Favourable :agent)
     (error usually medium Favourable :agent)
     (faux_pas usually medium Pejorative :agent)
     (goof usually medium Favourable :agent)
     (slip usually medium Favourable :agent)
     (solecism usually medium Favourable :agent)))
```

```
(defcluster generic_absorb_v
    :syns (absorb assimilate digest imbibe incorporate ingest )
    :senses (absorb#v#1 absorb#v#2 assimilate#v#1 assimilate#v#4 digest#v#1
             imbibe#v#2 imbibe#v#3 imbibe#v#4 ingest#v#1 ingest#v#2)
    :core (ROOT GENERIC_ABSORB (OR |absorb<sorb| |ingest<larn|
                              |imbibe,assimilate| |digest<treat| |imbibe<have|
                              |imbibe<take in| |ingest,have|) )
    :covers (ROOT)
    :periph (
       (P1  (C1 / |taking in|))
       (P2  (C1 / thoroughness :GPI action))
       (P3  (C1 / disappearance :MOD complete))
       (P4  (C1 / action))
       (P5  (C1 / |taking in|))
       (P6  (C1 / receptivity))
       (P7  (C1 / alter :OBJECT food))
       (P8  (C1 / agent :MOD third))
       (P9  (C1 / taking))
       (P10  (C1 / loss :GPI (MM1 / identity :MOD separate)))
       (P11  (C1 / mentally))
       (P12  (C1 / (*OR* liquid moisture)))
     )
   :distinctions
    (
     (absorb low Formality)
     (absorb usually medium Suggestion P1)
     (absorb sometimes medium Implication P2)
     (absorb sometimes high Denotation P3)
     (ingest usually medium Denotation P4)
     (ingest usually medium Denotation P5)
```

```
     (ingest usually medium Suggestion P6)
     (digest sometimes medium Denotation P7)
     (assimilate high Force)
     (assimilate usually medium Implication P8)
     (incorporate usually medium Denotation P9)
     (incorporate usually high Denotation P10)
     (digest usually medium denotation P11)
     (imbibe usually medium denotation P12)
     (absorb usually medium Favourable :agent)
     (assimilate usually medium Favourable :agent)
     (digest usually medium Favourable :agent)
     (imbibe usually medium Favourable :agent)
     (incorporate usually medium Favourable :agent)
     (ingest usually medium Favourable :agent)))
```

```
(defcluster generic_afraid_a
    :syns (afraid aghast alarmed anxious apprehensive fearful frightened
          scared terror-stricken )
    :senses (afraid#a#1 aghast#a#1 alarmed#a#1 apprehensive#a#3
            fearful#a#1 fearful#a#2 fearful#a#3 frightened#a#1 frightened#a#2
            scared#a#1 terror-stricken#a#1)
    :core (ROOT GENERIC_AFRAID (OR |afraid>unnerved| |aghast| |alarmed|
                   |uneasy<afraid| |fearful<afraid| |dreaded| |coward,fearful|
                   |fearful<unmanly| |scared| |panicky| |terror-struck|) )
    :covers (ROOT)
    :periph (
       (P1  (C1 / fear))
       (P2  (C1 / nothing))
       (P3  (C1 / fear :GPI (MM1 / harm :MOD bodily)))
       (P4  (C1 / fears :MOD vague))
       (P5  (C1 / causes))
       (P6  (C1 / (*AND* tense worried)))
       (P7  (C1 / (*OR* terror dread)))
       (P8  (C1 / awareness :GPI (MM1 / danger :MOD impending)))
       (P9  (C1 / feelings :GPI fear :MOD strong))
     )
    :distinctions
    (
     (afraid usually medium Denotation P1)
     (afraid usually medium Denotation P2)
     (frightened usually medium Suggestion P3)
     (scared usually medium Suggestion P3)
     (frightened usually medium Denotation P4)
     (scared usually medium Denotation P4)
     (frightened usually medium Denotation P5)
     (anxious usually medium Denotation P6)
```

```
(fearful sometimes medium Denotation P7)
(apprehensive usually medium Suggestion P8)
(aghast usually medium Denotation P9)
(alarmed usually medium Denotation P9)
(terror-stricken usually medium Denotation P9)
(terror-stricken usually medium Suggestion P1)
(afraid usually medium Favourable :agent)
(aghast usually medium Favourable :agent)
(alarmed usually medium Pejorative :agent)
(anxious usually medium Favourable :agent)
(apprehensive usually medium Favourable :agent)
(fearful usually medium Pejorative :agent)
