Making Lexical Sense of Japanese–English Machine Translation: A Disambiguation Extravaganza

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
GRADUATE SCHOOL OF INFORMATION SCIENCE AND ENGINEERING,
TOKYO INSTITUTE OF TECHNOLOGY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF ENGINEERING

Supervised by Prof. Hozumi Tanaka and Assoc. Prof. Tohru Noguchi

Timothy Baldwin

December 2000

Contents

List of Tables

List of Figures

Acknowledgements

A vast number of people have contributed both directly and indirectly to the development and culmination of this research.

The one person to whom I am most indebted in gaining access to the natural language processing domain is Professor Tanaka, who instigated my stay in Japan and made available the formidable resources of his research laboratory during my five and a half year stay as a student. His constant encouragement, infinite patience and understanding, and willingness to let me find my own wings, have all contributed to my humble development as an NLP practitioner. Associate Professor Noguchi demonstrated superhuman benevolence in putting up with my frenetic journey through the field, and provided superlative feedback when it mattered most. Associate Professor Tokunaga also provided valuable advice and support throughout the course of this research. Thanks go also to the remaining two members of my thesis evaluation committee, Professor Furui and Associate Professor Okumura.

The one individual to have had the greatest influence on the development of this research is without doubt Francis Bond. He was instrumental in setting up a joint research agreement with the NTT Communication Science Laboratories, which gave me access to their peerless NTT system resources, and also to other members of the NTT MT group. It is not hard to see the NTT mark on much of the work contained in this thesis. Above and beyond providing the means to do this research, Francis was a constant source of inspiration, a joy to do work with and a good friend. I am eternally grateful for his openness in sharing research ideas and keeping me abreast of research developments throughout the world.

Other individuals whose comments and suggestions helped shape this research are Oscar Beatriz, Atsushi Fujii, Hiromi Nakaiwa, Emmanuel Planas and Yorick Wilks.

Christoph Neumann was priceless in lightening up the tone of the laboratory, and also made key contributions to this research. Kiyoaki Shirai's tireless efforts in buffering much of the administrative burden that should by all rights have fallen my way, and unfailing preparedness to hear out the most trivial of questions infinitely enhanced life in the Tanaka laboratory. Masahiro Ueki and Yusuke Shin'yama were invaluable system oracles, and enhanced my working knowledge of UNIX/Linux no end. I would also like to express my gratitude to past and present members of the Tanaka and Tokunaga laboratories, as well as long-suffering friends around campus, for making my stay in Japan a pleasant and rewarding one.

Closer to home, I doubt that I would have made it this far, and certainly wouldn't have enjoyed either the getting here or appreciated what it meant to be here, without the unfailing support, love and occasional reprimand from my wife Mari. While not involving herself directly in my university life, she laid the foundations for my work and stood in for me at all the right times. Without a doubt the most significant results of my time as a PhD student are Kate and Kyle. They have made my life richer and more fulfilling, and their beaming smiles have lit my path through times of despondency, and made times of joy that much more memorable. In this respect also, Mari has provided the grounding for an unforgettable few years, in bringing two happy, bouncy babies into

the world and doing marvels in bringing them up.

I dedicate this thesis to the memories of Zara and Marina: to all they were and all they could have been in this life.

Abbreviations and lexical conventions

Throughout this publication, Japanese examples are represented by way of the Kunreishiki transliteration system, modified slightly to distinguish the single-character kana *n* ("ん/ン") from other characters containing the phoneme /n/ (i.e. *na*, *ni*, *nu*, *ne*, *no*, *nya*, *nyu*, *nyo*). In the former case, we use a capital *N*, and in the latter case a lower case *n*, as in *bunaN* "safe". Note that the "−" symbol represents a long vowel sound, as in *kibō* "wish/desire".

The following abbreviations are used in glosses:

ABL	Ablative case marker
ACC	Accusative case marker
COM	Comitative case marker
DAT	Dative case marker
LOC	Locative case marker
NML	Nominaliser
NOM	Nominative case marker
PAST	Past tense
POT	Potential verb morpheme
PRES	Non-past tense
PROG	Progressive aspect
REN	Ren'yo conjugational form
TOP	Topic case marker

Chapter 1

Introduction

1.1 Motivation

As with any natural language processing task, disambiguation is a key technology in machine translation (MT). Disambiguation is broadly defined as the task of interpreting or otherwise determining the correct analysis for a given item from a finite set of item abstractions. In word sense disambiguation, for example, an input word is disambiguated according to a fixed sense set, by way of mapping it a unique sense candidate contained in this set. Uniqueness of interpretation is a key aspect of disambiguation in domains such as MT.[1] That is, we assume a 1-to-1 mapping from source items to target concepts, or in other words, that every source item can be optimally described by way of one and only one target concept.[2]

Disambiguation comes in many guises and impinges on all aspects of MT, including lexical, morphological, syntactic, semantic and transfer ambiguity. It can be restricted to ambiguity within either the source or target language, or alternatively affect source–target language transferral. Example intra-language MT disambiguation tasks are part-of-speech tagging (the disambiguation of words or morphemes according to a tag set: Papageorgiou (1994), Elworthy (1994), Brill (1995), Kurohashi and Nagao (1998), Matsumoto *et al.* (1999), *inter alia*), parsing (the disambiguation of word sequences according to a treebank of possible annotated structures: Collins (1996), Collins (1997), Charniak (2000), Henderson and Brill (2000), *inter alia*), and anaphora resolution (the disambiguation of the referent of words across a document and greater world context: Nakaiwa and Ikehara (1995), Nakaiwa *et al.* (1995), Azzam *et al.* (1998), Ferrández *et al.* (1998), Murata and Nagao (1998), *inter alia*). Intra-language disambiguation can also take place in the target language as part of lexical selection, in selecting between alternates of a given verb (Dorr and Olsen 1996; Stede 1996) or resolving "boundary friction" in choosing the optimal path through a target language lexicalisation lattice (Dagan and Itai 1994). Our particular interest, however, lies in inter-language disambiguation, that is directly mapping the source language onto the target language through the medium of direct lexicalisations (for transfer-based MT) or abstractions over the data (for interlingua-based MT). Our motivation in restricting the focus of our research efforts in this manner, is to zoom in on issues specific to the realm of MT. Intra-language disambiguation techniques are common to a wide range of sub-fields of natural language processing, including parsing, information extraction, text understanding and question answering, in the same basic form as in MT. We thus

[1] Domains where this stipulation does not hold do exist, including information retrieval, where each member of a document set is evaluated for relevance to the query, and no attempt is generally made to internally rank the relevant documents against each other.

[2] Possibly including the possibility of an "unknown item" category, in the case that the source item cannot be described aptly by way of the domain abstraction.

gain little by talking of them specifically within the bounds of MT. Inter-language disambiguation techniques, on the other hand, are the exclusive domain of machine translation, or at least research founded around MT techniques (such as cross-lingual information retrieval).

The focus of this research is key disambiguation and lexical selection techniques in Japanese–English MT. Rather than restricting ourselves to one MT paradigm, we test component technologies for translation memories, example-based MT, and transfer-based MT. **Translation memories** are essentially a database of source language strings paired with target language translations, linked to a retrieval mechanism providing access to the translation records most similar to a given input from the memory. The job of the translation memory system is simply to retrieve a set of translations from the memory, and how those translations are then used is left up to the system user. **Example-based** MT represents a step up over translation memory and offer flexibility of output. As with translation memory systems, translations or translation fragments corresponding to sections of the input are first retrieved from the translation memory, but the system then goes on to extract/modify components from the retrieved translations and glue these together in forming a final translation which directly reflects the original input. **Transfer-based MT** is in a way the next step along this evolutionary path, in that the system has an internal knowledge base of source–language word/phrase correspondences ("transfer pairs"), and arranges the target language equivalents of source language linguistic units, in forming the final translation. Unlike example-based MT, the linguistic structure/features of each transfer pair are also hard-wired into transfer-based systems and applied directly in translation generation. Transfer-based systems thus tend to be more linguistically aware and highly abstracted than their example-based counterparts.

Translation memory and example-based MT disambiguation techniques are necessarily lexically-oriented, whereas transfer-based MT operates on a higher plane of linguistic sophistication, often involving semantic and pragmatic processing. In order to achieve an appreciation of the full spectrum of types of interlingual disambiguation involved in MT, we pinpoint three areas of disambiguation, suggested to be approximately evenly spaced along the continuum of disambiguation complexity: lexical disambiguation, feature-based disambiguation, and fully-fledged semantic disambiguation (in increasing order of disambiguation complexity).

Lexical disambiguation is the shallowest form of disambiguation considered, and operates over the surface realisation (i.e. spell-out) of target items, without any explicit form of abstraction. The particular instance of lexical disambiguation we target is translation retrieval, that is the retrieval of translation candidates for a lexical input, from a translation memory. Here, disambiguation is by way of the finite set of translations contained in the memory. Translation retrieval constitutes the full extent of a translation memory system, and is a key component of example-based MT.

Feature-based disambiguation involves limited conceptual abstraction of the data, but according to a restricted set of features. Unlike lexical disambiguation where the actual lexical make-up of target objects doubles as their fingerprint for disambiguation, with feature-based disambiguation, item representation is based around this feature set. Disambiguation is thus from the feature vector for a given input to a fixed set of classes typifying the range of categories of data. We exemplify feature-based disambiguation by way of the Japanese relative clause construction interpretation task, in which we classify the interpretation of Japanese relative clause constructions and transfer this across to selection of the equivalent English construction type. The actual disambiguation mechanism employed is supervised machine learning, in the form of the C4.5 (Quinlan 1993) and TiMBL (Daelemans *et al.* 2000) systems, interleaved with home-grown feature selection and construction methods.

Fully-fledged semantic disambiguation interfaces with a full component of lexical semantics, potentially including word sense, conceptual similarity, domain type, and stylistic considera-

tions. Concepts are represented according to semantic hierarchies or networks, and it is over these fields of operation that fully-fledged semantic disambiguation operates. Disambiguation is relative to a set of concepts or senses. The particular task used to illustrate the nature of fully-fledged semantic disambiguation is verb sense disambiguation, in a Japanese–English transfer-based MT context. Verb sense is represented by way of a case frame, annotated with selectional constraints. Each Japanese verb sense and associated case frame is linked to an English structural skeleton, headed by the corresponding English term. Disambiguation is thus according to translation correspondence. The fully-fledged semantic nature of this task arises from the selectional constraints characterising each verb sense, which represent sets of nodes in a thesaurus.

1.2 Key terms

Due to the heterogeneous nature of the disambiguation tasks targeted in this thesis, we will tend to explain concepts within the local context of each individual task. There are a number of linguistic concepts, however, which underpin much of this research and are worthy of explanation at this point.

First and foremost, we refer variously to **arguments** in the sense of consitutents governed by the predicate of a given clause. Here, we make no distinction between different argument types along the lines of grammatical roles or relations, for example, such that the subject of a clause and an adverbial are treated equivalently as arguments. The reader is cautioned that this is a much broader definition of the term than is given within certain schools of theoretical linguistics, where arguments are non-subject complements.

In line with this definition of arguments, **argument status** is a measure of the degree of intrinsic participation of each argument within the state of affairs described by the predicate, and correlates with the more conventional concept of complementhood/adjuncthood. Our motivation in invoking a new term in this context is that argument status is described by way of a total of four categories, rather than the conventional two.

1.3 Thesis structure

A chapter is devoted to a hand-picked instance of each of the lexical, feature-based and fully-fledged semantic disambiguation tasks. Lexical disambiguation methods are described in Chapter 2 by way of the Japanese–English translation retrieval task, and the relative performance of various parameter combinations analysed in detail. Chapter 3 looks at feature-based disambiguation in the form of Japanese relative clause construction analysis, and discusses various methods for incrementally improving the classification accuracy of feature-based disambiguation methods. Finally, Chapter 4 exemplifies fully-fledged semantic disambiguation by way of a selectional constraint-based verb sense disambiguation task, and presents a range of disambiguation techniques targeted at semantic taxonomies. Finally, the thesis is concluded in Chapter 5.

Chapter 2

The Hare and the Tortoise: Slow and Supposedly Steady does not Necessarily Win the Translation Retrieval Race

2.1 Introduction

Translation memories (TM's) are a well-established technology within the human and machine translation fraternities, due to the high translation precision they afford. Essentially, TM's are a list of **translation records** (source language strings paired with a unique target language translation), which the TM system accesses in suggesting a list of target language **translation candidates** for a given source language input.[1] In example-based machine translation (EBMT), either a single translation record is retrieved from the TM based on a match with the overall source language input, or the input is partitioned into coherent segments, and individual translations retrieved for each (Sato and Nagao 1990; Nirenburg *et al.* 1993); this is the first step toward generating a customised translation for the input. With stand-alone TM systems, on the other hand, the system selects an arbitrary number of translation candidates falling within a certain empirical corridor of similarity with the overall input string, and simply outputs these for manual manipulation by the user in fashioning the final translation. **Translation retrieval** is a description of this process of selecting from the TM a set of translation records of maximum similarity to a given input. Note that although we will tend to refer to translation retrieval exclusively within the bounds of "TM systems" throughout this chapter, all methods and results described herein apply equally to the retrieval component of an overall EBMT system. Equivalently, the methods described here are equally applicable to full strings, phrases and any other segment granularity.

Naturally, at the time of retrieval, TM systems have no way of accessing the target language (L2) translation of the source language (L1) input, and hence the list of L2 translation candidates is determined based solely on L1 similarity between the current input and translation examples within the TM. This is based on the integral assumption that L1 inter-string structural and semantic similarities will be reflected in the L2 translations. Clearly, there is a design decision here as to just how linguistically robust the retrieval mechanism should be in determining L1 similarity, and how hard the system should try to find a translation candidate in the case that no similar translation

[1] See Trujillo (1999) for a basic overview of translation memories and the issues and technologies surrounding them, and Planas (1998) for a thorough review of commercial TM systems

record is immediately evident. The most naive retrieval mechanism is full-string boolean match, which will naturally produce high-quality results in the case of a hit. For the greater part, however, we cannot reasonably expect to find a duplicate of the L1 string in the TM, and must rely on some form of "fuzzy match" to retrieve similar but lexically differentiated translation candidates. In this, we are confronted with a trade-off between exhaustivity and speed, that is predictive accuracy and access/retrieval speed. Computational overhead tends to be at a premium with TM systems, as the user expectation is that the system will not impede the conventional translation process, but rather operate invisibly as an added extra in retrieving translation candidates and incrementally updating the TM on-stream. Similarly for EBMT, any additional computational overhead would be better invested in the downstream operation of translation adaptation, rather than in marginally boosting retrieval performance. We are thus interested in getting the fastest possible access times, while maintaining near-optimal retrieval performance.

This research owes much to the legacy of information retrieval (IR) research, due to the strong parallels between the IR task of extracting documents similar to a given query from a document set, and the TM task of extracting translation records similar to a given input from the TM. Many of the basic methods used herein are therefore taken directly from IR research. There are, however, subtle differences between IR and translation retrieval, an item we look to quantify, and apply in remodelling IR methods for our purposes.

In this chapter, we choose to focus on retrieval performance within non-segmenting languages, targeting Japanese as our source language for the greater part of this chapter. Non-segmenting languages are those which do not involve delimiters (i.e. spaces) between words, and include Japanese, Chinese and Thai. Our particular interest in non-segmenting languages relates to the accuracy/speed trade-off, in determining the relative impact of the orthogonal parameters of segmentation, segment order and segment contiguity on retrieval performance. That is, we seek to determine whether the unavoidable overhead associated with pre-segmenting the input into words or morphemes is commensurate with the resultant retrieval performance. This is achieved through comparison of **character-based indexing** (where the string is blithely segmented into its constituent characters or character chunks of fixed size) and **word-based indexing** (where a segmentation module is used to systematically segment the string into words), and compare their relative virtues. Similarly, we variously verify whether we gain anything by adopting more expensive segment order-sensitive approaches, over treating each L1 string as a "bag of words", and performing a simple boolean match for each component segment. We also test the effects of explicitly modelling segment contiguity as compared to treating matches over contiguous segments identically to matches over displaced segments.

In addition to adjusting the stringency of the match mechanism, there is room to adaptively speculate as to the L2 impact of different L1 sub-string correspondences. Ideally, we would like the TM system to be able to predict which terms in the input are of greater import in terms of translation similarity, preferably at no extra burden to the user. We look into this possibility through the medium of both fixed weighting schemes and a dynamic term weighting method.

In order to evaluate different matching methods, indexing options and segment weighting schemes, we require some framework for empirical evaluation of the system output. Traditionally, TM system evaluation has taken the form of subjective evaluation of the "usefullness" of the top n ranking outputs, generally scored according to some discrete scale. We propose a fully automated, objective evaluation method, based on identification of the optimal translation candidate(s) within the TM, and comparison of these with the actual system output. That is, we partition the TM dataset into training and test data, use the training data as the current TM, and treat the translation given for each input within the test data as the held-out model translation. We then determine those translation candidates of maximal L2 similarity within the TM, thereby modelling

the best possible translation output the system would be able to achieve within the constraints of the current TM context. By comparing the actual system output to these optimal translation candidates, we are able to empirically rate retrieval performance. This method both allows us to quantitatively compare the accuracies for different methods, and provides some insight into the degree of discrepancy in the case of non-optimal output.

To preempt the findings presented below, over a series of experiments we find that segmentation is detrimental to retrieval accuracy or, in other words, that it is better to retrieve over characters than over words. Furthermore, the bag-of-words methods we test are equivalent in translation accuracy to segment order-sensitive methods, but superior in retrieval speed. Finally, a local model of segment contiguity is beneficial for retrieval over character-based segments, and effective to a lesser degree for word-based indexing. We thus provide clear evidence that naive hare-like methods are at least as good as, and in some cases, superior to more stringent (tortoise-like) retrieval methods. That is, translation retrieval is one task where "slow and supposedly steady" does not necessarily correlate with enhanced translation performance.

Below, we first discuss the relationship between translation retrieval and IR (Section 2.2), before going on to describe parameters we consider to affect translation retrieval accuracy, namely segmentation, segment order and segment type (Section 2.3). We then present a range of both bag-of-words and segment order-sensitive string comparison methods in Section 2.4, and discuss methods for accelerating retrieval speed. In Section 2.5, we detail the evaluation methodology, before going on to evaluate the different methods with Japanese–English translation retrieval (Section 2.6) and English–Japanese translation retrieval (Section 2.7). Finally, we conclude the chapter in Section 2.8.

2.2 Translation Retrieval vs. Information Retrieval

Information retrieval (IR) can be defined as the task of searching a document set for documents relevant to a given query, and returning an arbitrary number of relevant documents, preferably ranked in some way. Queries constitute a description of the information content the user is after, in the form of a number of sentences or a string of keywords, as compared to documents, which tend to be the length of a full newspaper article or longer. Queries and documents are thus not directly comparable as in translation retrieval, and queries are seen as an abstraction of the user's informational needs. As a result, some form of "query interpretation" generally takes place prior to the document retrieval process, in the first place by filtering out "stop words" (a pre-determined set of closed-class words or words otherwise considered to be semantically vacuous). The reason for this is to reduce the search space and minimise spurious matches, as stop words tend to be contained in most documents with high frequency, irrespective of genuine relevance to the query. Additionally, the user does not expect the concept they seek to be expressed in an identical manner to the query, such that we gain by losing the linguistic structure of the query.

Due to the extended length of documents and potentially huge size of the document set, the actual retrieval process in IR is inevitably restricted to a boolean match over each of the terms derived from the query, and subsequent scoring of the match according to the three key factors of: (i) the intra-document frequency of a given term contained within the query ("term frequency" or *tf*), (ii) the inverse inter-document frequency of that term ("inverse document frequency" or *idf*), and (iii) the normalised document length (Salton and McGill 1983). In addition, a number of smoothing techniques are commonly employed to ensure high retrieval recall, including term expansion, whereby synonyms, collocates, and words displaying similar structural behaviour to terms contained in the original query, are added in to final query (Mandala *et al.* 1999).

In contrast to IR, translation retrieval revolves around more direct term correspondence between

the input and translation records contained in the TM. Unlike IR, the input is a direct expression of the user's L1 requirements, rather than an abstraction of informational need. In this sense, all terms have some part to play in translation retrieval, and a fully-matching translation record can realistically be expected to provide the closest translation correspondence. It would thus be inappropriate, and potentially harmful, to filter out functional words in the manner of IR, as we would simply lose disambiguating potential. The English strings "further up the street", "he went across the street when she did" and "the second street on the right", for example, would all collapse into "street" under the SMART (Salton 1971) stop word listing (see Section B.1), thereby deleting obviously valuable discriminatory information. That is not to say, however, that all terms contribute equally to retrieval accuracy, just that we should avoid completely factoring out any term from the input.

In the following, we discuss the effects of term frequency, inverse document frequency and document length normalisation in the context of translation retrieval.

2.2.1 Term Frequency

In IR, documents containing a given term with high frequency are commonly scored up over those containing the same term with lower or zero frequency, through the advent of term frequency (*tf*). This can be formalised in its simplest form as follows, where $freq_d(t_i)$ is the term frequency of term t_i in document d:

$$tf(t_i) = \frac{freq_d(t_i)}{\sum_{t_j \in d} freq_d(t_j)} \tag{2.1}$$

tf operates around the assumption that the frequency of a term reflects its salience to the topic of the containing document, and is generally normalised over the total number of terms in the document as indicated in Equation 2.1.

Term frequency has direct applicability within translation retrieval, in that translation records of maximum similarity to the input are going to contain the same terms in the same basic frequencies. Due to the direct linguistic comparability of the input and translation records, however, we would like to penalise both depleted and inflated segment counts over the input. For this purpose, we set the segment frequency in the input as a ceiling, and either explicitly or implicitly (in the case of the edit distance methods) normalise over both the input and translation record segment lengths, rather than just the document length as in IR. In this way, any segment frequency below the ceiling is going to score below segment frequencies close to the ceiling, due to a diminished raw *tf* score; additionally, the truncation of any segment frequency above the ceiling and subsequent normalisation over string length deflates the *tf* for high segment counts. The string length normalisation methods for each of the comparison methods are individually presented in Section 2.4.

2.2.2 Inverse Document Frequency

The second key IR technique is inverse document frequency (*idf*), where we set a fixed weight for each term according to the scaled inverse proportion of documents containing that term. Conventionally, a log scale is used to diminish the weight associate with outliers, as in:

$$idf_{IR}(t) = \log \frac{|D|}{|\{d : d \in D, d \ni t\}|} \tag{2.2}$$

where t represents a given term in the query, and D is the document set. Under this formalisation, a term contained exclusively within a small selection of documents will receive a higher weight than

one distributed uniformly over all documents, under the assumption that the former term is more specialised in usage, and hence a better indicator of true document content.

In translation retrieval, the inverse document (i.e. translation record) frequency of each segment in the input can similarly be used to weight segment hits in the TM, and the combination of *idf* weights for matching segments, used to rate the overall quality of match. We must tread cautiously in applying inverse document frequency directly to the task of translation retrieval, however, in order to avoid the situation of an especially highly-weighted segment winning out unconditionally over any combination of other lower-scoring segments. This could occur for a segment occurring in only one of a large number of translation records, finding its way into an input otherwise comprised of uniformly distributed segments. Additionally, we want to avoid any segment having a weight of zero, as all segments have some role to play in determining the degree of string correspondence. We thus reformulate the original idf_{IR} as idf_{TR} for translation retrieval purposes:

$$idf_{TR}(s) = \min\left(\log \frac{|TM|+1}{|\{tr : tr \in TM, tr \ni s\}|}, max\right) \tag{2.3}$$

where s is a segment contained in the TM, TM is the translation memory, and tr is a translation record contained therein; max is a fixed constant which acts as a ceiling on *idf* weights, and the addition of one to the TM size guarantees the generation of a positive *idf* value. This provides a dynamic segment weighting method, customisable to any domain or language. It is equally possible to enforce static weights over different segment types, as we discuss in Section 2.3.3. In this respect, *idf* is an optional extra in translation retrieval, rather than an integral element.

2.2.3 Document Length Normalisation

The need for document length normalisation in IR is attributable to the often large-scale variance of document lengths in the document set. There is also subtle interplay between the probability of longer documents being retrieved (due to their greater word coverage), and the probability of them being relevant to the query (due to their inflated coverage of a range of topics), with the latter increasingly outstripping the former for longer documents (Singhal *et al.* 1996). Document length normalisation is thus used to correct this imbalance and operates over the document set, independently of the query length. In translation retrieval, on the other hand, retrieval centres around maximal correspondence between the input and translation records, and operates over strings and not documents. Consequently, normalisation must be over the lengths of *both* of the strings being compared, and not just the translation records. In translation retrieval, string length normalisation is especially important for methods which consider only congruencies between two strings, and not divergences. In such methods, the lengths of the two strings in question provide a description of the maximum possible correspondence between the strings, and can hence be used to normalise the level of congruence according to the associated degree of divergence. Note that string length normalisation is optional for methods which model only the level of divergence between two strings (i.e. edit distance methods — see below), as the degree of divergence constitutes an inherent description of relative string length (in that, given the same level of congruence, strings of similar length will always have a lower lever of divergence than strings of differing length).

To summarise, the three primary IR term weighting methods of *tf*, *idf* and document length normalisation, are, for the purposes of translation retrieval, translated into truncated *tf*, truncated *idf* as an optional extra, and combined translation record and input string length normalisation.

2.3 Parameters Affecting Translation Retrieval Performance

In this section, we review three parameter types that we suggest impinge on retrieval performance, namely segmentation, segment order, and segment type.

2.3.1 Segmentation

Despite non-segmenting languages such as Japanese not making use of segment delimiters, it is possible to artificially partition off a given string into constituent morphemes through the process of "segmentation", using systems such as ChaSen (Matsumoto *et al.* 1999), JUMAN (Kurohashi and Nagao 1998) and ALT-JAWS[2]. The resultant segments constitute affixes, case markers and stand-alone words, which we will collectively term as "words" for the remainder of this chapter. Using segmentation to divide strings into component words has the obvious advantage of clustering characters into semantic units, which in the case of ideogram-based languages such as Japanese (in the form of kanji characters) and Chinese, generally disambiguates character meaning. The kanji character 弁 [*ben*], for example, can mean any of "to discern/discriminate", "to speak/argue" and "a valve" in Japanese, but word context easily resolves such ambiguity. In this sense, our intuition is that segmented strings should produce better results than non-segmented strings.

Looking to past research on string comparison methods for TM systems, almost all systems involving Japanese as the source language rely on segmentation (Nakamura 1989; Sumita and Tsutsumi 1991; Kitamura and Yamamoto 1996; Tanaka 1997), with Sato (1992) and Sato and Kawase (1994) providing rare instances of character-based systems.

By avoiding the need to segment text, we:

- alleviate the computational overhead associated with segmentation modules
- avoid the need to commit ourselves to a particular analysis type in the case of ambiguity or unknown words
- avoid the need for stemming/lemmatisation
- to a large extent get around problems related to the normalisation of lexical alternation

All of these are issues which hit at the heart of morphological analysis.

The alleviation of the need to segment both the input and translation records in the TM (i.e. reliance on character-based indexing) could accelerate both the retrieval and TM update processes. Admittedly, for an on-line system, we can expect translation records in the TM to be pre-segmented, but there is still the need to segment each input, using the same segmentation method as was used to segment the TM in order to maintain consistency.

For an application such as a TM, the segmentation module has no recourse to user feedback in disambiguating lexical ambiguity, and must commit itself to the most plausible analysis, including segmenting unknown words appropriately and consistently. Below, we discuss this issue within the domain of a technical text, and rate the actual performance of the ChaSen system in segmenting unknown words.

Stemming and lemmatisation tend not to pose too great a problem for Japanese, as in the segmentation output, non-conjugating stems and morphemes/conjugational affixes tend to be segmented off individually, presenting the stem as a single segment. If we wish to filter affixes and

[2] `http://www.kecl.ntt.co.jp/icl/mtg/resources/altjaws.html`

other functional words out of the output from the segmentation module, however, we require part-of-speech (POS) information, placing an additional burden on the segmentation module and forcing it to commit to a higher level of information.

Lexical alternation characteristically occurs for nominalised verbs with kanji stems, when conjugational information which would ordinarily be expressed as a hiragana suffix to the kanji stem, is optionally conflated with the stem.[3] Due to the lexical inconsistency of such words, it is necessary to normalise the spell-out in some way for segmented text. We investigate this possibility through the use of ALT-JAWS in Section 2.6.4. With unsegmented text, on the other hand, we maintain the same kanji content, allowing for a match at this level irrespective of whether we have a hiragana suffix or not.

While segmentation provides implicit disambiguation of kanji sense, treating each kanji character as an individual segment in character-based indexing has the advantage of providing a primitive semantic class index. In technical domains in particular, it often occurs that a particular kanji occurs in only one of its semantic realisations, a fact which is borne out by our 弁 [*ben*] example, for which all 186 occurrences in the TM corpus used in evaluation are in the "valve" sense.[4] Prising kanji characters apart from their word contexts gives us direct access to this semantic class-type information, and gives a reasonable indication of similarity in the case of partial kanji overlap between two strings. This is particularly useful in cases where the same kanji is used in different words, as would not match under character-based indexing. With word-based indexing, we would have to rely on some form of higher-level semantic processing (e.g. linked to a thesaurus) to derive the same semantic link between the two words with kanji overlap.

Note that unless otherwise stated, all methods described below are applicable to both word- and character-based indexing. To avoid confusion between the two lexeme types, we will collectively refer to the elements of indexing as "segments".

2.3.2 Segment Order

Our expectation is that translation records that preserve the segment order observed in the input string will provide closer-matching translations than translation records containing those same segments in a different order. Naturally, enforcing preservation of segment order is going to place a significant burden on the matching mechanism, in that a number of different substring match schemata are inevitably going to be produced between any two strings, each of which must be considered on its own merits.

As far as we are aware, there is no TM system operating from Japanese that does not rely on word/segment/character order to some degree. Tanaka (1997) uses pivotal content words identified by the user to search through the TM and locate translation records which contain those same content words in the same order and preferably the same segment distance apart. Words in the local context of the pivot words in the original text, are weighted according to a decay function over the distance from the pivot words. Nakamura (1989) similarly gives preference to translation records in which the content words contained in the original input occur in the same linear order, although there is the scope to back off to translation records which do not preserve the original

[3] See Fujii and Croft (1993) and Baldwin and Tanaka (1999) for details of other, less-widespread types of lexical alternation, including katakana spelling inconsistencies, kanji–kana inter-replacement, and kanji spelling inconsistencies (i.e. there being multiple versions of the same basic kanji character, differentiated stylistically if at all), all of which produce analogous lexical inconsistency side-effects. Character-based indexing would provide a solution in the instance of katakana spelling inconsistencies, but not the latter cases. Lexical normalisation, on the other hand, offers a solution to all of these forms of lexical alternation.

[4] Parallels can be drawn here with "one-sense-per-discourse" constraint of Gale *et al.* (1992), which states that a given word will only ever occur in a single sense, for a given document or domain.

word order. Sumita and Tsutsumi (1991) take the opposite tack in iteratively filtering out NPs and adverbs to leave only functional words and matrix-level predicates, and find translation records which contain those same key words in the same ordering, preferably with the same segment types between them in the same numbers. Nirenburg *et al.* (1993) propose a stratified word order-sensitive metric based on "string composition discrepancy", and incrementally relax the restriction on the quality of match required to include word lemmata, word synonyms and then word hypernyms, increasing the match penalty as they go. Sato and Kawase (1994) employ a more local model of *character* order in modelling similarity according to N-grams fashioned from the original string.

The greatest advantage in ignoring segment order is computational, in that we significantly reduce the search space. The arguments for segment order are largely based around the intuition that it should aid translation retrieval, a claim we look to verify. Below, we analyse whether the gain in speed to bag-of-words methods outweighs any losses in retrieval accuracy over segment order-sensitive methods. We also look to local models of segment order which partially supplement the segment order insensitivity of bag-of-words methods, thereby maintaining and potentially enhancing retrieval speed while allowing for limited segment order sensitivity.

Japanese is a relatively free word order language, which could possibly dilute the performance gains to sequential match methods. Having said this, while Japanese certainly has the scope for word order variation (or strictly speaking, case-marked noun and adverb phrase permutation within a clause, as the predicate is always clause-final in written Japanese), it is rarely seen in formal or technical documents. In this sense, we can largely ignore the effects of Japanese word order variation for written language domains, such as we target in evaluation.

2.3.3 Segment Type

As touched upon above, Japanese is made up of multiple script types, namely the native **katakana** and **hiragana** syllabries, and the **kanji** ideographic system. Katakana and hiragana (collectively termed **kana**) are isomorphic and fully inter-replaceable schematically, although there is a clear division of labour between the two, with katakana generally used for words of foreign origin, and hiragana for adverbs, functional words and conjugational affixes. Most nouns and verb and adjective stems are lexicalised in kanji, meaning that words with some kanji content are generally open class, and those made up entirely of hiragana are generally closed class.

This observation constitutes a heuristic of immediate applicability, and led Fujii and Croft (1993) to delete all hiragana from documents in character-based IR, and base retrieval around single kanji characters and contiguous katakana strings. An alternative to this extreme view of character type is to weight down segments made up entirely of hiragana, over those containing one or more kanji. In this manner, we can retain all segments but weight each according to its translation salience. We test a number of static weighting schemes in this regard in Section 2.6.5. We can also adopt dynamic term weighting methods from the realm of IR, as proposed below in Section 2.2. Static weighting schemes offer advantages in terms of scalability and the ease with which the TM can be updated. Dynamic weighting schemes, on the other hand, have superior domain adaptability and can be fine-tuned to the data.

2.3.4 Match stratification

One method which allows us to model multiple parameter types in parallel, is stratification of the match process (e.g. see (Nirenburg *et al.* 1993; Planas and Furuse 1999; Planas and Furuse 2000)). Under stratified (multi-stratum) translation retrieval, the types of string comparison methods described herein are combined over multiple parallel levels of representation, possibly including the lexical, lemma, part-of-speech (POS) and semantic levels. By operating over multiple data types,

we are able to pick up on subtle structural and conceptual correspondences not available under straight lexical comparison. This provides a valuable smoothing mechanism and a direct means of coping with effects such as lexical normalisation, by way of a simple extension to the existing matching apparatus. Clearly, however, we are faced with the same opposition of parameter types as are established in this research. We thus choose to ignore stratification for the purposes of this research, and focus on comparison of the different parameter types.

One area in which stratification may affect our results is the relative retrieval performance of the different string comparison methods, where the smoothing nature of stratification may cover the inadequacies of different parameter configurations. Here, we make the claim that any reduction in error availed through stratification would be proportionally equivalent for the different methods. Any ranking of the different methods obtained through analysis of lexical match and any conclusions drawn based thereupon, should therefore hold under stratification.

2.4 String Comparison Methods

Due to our interest in the effects of segment order, segmentation and segment type, we must have a selection of string comparison methods compatible with the various permutations of these three parameter types. We choose to look at a number of bag-of-words[5] and sequential segment matching methods which are compatible with both character-based and word-based indexing, and test different segment weighting schemes within each such system configuration.

It is possible to plot different string comparison techniques on a continuum of match stringency, as illustrated in Figure 2.1. The bag-of-words, or boolean segment, matching mechanism is suggested to lie toward the computationally lightweight end of this continuum, outdone in simplicity only by full-string boolean match. Sequential segment match, is computationally weightier than boolean segment match, as multiple sequential segment correspondences can exist between the two strings in question, which must be identified and individually evaluated. Perhaps the most exhaustive matching method available is contiguous sequential segment match, where segment contiguity and sequentiality are balanced off against each other.

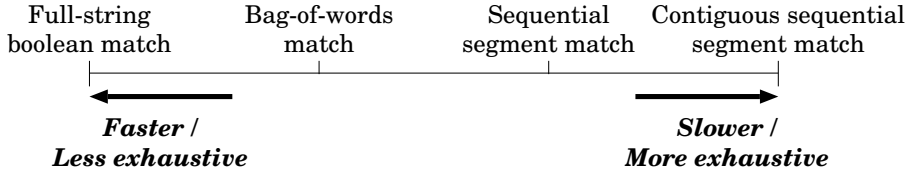

Figure 2.1: The continuum of match exhaustivity vs. speed

The particular bag-of-word approaches we target are the vector space model (Manning and Schütze 1999:p300) and "token intersection", a simple ratio-based segment comparison method. For segment order-sensitive approaches, we test two varieties of edit distance and similarity, and also "weighted sequential correspondence".

All of the comparison techniques other than edit distance, empirically describe the similarity $sim \in [0, 1]$ between two strings $S = s_1 s_2 ... s_m$ and $T = t_1 t_2 ... t_n$.[6] The different implementations

[5]The term "bag-of-words" is misleading in the sense that it is orthogonal to the issue of character- vs. word-based indexing. While noting this anomaly, we retain the term due to it being well-established nomenclature within NLP circles.

[6]Note that the ordering of the strings is arbitrary, and that all the comparison methods described herein are commutative for the given implementations.

of edit distance determine the distance rather than similarity between arbitrary strings in terms of the number of primitive edit operations required to transform one string into the other. For the similarity-based methods, *greater* values signify closer correspondence between the strings, whereas with edit distance, *smaller* values signify closer correspondence. It is a trivial process, however, to transform edit distance into edit similarity (see below).

One feature of all comparison methods given here is that they have fine-grained discriminatory potential and define a partial ordering of translation records according to their relative similarity to the input. This was a deliberate design decision, and aimed at system customisability, i.e. the ability to adjust the system to reliably output n translations of maximum similarity to the input, or alternatively use the scores returned by the various methods to threshold over a user-defined cutoff of desired similarity. It also allows us to pinpoint a single translation record of maximum similarity (leaving the issue of ties aside for the time being), a quality that is desirable for EBMT applications. In this, we set ourselves apart from the research of Sumita and Tsutsumi (1991), for example, who judge the system to have been successful if there are a total of 100 or less outputs, a selection of which are useful.

All methods are formulated to operate over arbitrary *sweight* schemata, although we leave the issue of exactly what form the *sweight* schema should take until Section 2.6.

2.4.1 Bag-of-Words String Comparison

Here, we describe our adopted implementations of the vector space model and token intersection, the two bag-of-words methods. We illustrate the scoring process for both bag-of-words and segment order-sensitive methods according to the task of translation retrieval for string (1), from the toy TM made up of strings (2a), (2b), (2c) and (2d), as below (segment delimiters given as '·'):[7]

(1) 冬 · の · 雨 [*huyu·no·ame*] "winter rain"
(2a) 夏 · の · 雨 [*natu·no·ame*] "summer rain"
(2b) 雨 · の · 夏 [*ame·no·natu*] "a rainy summer"
(2c) 雨 · の · 冬 [*ame·no·huyu*] "a rainy winter"
(2d) 真 · 冬 · の · 雨 [*ma·huyu·no·ame*] " mid-winter rain"

We assume a unit *sweight* for all contained segments for the purposes of illustration. Based on the L2 similarity between strings (2) and string (1), we would hope that either (2a) or (2d) would come out as having the highest level of correspondence to (1).