(frightened usually medium Pejorative :agent)
(scared usually medium Favourable :agent)))
```

Appendix D

Example of French LKB Entry for the Near-Synonyms of *erreur*

```
(defcluster generic_erreur_n
    :syns (erreur egarement illusion aberration malentendu mecompte bevue
           betise blague gaffe boulette brioche maldonne sophisme lapsus
           meprise bourde)
    :periph ((P1 (c1 / |take amiss| :object thing))
             (P2 (c2 / |grief,sorrow|))
             (P3 (c3 / |betise|))
             (P4 (c4 / |hand,deal|
                     :object |card<paper|
                     :polarity -
                     :mod |the right way|))
             (P5 (c5 / (OR |voluntary>unforced| |involuntary>enforced|)))
             (P6 (c6 / |mind<view|  :mod |wrong>false|))
             (P7 (c7 / |glaring,gross|))
             (P8 (c8 / |view<belief| :mod |false>untrue|))
             (P9 (c9 / |significant,substantial|))
             (P10 (c10 / |dementia|))
             (P11 (c11 / |visual aspect| :mod |false>untrue|))
             (P12 (c12 / |abnormalcy| :mod |rational<mental|))
             (P13 (c13 / |figuring|))
             (P14 (c14 / assessment :MOD |ignorance|))
             (P15 (c15 / |conduct<activity| :mod |societal,social|))
             (P16 (c16 / |logical thinking| :mod |wrong>false|))
             (P17 (c17 / |speech<module|))
             (P18 (c18 / |action| :MOD |regretable|))
     )
    :distinctions
```

```
    ((meprise usually medium Denotation P1)
;     "prend une chose pour une autre"
     (meprise usually medium Denotation P2)
;     "generalement faire grief"
     (bevue usually medium Implication P3)
;     "irreflexion, etourderie, souvent meme betise"
     (maldonne usually medium Denotation P4)
;     "ne distribue pas les cartes comme il se doit"
     (maldonne usually medium Denotation P5)
;     "erreur voluntaire or involuntaire" ?
     (aberration always medium Denotation P6)
;      "erreur de jugement"
     (blague  low Formality)
;       "familier"
     (gaffe  low Formality)
;       "familier"
     (gaffe usually medium Denotation P7)
;       "bevue grossiere" -- rude
     (boulette  low Formality)
;       "populaire"
     (erreur usually medium Denotation P8)
;       "fausse opinion"
     (egarement usually medium Denotation P9)
;       "erreur considerable"
     (egarement usually medium Denotation P10)
;    "due a une sorte d'extravangace, demence, par raport au vrai et au bien"
     (illusion usually medium Denotation P11)
;       "erreur des senses et de l'espirit due a une fausse apparence"
     (aberration usually medium Denotation P12)
;       "anomalie d'une des nos intelectuelle fonctionnes => juger mal"
     (malentendu usually medium Denotation P1)
;       "prend une chose pour une autre"
     (mecompte usually medium Denotation P13)
;       "erreur dans une compte, calcul"
     (bevue usually medium Denotation P7)
;       "erreur grossiere"
     (bevue usually medium Denotation P14)
;       "due a ignorance out a balourdise"
     (bourde  low Formality)
;       "familier"
     (betise  low Formality)
;       "tres familier"
     (gaffe usually medium Denotation P15)
;       "maladresse, demarche de conduite" -- social behaviour
     (boulette usually medium Denotation P9)
;       "grosse bevue" -- considerable
     (brioche  low Formality)
```

```
;       "populaire"
     (sophisme usually medium Denotation P16)
;       "raisonnement faux"
     (lapsus usually medium Denotation P17)
;       "erreur de langage"
     (erreur usually medium Denotation P18)
;       "action inconsideree, regrettable, maladroite"
    )
)
```

Appendix E

Implementation Notes

Most of the programs implemented for this thesis were written in Perl. An exception is Chapter 7, where the main programs were implemented in Lisp and CLOS (object-oriented Common Lisp). Off-the-shelf tools were used when possible. Following is a list of the main programs and supporting tools developed for each chapter.