Vector Space Model

Within our implementation of the **vector space model** (VSM), the segment content of each string is described as a vector, made up of a single dimension for each segment token occurring within S or T. The value of each vector component is given as the weighted frequency of that token according to its *sweight* value. The string similarity of S and T is then defined as the cosine of the angle between vectors $\vec{S}$ and $\vec{T}$, respectively, calculated as:

$$\cos(\vec{S},\vec{T}) = \frac{\vec{S}\cdot\vec{T}}{|\vec{S}||\vec{T}|} = \frac{\sum_j s_j t_j}{\sqrt{\sum_j s_j^2}\sqrt{\sum_j t_j^2}} \tag{2.4}$$

Note that VSM considers only (weighted) segment frequency and is insensitive to segment order.

[7] So as to filter out any differences between character- and word-based indexing, the strings have been chosen such that all "words" are in fact single characters. Retrieval performance is thus identical for the two indexing paradigms.

The similarity for string (1) over both strings (2a) and (2b) is calculated to be $\frac{1\times0+1\times1+1\times1+0\times1}{\sqrt{3}\sqrt{3}} = \frac{2}{3}$. (2d) fares slightly better, at a similarity of $\frac{1\times1+1\times1+1\times1+0\times1}{\sqrt{3}\sqrt{4}} = \frac{\sqrt{3}}{2} \approx 0.866$. The greatest similarity, however, is attained for (2c), at $\frac{1\times1+1\times1+1\times1}{\sqrt{3}\sqrt{3}} = \frac{3}{3} = 1$. That is, VSM considers strings (1) and (2c) to be identical. In this case, therefore, we obviously need to look beyond segment content to segment order, to escape from this misrepresentation of the data.

Token Intersection

The **token intersection** of S and T is defined as the cumulative intersecting frequency of tokens appearing in each of the strings, normalised according to the combined segment lengths of S and T using Dice's coefficient. Formally, this equates to:

$$tint(S,T) = \frac{2 \times \sum_{e \in S,T} \min\left(freq_S(e), freq_T(e)\right)}{len(S) + len(T)} \tag{2.5}$$

where each e is a segment occurring in either S or T, $freq_S(e)$ is defined as the *sweight*-based frequency of token e occurring in string S, and $len(S)$ is the segment length of string S, that is the *sweight*-based count of segments contained in S (similarly for T).

Note that segment order does not take any part in calculation.

The similarity for string (1) over both strings (2a) and (2b) is calculated to be $\frac{2\times(0+1+1+0)}{3+3} = \frac{4}{6} = \frac{2}{3}$, the same result as for VSM above. (2d) scores significantly higher at $\frac{2\times(1+1+1+0)}{3+4} = \frac{6}{7}$, but (2c) again produces the highest score, at a perfect $\frac{2\times(1+1+1)}{3+3} = \frac{3}{3} = 1$.

2.4.2 Segment Order-sensitive String Comparison

Next, we give a brief overview of the segment order-sensitive string comparison methods utilised in this research.

Edit Distance and Similarity

Essentially, the segment-based **edit distance** between strings S and T is the minimum number of primitive edit operations on single segments required to transform S into T (and vice versa). Conventionally, edit operations are *segment equality* (segments s_i and t_j are identical), *segment deletion* (delete segment s_i), *segment insertion* (insert segment a into a given position in string S), and *segment substitution* (substitute segment s_i for segment t_j). The cost associated with each operation is determined by the *sweight* values of the operand segments, with the exception of segment equality which is defined to have a fixed cost of 0. This produces the desired quality that deletion and insertion operations over the same segment types are equal in cost, hence maintaining commutativity. Edit distance over all four operations at unit cost for segment deletion, insertion and substitution (i.e. *sweight* = 1 for all segment types), leads to the "Levenshtein distance" between a given string pair.

Segment substitution can be seen to be a compound operation, first deleting a segment at a given position in string S, and then inserting a second segment into that same position. By maintaining segment deletion and insertion as separate operations, our intuition is that we should get a stronger sense of the true effort required to coerce an arbitrary string pair together, as a translator or EBMT system would have to do in adapting the final translation candidate to the needs of the original L1 input. Additionally, we are able to distinguish between segment substitutions for strings of equal length, and segment deletions/insertions for strings of disparate length (see the numeric example below). We test whether this extra discriminatory potential contributes to retrieval performance,

```
1  # INITIALISE TOP AND LEFT BOUNDARY VALUES
2  set array[0][0] = 0
3  for i = 1 .. m
4    set array[i][0] = array[i-1][0] + sweight(s_i)
5  endfor
6  for j = 1 .. n
7    set array[0][j] = array[0][j-1] + sweight(t_j)
8  endfor
9
10 # ITERATIVELY CALCULATE DISTANCES ROW-WISE
11 for i = 1 .. m
12   for j = 1 .. n
13     if s_i = t_j
14       set array[i][j] = min(array[i-1][j] + sweight(s_i),
15                             array[i][j-1] + sweight(t_j),
16                             array[i-1][j-1])
17     endif
18     else
19       set array[i][j] = min(array[i-1][j] + sweight(s_i),
20                             array[i][j-1] + sweight(t_j),
21                             array[i-1][j-1] + max(sweight(s_i),sweight(t_j)))
22     endelse
23   endfor
24 endfor
25
26 # RETURN OVERALL EDIT DISTANCE
27 return array[m][n]
```

Figure 2.2: The classic edit distance algorithm

by evaluating both classic **4-operation edit distance** and simplified **3-operation edit distance** (classic edit distance without the substitution operator).

Despite the high match stringency of edit distance, it has benefited from exhaustive optimisation (Wagner and Fisher 1974; Masek and Paterson 1980; Kukich 1992), culminating in efficient dynamic programming (DP) algorithms. One particularly accessible algorithm which has gained widespread acceptance is that of Wagner and Fisher (1974), which is an implementation of classic 4-operation edit distance and runs in $\Theta(mn)$ time (as do the other two DP algorithms presented below), where m and n are the segment lengths of the target strings. The algorithm can be formalised as detailed in Figure 2.2, for target strings $S = s_1 s_2 ... s_m$ and $T = t_1 t_2 ... t_n$. In its presented form, the algorithm is applicable to any *sweight* schema made up of positive and zero values.

As a distance, smaller values indicate greater similarity for edit distance, and identical strings have edit distance 0. It is possible to normalise operation edit distance $edist_{4op}$ over the lengths $len(S)$ and $len(T)$ of target strings S and T, and further convert the normalised edit distance $edist^*_{4op}$ into **4-operation edit similarity** sim_{4op} by way of the trivial transformations in Equation 2.6:

$$sim_{4op}(S,T) = 1 - edist^*_{4op}(S,T) = 1 - \frac{edist_{4op}(S,T)}{\max(len(S), len(T))} \tag{2.6}$$

To return to our example strings from above, 4-operation edit distance generates a distance of 1 (and similarity of $\frac{2}{3}$) for (2a), a distance of 1 (and similarity of $\frac{3}{4}$) for (2d), and a distance of 2 (and similarity of $\frac{1}{3}$) for (2b) and (2c). For the first time, therefore, we are able to produce the desired result of (2a) and (2d) being most similar to (1) (or in the case of 4-operation edit similarity, (2d) providing the closest match), due to the segment order-sensitivity of this method. Note that the edit distances of these two strings are identical, despite them being due to a substitution in the first case and a deletion in the second. With 4-operation edit similarity, on the other hand, normalisation according to the segment lengths of the two strings discriminates between these two cases. In this sense, normalisation of 4-operation edit distance into a scaled similarity offers limited means for discrimination between segment substitution and deletion/insertion.

The simplified **3-operation edit distance** algorithm is presented in Figure 2.4.2, and can be seen to largely parallel the basic algorithm. In fact, the only difference is for cases of non-matching segments, when we disallow substitutions (at line 21 within the algorithm description). Again, as presented, the algorithm can operate over any *sweight* schema.

As for 4-operation edit distance, it is possible to normalise the final 3-operation edit distance $edist_{3op}$ over the lengths $len(S)$ and $len(T)$ of target strings S and T, and use this to generate **3-operation edit similarity** sim_{3op}, through Equation 2.7:

$$sim_{3op}(S,T) = 1 - edist^*_{3op}(S,T) = 1 - \frac{edist_{3op}(S,T)}{len(S) + len(T)} \tag{2.7}$$

3-operation edit similarity computed in this fashion is identical to the "sequential correspondence" method of Baldwin and Tanaka (2000b).

To return to our example strings from above, 3-operation edit distance generates a distance of 1 (and similarity of $\frac{6}{7}$) for (2d), a distance of 2 (and similarity of $\frac{2}{3}$) for (2a), and a distance of 4 (and similarity of $\frac{1}{3}$) for both (2b) and (2c). Here, (2d) is a clear winner over (2a) in terms of both edit distance and normalised similarity, unlike 4-operation edit distance, where the two scored equally.

Weighted Sequential Correspondence

Weighted sequential correspondence (originally proposed in Baldwin and Tanaka (2000b))—the last of the segment order-sensitive methods—goes one step further than edit distance in analysing

```
1  # INITIALISE TOP AND LEFT BOUNDARY VALUES
2  set array[0][0] = 0
3  for i = 1 .. m
4    set array[i][0] = array[i-1][0] + sweight(s_i)
5  endfor
6  for j = 1 .. n
7    set array[0][j] = array[0][j-1] + sweight(t_j)
8  endfor
9
10 # ITERATIVELY CALCULATE DISTANCES ROW-WISE
11 for i = 1 .. m
12   for j = 1 .. n
13     if s_i = t_j
14       set array[i][j] = min(array[i-1][j] + sweight(s_i),
15                             array[i][j-1] + sweight(t_j),
16                             array[i-1][j-1])
17     endif
18     else
19       set array[i][j] = min(array[i-1][j] + sweight(s_i),
20                             array[i][j-1] + sweight(t_j))
21                                              # SUBSTITUTION OPERATION REMOVED HERE
22     endelse
23   endfor
24 endfor
25
26 # RETURN OVERALL EDIT DISTANCE
27 return array[m][n]
```

Figure 2.3: The 3-operation edit distance algorithm

```
1  # INITIALISE TOP AND LEFT BOUNDARY VALUES
2  for i = 0 .. m
3    set m[i][0] = 0
4    set array[i][0] = 0
5  endfor
6  for j = 1 .. n
7    set m[0][j] = 0
8    set array[0][j] = 0
9  endfor
10
11 # ITERATIVELY CALCULATE SCORES ROW-WISE
12 for i = 1 .. m
13   for j = 1 .. n
14     if s_i = t_j
15       set m[i][j] = min(m[i-1][j-1]+1,MAX)
16       set array[i][j] = max(array[i-1][j], array[i][j-1],
17                                            array[i-1][j-1] + m[i][j]*sweight(s_i))
18     endif
19     else
20       set m[i][j] = 0                        #SATO(92): set m[i][j] = m[i-1][j-1]
21       set array[i][j] = max(array[i-1][j],
22                             array[i][j-1],
23                             array[i-1][j-1])
24     endelse
25   endfor
26 endfor
27
28 # RETURN OVERALL CORRESPONDENCE SCORE
29 return array[m][n]
```

Figure 2.4: The substring match scoring algorithm for weighted sequential correspondence

not only segment sequentiality, but also the contiguity of matching segments. Given the input $\alpha_1\alpha_2\alpha_3\alpha_4$, the edit distance and similarity methods would suggest $\alpha_1\beta_1\alpha_2\beta_2\alpha_3\beta_3\alpha_4$ and $\alpha_1\alpha_2\alpha_3\alpha_4\beta_1\beta_2\beta_3$ as equally close matches, for example, despite the second of these being more likely to produce a translation at least partially resembling that of the input string.

We get around this by associating an incremental weight (orthogonal to our *sweight* weights) with each matching segment assessing the contiguity of left-neighbouring segments, in the manner described by Sato (1992) for character-based matching. Namely, the kth segment of a matched substring is given the multiplicative weight $\min(k, Max)$, where Max is a positive integer.[8] This weighting up of contiguous matches is facilitated through the DP algorithm given in Figure 2.4. The essence of the method is to use array `m` to keep track of the truncated number of contiguous segment matches leading up to the current segment (calculated according to $\min(k, Max)$), and weight the current segment match based on the match contiguity score pre-stored in `m[i][j]`.

The final similarity is determined as:

$$wseq^*(S,T) = \frac{2 \times wseq(S,T)}{len(S) + len(T)} \tag{2.8}$$

where $wseq(S,T)$ is the maximum weighted sequential correspondence between S and T, as determined by the algorithm in Figure 2.4.

One key area in which our algorithm differs from that of Sato (1992), is that we reset `m[i][j]` to zero in the case of a non-matching segment, whereas Sato carries the degree of contiguity over from `m[i-1][j-1]` (in the manner indicated in line 21 of the algorithm description). Thus, if we have a contiguously matching sub-string of length Max or more early on in the match process, any subsequent segment-level matches will be treated as if they were contiguous with the original matching sub-string. In our implementation, we reset the match counter to zero whenever a non-matching segment is encountered, and thereby measure the true contiguity of each matching sub-string. In evaluation, we analyse the implications of this difference on retrieval accuracy.

Turning once again to our example TM, the similarity for (2a) is $\frac{2\times3}{6+6} = \frac{1}{2}$, the similarity for (2d) is $\frac{2\times6}{6+10} = \frac{3}{8}$, and that for (2b) and (2c) is $\frac{2\times1}{6+6} = \frac{1}{6}$. In this way, results strikingly similar to those for 3-operation edit distance are attained.

2.4.3 Enhanced segment order sensitivity: N-gram methods

All of the above methods can be run over any type of ordered, segmented data, noting that the segment order is ignored by the bag-of-words methods. By adjusting the data to fuse together clusters of N adjacent segments into N-grams, it is possible to generate an explicit model of local segment context. This is particularly useful for the bag-of-words methods as it provides a crude model of both segment order and contiguity, but is also valuable for the edit distance methods as it complements the implicit modelling of segment order with local segment contiguity. Here, N-grams can be fashioned either from individual characters in character-based indexing, or from words in word-based indexing. The modelling of local segment contiguity in this manner is thus orthogonal to the two indexing paradigms.

N-gram models have seen widespread usage in NLP, IR and speech recognition research, due to their easy applicability and high versatility. The main failing documented in past research has been that smaller values of N often do not produce sufficient segment discrimination and are unable to model long-distance segment dependency, whereas larger values of N quickly run into problems with data sparseness (Brown *et al.* (1992), Robertson and Willett (1998), Manning and Schütze

[8] Observe that the case of $Max = 1$ corresponds to the complement of 3-operation edit distance, or sequential correspondence in Baldwin and Tanaka (2000b).

(1999), *inter alia*). One commonly-used method to combine the relative advantages of different N-gram orders is to use some form of smoothing or backing-off, combined different N-gram sizes into a single framework. In our case, rather than introducing any new computational apparatus, we adjust the input to the different string comparison methods, in the form of simple unigram (1-gram), pure bigram (2-gram), and mixed unigram/bigram modelling. The unigram method equates to the basic word and character segmentation methods described above. With the bigram method, we mutate each string by clustering each adjacent segment pair into a single segment. In the mixed unigram/bigram method, on the other hand, we generate bigrams from adjacent segments while maintaining the base unigrams (in their original order), interleaving the two so as to preserve the original segment ordering; in terms of smoothing, this roughly equates to weighting unigrams and bigrams equally. From string (2a), for example, we would generate the following variants (common to both character- and word-based indexing):

Basic unigram version:	夏 · の · 雨
Bigram variant:	夏の · の雨
Mixed unigram/bigram variant:	夏 · 夏の · の · の雨 · 雨

Due care must be exercised in mutating the strings to maintain a final segment order consistent with that of the original string, in order for the segment order-sensitive methods to operate effectively. The derived strings are then supplied to the various string comparison methods in the indicated formats. No special tweaking of the string comparison methods in needed in order to run them over N-gram enhanced strings, just as long as a single segment contiguity model is applied consistently for all strings.

2.4.4 Retrieval Speed Optimisation

In their existing forms, the bag-of-words methods hold an absolute computational advantage over the segment order-sensitive methods in terms of the cost for a single string comparison, despite the relative frugality of the DP implementations of the segment order-sensitive methods. All is not lost, however, as we are able to reduce the number of string comparisons required to determine the best match, under the assumption that we are after only the best-scoring match(es) and not a full ranking of all translation candidates. This is achieved through analysis of the segment length of translation candidates (segment order-sensitive methods only), and segment overlap with the input (all methods). There is also scope for minimal savings with the edit distance and similarity methods, in prematurely exiting out of sub-optimal string matches. All acceleration methods discussed here are lossless, and simply prune off sub-optimal translation records.

First, by pre-computing and storing the weighted segment length of each string (weighted according to both the *sweight* values and the sequentiality increment in the case of weighted sequential correspondence), we can use the current top-ranking score α and weighted segment length of the input $len(IN)$, to determine upper and lower bounds on string lengths that could possibly better that score for the segment order-sensitive methods. While the bounds can be derived trivially from the basic scoring functions for each method, in the interests of completeness, we present the various bounds for each method in Table 2.1.

Clearly, we will not be able to rely on previous matches in setting α (the score for the current best match) for the first iteration. For the (unnormalised) edit distance methods, we get around this by initialising α to $len(IN)$, which is equivalent to setting the empty string as the closest match. For the remaining similarity-based methods, however, we hold back from applying the bounds until after we have calculated the similarity for the first string, and can use the resultant

String comparison method	*Upper bound on string length*	*Lower bound on string length*	*Halting condition*
4-op edit distance	$len(IN) + \alpha$	$len(IN) - \alpha$	$\beta > \alpha$
4-op edit similarity	$\frac{len(IN)}{\alpha}$	$\alpha len(IN)$	$1 - \frac{\beta}{\max(len(IN), len(TM_i))} < \alpha$
3-op edit distance	$len(IN) + \alpha$	$len(IN) - \alpha$	$\beta > \alpha$
3-op edit similarity	$\frac{(2-\alpha)len(IN)}{\alpha}$	$\frac{\alpha len(IN)}{2-\alpha}$	$1 - \frac{\beta}{len(IN)+len(TM_i)} < \alpha$
Weighted seq corr	$\frac{(2-\alpha)len(IN)}{\alpha}$	$\frac{\alpha len(IN)}{2-\alpha}$	$\frac{2\beta}{len(IN)+len(TM_i)} < \alpha$

Table 2.1: Bounds on string lengths and halting conditions for the various string comparison methods

α value. Note that α for the straight edit distance methods is an (unnormalised) edit distance, whereas that for all other methods is a scaled similarity in the range $[0, 1]$.

It is important to realise that all bounds are static for a given α and $len(IN)$, and need be recomputed only when the current best match is bettered. Note also that the string lengths are fixed and can be cached for as long as the *sweight* schema remains unchanged. For static segment weighting methods, therefore, this dependence on length poses no additional burden, and the database of translation record lengths can be updated incrementally whenever additions of translation records are made. With the truncated *idf* dynamic weighting method, on the other hand, we must recompute the segment length of each translation record overlapping in segment content with the incoming translation records, for every TM update. This form of optimisation is thus less effective for dynamic weighting methods.

We can also limit the search space for all methods by using an "inverted file" description of the TM to identify those translation records with some segment overlap with the input. An inverted file is simply a list of all segments realised within the TM, and for those translation candidates containing a given segment, a description of segment frequency. Despite the minimalist nature of the inverted file (e.g. it does not contain information about segment order[9]), it (1) allows us to exclude all translation records with no common segment component with the input, from the search process, and (2) provides an immediate indication of the quality of match possible with each translation record overlapping in segment content with the input. All of this is possible within a compact, easy-to-maintain format.