In Chapter 2, Perl programs were implemented for the following tasks:

- preprocessing the CTRW dictionary (XML markup, sentence segmentation).
- preparing data for the decision-lists algorithm.
- the decision-lists algorithm.
- extracting knowledge from CTRW (including comparisons and coreference resolution).
- extracting random clusters of near-synonyms from CTRW to be used in evaluation.
- evaluating the accuracy of the extraction results.

Existing tools used in Chapter 2 were: Abney's chunker[1], Collins's parser[2], and Charniak's parser[3] (also used in Chapter 7).

In Chapter 3, Perl programs were developed for:

[1]http://gross.sfs.nphil.uni-tuebingen.de:8080/release/cass.html
[2]http://www.ai.mit.edu/people/mcollins/code.html
[3]ftp://ftp.cs.brown.edu/pub/nlparser/

- computing indicators for all the senses of the near-synonyms.
- computing context vectors (including extracting co-occurrence counts from the BNC for the dimension and feature words).
- adapting Corelex for WordNet1.7.
- computing Resnik's coefficient and Resnik's similarity measure.
- preparing data for the human judges.
- producing a standard solution and computing the inter-judge agreement and kappa statistic.
- evaluating the accuracy of the automatic sense disambiguation.

Existing tools used in Chapter 3 were: the QueryData[4] interface to WordNet and the C4.5 decision tree learning tool[5].

In Chapter 4, Perl programs were needed for:

- the mutual information filter using the Web as a corpus.
- the differential t-test classifier (into preferred collocations, less-preferred collocations, and anti-collocations).
- preparing data for the human judges to annotate.
- building standard solutions for the MI filter and for the classifier, and computing the inter-judge agreement and kappa statistic.
- evaluating the accuracy of the results.

The Bigram Statistics Package[6] was used in Chapter 4 to gather co-occurrence counts from the BNC and to rank collocations according to various measures. C4.5 was used for learning a decision tree for the differential t-test classifier. The WWW-Search package[7] was used to compute Web counts through the AltaVista search engine.

In Chapter 5, the Perl programs were implemented by the research assistant Olga Feiguina (except for the part dealing with WordNet) for the following tasks:

- extracting definitions of near-synonyms from the *Macquarie Dictionary*.

[4] http://www.ai.mit.edu/∼jrennie/WordNet/
[5] http://www.cse.unsw.edu.au/∼quinlan/
[6] http://www.d.umn.edu/∼tpederse/code.html
[7] http://www.perl.com/CPAN/authors/id/M/MT/MTHURN/

- extracting denotational distinctions from the definition.
- extracting attitudinal distinctions from the *General Inquirer*.
- extracting stylistic distinctions (formality) from WordNet.
- merging parts of of the LKB of NS acquired from different sources.
- checking the consistency of the final LKB of NS, and resolving conflicts.

Chapter 6 needed Perl programs for the following tasks:

- transforming the peripheral strings into configurations of concepts.
- computing the coverage and the correctness of the transformation rules (evaluation).
- mapping WordNet senses to Sensus concepts.

In Chapter 7, Irene Langkilde-Geary's HALogen[8] (implemented in Lisp, except the statistical ranker which is implemented in C) was extended with the Lisp programs for:

- loading the LKB of NS (Lisp and CLOS code adapted from I-Saurus).
- computing preference satisfaction scores (Lisp and CLOS code adapted from I-Saurus).
- integrating the near-synonym choice module with HALogen.
- computing similarity of conceptual configurations and lexical similarity.
- lexical analysis of nuances for English and French near-synonyms.

and Perl programs were developed for:

- the near-synonym collocation module, which performs operations on the forest representation.
- evaluating the accuracy of the near-synonym choice module and of the near-synonym collocation module.
- extracting sentences from Hansard that contain English and French near-synonyms.
- extracting sentences from the BNC that contain collocations of near-synonyms.
- scripts to automate the production of interlingual representations (batch mode) by using the input construction tool (this tool was implemented in Lisp and Perl by Irene Langkilde-Geary).

[8]http://www.isi.edu/licensed-sw/halogen/

Bibliography

Steven Abney. Partial parsing via finite-state cascades. In *Proceedings of the 8th European Summer School in Logic, Language and Information (ESSLLI'96), Robust Parsing Workshop*, pages 124–131, 1996.

J.D. Apresjan, V.V. Botiakova, T.E. Latiskeva, M.A. Mosiagina, I.V. Polik, V.I. Rakitina, A.A. Rozenman, and E.E. Sretenskaia. *Anglo-Russkii Sinonimicheskii Slovar*. Izdatelstvo Russkii Jazik, 1980.

Juri Apresjan. *Systematic Lexicography*. Translation, Oxford University Press, 2000.

Antti Arppe. The usage patterns and selectional preferences of synonyms in a morphologically rich language. *Journées internationales d'Analyse statistique des Donées Textuelles (JADT)*, 6:21–32, 2002.

René Bailly, editor. *Dictionnaire des Synonymes de la Langue Française*. Larousse, Paris, 1973.

Caroline Barrière and Fred Popowich. Concept clustering and knowledge integration from a children's dictionary. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'96)*, Copenhagen, Denmark, 1996.

Henri Bénac, editor. *Dictionnaire des Synonymes*. Librarie Hachette, Paris, 1956.

J.R.L. Bernard, editor. *The Macquarie Thesaurus – Companion to The Macquarie Dictionary*. Sydney, Australia: Macquarie Library, 1987.

Alexander Budanitsky and Graeme Hirst. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Proceedings of the Workshop on WordNet and Other Lexical Resources, Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001)*, Pittsburgh, USA, 2001.

Sharon Caraballo. Automatic acquisition of a hypernym-labeled noun hierarchy from text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 120–126, Maryland, USA, 1999.

Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational Linguistics*, 22 (2):249–254, 1996.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics and the 6th Conference on Applied Natural Language Processing (NAACL-ANLP 2000)*, Seattle, USA, 2000.

Kenneth Church, William Gale, Patrick Hanks, and Donald Hindle. Using statistics in lexical analysis. In Uri Zernik, editor, *Lexical Acquisition: Using On-line Resources to Build a Lexicon*, pages 115–164. Lawrence Erlbaum, 1991.

Kenneth Church and Patrick Hanks. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16 (1):22–29, 1991.

Michael Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, 1996.

Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-99)*, Maryland, 1999.

A. Delbridge, J.R.L. Bernard, D. Blair, W.S. Ramson, and Susan Butler, editors. *The Macquarie Dictionary*. Sydney, Australia: Macquarie Library, 1987.

Chrysanne DiMarco and Graeme Hirst. Usage notes as the basis for a representation of near-synonymy for lexical choice. In *Proceedings of the 9th annual conference of the University of Waterloo Centre for the New Oxford English Dictionary and Text Research*, pages 33–43, Oxford, 1993.

Chrysanne DiMarco, Graeme Hirst, and Manfred Stede. The semantic and stylistic differentiation of synonyms and near-synonyms. In *Proceedings of AAAI Spring Symposium on Building Lexicons for Machine Translation*, pages 114–121, Stanford, CA, 1993.

Bonnie J. Dorr. *The Use of Lexical Semantics in Interlingual Machine Translation*. The MIT Press, 1993.

Ted Dunning. Accurate methods for statistics of surprise and coincidence. *Computational Linguistics*, 19 (1):61–74, 1993.

Philip Edmonds. *Semantic representations of near-synonyms for automatic lexical choice*. PhD thesis, University of Toronto, 1999.

Philip Edmonds and Graeme Hirst. Near-synonymy and lexical choice. *Computational Linguistics*, 28 (2):105–145, 2002.

Stefan Evert and Brigitte Krenn. Methods for the qualitative evaluation of lexical association measures. In *Proceedings of the 39th Annual Meeting of the of the Association for Computational Linguistics (ACL 2001)*, Toulouse, France, 2001.