For the bag-of-words approaches, the inverted file supplies data to plug directly into the scoring functions, in combination with the weight for each segment. For the segment order-sensitive approaches, on the other hand, the segment overlap between each translation record and the input, combines with the scoring function to determine the optimal match quality (attained only in the case that the overlapping segments occur in identical order in the two strings). The method for computing the optimal score for each translation record differs according to the string comparison method, and is given on the lefthand side of the various inequalities in the "halting condition" column of Table 2.1. Here, β is the segment overlap between the input and current translation

[9]Although it is possible to use N-gram methods to encode segment order information, as per Nagao and Mori (1994).

record, α is the current best matching score, and $len(IN)$ and $len(TM_i)$ are the lengths of the input and current translation record, respectively.

In this way, we can order translation records according to the degree of match potentiality, and stop matching when the highest ranking translation record not yet processed, has an optimal score inferior to the best-scoring match to that point, in the manner presented in the "halting condition" column of Table 2.1.

While this method reduces the seach space considerably for most methods, there is a significant computational overhead to establishing an optimal score for each translation record sharing some segment content with the input, and subsequently sorting the translation records according to their optimal score. Over the small-scale TM used in evaluation, this overhead was not probitively expensive, but it could prove too great for larger-scale TM's (see Section 2.6.8 for further discussion of this point). One lossy method of reducing the burden of pre-processing, could be to do a beam search over the translation records with the greatest segment overlap with the input (irrespective of the correlation of this overlap to the method-specific scoring system), thus partially eliminating expensive floating point operations. Alternatively, a beam search could be employed over the n segments with the highest *sweight* values, reducing the search space further with little danger of disallowing optimal matches. By imposing tighter (but lossy) string length bounds and halting conditions, further reductions in the search space would be possible.

Local optimisations to each method are also possible. For the edit distance methods, for example, we can build into the algorithms a mechanism to iteratively determine whether the current translation record has the potential to better the current best match. This is achieved by caching the minimum edit distance for each sub-string pair `i` and `j`, along the row `array[i][0..j]` and column `array[0..i][j]`. If the combined minimum edit distance along these edges exceeds the current optimum edit distance, then we halt the matching process. Our motivation in this is that all edit operations are additive, and it is hence not possible for any reduction in the minimum local edit distance to occur over the remaining components of the target strings. If a local sub-optimal match is detected, then we exit out of the matching process, and return an arbitrarily high edit distance (at least as high as the edit distance to that point), to ensure that the TM system discards that translation candidate. More localised optimisation is also possible using this same basic method, in triangularising the edit distance matrix to avoid local computations over regions which have no chance of contributing to an overall best analysis (see Planas and Furuse (2000) for a more thorough description of this general method).

2.5 Evaluation Specifications

In this section, we describe the evaluation methodology. In proceeding sections, we go on to apply this methodology in evaluating the various string comparison methods, under the two indexing algorithms and in combination with the various models of segment contiguity. We additionally test various *sweight* schemata, based on character type and IDF. In Section 2.6, we present the results for Japanese–English translation retrieval, before reversing the language direction and looking briefly at English–Japanese translation retrieval under word-based indexing only in secrefej-tm.

2.5.1 Details of the Dataset

As our main dataset, we used 3033 unique Japanese–English translation records relating to the servicing of construction machinery.[10] In the dataset, translation records vary in size from single-

[10]Each translation record is unique as an ordered L1–L2 string pair, but there is scope for L1 or L2 coincidence between translation records, an effect which was observed to a limited degree in the data.

word glossary terms to multiple-sentence strings, at an average Japanese segment length of 14.4 and character length of 27.7, and an average English word length of 13.3. Note that our dataset constitutes a controlled language, that is, a given word will tend to be translated identically when used in a particular sense (i.e. there is no literary flair), and only a limited range of syntactic constructions are employed. We discuss the impact of this fact on retrieval performance below.

In Section 2.6.8, we use a second dataset of expanded size, from the economic domain. Details of the dataset and its use can be found there.

The translation records making up the dataset were generated from a glossary of technical terms and a collection of technical reports. Each line in the original glossary consisted of a Japanese technical term and a list of corresponding translations, from which a single translation record was generated by taking the first translation candidate. The technical reports posed a more serious alignment problem, as descriptions were given in full sentence form, and the Japanese and English versions were found in independent files. Fortunately, individual technical reports were labelled with the same index across languages, and the same logical structure was preserved between corresponding technical reports. It was thus possible to align reports at the logical segment level ("Title", "Problem description", "Solution", etc.). The number of sentences in aligned logical segments was then counted, and in the case of an identical count, individual sentences were extracted and sequentially aligned between Japanese and English (similarly to the approach of Veale and Way (1997)). Failing this and in the case that one of the languages contained a single sentence, the multiple sentences in the opposing language were aligned to that single sentence. Only in the case that a divergent, non-singular number of sentences was obtained for the two languages, was manual alignment called upon. We did not attempt to automatically align sentences in this final case, as the number of such cases was minimal (around 50). Instead, we manually aligned this residue, a process which was completed within one person hour. Naturally, we recognise the worth of automatic alignment methods for larger texts with less logical structure, and refer the reader to the work of Gale and Church (1993), Fung and Church (1994), Wu (1994) and Xu and Tan (1999). Such methods would prove their worth in on-line applications where TM updates are fully automated.

Demarkation of translation records was assumed in evaluation, and no further effort was made to subdivide partitions. For real-world TM systems, the automation of this process comprises an important component of the overall system, integrally linked to translation retrieval. While acknowledging the importance of this step and its interaction with retrieval performance (e.g. see Nirenburg *et al.* (1993)), we choose to sidestep it for the purposes of this chapter, and leave it for future research.

For Japanese word-based indexing, segmentation was carried out primarily with ChaSen v2.0 (Matsumoto *et al.* 1999), and where specifically mentioned, JUMAN v3.5 (Kurohashi and Nagao 1998) was also used as a control. A third system, ALT-JAWS,[11] was used to test the effects of lexical normalisation on retrieval accuracy. For all three segmenters, no attempt was made to post-edit the segmented output, in interests of maintaining consistency in the data. We do, however, rate the segmentation performance of the different systems in Section 2.6.4 in order to determine the source of any disparity in retrieval performance.

2.5.2 Semi-stratified Cross Validation

Retrieval accuracy was determined by way of 10-fold semi-stratified cross validation over the dataset. As part of this, all Japanese strings of length 5 characters or less (a total of 531 translation records) were extracted from the dataset, leaving a residue of 2502 translation records over which

[11] `http://www.kecl.ntt.co.jp/icl/mtg/resources/altjaws.html`

cross validation was performed. Our motivation in filtering off the set of 531 shorter strings was that they originated from the glossary, making it improbable that a useful match would be found for them in the TM. That is not to say that they could not provide high quality partial matches for other translation records, however, and they were included in the training data (i.e. TM) on each iteration.

As described in Section 3.6, in N-fold stratified cross validation, the dataset is divided into N equally-sized partitions of uniform class distribution (cf. standard cross validation, where class distribution does not enter into calculations). Evaluation is then carried out N times, taking each partition as the held-out test data, and the remaining partitions as the training data on each iteration; the overall accuracy and other evaluative data is appropriately sampled over the N data configurations.

For our purposes, the test data equates to a set of held-out inputs, and the training data (plus the glossary) the TM. As our dataset is not pre-classified according to a discrete class description, we are not able to perform true data stratification over the class distribution. Instead, we carry out "semi-stratification" over the L1 segment lengths of the translation records. The same partitioning of data was used throughout evaluation unless otherwise specified.

2.5.3 Evaluation of the Output

We evaluate the appropriateness of the translation candidate(s) settled upon by the different comparison methods for a given input, by taking the held-out translation for the input as the model translation and analysing whether the translation candidate(s) are optimally close to the model translation, given the current TM content. By treating the held-out translation for each input as the model translation, it is possible to apply the same string comparison methods as are used in translation retrieval, to determine that set of translations most closely corresponding to the actual translation. Determination of the set of "optimal" translations, therefore, consists of using the held-out translation as the input in searching through the L2 component of the TM, and using an arbitrary string comparison method to judge which translations in the TM most closely resemble the model translation. That is, translation "optimality" is defined numerically according to L2 similarity or distance from the model translation, as determined by a given string comparison method.

Below, we describe the evaluation methodology in greater detail.

First, we use an arbitrary string comparison method to determine the translation record(s) of minimum L2 distance/maximum L2 similarity to the (unique) model translation. In this, we consider only those translation records in the test TM associated with each input. This methodology correlates to pinpointing the "most useful" translation(s) the TM system could possibly return from the current TM, given that this is the only translation data available to the TM system.

Having identified the translation candidates most similar to the model translation, we next consider whether the effort required to transform any one of these candidates into the model translation will be greater than the effort required to translate the input from scratch, that is whether their "translation utility" will be sufficiently high. This is achieved in a model-specific manner, either by thresholding over a static value (see below), or in the case of the edit distance methods, by thresholding over the edit distance between the input and the empty string (i.e. the segment length of the input). Combining this stipulation with the basic method as described above, the set of optimal translation candidates for a given TM and model translation IN^{L2} can

be formalised as follows in the case of a distance-based string comparison method:[12]

$$\{S^{L2} | \langle S^{L1}, S^{L2} \rangle \in TM \cup \{\langle \epsilon^{L1}, \epsilon^{L2} \rangle\}, dist(S^{L2}, IN^{L2}) \leq len(IN^{L2}),$$
$$\forall \langle T^{L1}, T^{L2} \rangle \in TM - \langle S^{L1}, S^{L2} \rangle : dist(S^{L2}, IN^{L2}) \leq dist(T^{L2}, IN^{L2})\}$$

In the case that no translation candidate is below the translation utility cut-off, we return the empty string, and in the case that the optimal translation candidate(s) score at the same level as the cut-off, the empty string is returned along with the best-scoring translation candidate(s).

Note that the string comparison method used in evaluation of the optimality of the L2 output is independent of the string comparison method used in translation retrieval. In Section 2.6, we look at the idiosyncrasies of a number of string comparison methods when used for final evaluation.

Having established the optimal TM system output for each input, we proceed to ascertain whether the actual system output coincides with one of the optimal outputs, and rate the accuracy of each method according to the proportion of optimal outputs. In order to factor out the effects of differing degrees of multiplicity of output, we break ties randomly. This guarantees a unique translation output. Our primary motivation in this was to make the final results for each method directly comparable. This differs from the methodology of Baldwin and Tanaka (2000b) in adjudging the system to have been "correct" if an optimal translation candidate is contained in the potentially multiple set of top-ranking outputs.

We additionally evaluate the margin of error in the case that the translation output is non-optimal, i.e. that it does not correspond to one of the optimal translation candidates. This is achieved by determining the minimum distance/maximum similarity between the unique output and the set of optimal outputs, hence modelling the degree by which the output is amiss.

The following L2 *sweight* schemata were adopted in evaluation:

English *sweight* Schema:

Segment type	*sweight*
punctuation	0
stop words	0
other words	1

Japanese *sweight* Schema:

Segment type	*sweight*
punctuation	0
other segments	1

Stop words are defined as those contained within the SMART (Salton 1971) stop word list.[13] See below for a justification of these *sweight* settings.

The thresholds on "translation utility" are as follows:

Comparison method	*Threshold*
Vector space model	0.5
Token intersection	0.4
3-operation edit distance	$len(IN)$
3-operation edit similarity	0.4
4-operation edit distance	$len(IN)$
4-operation edit similarity	0.4
Weighted sequential correspondence	0.2
Sato92	0.2

[12] With a similarity-based string comparison method, the set of optimal translation candidates is defined similarly, except that the distance function is replaced by a similarity function, and the various *dist* inequalities are reversed in direction.

[13] ftp://ftp.cornell.cs.edu/pub/smart/english.stop

where IN is the input string, and *len* is the conventional segment length operator. See Section 2.6.8 for a justification of these particular static weight settings.

We set ourselves apart from conventional research on TM retrieval performance in adopting this objective numerical evaluation method. Traditionally, retrieval performance has been gauged by the subjective usefulness of the closest matching element of the system output (as judged by a human), and described by way of a discrete set of translation quality descriptors (e.g. Nakamura (1989), Sumita and Tsutsumi (1991), Sato (1992)). Perhaps the closest evaluation attempts to what we propose are those of Planas and Furuse (1999) in setting a mechanical cutoff for "translation usability" as the ability to generate the model translation from a given translation candidate by editing less than half the component words, and Nirenburg *et al.* (1993) in calculating the weighted number of key strokes required to convert the system output into an appropriate translation for the original input. The method of Nirenburg *et al.* (1993) is certainly more indicative of true target language usefulness, but is dependent on the competence of the translator editing the TM system output, and not automated to the degree our method is.

Perhaps the main drawback of our approach to evaluation is that we assume a unique model translation for each input, where in fact, multiple translations of equivalent quality, differentiated only stylistically, could reasonably be expected to exist for each L1 string. In our case, the highly technical nature of our target domain of technical field reports offers some respite in this regard, in that it constitutes a controlled language and does not display the same degree of diversity as we would expect in free text. In this sense, the proposed evaluative framework is most robust under relatively strictured domains of the type targeted in this research. For free text domains, the allowance of multiple model translations would go some way in modelling stylistic variation, although this would leave the issue of how the TM system should choose between multiple model translations for a given L1 string, in positing translation candidates from the TM.

2.6 Japanese–English Translation Retrieval Results

2.6.1 Experiment I: 3-operation edit distance-based evaluation

First, we look at Japanese–English translation retrieval for the following basic L1 *sweight* schema:

Segment type	*sweight*
punctuation	0
other segments	1

The results for the different string comparison methods under character-based and word-based indexing are given in Tables 2.2 and 2.3, respectively, as determined using 3-operation edit distance to derive translation optimality. In each table, the bag-of words methods are partitioned off from the segment order sensitive methods, "Weight seq corr" refers to weighted sequential correspondence and "Sato92" to the Sato (1992) variant thereof. The bag-of-words methods and edit distance and similarity methods were variously tested under unigram, bigram, and mixed unigram/bigram models of segment contiguity,[14] whereas weighted sequential correspondence and Sato92 were tested only with the unigram model, due to their innate modelling of segment contiguity. "Accuracy" is

[14]While we do not present the results here, both trigram and mixed bigram/trigram models were also tested over character- and word-based indexing. In the case of character-based indexing, trigrams were found to be inferior to bigrams but superior to unigrams, and mixed bigram/trigrams were found to be slightly superior to simple trigrams but inferior to both bigrams and mixed unigram/bigrams. With word-based indexing, both trigrams and mixed bigram/trigrams were found to be inferior to the basic unigram model.

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	55.67	11.62	0.97	1.73
	Token int	55.47	10.92	0.94	2.37
	3-op edit dist	**64.43**	8.85	0.81	2.98
	3-op edit sim	**58.83**	9.72	0.95	10.55
	4-op edit dist	**53.75**	10.17	0.79	17.40
	4-op edit sim	**56.67**	9.59	0.90	26.28
	Weight seq corr	**58.71**	10.16	0.96	127.56
	Sato92	**57.27**	10.47	0.95	132.77
2-GRAM	VSM	**60.27**	9.15	0.97	0.49
	Token int	**59.95**	9.16	0.95	0.62
	3-op edit dist	**68.54**	8.48	0.82	0.76
	3-op edit sim	**60.15**	9.09	0.96	0.97
	4-op edit dist	**55.27**	10.04	0.77	1.67
	4-op edit sim	**58.43**	9.04	0.94	1.98
1+2-GRAM	VSM	**60.19**	10.27	0.97	2.11
	Token int	**59.63**	9.80	0.97	2.96
	3-op edit dist	**67.91**	8.59	0.85	3.50
	3-op edit sim	**60.71**	9.42	0.96	9.03
	4-op edit dist	**55.19**	9.99	0.84	20.89
	4-op edit sim	**58.59**	9.25	0.95	32.22

Table 2.2: Results for the different comparison methods under **character-based indexing**, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using **3-operation L2 edit distance**

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	54.75 (−1.7%)	10.97	0.93	<u>0.66</u>
	Token int	54.75 (−1.3%)	10.50	0.89	0.91
	3-op edit dist	<u>59.67</u> (−7.4%)	<u>9.29</u>	0.72	1.00
	3-op edit sim	55.67 (−5.4%)	9.78	0.90	1.75
	4-op edit dist	50.88 (−5.4%)	10.95	0.66	3.09
	4-op edit sim	53.23 (−6.1%)	9.89	0.84	4.04
	Weight seq corr	53.99 (−8.0%)	9.94	0.93	28.99
	Sato92	54.07 (−5.6%)	10.66	0.92	29.61
2-GRAM	VSM	52.87 (−12.3%)	8.55	0.95	<u>0.26</u>
	Token int	53.27 (−11.1%)	8.67	0.93	0.31
	3-op edit dist	<u>55.11</u> (−19.6%)	<u>8.52</u>	0.83	0.34
	3-op edit sim	53.19 (−11.6%)	8.68	0.93	0.38
	4-op edit dist	51.08 (−7.6%)	10.82	0.70	0.59
	4-op edit sim	52.79 (−9.6%)	8.80	0.92	0.55
1+2-GRAM	VSM	55.87 (−7.2%)	9.04	0.96	<u>0.90</u>
	Token int	56.11 (−5.9%)	9.23	0.94	1.16
	3-op edit dist	<u>62.59</u> (−7.8%)	<u>8.85</u>	0.83	1.32
	3-op edit sim	56.15 (−7.5%)	9.13	0.94	1.99
	4-op edit dist	50.68 (−8.2%)	10.84	0.73	3.65
	4-op edit sim	53.95 (−7.9%)	9.23	0.91	4.68

Table 2.3: Results for the different comparison methods under **word-based indexing**, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using **3-operation L2 edit distance**

an indication of the proportion of inputs for which an optimal translation was produced; character-based indexing accuracies in bold indicate a significant[15] advantage over the corresponding word-based indexing accuracy (same string comparison method and N-gram model), and the underlined figure for each indexing paradigm indicates the highest achieved accuracy. "Non-opt dist" refers to the mean minimum 3-operation edit distance between the translation candidate and optimal translation(s) in the case of the translation candidate being non-optimal (i.e. the edit distance is not calculated in the case of an optimal translation output); as for accuracy, the best-achieved non-optimal edit distance for each indexing paradigm is underlined. "Unique outputs" describes the proportion of inputs for which a unique translation candidate was produced. "Ave. time" describes the average time taken to determine the translation candidate(s) for a single input, relative to the time taken for word unigram-based 3-operation edit distance retrieval. We use this particular system configuration, as timed over the construction machinery dataset, as our touchstone for time calculation throughout this chapter.

Perhaps the most striking result is that character-based indexing produces a superior match accuracy to word-based indexing for *all* comparison methods operating over *all* N-gram models of local context, to a level of statistical significance in all cases except for the vector space model and token intersection, when coupled with a unigram model. The relative gain in accuracy due to character-based indexing, averaged over the different methods and N-gram models, is 9.2%.

3-operation edit distance outperforms all other methods for both character- and word-based indexing, peaking at around 69% for character bigrams. There is a marked drop-off in accuracy between 3-operation edit distance and 3-operation edit similarity for both character- and word-based indexing, which was found to be statistically significant in most cases. Similarly, 3-operation edit distance has a clear advantage over 4-operation edit distance, supporting our prediction that the introduction of the substitution operator with 4-operation edit distance leads to a loss of discriminatory granularity. Interestingly, 4-operation edit similarity fared slightly better than 4-operation edit distance, which we postulate is due to normalisation providing the means to differentiate between the segment substitution and segment insertion/deletion operations (see Section 2.4.2). Weighted sequential correspondence and the Sato92 method performed at roughly the same level as 4-operation edit similarity.

An equally important component of evaluation is non-optimal edit distance, i.e. the magnitude of error in the case of a non-optimal output. Here again, 3-operation edit distance was found to err most conservatively in all cases. We can thus make the statement that, of all methods tested, 3-operation edit distance is superior in terms of both raw accuracy and the margin of error in the case of a non-optimal output, within the confines of the given evaluation framework. Looking to the other methods, the bag-of-words methods tended to produce a high non-optimal edit distance when operating over unigrams, but near or surpass the non-optimal edit distance of the other methods for the bigram and mixed bigram models.

With the segment contiguity model, bigrams were found to perform best under character-based indexing, nosing out mixed unigrams/bigrams, and mixed unigrams/bigrams were found to be superior to straight unigrams and bigrams for word-based indexing. In comparing character bigrams to mixed word unigrams/bigrams, that is the best N-gram models for each indexing paradigm, our statement as to the superiority of character-based indexing is upheld. Bigrams produced a marked acceleration in retrieval time (i.e. they reduce the number of translation records required to be processed before settling on the final translation candidate set), whereas mixed unigrams/bigrams were marginally slower then simple unigrams.

[15] As determined by the paired t test ($p < 0.05$).

All similarity-based methods were able to produce a unique translation candidate over 90% of the time, while the edit distance methods were less successful at committing to a single translation candidate, returning unique translation candidates a lesser 85% of the time on average. Note that these figures are simply a reflection of the discriminatory potential of the different methods, and that a unique translation candidate was randomly selected from the set of outputs for use in evaluating accuracy and edit discrepancy. As such, the higher number of instances of translation ambiguity for 3-operation edit distance, for example, did not impinge on its superior accuracy and non-optimal edit distance, but rather point to the susceptibility of the method to be scored down due to non-optimal translation candidates being chosen ahead of optimal translation candidates, by the tie-breaking mechanism. For methods returning multiple translation outputs more often, therefore, there is scope for further improvement in raw accuracy when the basic method is combined with a more refined tie-breaking mechanism.

Looking to the relative speeds of the different methods, word-based indexing was found to be faster than character-based indexing across the board for all N-gram orders, largely because the number of character N-grams per string is always going to be greater than or equal to the number of word N-grams. Additionally, word-based indexing produces greater segment discrimination than character-based indexing for the same N-gram order, by way of increasing the number of segment types. This, coupled with the acceleration methods described in Section 2.4.4, reduces the number of string comparisons required to settle on the final translation candidate set (i.e. prunes the string search space). Note that the times quoted above include the time required to segment the input online, but that the search space reduction afforded by word-based indexing and associated speed-up, is much greater than the extra time taken in segmentation. While recognising that segmentation has a part to play in shortening retrieval times, character bigrams were found to be faster than word unigrams. While word bigrams were faster again, they were associated with a sharp drop in retrieval accuracy. Speed alone is thus not a true justification of segmentation.

2.6.2 Experiment II: Weighted sequential correspondence-based evaluation

In the preceding section, the accuracy and non-optimal edit distance results were derived through the use of 3-operation L2 edit distance. Intuitively speaking, 3-operation L2 edit distance would seem a valid choice of evaluative method, but there is the possibility that it has somehow skewed the results. This is a particularly pertinent concern for the corpus used, due to it constituting a controlled language. With controlled languages, the translation of a given string is highly predictable, in that there is little scope for creativity or originality in translation. For two L1–L2 string pairs $\langle S^{L1}, S^{L2}\rangle$ and $\langle T^{L1}, T^{L2}\rangle$, therefore, the same basic lexical consistencies will be observed between S^{L1} and T^{L1} as between S^{L2} and T^{L2}. The upshot of this is that when the same string comparison method is used in translation retrieval and final evaluation of the output, there is a real chance that it will pick up on the same idiosyncratic consistencies in L1 and L2, irrespective of the correlation between these consistencies and translation optimality. This can lead to the method having an artificially high estimation of itself over other string comparison methods.

In order to investigate the extent of this effect, in this section, we rate the output using a second string comparison method, namely L2 weighted sequential correspondence. While edit distance prefers strings which preserve the same basic lexical order and have the same basic composition of lexical items, as pointed out above, it is blind to the effects of segment contiguity. Weighted sequential correspondence, on the other hand, gives preference to strings where there is a high degree of sub-string overlap, possibly at the expense of overall similarity. One could plausibly argue that having a component of fully reusable substrings in the output makes the job of the translator easier, and that weighted sequential correspondence is more representative of the real-world value of the

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	55.59	10.87	0.97	1.73
	Token int	57.15	10.32	0.94	2.37
	3-op edit dist	**52.24**	7.96	0.81	2.98
	3-op edit sim	**62.31**	8.44	0.95	10.55
	4-op edit dist	**52.44**	9.58	0.79	17.40
	4-op edit sim	**60.99**	8.25	0.90	26.28
	Weight seq corr	**59.95**	9.14	0.96	127.56
	Sato92	58.23	9.63	0.95	132.77
2-GRAM	VSM	**62.23**	7.41	0.97	0.49
	Token int	**64.83**	7.36	0.95	0.62
	3-op edit dist	52.08	7.65	0.82	0.76
	3-op edit sim	**64.87**	7.28	0.96	0.97
	4-op edit dist	**52.99**	9.41	0.77	1.67
	4-op edit sim	**63.07**	7.29	0.94	1.98
1+2-GRAM	VSM	61.15	9.04	0.97	2.11
	Token int	**63.35**	8.36	0.97	2.96
	3-op edit dist	51.75	7.78	0.85	3.50
	3-op edit sim	**64.63**	7.72	0.96	9.03
	4-op edit dist	**52.63**	9.46	0.84	20.89
	4-op edit sim	**63.03**	7.63	0.95	32.22

Table 2.4: Results for the different comparison methods under **character-based indexing**, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using L2 **weighted sequential correspondence**

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	55.95 (+0.6%)	9.95	1.10 (0.93)	<u>0.66</u>
	Token int	56.63 (−0.9%)	9.47	1.17 (0.89)	0.91
	3-op edit dist	49.20 (−5.8%)	8.49	1.78 (0.72)	1.00
	3-op edit sim	<u>58.75</u> (−5.7%)	<u>8.43</u>	1.16 (0.90)	1.75
	4-op edit dist	47.80 (−8.8%)	10.28	2.51 (0.66)	3.09
	4-op edit sim	56.39 (−7.5%)	8.56	1.40 (0.84)	4.04
	Weight seq corr	57.15 (−4.7%)	8.74	1.13 (0.93)	28.99
	Sato92	55.59 (−4.5%)	9.57	1.14 (0.92)	29.61
2-GRAM	VSM	55.79 (−10.3%)	<u>7.00</u>	1.07 (0.95)	<u>0.26</u>
	Token int	55.79 (−13.9%)	7.03	1.09 (0.93)	0.31
	3-op edit dist	50.99 (−2.1%)	7.47	1.26 (0.83)	0.34
	3-op edit sim	<u>56.07</u> (−13.6%)	7.02	1.09 (0.93)	0.38
	4-op edit dist	46.60 (−12.1%)	10.03	2.72 (0.70)	0.59
	4-op edit sim	55.59 (−11.9%)	7.16	1.16 (0.92)	0.55
1+2-GRAM	VSM	58.91 (−3.7%)	<u>7.42</u>	1.05 (0.96)	<u>0.90</u>
	Token int	<u>59.35</u> (−6.3%)	7.53	1.07 (0.94)	1.16
	3-op edit dist	50.11 (−3.2%)	7.94	1.26 (0.83)	1.32
	3-op edit sim	59.19 (−8.4%)	7.46	1.07 (0.94)	1.99
	4-op edit dist	47.80 (−9.2%)	10.35	2.15 (0.73)	3.65
	4-op edit sim	57.47 (−8.8%)	7.78	1.19 (0.91)	4.68

Table 2.5: Results for the different comparison methods under **word-based indexing**, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using L2 **weighted sequential correspondence**

output. While we make no subjective claim in this regard, we do recognise that such a trade-off between overall correspondence and local reusability exists in determining translation utility, and that 3-operation edit distance approximates the former and weighted sequential correspondence the latter.

The results as evaluated by weighted sequential correspondence are presented in Tables 2.4 and 2.5. The basic format utilised here is identical to that used to present the results under 3-operation edit distance, including the non-optimal edit distance being calculated according to 3-operation edit distance. That is, we use weighted sequential correspondence to generate the set of optimal translation candidates, and in the case of the output not being found therein, use 3-operation edit distance to calculate the distance between the output and model translation.

As with evaluation based on 3-operation edit distance, the results show character-based indexing to be superior to word-based indexing in all cases bar VSM in combination with unigrams; in over half of the system configurations tested, this difference was found to be statistically significant. Once again, therefore, segmentation was found to be superfluous at best, and potentially damaging to retrieval performance.

In using weighted sequential correspondence rather than 3-operation edit distance to evaluate translation optimality, the relative performance of 3-operation edit distance drops by a considerable margin, suggesting a strong affinity between L1 3-operation edit distance in retrieval and L2 3-operation edit distance in evaluation. Interestingly, the same does not apply to weighted sequential correspondence, which was rated at the same relative level in the two experiments. According to weighted sequential correspondence, the most accurate method is 3-operation edit similarity by a slight margin, although the two bag-of-words methods also performed strongly. Indeed, in terms of non-optimal edit distance, the bag-of-words methods on occasion erred more conservatively than the segment order-sensitive methods. Overall, there was little to separate the different string comparison methods in terms of non-optimal edit distance, with the one exception of 4-operation edit distance which was found to be further off-target than the other methods by an average of three edit operations (L2 word deletions and insertions). 4-operation edit similarity rated much better, but was still inferior to 3-operation edit similarity.

Of the different segment contiguity models, character bigrams and mixed word unigrams/bigrams produced the most solid performances for the two indexing paradigms, mirroring the results of the first experiment.

Note that the running times for the different system configurations are identical to those for experiment I, as all we have altered is the means for evaluating the results of the same retrieval task.

While we do not present the results here, we also tested the use of both bag-of-words methods (i.e. VSM and token intersection) in rating the output, but found them to be more erratic than either of the two methods applied above.

2.6.3 Cross-method integrated evaluation

It is impossible to make anything other than a subjective judgement as to the superiority of 3-operation edit distance and weighted sequential correspondence as our chosen model of translation optimality in evaluating retrieval performance. While weighted sequential correspondence appears to be less biased than 3-operation edit distance toward any one string comparison method in retrieval, as a general trend, it would seem to score similarity-based comparison methods higher than distance-based methods. Rather than committing ourselves to either evaluation method, we choose to hedge this issue in describing retrieval accuracy by way of the mean of the accuracies produced by these two methods. This is the method that we adopt for the remainder of this

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	53.35	10.75	0.97	1.73
	Token int	53.59	10.12	0.94	2.37
	3-op edit dist	**55.45**	8.00	0.81	2.98
	3-op edit sim	57.87	8.61	0.95	10.55
	4-op edit dist	**50.76**	9.50	0.79	17.40
	4-op edit sim	56.49	8.52	0.90	26.28
	Weight seq corr	56.73	9.18	0.96	127.56
	Sato92	55.15	9.58	0.95	132.77
2-GRAM	VSM	**60.41**	7.95	0.97	0.49
	Token int	**60.85**	7.88	0.95	0.62
	3-op edit dist	**58.17**	7.61	0.82	0.76
	3-op edit sim	**61.09**	7.83	0.96	0.97
	4-op edit dist	**51.66**	9.32	0.77	1.67
	4-op edit sim	**59.55**	7.82	0.94	1.98
1+2-GRAM	VSM	58.95	9.24	0.97	2.11
	Token int	59.09	8.63	0.97	2.96
	3-op edit dist	57.65	7.77	0.85	3.50
	3-op edit sim	**60.57**	8.15	0.96	9.03
	4-op edit dist	**51.42**	9.38	0.84	20.89
	4-op edit sim	**59.05**	8.07	0.95	32.22

Table 2.6: Results for the different comparison methods under **character**-based indexing, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using the **combined 3-operation edit distance and weighted sequential correspondence methods**

	Method	***Accuracy***	***Non-opt edit dist***	***Unique outputs***	***Ave. time***
1-GRAM	VSM	53.89 (+1.0%)	10.08	0.93	0.66
	Token int	53.83 (+0.4%)	9.60	0.89	0.91
	3-op edit dist	52.38 (−5.5%)	8.51	0.72	1.00
	3-op edit sim	55.45 (−4.2%)	8.75	0.90	1.75
	4-op edit dist	47.72 (−6.0%)	10.30	0.66	3.09
	4-op edit sim	53.71 (−4.9%)	8.93	0.84	4.04
	Weight seq corr	54.23 (−4.4%)	9.00	0.93	28.99
	Sato92	52.97 (−3.9%)	9.72	0.92	29.61
2-GRAM	VSM	54.41 (−9.9%)	7.57	0.95	0.26
	Token int	54.39 (−10.6%)	7.61	0.93	0.31
	3-op edit dist	53.01 (−8.9%)	7.75	0.83	0.34
	3-op edit sim	54.51 (−10.8%)	7.61	0.93	0.38
	4-op edit dist	47.60 (−7.9%)	10.16	0.70	0.59
	4-op edit sim	54.23 (−8.9%)	7.74	0.92	0.55
1+2-GRAM	VSM	56.65 (−3.9%)	7.95	0.96	0.90
	Token int	56.65 (−4.1%)	8.07	0.94	1.16
	3-op edit dist	55.93 (−3.0%)	8.08	0.83	1.32
	3-op edit sim	56.69 (−6.4%)	8.00	0.94	1.99
	4-op edit dist	47.82 (−7.0%)	10.30	0.73	3.65
	4-op edit sim	54.91 (−7.0%)	8.23	0.91	4.68

Table 2.7: Results for the different comparison methods under **word**-based indexing, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model, as evaluated using the **combined 3-operation edit distance and weighted sequential correspondence methods**

chapter. One motivation in this is to balance off a similarity- and distance-based measure against each other, and also to combine two relatively heterogeneous methods capable of capturing distinct effects.

The combined (averaged) results from experiments I and II are presented in Table 2.7. Unsurprisingly, the general trends observed for the individual evaluation methods are repeated here. Namely:

- Character-based indexing is consistently superior to word-based indexing, particularly when combined with an N-gram model with a bigram component
- In terms of raw translation optimality, there is very little to separate bag-of-words methods from segment order-sensitive methods. In other areas, the margin of error for segment order-sensitive methods in the case of a non-optimal output tends to be lesser than for bag-of-words methods. Bag-of-words methods, on the other hand, are associated with lower computational overheads.
- With character-based indexing, bigrams offer slight gains in translation accuracy at the same time as greatly accelerating the retrieval process. With word-based indexing, mixed unigrams/bigrams offer slight gains in translation accuracy and reduce edit discrepancy, at slightly inflated computational cost to unigrams.
- Weighted sequential correspondence (and its derivative Sato92) are moderately successful in terms of accuracy and non-optimal edit distance, but are grossly expensive and not immediately compatible with N-gram methods. More lightweight segment order-sensitive methods such as 3-operation edit distance and similarity, on the other hand, can be combined with N-gram segment contiguity models to approximate the segment contiguity-awareness of weighted sequential correspondence, while avoiding the unwarranted computational overhead. Weighted sequential correspondence is superior to Sato92 in terms of overall retrieval performance.
- 3-operation edit distance and similarity are superior to their 4-operation counterparts in all respects.

Based on the results of the above two experiments, we judge bigrams to be the best segment contiguity model for character-based indexing, and mixed unigrams/bigrams to be the best segment contiguity model for word-based indexing, and for the remainder of this chapter, present only these two sets of results. In keeping with our comments on the incompatibility of weighted sequential correspondence and Sato92 with anything other than a unigram model, we continue to run these two methods only on basic unigram data, but present the results along with the data for other methods under a bigram or mixed unigram/bigram segment contiguity model.

2.6.4 Experiment III: Segmentation accuracy

Above, we observed that segmentation consistently brought about a degradation in translation retrieval for the given dataset. Automated segmentation inevitably leads to errors, which could possibly impinge on the accuracy of word-based indexing. Alternatively, the performance drop could simply be caused somehow by our particular choice of segmentation module, that is ChaSen. In this section, we look to clarify such issues primarily by evaluating the retrieval and segmentation performance of the ChaSen, JUMAN and ALTJAWS systems and considering the ramifications of system errors on overall retrieval accuracy. In conjunction with this, we test whether lexical

	Method	*Accuracy*	*Non-opt edit dist*	*Unique outputs*	*Ave. time*
Word-based	VSM	56.94 (+0.5%)	7.84	0.96	0.76
	Token int	57.45 (+1.4%)	7.89	0.95	1.03
	3-op edit dist	56.19 (+0.5%)	7.92	0.84	1.19
	3-op edit sim	57.28 (+1.0%)	7.88	0.95	1.61
	4-op edit dist	47.64 (−0.4%)	10.18	0.73	2.76
	4-op edit sim	55.94 (+1.9%)	7.97	0.92	3.52
	Weight seq corr	55.04 (+1.5%)	8.64	0.92	23.50
	Sato92	53.58 (+1.1%)	9.07	0.91	23.39

Table 2.8: Results for data segmented with JUMAN, under word-based indexing

	Method	*Accuracy*	*Non-opt edit dist*	*Unique outputs*	*Ave. time*
Without Lexical Norm.	VSM	**58.69** (+3.6%)	7.61	0.96	0.60
	Token int	**59.33** (+4.7%)	7.68	0.95	0.87
	3-op edit dist	58.31 (+4.3%)	7.61	0.82	1.03
	3-op edit sim	**59.09** (+4.2%)	7.66	0.95	1.39
	4-op edit dist	49.44 (+3.4%)	9.68	0.72	2.43
	4-op edit sim	**57.59** (+4.9%)	7.77	0.93	2.70
	Weight seq corr	**57.24** (+5.5%)	8.35	0.93	20.43
	Sato92	**55.96** (+5.6%)	8.83	0.92	19.43
With Lexical Norm.	VSM	58.86 (+3.9%)	7.85	0.96	0.57
	Token int	**59.60** (+5.2%)	7.86	0.94	0.74
	3-op edit dist	59.06 (+5.6%)	7.77	0.83	0.85
	3-op edit sim	59.29 (+4.6%)	7.83	0.94	1.28
	4-op edit dist	**50.83** (+6.3%)	9.85	0.73	2.20
	4-op edit sim	**58.09** (+5.8%)	7.94	0.93	2.84
	Weight seq corr	**57.41** (+5.8%)	8.54	0.93	16.25
	Sato92	**56.52** (+6.7%)	9.05	0.92	17.25

Table 2.9: Results for data segmented and optionally lexically normalised with ALTJAWS, under word-based indexing

normalisation has any impact on retrieval performance, using ALTJAWS to convert each segment into canonical form.

First, we used JUMAN to segment the original set of 3033 translation records, and evaluated the resultant dataset in the exact same manner as for the ChaSen output. Note that in semi-stratified cross validation, a slightly different partition composition was generated for JUMAN than for ChaSen, due to relative differences in the number of segments returned for each string. Similarly, we ran a development version of ALTJAWS over the 3033 translation records to produce two datasets, the first simply segmented and the second both segmented and lexically normalised.[16] Lexical normalisation took the form of taking each segment and converting it into canonical form according to the ALTJAWS analysis.[17] Instances where the canonical form differed from the original spell-out were predominantly verbs in non-regular form (e.g. *kōkaN-site* "replace" ⇒ *kōkaN-suru* ")", but also included loan-word nouns with an optional long final vowel (e.g. *monitā* "monitor" ⇒ *monita* ")" and words with multiple kanji realisations for the same basic sense (e.g. 充分 [*zyūbuN*] "sufficient" ⇒ 十分). As for the JUMAN output, we evaluated the normalised output of ALTJAWS using semi-stratified cross validation.