Hong Gao. *The Physical Foundation of the Patterning of Physical Action Verbs*. PhD thesis, Lund University, 2001.

Philip B. Gove, editor. *Webster's New Dictionary of Synonyms*. G.&C. Merriam Co., 1984.

S. I. Hayakawa, editor. *Choose the Right Word.* Second Edition, revised by Eugene Ehrlich. HarperCollins Publishers, 1994.

Marti Hearst. Automatic acquisition of hyponyms from large corpora. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 538–545, Nantes, France, 1992.

Graeme Hirst. Near-synonymy and the structure of lexical knowledge. In *Working notes, AAAI Symposium on Representation and Acquisition of Lexical Knowledge: Polysemy, Ambiguity, and Generativity*, pages 51–56, Stanford University, 1995.

Eduard Hovy. Pragmatics and language generation. *Artificial Intelligence*, 43:153–197, 1990.

Nancy Ide and Jean Véronis. Knowledge extraction from machine-readable dictionaries: An evaluation. In P. Steffens, editor, *Machine Translation and the Lexicon*, pages 19–34. Springer-Verlag, 1994.

Diana Zaiu Inkpen and Graeme Hirst. Building a lexical knowledge-base of near-synonym differences. In *Proceedings of the Workshop on WordNet and Other Lexical Resources, Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001)*, pages 47–52, Pittsburgh, USA, 2001a.

Diana Zaiu Inkpen and Graeme Hirst. Experiments on extracting knowledge from a machine-readable dictionary of synonym differences. In *Proceedings of the 2nd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2001)*, pages 265–280, Mexico City, Mexico, 2001b.

Diana Zaiu Inkpen and Graeme Hirst. Acquiring collocations for lexical choice between near-synonyms. In *Proceedings of the Workshop on Unsupervised Lexical Acquisition, 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 67–76, Philadelphia, USA, 2002.

Diana Zaiu Inkpen and Graeme Hirst. Automatic sense disambiguation of the near-synonyms in a dictionary entry. In *Proceedings of the 4th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2003)*, pages 258–267, Mexico City, Mexico, 2003.

Mario Jarmasz and Stan Szpakowicz. Roget's thesaurus and semantic similarity. In *Proceedings of the International Conference RANLP-2003 (Recent Advances in Natural Language Processing)*, Borovets, Bulgaria, 2003.

Kevin Knight and Steve Luk. Building a large knowledge base for machine translation. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA, 1994.

Anna Korhonen. Semantically motivated subcategorization acquisition. In *Proceedings of the Workshop on Unsupervised Lexical Acquisition, 40th Annual Meeting of the of the Association for Computational Linguistics (ACL 2002)*, Philadelphia, USA, 2002.

Irene Langkilde. Forest-based statistical sentence generation. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics and the 6th Conference on Applied Natural Language Processing (NAACL-ANLP 2000)*, Seattle, USA, 2000.

Irene Langkilde and Kevin Knight. The practical value of N-grams in generation. In *Proceedings of the 9th International Natural Language Generation Workshop*, pages 248–255, Niagara-on-the-Lake, Canada, 1998.

Irene Langkilde-Geary. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, pages 17–24, New York, USA, 2002a.

Irene Langkilde-Geary. *A Foundation for a General-Purpose Natural Language Generation:*

Sentence Realization Using Probabilistic Models of Language. PhD thesis, University of Southern California, 2002b.

Doug Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Cummunications of the ACM*, 38 (11):33–38, 1995.

Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of SIGDOC Conference*, pages 24–26, Toronto, Canada, 1986.

Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. Identifying synonyms among distributionally similar words. In *Proceedings of the Eighteenth Joint International Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

Donald A.B. Lindberg, Betsy L. Humphreys, and Alexa T. McCray. The unified medical language system. *Methods of Information in Medicine*, 32 (4):281–289, 1993.

Kavi Mahesh and Sergei Nirenburg. A situated ontology for practical NLP. In *Proceedings of Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

Kathleen McKeown and Dragomir Radev. Collocations. In Robert Dale, Hermann Moisl, and Harold Somers, editors, *Handbook of Natural Language Processing*. Marcel Dekker, 2000.