The results for JUMAN are presented in Table 2.8, and those for ALTJAWS with and without lexical normalisation are presented in Table 2.9. The basic format of the tables coincides with that used above, and the figures given for the gain in accuracy are once again relative to the original character-based indexing accuracies. We present only the word-based indexing results, as the results for character-based indexing are equivalent to those for ChaSen presented above.[18] These results should be compared with those presented in Table 2.7 for ChaSen.

Looking first to the results for JUMAN, there is a slight but appreciable gain in accuracy over ChaSen, for all string comparison methods other than 4-operation edit distance. While this step-up in performance was not found to be statistically significant in any case, it was certainly consistent on the whole, suggesting JUMAN as an equally plausible candidate segmentation module to ChaSen. We consider reasons for this performance discrepancy below.

With ALTJAWS, also, a consistent gain in performance is evident with simple segmentation, the degree of which is significantly higher than for JUMAN. The addition of lexical normalisation enhances this effect marginally, although not to a degree where we are able to draw any hard conclusions as to the effectiveness of lexical normalisation in itself. One area in which ALTJAWS has an edge over ChaSen and JUMAN is the handling of compound nouns (and particularly katakana words), an effect we analyse in more detail below.

Based on the above, we can state that the choice of segmentation system does have a modest impact on retrieval accuracy, and also that the effects of lexical normalisation are highly localised. Our segmentation module of choice, ChaSen, was found to be the worst performer of all three systems tested, prompting doubt as to our commitment to it. In response to this, we emphasise that the main interest of this chapter is in verifying the relative merits of different methods and proce-

[16]The reason that we restrict discussion of lexical normalisation to ALTJAWS is that full normalisation is not possible with the remaining two systems. Note that it is possible to convert each segment to its "base" form with ChaSen and JUMAN, that is the form in which it would occur in in a dictionary entry, but not to reduce it one step further to a canonical form. ChaSen and JUMAN thus would not pick up on the equivalence of 変わる and 変る, lexical alternates of *kawaru* "to change/alter", whereas ALTJAWS would. We recognise that there is some merit in performing partial lexical segmentation, but choose to confine our focus to fully-fledged lexical normalisation for the purposes of this chapter.

[17]With foreign loan words such as *siriNda* "cylinder", the canonical form returned by ALTJAWS is the original English rather than a canonical Japanese form. We chose to ignore all such English canonical forms and retain the original Japanese due to the transliteration process being somewhat unreliable.

[18]Strictly speaking, there is some variation in results for character-based indexing between the different datasets, firstly due to variation in the partitioning of the data for each, and secondly due to random tie-breaking resulting in different final translation candidates being extracted from the same original translation candidate set.

dures in translation retrieval, and not in generating the absolute best integrated system possible. ChaSen simply provides a point of reference for this purpose. It is, however, worthwhile delving deeper into the expected impact of the segmentation module on final translation output, and in the following, we look to quantify the relationship between retrieval performance and segmentation accuracy.

In the next step of evaluation, we took a random sample of 200 translation records from the original dataset, and ran each of ChaSen, JUMAN and ALTJAWS over the Japanese component of each translation record. We then manually evaluated the output in terms of segment precision and segment recall, defined respectively as:

$$\text{Segment precision} = \frac{\#\text{ correct segments in system output}}{\text{Total }\#\text{ segments in system output}} \tag{2.9}$$

$$\text{Segment recall} = \frac{\#\text{ correct segments in system output}}{\text{Total }\#\text{ segments in correct analysis}} \tag{2.10}$$

We further calculated the average character and segment count per sentence, and average sentence segmentation accuracy (i.e. the proportion of sentences for which an overall correct analysis was obtained).

One slight complication in evaluating the output of the three systems is that they adopt incongruent models of conjugation. ChaSen and ALTJAWS consistently partition off the stem of (both conjugating and non-conjugating) verbs and adjectives, and also treat any verbal morphemes (e.g. the passive morpheme) and conjugating affixes as individual segments. JUMAN, on the other hand, segments up conjugating lexemes only in the case that they are demarcated by verbal morphemes, choosing to consider cases of simple inflection as a single segment. There are also subtle differences between ChaSen and ALTJAWS, such as ChaSen tending to separate off each individual auxiliary verb, but ALTJAWS bundling complex sequences of auxiliaries into a single segment. The verb complex site-i-ta [*to do*-PROG-PAST] "was doing", for example, would be divided into three segments (the stem "to do", the stem of the progressive auxiliary, and the past tense auxiliary) by ChaSen, two segments (the stem "to do" and a single segment for the two auxiliaries) by ALTJAWS, and a single segment by JUMAN.

In evaluating the accuracy of the different systems, therefore, we had to make allowance for the idiosyncrasies of the particular segmentation paradigm adopted by each system. In practice, all errors for the three systems occurred in the segmentation of noun complexes, and slight variance in the treatment of single-entry verb complexes was permitted (e.g. both *sasi-kaeru* and *sasikaeru* were judged to be acceptable segmentations for *sasikaeru* "to swap over/inter-replace"). Any fall-off in segmentation performance should thus be seen as a reflection of the ability of each system to handle noun phrase segmentation, and in particular the robustness of the system over compound nouns and unknown words.

A performance breakdown for JUMAN, ChaSen and ALTJAWS is presented in Table 2.10, where segment precision and recall are defined as above, and sentence accuracy is an indication of the proportion of sentences which contained no errant segments. "Total segment types" is the total number of distinct segments found in the overall dataset, adjusted in the manner described below. The finer segment granularity of ChaSen over JUMAN and ALTJAWS is borne out by its slightly higher average segment number. ALTJAWS was found to outperform the remaining two systems in terms of segment precision, at a reduction in error of over 17%. ChaSen and JUMAN performed at the exact same level of segment precision, although one must bear in mind that this statistic for ChaSen includes 200 $(13.0 - 12.0 \times 200)$ additional segments judged to be 100% correct (conjugating affixes, etc. attributable to the finer segmentation granularity of ChaSen);

when these are factored out from calculations, the segment precision for ChaSen falls back to 98.2%, and JUMAN is adjudged to have a slight empirical advantage. Looking next to segment recall, ChaSen significantly outperformed both ALTJAWS and JUMAN (operating at an error rate roughly 17% lower than ALTJAWS, and exactly half that of JUMAN). In practice, this difference was due to JUMAN being less adept at segmenting compound nouns, and tending to bundle multi-word katakana word sequences such as *gēto-rokku-barubu* "gate-lock valve" together into a single segment. ALTJAWS was remarkably successful at segmenting katakana word sequences, achieving a segment precision of 100% and segment recall approaching 99%. For the other two systems, on the other hand, this was the source of almost all errors in recall, and roughly half of errors in precision. This is thought to have been the main cause for the different translation retrieval results over data segmented by the three systems. The reason for JUMAN's slight edge over ChaSen is partially due to the higher number of verb morpheme segments contained in the ChaSen output. Under the weighting schema adopted at this point, all segments are treated equally, such that the higher proportion of verbal segments in ChaSen output has the potential to create a bias in string comparison towards strings which share the same verbal complexes, and downplay the importance of nouns.

	ChaSen	JUMAN	ALTJAWS
Average segments per translation record	13.0	12.0	11.7
Segment precision	98.3%	98.3%	98.6%
Segment recall	98.1%	96.2%	97.7%
Sentence accuracy	70.5%	59.0%	72.0%
Total segment types	650	656	634

Table 2.10: Segmentation performance of ChaSen, JUMAN and ALTJAWS

One point which is not immediately reflected in the precision/recall analysis is the level of consistency in case of error. For translation retrieval, segmentation *consistency* is almost as important as segmentation accuracy, in that we are mainly interested in producing a match between identical words or word clusters, irrespective of just how that match is reached. Indeed, this is a major factor in the high performance of character-based indexing. While consistency is guaranteed in the case of a correct analysis, we wish to gain an insight into the consistency of the three systems in the case of error. To this end, we further calculated the total number of segment types in the output, expecting to find a core set of correctly-analysed segments, of relatively constant size across the different systems, plus an unpredictable component of segment errors, of variable size. The system generating the fewest segment types can thus be said to be the most consistent, as what errors it makes are reproducible on alternate data.

Here again, we face difficulty in comparing the output of the three systems directly due to their heterogeneous segmentation output styles. To gain a truly indicative view of the degree of variation in segment types independent of the handling of predicates, we normalise the output of the three systems by configuring them to output the base form of each morpheme, and filtering any conjugational affixes and verbal morphemes from the output. This results in a total of 634 segment types for ALTJAWS, as compared to 650 for ChaSen and 656 for JUMAN. Based on this, ALTJAWS errs more consistently than the remaining two systems, and there is very little to separate ChaSen and JUMAN. Indeed, while ChaSen produced fewer segment types than JUMAN, it is important to realise that the recall gain to ChaSen exceeds this small disparity. Ostensibly what this means is that ChaSen is more erratic in case of error, but that its higher segment recall covers for this. In conclusion, therefore, ALTJAWS holds a marginal advantage over ChaSen and JUMAN, with little separating the latter two systems.

	Method	$\mathbf{W}_1^{\mathbf{JE}}$ [+k]	$\mathbf{W}_1^{\mathbf{JE}}$ [−k]	$\mathbf{W}_2^{\mathbf{JE}}$ [+k]	$\mathbf{W}_2^{\mathbf{JE}}$ [−k]	$\mathbf{W}_3^{\mathbf{JE}}$ [+k]	$\mathbf{W}_3^{\mathbf{JE}}$ [−k]
CHAR-BASED	VSM	60.39	60.47	60.31	60.33	60.33	60.31
	Token int	60.83	61.11	60.99	60.91	60.95	60.91
	3-op edit dist	58.23	58.25	57.99	58.81	58.05	58.09
	3-op edit sim	61.13	61.17	61.29	61.39	61.17	61.15
	4-op edit dist	51.74	51.73	51.58	51.45	51.94	51.77
	4-op edit sim	59.51	59.73	59.59	59.69	59.61	59.49
	Weight seq corr	49.20	56.39	53.35	57.19	55.07	57.29
	Sato92	46.90	54.99	51.75	55.83	53.67	55.63
	Ave. gain	−7.2%	+0.0%	−3.1%	+0.7%	−1.3%	+0.5%
WORD-BASED	VSM	57.39	57.21	57.29	57.23	57.19	57.35
	Token int	58.29	58.39	58.33	58.11	57.41	57.45
	3-op edit dist	57.47	57.31	57.51	57.31	56.97	57.15
	3-op edit sim	58.29	58.17	58.21	57.85	57.53	57.47
	4-op edit dist	49.34	48.94	49.18	49.02	48.72	48.80
	4-op edit sim	56.83	56.69	56.61	57.01	56.01	56.27
	Weight seq corr	56.49	56.51	56.09	55.91	55.47	55.41
	Sato92	56.01	55.99	55.37	55.35	54.65	54.89
	Ave. gain	+6.5%	+4.9%	+5.5%	+5.3%	+3.6%	+4.0%

Table 2.11: The retrieval performance over different character-based segment weighting schemata

To return to evaluation of the translation retrieval task for the three systems, there would seem to be a direct correlation between segmentation accuracy and retrieval performance, with segmentation accuracy on key terms such as compound nouns having a particularly keen effect on translation retrieval. In this respect, ALTJAWS is superior to both ChaSen and JUMAN for the target domain. Additionally complementing segmentation with lexical normalisation for each segment would seem to produce slight performance gains, although not to an immediately tangible level. It is important to compare the final results for ALTJAWS with lexical normalisation, back to the original results for character-based indexing, and realise that the absolute best word-based indexing system configuration is by and large inferior to character bigrams. Thus, for all the effort expended in tweaking the performance of word-based indexing, character-based indexing coupled with a simple bigram model, maintains a performance advantage. Our original observation as to the superiority of character-based indexing over word-based indexing still holds, therefore.

2.6.5 Experiment V: The effects of different character type-based segment weighting schemata

Having variously tested the relative merits of the different string comparison methods and the effects of segmentation, we next turn to methods of enhancing retrieval performance through segment weighting. The first such approach is segment weighting based on the character composition of that segment, through a static weighting schema.

To reiterate the predictions of Section 2.3.3, for Japanese we predict that segments made up entirely of hiragana will have less bearing on retrieval performance than other segment types. It is also possible to take this a step further in suggesting that segments not containing any kanji characters will have less import on retrieval performance than those containing kanji. To test this

hypothesis, we define the class of "light segments" to be those segments made up exclusively of hiragana, numerals, Latin characters and optionally katakana, and stipulate a range of static *sweight* values for light and other segments to balance up the impact each will have on translation retrieval.[19] This is done in the form of the following three scoring schemata, where "other segments" essentially refers to segments containing kanji characters, or alternatively kanji or katakana characters:

Segment type	W_1^{JE}	W_2^{JE}	W_3^{JE}
punctuation	0	0	0
light segments	0.01	0.2	0.5
other segments	1	1	1

The translation accuracy over each weighting schema is then tested and compared to the basic translation accuracy from Tables 2.6 and 2.7 above (that is with the *sweight* for light segments set equal to that for other non-punctuation segments, at 1). Final translation accuracies are given in Table 2.11, including the average gain in accuracy over all methods. Each [+k] column indicates the case of katakana having been included in the definition of light segments, and each [−k] column the case of exclusion of katakana from the definition. Underlined accuracies represent the best performance for that string comparison method under the given indexing paradigm.

We first analyse the results for character-based indexing, where the respective *sweight* values translate across to weights on character bigrams, recalling that character-based indexing is applied exclusively to character bigrams at this stage of evaluation. The average gain in translation accuracy for all three weighting schemata when katakana are included in the definition of light segments, is negative. The relative drop-off in accuracy lessens as the *sweight* for light segments increases, finally reaching the level of the basic method for an *sweight* value of 1 for light segments. The string comparison methods most keenly affected by a depleted *sweight* value for light segments were weighted sequential correspondence and Sato92, probably because of katakana sequences being weighted down against kanji strings. When katakana characters are excluded from the definition of light segments, on the other hand, slight gains in translation accuracy are observable, peaking for W_2^{JE}[−k]. Combining these two trends together we can conclude that katakana characters have an important role to play in translation accuracy for the given domain, due to their predominant usage in technical terms. Our original hypothesis that hiragana characters are lesser in importance to either kanji or katakana, is tentatively maintained in the modest accuracy gains when katakana characters are treated on a par with kanji characters. While we do not present the results here, the various weighting schemata were also tested over other segment contiguity models. With character unigrams, the drop-off in accuracy when katakana were included in the definition of light segments, was even more accentuated than that documented above for character bigrams.

Turning next to the accuracies for word-based indexing, we see a marked improvement over the relative gains for character-based indexing. Contrary to the results for character-based indexing, here the best results are achieved when katakana characters are treated as going to make up light segments, with the best individual accuracy gain of 6.5% witnessed for W_1^{JE}[+k]. The relative gain for the [+k] system configurations drops continuously as the *sweight* associated with light segments increases, again contrasting with the results for character-based indexing. With katakana excluded from our definition of light segments, on the other hand, the results mirror those for character-based indexing, peaking for W_2^{JE}[−k] at a gain of 5.3%.

We have no immediate explanation for the contradictory results under character- and word-based indexing with respect to the proper treatment of katakana, other than to say that weighting

[19] Numerals and Latin characters are included in the definition of light segments for the given domain, as they are generally used to stipulate part numbers, pressures, tightening torques and the like. There is a high degree of variation in the values of all of these segment types, and little expectation of full lexical match.

	Method	$max = 1$	$max = 2$	$max = 5$	$max = 10$	$max = \infty$
CHAR-BASED	VSM	60.19	60.29	59.45	58.75	58.71
	Token int	60.33	60.47	60.63	60.67	60.59
	3-op edit dist	58.13	58.31	59.41	59.81	59.79
	3-op edit sim	60.73	60.69	60.81	60.71	60.71
	4-op edit dist	51.72	51.58	52.17	52.73	52.71
	4-op edit sim	59.19	59.65	59.27	59.13	59.21
	Weight seq corr	56.87	57.43	58.27	57.83	57.89
	Sato92	54.85	55.57	56.11	55.81	55.83
	Ave. gain	−0.3%	+0.1%	+0.6%	+0.5%	+0.5%
WORD-BASED	VSM	56.41	57.07	57.25	56.87	56.93
	Token int	55.89	56.69	57.97	57.97	58.01
	3-op edit dist	55.97	56.39	57.69	57.89	57.97
	3-op edit sim	56.93	57.01	58.21	57.79	57.79
	4-op edit dist	47.96	48.68	49.44	50.10	50.18
	4-op edit sim	54.59	55.71	56.71	57.23	57.19
	Weight seq corr	53.77	54.13	55.71	55.53	55.53
	Sato92	52.69	53.23	55.19	55.15	55.17
	Ave. gain	−0.3%	+0.7%	+2.8%	+2.9%	+3.0%

Table 2.12: The retrieval performance under IDF-based segment weighting, with varying ceiling values

based on character type has much greater merits for word-based indexing than character-based indexing. Note that while considerably higher gains in accuracy were observed for word-based indexing, the best results for word-based indexing were inferior to the worst results for character-based indexing, with the exception of the weighted sequential correspondence and Sato92 methods.

2.6.6 Experiment VI: The effects of IDF

Above, IDF was suggested as one form of dynamic weighting, a possibility we empirically validate here. Unlike character type-based weighting, IDF derives a distinct weight for each segment type, based on analysis of the frequency of that segment in the overall TM. As a form of dynamic weighting, it can adapt on-line to the statistical idiosyncrasies of any dataset, with the single drawback of on-line TM updates potentially being expensive due to the need to recompute individual segment weights and string lengths. It is our hope, however, that the gains availed by the intricate modelling of the statistical make-up of the TM will more than make up for this.

The formula for determining individual IDF *sweight* values is given in equation (2.3), in which *max* is imposed as a ceiling on the size of IDF scores, suggested as necessary to avoid the case of one segment of inflated *sweight* single-handedly determining the potential for translation retrieval of a translation record. We vary the magnitude of *max* to ascertain the optimal level at which it should be set for the given dataset, with the different values as indicated in Table 2.12. The final column of $max = \infty$ represents the conventional case of there being no cap on the maximum IDF value. As with character type-based weighting, in Table 2.12 we present only the various translation accuracies. The best translation accuracy achieved for each string comparison method, under the two indexing paradigms, is underlined, and the mean gain over the base method from Tables 2.6 and 2.7 is indicated for each indexing paradigm. In line with the findings of the basic

	Method	*Accuracy*	*Non-opt edit dist*	*Unique outputs*	*Ave. time*
CHAR-BASED	VSM	**60.72**	8.42	0.97	1.03
	Token int	**60.68**	8.41	0.96	1.40
	3-op edit dist	56.67	7.92	0.84	1.64
	3-op edit sim	**61.28**	8.17	0.96	2.39
	4-op edit dist	**52.28**	9.37	0.80	4.49
	4-op edit sim	**59.96**	8.14	0.94	5.59
	Weight seq corr	55.97	9.54	0.97	199.91
	Sato92	53.69	10.10	0.97	216.71
WORD-BASED	VSM	57.66 (−5.0%)	8.11	0.96	0.82
	Token int	57.90 (−4.6%)	8.32	0.94	1.08
	3-op edit dist	57.22 (+1.0%)	8.20	0.84	1.29
	3-op edit sim	58.08 (−5.2%)	8.24	0.94	1.77
	4-op edit dist	48.42 (−7.4%)	10.28	0.73	3.36
	4-op edit sim	55.79 (−7.0%)	8.36	0.91	4.41
	Weight seq corr	54.99 (−1.8%)	9.15	0.92	25.55
	Sato92	53.27 (−0.8%)	9.81	0.92	26.34

Table 2.13: Results for fully alphabetised data

model, character-based indexing is based on character bigrams, and word-based indexing on mixed word unigrams/bigrams.

Looking initially at the results for character-based indexing, the different methods peak at *max* values between 2 and 10 and drop back imperceptibly as *max* tends towards infinity. The two lower *max* values produce either a degradation or only slight improvement in accuracy, and real gains are seen for the greater part only when *max* exceeds 2. Comparing the results for IDF to those for character type-based static weighting from Table 2.11, we see that the magnitude of optimal gain in translation accuracy for the two weighting methods is almost identical, at a meager 0.6%.

The results for word-based indexing are more encouraging, with the gain in accuracy gradually edging up towards 3.0% as *max* tends to infinity. That is, the *sweight* ceiling is unneeded, and in fact counter-productive, in the case of word-based indexing. Here again, it is revealing to refer back to Table 2.11 and realise that while translation accuracy has been boosted significantly, the maximum relative gain availed through IDF is under half that achieved with character type-based weighting. In this sense, the relative usefullness of IDF is questionable. Clearly this does not rule out dynamic weighting schemata altogether, however, and we leave the quest for more effective dynamic weighting methods as an item for future research.

Note that, as was the case for character type-based weighting, while the translation accuracy disparity between character- and word-based indexing was narrowed through the use of IDF, character-based indexing remained the clear favourite indexing paradigm.

2.6.7 Experiment VII: The effects of kanji

In Section 2.6.5, we observed that translation performance gains were possible through treating kanji-containing segments as "first-rate citizens" and weighting them up over other segment types. In this section, we look closer at the role that kanji play in translation retrieval, by analysing retrieval performance in the absence of kanji; this is achieved by replacing all kanji characters by

their katakana (alphabetised) equivalents.

An additional motivation for removing kanji is to study the semantic smoothing effects of individual kanji characters, alluded to above (2.3). To take an example, the single-segment nouns 操作 [*sōsa*] and 作動 [*sadō*] both translate into English as "operation" for the given domain, but would not match under word-based indexing. Character-based indexing, on the other hand, would recognise the overlap in character content, and in the process pick up on the semantic correspondence between the two words (assuming that the segment contiguity model includes unigrams).

To test the effect of kanji characters (i.e. ideograms) on translation retrieval performance, we used ChaSen to convert all kanji and hiragana into katakana, generating an essentially alphabetic version of each string, analogous to the case of Thai. In one version of this alphabetised data, the original segmentation was retained, and in a second version, each string was segmented off into individual katakana characters; the rough equivalent of this latter dataset for English, is to delete all word boundaries and chunk the input into syllable chunks, with no differentiation between word and syllable boundaries. We then ran the same methods over this modified input, using exactly the same technique as for the original experiment (including bigrams for character-based indexing and mixed unigrams/bigrams for word-based indexing). The results are presented in Table 2.13, with times calculated relative to 3-operation edit distance over word unigrams from the first experiment.

As for the original experiment, character-based indexing was found to be superior to word-based indexing for the given N-gram models, to a level of statistical significance in most cases. It is worth emphasising here that the system is operating over fully alphabetised Japanese data, such that with character-based indexing, we have only a jumble of what are essentially meta-phoneme pairs to go on in forming a decision as to an appropriate translation candidate for the given input. Even in these impoverished conditions, segmentation is by and large detrimental.

Our original motivation in removing kanji from the data was to test whether the superiority of character-based indexing to word-based indexing was a side-effect of semantic smoothing due to kanji overlap between synonyms. The fact that the same basic set of results as for fully lexically discriminated data, was producible for homogeneous alphabetised data would suggest that it is not kanji that had bolstered the performance of character-based indexing. Having said this, returning to our "operation" synonym example from above, we notice that the overlap in kanji is reflected in overlap in pronunciation, which is retained in the katakana version. In fact, different readings for the same kanji will tend to have subtly or otherwise different meanings, such that by transposing the data into a near-phonetic representation, we are refining the synonym detection process for words sharing some kanji content. That is not to say, however, that words with similar phonetic content will generally be semantically associated in some way, in which sense, it is surprising that the fall-off in performance was as slight as it was. In light of this and the low level of drop-off in performance in the absence of kanji, we can postulate that the character bigram method in particular is highly robust and capable of handling a broad range of input types. This suggests ramifications for alphabet-based, non-segmenting languages such as Thai, where our expectation would be for analogous results to be produced.

As an aside, it is important to pick up on the large-scale slowdown for weighted sequential correspondence and the closely related Sato92 is particularly marked, largely due to them operating over unigrams. This does not occur to the same degree for word-based indexing because segmentation produces segment differentiation, and the remaining string comparison methods are relatively impervious to slowdown under character-based indexing, due to them operating over highly-discriminated character bigrams.

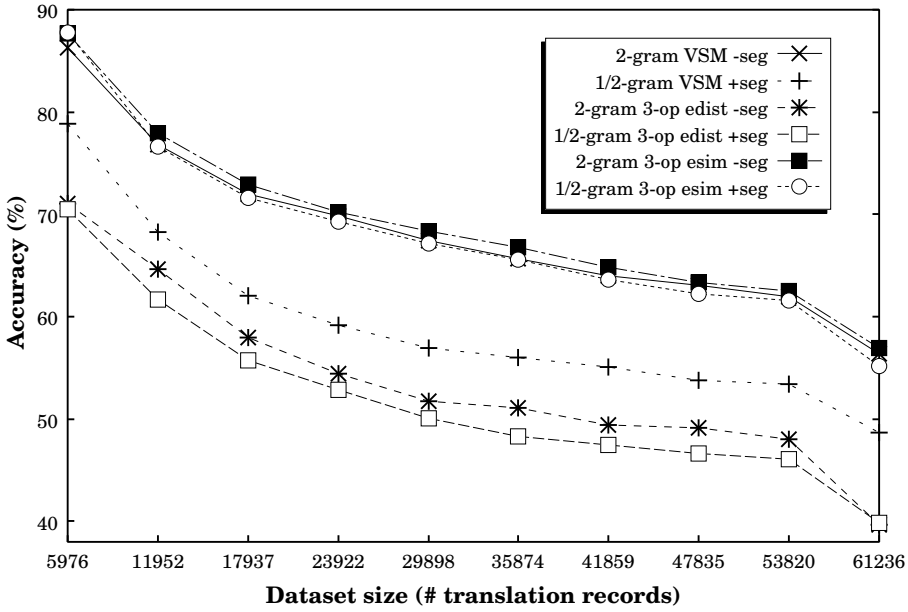

Figure 2.5: The translation accuracies of the vector space model, 3-operation edit distance and 3-operation edit similarity over datasets of increasing size

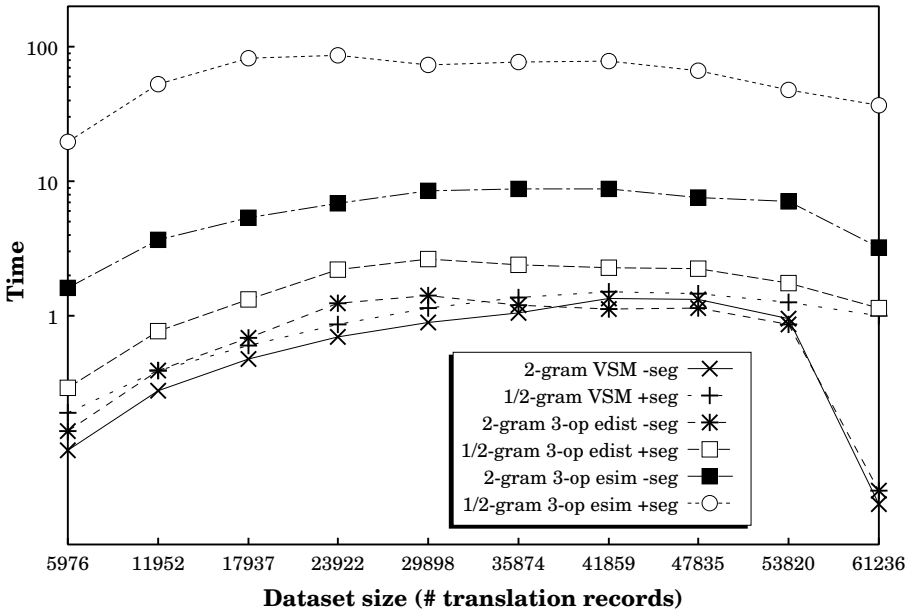

Figure 2.6: The relative retrieval speeds of the vector space model, 3-operation edit distance and 3-operation edit similarity over datasets of increasing size

2.6.8 Scalability of performance

All results to date have arisen from evaluation over a single dataset of fixed size. While this has provided a valuable insight into a wide range of phenomena and the practical applicability of different weighting schemata, we have no way of knowing how the results will scale with TM's of expanded size. It could well happen, for example, that the performance disparity that exists between character- and word-based indexing is narrowed and even reversed as the size of data increases. An orthogonal issue which we have tended to neglect to this point is the inherent speed of the different methods. Does, for example, a 10-times difference in retrieval speed for a given TM size, translate into a 100-times difference for a TM 10 times the size, or is relative speed independent of TM size? In order to address these questions, we employ a second parallel corpus of expanded size, and test the translation performance of a number of different system configurations over increasingly large subsets of the original corpus, to simulate TM size variation.

The particular parallel corpus we used was developed by JEIDA, and originates from government white papers on the Japanese economy, translated into English. We extracted a total of 61,236 translation records from the JEIDA corpus,[20] over 20 times the number of translation records in the construction machinery corpus from above. One reservation with this dataset is that the granularity of alignment is fairly coarse, with a single segment often extending over multiple sentences. The average segment and character lengths of the Japanese component of each translation record are 45.3 and 76.3 respectively, vastly greater than the figures of 14.3 and 27.7 for the construction machinery corpus used for evaluation up to this point. On the English side, the average word length of a single translation record is 35.7, again representing a steep increase from the 13.3 word length of the original corpus. The implication of these inflated figures is that it will frequently occur that we cannot find a translation record similar enough to the input to be useful, and the bulk of translation candidates will simply be empty strings. Naturally, this trend will be lessened as the size of the TM increases and we gain access to a more linguistically diverse range of translation records. We return to this point below.

We simulate TM's of differing size while nullifying the effects of genre and stylistic change, by randomly splitting the JEIDA corpus into ten partitions, and running the various system configurations first over partition 1, then over combined partitions 1 and 2, and so on until all ten partitions are combined together into the full corpus. For each of the ten datasets produced in this manner, we computed the translation accuracy and retrieval speed, in the same manner as indicated above. That is, translation accuracy was calculated to be the proportion of optimal outputs produced by a given method, averaged over the accuracies for translation optimality as determined according to word bigram-based edit distance and that for unigram weighted sequential correspondence. Retrieval speed was calibrated based on the mean retrieval speed for word unigrams under 3-operation edit distance over the original corpus (i.e. relative speeds are directly comparable to those presented to this point).

Due to computational constraints, we were unable to test all string comparison methods over the JEIDA corpus, and instead singled out three representative methods which performed particularly well in preceding evaluation, namely the vector space model, 3-operation edit distance and 3-operation edit similarity.[21] That is, we tested one bag-of-words and two segment order-sensitive methods, and one distance- and two similarity-based methods, selecting those methods

[20] While the source corpus contains more translation records than this and is exhaustively aligned, some alignment links are rated as being of better quality than others. Rather than compromising the quality of the overall TM, we chose to take only those translation records of a certain quality, resulting in the indicated count. Despite us using only a fraction of the total content of the corpus, we will refer to the derived TM as the JEIDA corpus from here on.

[21] Token intersection was also tested, in fact, and found to perform almost identically to VSM. Detailed results are omitted from this chapter.

which tended to perform best in each such category.

The transition in translation accuracy for the different methods over the ten datasets of varying size, is indicated in Figure 2.5, with each string comparison method tested under character bigrams ("2-gram −seg") and mixed word unigrams/bigrams ("1/2-gram +seg") as above. Note that while we do not present the results here, other models of segment contiguity were tested, in order to verify the superior performance of these methods over the two indexing paradigms. Results remarkably similar to those for the construction machinery dataset were obtained and character bigrams and word unigrams/bigrams found to outperform other methods within the respective indexing paradigms.

A striking feature of the various graphs is that they are right-decreasing, which is essentially an artifact of the size of each translation record and resultant data sparseness. That is, for smaller datasets, in the bulk of cases, no translation record in the TM is similar enough to the input to warrant consideration as a translation candidate. This is reflected in the following breakdown of the mean ratio of optimal translation sets made up solely of the null string, for each dataset (averaged across the optimal translation data from weighted sequential correspondence and bigram-based 3-operation edit distance):

TM size in translation records (% of total)	*Proportion of singleton null-string optimal translation sets*
5976 (10%)	82.3%
11952 (20%)	74.3%
17937 (30%)	69.4%
23922 (40%)	65.7%
29898 (50%)	62.7%
35874 (60%)	59.9%
41859 (70%)	57.3%
47835 (80%)	55.2%
53820 (90%)	53.4%
61236 (100%)	50.6%

One key trend in Figure 2.5 is the relative disparity in accuracy between character- and word-based indexing for each of the three string comparison methods. For all methods, the absolute margin between the character- and word-based implementations is maintained at a relatively constant level as the TM size grows, with character-based indexing coming out on top for all methods. This throws weight behind our claim as to the superiority of character-based indexing over word-indexing, but also goes one step further in suggesting that we can expect this trend to be reproduced for all TM sizes and a variety of data types. It is interesting to note that the disparity between character- and word-based indexing is greatest for VSM, and that with the two segment order-sensitive methods, the spread is much smaller (but still significant). This is contrary to our earlier results, where token intersection and VSM showed the smallest gains for character-based indexing. As for the original dataset, 3-operation edit distance outstripped 3-operation edit similarity in terms of raw accuracy. VSM turned in a strong performance under character-based indexing, approximately equally the accuracy of character-based 3-operation edit similarity for all dataset sizes. Based on this result, we must conclude that there is little to distinguish bag-of-words from segment order-sensitive methods in terms of retrieval accuracy.

As with the original dataset from above, 3-operation edit similarity was the strongest performer, with VSM coming second and 3-operation edit distance a distant third. The sizeable disparity of around 20% between 3-operation edit similarity and 3-operation edit distance was unrivalled in

other evaluation, casting doubts as to the robustness of 3-operation edit distance over different data types.

Next, we turn to consider the mean retrieval times for each method, under the two indexing paradigms. Times are presented in Figure 2.6, calibrated according to 3-operation edit distance run over word unigrams for the construction machinery dataset. The reader should note that a logarithmic scale has been used on the y-axis in order to fit the full fan-out of retrieval times onto a single graph. VSM and 3-operation edit distance were the most consistent performers, both maintaining retrieval speeds in line with those for the original dataset at around or under 1.0 (i.e. the same time as 3-operation edit distance run over word unigrams for the construction machinery dataset). Most importantly, only minor increases in retrieval speed were evident as the TM size increased, which were then reversed as the retrieval time dropped back for the larger datasets. This same convex shape was observed for the 3-operation edit similarity curve, although the slope of the curve was much steeper over the smaller datasets, with the final running time for mixed word unigrams/bigrams over 100 times that for VSM or 3-operation edit distance over the same dataset. For character bigrams also, the final retrieval time was markedly slower than those for the other two methods, at around one order of magnitude. Clearly, this blow-out in retrieval time for 3-operation edit similarity is not desirable if we wish our TM system to be scalable, in which sense the coupling of 3-operation edit distance with lower order N-gram models is computationally inappropriate.

To combine the findings for accuracy and speed, VSM under character-based indexing suggests itself as the pick of the different system configurations, combining both speed and consistent accuracy.

To sum up the findings of this section, character-based indexing was found to be superior to word-based indexing once more, but for a distinct dataset this time. Perhaps more importantly, VSM and 3-operation edit distance returned almost constant mean retrieval times over datasets of varying size, demonstrating their scalability over large TM's. 3-operation edit similarity proved to be considerable slower, particularly for a mixed N-gram model, but the blow out was at least found to be bounded, at around 10-times the speed of the other two methods for character-based indexing.

2.6.9 The relative score ranking for the similarity-type methods

In stipulating the numerical cutoffs on translation utility for the similarity methods in Section 2.5.3, we gave no justification for the proposed values. In this section, we give a visual depiction of why the suggested cutoffs are appropriate, as part of analysing the distribution of scores for each of the similarity methods. While this is peripheral to the main thread of this chapter, it does give rise to some interesting results.

Essentially what we did was to run each of the similarity methods, namely VSM, token intersection, 3-operation edit distance, 4-operation edit distance and weighted sequential distance, over the construction machinery corpus, and taking each translation record in turn, calculate the similarity of that L1 string to all other L2 strings. We then ranked the resultant similarities in decreasing order of magnitude, and combined all such "similarity chains" together by averaging similarities of corresponding rank together. That is, for each string S_i, similarity chains of the form $sim_1^{S_i}, sim_2^{S_i}, ..., sim_{n-1}^{S_1}$ (where $sim_1^{S_i} \geq sim_2^{S_i} \geq ... \geq sim_{n-1}^{S_i}$) were formed based on string comparison with each remaining string S_j in the dataset, from which the combined similarity chain $sim_1^S, sim_2^S, ..., sim_{n-1}^S$ was produced, where $sim_i^S = \frac{\sum_{j=1}^{n} sim_i^j}{n}$. This combined similarity chain gives us an idea of the rate of fall-off in similarity values. Further, by comparing the combined similarity chains for an array of similarity-based string comparison methods, we can compare the

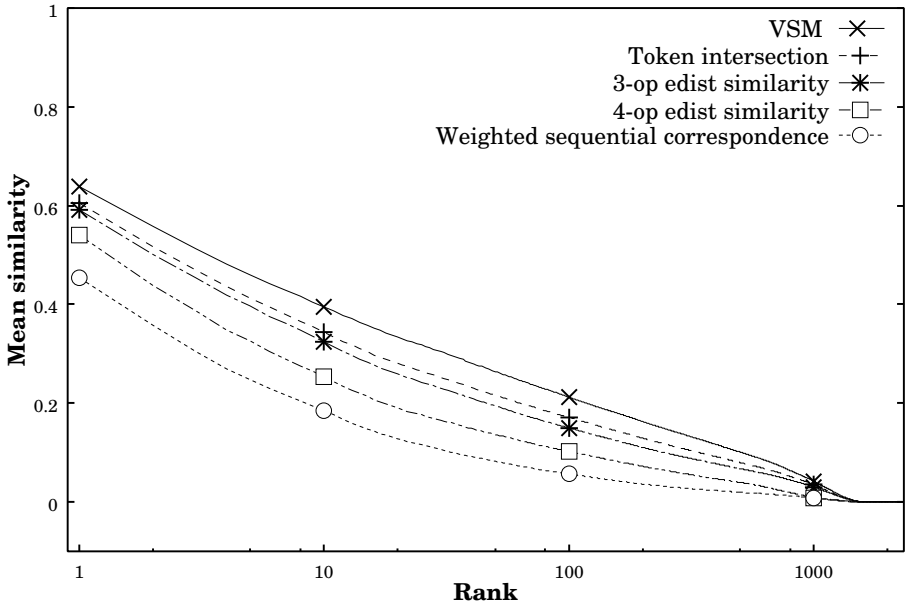

Figure 2.7: Combined similarity chains for the various similarity-based string comparison methods

average degree of similarity for the k-th best matching string, for example.