Igor Mel'čuk and Leo Wanner. Towards a lexicographic approach to lexical transfer in machine translation. *Machine Translation*, 16 (1):21–87, 2001.

George A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38 (11):39–41, 1995.

Yoshiki Niwa and Yoshihiko Nitta. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'94)*, pages 304–309, Kyoto, Japan, 1994.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. Technical report RC22176, IBM Research Division, Thomas J. Watson Research Center, 2001.

Darren Pearce. Synonymy in collocation extraction. In *Proceedings of the Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, USA, 2001.

Ted Pedersen. Fishing for exactness. In *Proceedings of the South-Central SAS Users Group Conference (SCSUG-96)*, Austin, Texas, 1996.

Ted Pedersen and Satanjeev Banerjee. An adapted Lesk algorithm for word sense disambiguation using WordNet. In *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2002)*, Mexico City, Mexico, 2002.

Ted Pedersen and Satanjeev Banerjee. The design, implementation, and use of the ngram statistical package. In *Proceedings of the 4th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2003)*, pages 370–381, Mexico City, Mexico, 2003.

Penman Natural Language Group. The Penman reference manual. Technical report, Information Science Institute of the University of Southern California, 1989.

Sabine Ploux and Hyungsuk Ji. A model for matching semantic maps between languages (French/English, English/French). *Computational Linguistics*, 29 (2):155–178, 2003.

Philip Resnik. Mining the web for bilingual text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 527–534, Maryland, USA, 1999a.

Philip Resnik. Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999b.

Philip Resnik, Mari Broman Olsen, and Mona Diab. Creating a parallel corpus from the book of 2000 tongues. In *Proceedings of the Text Encoding Initiative 10th Anniversary User Conference (TEI-10)*, Providence, 1997.

Stephen Richardson, William Dolan, and Lucy Vanderwende. MindNet: Acquiring and structuring semantic information from text. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics joint with 17th International Conference on Computational Linguistics (ACL-COLING'98)*, pages 1098–1102, Montreal, Quebec, Canada, 1998.

German Rigau, Horacio Rodriguez, and Eneko Agirre. Building accurate semantic taxonomies from monolingual MRDs. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics joint with 17th International Conference on Computational Linguistics (ACL-COLING'98)*, pages 1103–1109, Montreal, Quebec, Canada, 1998.

Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 474–479, Orlando, FL, 1999.

Peter Mark Roget, editor. *Roget's Thesaurus of English Words and Phrases*. Longman Group Ltd., Harlow, Essex, England, 1852.

Adrian Room, editor. *Room's Dictionary of Distinguishables*. Routhledge & Kegan Paul, Boston, 1981.

Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24 (1): 97–123, 1998.

Sidney Siegel and N. John Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, Inc., 1988.

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

Manfred Stede. A generative perspective on verb alternations. *Computational Linguistics*, 24 (3):401–431, September 1998.

Mark Stevenson and Yorick Wilks. The interaction of knowledge sources in word sense disambiguation. *Computational Linguistics*, 27 (3):321–350, 2001.

Philip J. Stone, Dexter C. Dunphy, Marshall S. Smith, Daniel M. Ogilvie, and associates. *The General Inquirer: A Computer Approach to Content Analysis*. The MIT Press, 1966.

Peter Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the Twelfth European Conference on Machine Learning (ECML 2001)*, pages 491–502, Freiburg, Germany, 2001.

Anna Wierzbicka. *Understanding Cultures Through Their Key Words*. Oxford University Press, 1997.

Yorick Wilks. Providing machine tractable dictionary tools. In James Pustejovsky, editor, *Semantics and the Lexicon*, pages 341–401. Kluwer, London, 1993.

Yorick Wilks and Roberta Catizone. Lexical tuning. In *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2002)*, Mexico City, Mexico, 2002.

Yorick Wilks, Brian Slator, and Louise Guthrie, editors. *Electric words: dictionaries, computers, and meanings*. MIT Press, 1996.

David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, MA, 1995.

David Yarowsky and Richard Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 207–216, Hong Kong, 2000.

Subject Index

Citation Index