The similarity chains are presented in Figure 2.7, with a logarithmic scale used on the x-axis to indicate the rank at which each mean similarity was produced. Note that none of the curves are smoothed, and that the shapes are indicative of the actual sequence of mean similarity values produced for each method. The first thing to notice here is the relative ordering of the curves from top to bottom, with VSM returning inflated similarity values, and weighted sequential correspondence diminished values. This ordering is not particularly surprising when one goes back and carefully inspects the formulations for each of the methods. What is striking is that the relative depreciation in similarity values for our assortment of methods, is roughly equal, and moreso that the curves are approximately linear. This equates to a Zipf-like effect, in that the rank is roughly proportional to the square of the similarity value at that rank.

As interesting as this may be, our primary motivation in generating the combined similarity chains was to derive a correlation between the translation utility cut-offs and the morphology of the curves. The correlation comes in drawing a vertical line through the curves at an arbitrary point along their length. Notice that the suggested static cut-offs for the different methods roughly correspond to the point $x = 10$. That is, we are allowing each method the same basic scope to match, for the given dataset at least.

2.7 English–Japanese Translation Retrieval Results

Based on the combined results of Section 2.6, we drew the conclusion that character-based indexing is superior to word-based indexing, and also that there is little to separate bag-of-words and segment order-sensitive methods in general. These findings were based on analysis of Japanese, however, and we have little way of knowing how they translate across to other languages. For this reason, in this section, we consider a new language direction for translation retrieval, namely English–Japanese, so as to be able to reuse the data we have at hand. Another reason that we are interested in

	Method	$\mathbf{W}_1^{\mathbf{EJ}}$	$\mathbf{W}_2^{\mathbf{EJ}}$	$\mathbf{W}_3^{\mathbf{EJ}}$	$\mathbf{W}_4^{\mathbf{EJ}}$	$\mathbf{W}_5^{\mathbf{EJ}}$	$\mathbf{W}_6^{\mathbf{EJ}}$
1-GRAM	VSM	46.25	50.10	50.16	49.90	50.36	46.46
	Token int	49.48	50.60	50.32	49.96	49.70	47.06
	3-op edit dist	47.12	48.78	49.56	48.92	48.94	46.90
	3-op edit sim	50.94	51.16	51.16	51.08	50.74	49.56
	4-op edit dist	42.83	43.80	43.86	43.96	42.94	41.26
	4-op edit sim	49.24	49.88	50.52	50.28	49.80	48.08
	Weight seq corr	50.58	51.08	51.34	50.96	50.94	50.50
	Sato92	49.72	50.08	49.70	49.62	49.52	46.98
	Ave. gain	−2.2%	+0.2%	+0.5%	0.0%	−0.5%	−4.6%
2-GRAM	VSM	51.60	51.40	51.60	51.32	51.42	51.32
	Token int	52.02	52.02	51.86	52.65	51.62	51.86
	3-op edit dist	50.12	49.88	49.68	49.62	49.40	49.54
	3-op edit sim	51.68	52.26	51.92	51.54	52.06	51.74
	4-op edit dist	43.94	43.62	43.56	43.98	43.86	44.00
	4-op edit sim	50.36	50.10	49.94	50.30	50.20	49.90
	Ave. gain	+0.1%	−0.0%	−0.2%	0.0%	−0.2%	−0.3%
1+2-GRAM	VSM	49.59	52.98	53.38	53.27	52.73	50.64
	Token int	53.61	52.94	52.60	52.44	52.79	52.08
	3-op edit dist	52.36	51.62	51.70	52.18	51.08	50.30
	3-op edit sim	52.86	52.64	52.73	52.44	52.38	52.26
	4-op edit dist	43.72	43.20	43.84	43.46	43.32	42.18
	4-op edit sim	50.54	50.62	50.62	50.54	50.54	49.76
	Ave. gain	−0.4%	−0.1%	+0.2%	0.0%	−0.4%	−1.7%

Table 2.14: The retrieval performance for English–Japanese translation retrieval different character-based, using a unigram (1-GRAM), bigram (2-GRAM), and mixed unigram/bigram (1+2-GRAM) segment contiguity model

English–Japanese translation retrieval is to justify the *sweight* schema chosen for the evaluation of translation optimality above, and in particular our handling of stop words.

Naturally, words are explicitly segmented off in English, making any discussion of character- vs. word-based indexing meaningless. The bag-of-words vs. segment order sensitivity issue is still pertinent, however, as is local segment contiguity. We therefore consider the full range of string comparison methods, as well as each of word unigrams, word bigrams and mixed word unigrams/bigrams.

In evaluation of translation optimality, we employ the same method as described above for Japanese–English translation retrieval (combining the results for 3-operation edit distance and weighted sequential correspondence), excepting that character-based was used in place of word-based indexing; we employed the L2 *sweight* schema given in Section 2.5.3 for Japanese.

As with Japanese–English translation retrieval, we tested the effects of different static *sweight* schemata, operating over word type. Principally, we distinguish between punctuation, stop words and any other words, with stop words defined as those contained within the SMART (Salton 1971) stop word list, as was the case with optimal translation determination for Japanese–English translation retrieval. The full range of *sweight* schemata evaluated is as follows:

Segment type	W_1^{EJ}	W_2^{EJ}	W_3^{EJ}	W_4^{EJ}	W_5^{EJ}	W_6^{EJ}
punctuation	0	0	0	0	0	0
stop words	0	0.05	0.1	0.2	0.5	1
other words	1	1	1	1	1	1

Essentially, we vary the relative import of stop words by varying the associated *sweight* between 0 and 1, in the manner adopted for character type-based weighting in Section 2.6.5. Note that W_1^{EJ} (i.e. stop words filtered out of all strings) corresponds to the L2 *sweight* schema utilised in Japanese–English translation retrieval. A stop word in the case of an N-gram model of order 2 or greater is interpreted as an N-gram of stop words. That is, a segment is given the indicated stop word weight only in the case that it is made up entirely of stop words.

The results for the three segment contiguity models, as tested under the different *sweight* settings are presented in Table 2.14. The first thing to note is the high correlation between these results and those for Japanese–English translation retrieval, namely the superiority of word bigrams and mixed unigrams/bigrams to unigrams, and also the relative performances of the different string comparison methods. Additionally, the bag-of-words methods perform at an equivalent level to the better segment order-sensitive methods, with 3-operation edit similarity coming out as the overall method of choice. The strong performance of the bag-of-words methods is perhaps a little surprising, and discounts any suggestion that their strong showing when run over Japanese is related to its relatively free word order. These findings underline the language universality of the conclusions drawn for Japanese–English translation retrieval, although still further language pairs and translation directions must be tested before making too strong a claim in this regard.

One very important fact evidenced by the results is that there is a strong case for treating stop words differently to other words, and that for word bigrams at least, it is possible to filter stop words (i.e. adjacent pairs of stop words) out of strings all together. For the remaining two models of segment contiguity, which contain a component of unigrams, both scoring stop words too low and scoring stop words too high were found to be detrimental, with $\mathbf{W_3^{EJ}}$ providing the optimal trade-off between the two.

Our choice of word bigrams for the determination of translation optimality in Japanese–English translation retrieval, despite mixed word unigrams/bigrams providing slight enhancements in accuracy, is rationalised through consideration of retrieval speed. That is, while the scales are tipped marginally in favour of mixed word unigrams/bigrams in terms of retrieval accuracy, they point unequivocally to word bigrams when retrieval time is taken into account, particularly for large-

scale TM's such as the JEIDA dataset. The $\mathbf{W}_1^{\mathbf{EJ}}$ *sweight* setting would also appear optimal for word bigrams in terms of accuracy, and again have advantages in terms of speed (i.e. we can completely filter our stop word bigrams from the dataset, reducing both the search space and number of segment comparisons).

2.8 Discussion and Concluding Remarks

To finish up with, we spend a moment reflecting on the combined results for Japanese–English translation retrieval. Dichotomies established as key points of interest at the outset of evaluation where character- vs. word-based indexing, and bag-of-words vs. segment order-sensitive string comparison methods, as well as the effectiveness of N-gram based segment contiguity modelling. Lesser areas of interest included segment contiguity-blind vs. segment contiguity-aware string comparison methods, 3-operation edit distance/similarity vs. 4-operation edit distance/similarity, and weighted sequential correspondence vs. Sato92. Here, we review the findings and discuss related literature for each of these items.

First, character-based indexing was found to be superior to word-based indexing almost without exception, and generally to a level of statistical significance. The disparity between the two indexing paradigms was particularly noticeable when character bigrams were played off against mixed word unigrams/bigrams, the best-performing models of segment contiguity for the respective indexing paradigms. As noted above, most research relating to translation retrieval from Japanese has taken it for granted that segmentation will enhance retrieval performance. That is, the overhead associated with segmentation was implicitly assumed to be justifiable in terms of producing greater retrieval robustness, or in other words a "slower but steadier" method. While we were prepared to question this standpoint, we were certainly not expecting hare-like character-based indexing to come out the clear leader. Looking to related literature, there is limited support for our findings, particularly in relation to IR where Fujii and Croft (1993) pointed out at a relatively early stage that character-based indexing performs comparably with word-based indexing in Japanese IR. In research laying the foundations for this chapter, Baldwin and Tanaka (2000b) report similar findings for a Japanese–English translation retrieval task. To the author's knowledge, there is no empirical evidence to suggest that segmentation enhances accuracy in an analogous retrieval context.

Considering next segment order sensitivity in the string comparison method, the findings for retrieval accuracy were somewhat mixed, with bag-of-words methods coming out on top in a select few cases, but segment order-sensitive methods more generally holding a lead. All in all, however, there was very little to separate the two in most cases. In terms of speed, the bag-of-words methods were dependably fast, whereas depending on the segment order-sensitive method, sizeable blow-outs in time were observed, particularly for weighted sequential correspondence and Sato92, and to a lesser degree, the edit similarity methods. For real-world applications, speed is obviously going to be a concern. Given that there is little to distinguish the two method types in terms of retrieval accuracy, bag-of-words methods must be recognised as the more attractive option. Once again, therefore, the sleek, seemingly fragile method comes out on top.

These findings contradict those of Baldwin and Tanaka (2000b), who found segment order-sensitive methods to excel over bag-of-words methods. The principal cause for this was that calculation of translation optimality was based exclusively on 3-operation L2 edit distance, producing a bias towards 3-operation L1 edit distance in retrieval and against the bag-of-words methods. We claim that our results reflect more accurately the true performance of the different methods, due to evaluation being averaged across 3-operation edit distance and weighted sequential correspondence, two heterogeneous methods picking up on distinct effects in the output.

Recall that our comparison of bag-of-words and segment order-sensitive methods is based on

what is suggested to be a representative selection of comparison methods in each category. At no point do we claim that the methods targeted herein are optimal, and significant improvement may well be possible with more sophisticated matching means. We do suggest, however, that there is the most room for improvement with the bag-of-words methods, as the only types of optimisation employed are an inverted file and cache of string segment lengths, unlike the segment order-sensitive methods were much effort was put into getting the maximal possible performance from the basic method. Optimisation of the bag-of-words methods would thus open the way for more computationally expensive means of comparison, without sacrificing TM access times.

One area of research where the bag-of-words vs. segment order-sensitive debate has been the focus of attention of late, is question answering. While there are crucial differences between translation retrieval and question answering, the basic essence of the two tasks is the same, in that we are after the output corresponding most closely to the input.[22] There have been varying reports on the successes of bag-of-words and segment order-sensitive methods (Prager *et al.* 2000; Singhal *et al.* 2000; Voorhees 2000), and indications are that methods such as VSM are not sufficient for attaining the level of granularity required for question answering.

Segment contiguity modelling in the form of N-grams was found to be a valuable supplement to the various system configurations. Of unigrams, bigrams and mixed unigrams/bigrams, bigrams were found to perform best for character-based indexing, and mixed unigrams/bigrams for word-based indexing. Despite segment contiguity modelling being orthogonal to both the choice of indexing paradigm and the type of string comparison method, it interfaces closely with each of this, in that character bigrams are a computationally lightweight approximation of word unigrams, and bag-of-words methods coupled with higher-order N-gram models approach segment order-sensitive methods through gaining access to local segment ordering. Our results show unequivocally that the simple regular chunking of segments is more effective than segmentation at providing a description of character-connectivity, while retaining the same speed advantages as word-based indexing.

We are by no means alone in employing N-grams, and their use is well established in the IR community in particular, as well as in speech processing as a crude but effective "linguistic model" in filtering out unlikely word combinations.

Turning to more specific issues, weighted sequential correspondence was suggested as an enhancement over the edit distance and similarity methods, in that it fed off segment order at the same time as preferring contiguous matches over non-contiguous matches. In evaluation, this awareness of segment contiguity produced no real gain in retrieval accuracy, and simply made the method more cumbersome. In particular, it was found that combining a segment contiguity-blind, segment order-sensitive method with a higher-order N-gram model, we could both accelerate retrieval times and benefit from limited modelling of local contiguity. Given the gains availed by edit distance and similarity over weighted sequential correspondence when coupled with N-gram models, we feel confident in stating that weighted sequential correspondence has little practical worth. Note that work has been done on buffering the inflated retrieval times seen for weighted sequential correspondence, namely in the work of Sato (1994), where a massively-parallel computer was used to speed up retrieval. Even when the speed factor is removed, however, weighted sequential correspondence is inferior to the other methods in accuracy.

In analysing the performance of weighted sequential correspondence, we compared it to the closely-related method of Sato (1992), and found that the slight modification made to Sato's method led to significant performance gains. Similarly with edit distance and similarity, we compared the classic 4-operation methods to simplified 3-operation methods, generated through the removal of the substitution operator. Our prediction as to the superiority of the 3-operation methods was

[22] This is the case with the TREC question answering track, at least, where the answer is in the form of a fixed-sized window of the target text.

verified in evaluation, and 3-operation edit similarity found to be the best of the four variants.

To summarise, this research is concerned principally with the relative import of segment order and segmentation on translation retrieval performance for a TM system, as well as the value of N-gram-based local segment contiguity modelling. We simulated the effects of word order sensitivity vs. bag-of-words word order insensitivity by implementing a total of seven comparison methods: two bag-of-words approaches (the vector space model and "token intersection") and six word order-sensitive approaches (3-operation edit distance and similarity, 4-operation edit distance and similarity, and two versions of "weighted sequential correspondence"). Each of these metrics was then tested under character-based and word-based indexing and in combination with a range of simple and mixed N-gram models, and the relative performance of each such system configuration evaluated. Evaluation of the output took the form of determining the set of "optimal" translation candidates through automatic means, and calculating the degree of correspondence between this set and the actual translation output. Character-based indexing was found to be superior to word-based indexing in all cases tested, particularly when supplemented with a character bigram model. Segmentation was thus discounted as being an unwarranted luxury. The bag-of-words methods were found to be about equal in retrieval potential to the better of the segment order-sensitive methods, but the former tended to run faster.

In order to evaluate the impact the segmentation system had on final retrieval accuracy for word-based accuracy, we ran the various string comparison methods over the segment output of the ChaSen, JUMAN and ALTJAWS systems for the same dataset, as well as lexically normalised output from ALTJAWS. Our experiment showed a strong correlation between retrieval accuracy and reliability/consistency in the segmentation of compound nouns, and that lexical normalisation produces marginal gains in retrieval performance. We went on to test a range of static segment weighting schemata based on character type, and also dynamic weighting in the form of IDF, and showed that modest accuracy gains were possible through such methods. We also tried converting the dataset into katakana and seeing what difference this would make to translation retrieval, and found that we were able to reproduce the same basic results as for fully lexically-differentiated data. From this, we hypothesised that the role of kanji in retrieval was not as crucial as we had originally thought. After having established a range of results over a limited dataset, we experimented with a second dataset of a more practical size, and investigated the relation between retrieval speeds and accuracy for representative methods over different sized portions of the overall corpus. This resulted in the confirmation of our finding for the original dataset that (bigram-based) character-based indexing is superior to (mixed unigram/bigram-based) word-based indexing, and also that the retrieval speed for the vector space model and 3-operation edit distance remains constant as the TM size increases, unlike 3-operation edit similarity which was slower by on order of one or two. The retrieval accuracy of the vector space model and 3-operation edit similarity was roughly equivalent, whereas 3-operation edit distance performed poorly. On a sidetrack to the main heading of evaluation, we plotted the relative depletion in score for each of the similarity-based methods. Finally in evaluation, we reversed the language direction of translation retrieval, and for English–Japanese retrieval, verified the ranking of the string comparison methods derived in Japanese–English retrieval.

Chapter 3

Supervised Learning Meets Japanese Relative Clause Constructions

This work represents a continuation and validation of research commenced in Baldwin (1998b), and targets Japanese relative clause constructions (RCCs). As in the original research, our particular interest is in the classification of RCCs according to the semantic relationship between the relative clause body and head NP. In the original research, an elaborate rule system was devised to analyse this relationship, interfacing with verb classes, rudimentary semantic classification and case frame transposition. While commendable results were possible with the proposed system architecture, there were inevitable doubts as to the scalability of the rule set to other domains and open data, and also the optimality of the rule set. In order to gain an insight into such issues, we call on data-driven supervised learning techniques to determine what level of accuracy is attainable and examine the composition of the resultant classifier.

This chapter is made up of three main parts. First, we define the Japanese RCC interpretation task and outline the interpretational framework on which this research is based. Next, we describe a basic supervised learning-based approach to RCC interpretation in which we classify RCCs based around a pre-determined system of features. Finally, we consider expanding the original feature set, as part of which, we develop an automatic feature selection method to determine which features are pertinent to the classification task at hand.

Supervised learning is exemplified by way of two systems based on fundamentally different learning paradigms. The systems in question are C4.5 and TiMBL, the first of which is an eager learner, and the second a memory-based learner. By running these two systems in parallel, we aim to verify the universality of results.

3.1 Introduction

Japanese relative clause constructions have the general structure `[[S][NP]]`, and constitute a noun phrase. We will term the modifying `S` the "relative clause", the modified NP the "head NP", and the overall NP a "relative clause construction" or RCC.

Japanese RCCs are notoriously difficult to categorise semantically, and have been classified according to widely differing type hierarchies by different researchers (Teramura 1975–78; Sato 1989; Matsumoto 1997; Sirai and Gunji 1998). One of the few areas in which researchers concur is that Japanese RCCs can be broadly divided into the **case-slot gapping** and **attributive** types. With case-slot gapping RCCs, the head NP can be seen to instantiate a case slot position described by the matrix verb of the relative clause, and with attributive RCCs, the relative clause is simply

an attributive modifier of the head NP. Different claims have been made as to the roles of syntax, semantics and pragmatics (or frame semantics) in the make-up of these two main types, but we take the line that semantics is the predominant factor governing RCC construal. Essentially, we make the claim that the relative clause selects for the head NP while the head NP selects for the relative clause; case-slot gapping RCCs are produced in the case that the former process overshadows the first, and attributive RCCs are produced in the reverse case. There is a close relationship between syntax and semantics here, in that syntax describes the skeleton subcategorisation frame which semantics fleshes out by way of selectional restrictions on each case slot. Pragmatics also has a role to play in rating the plausibility of different interpretations, although we make the claim that this is very much the final touch and peripheral to the core construal process.

Our objective in this chapter is, given a taxonomy of Japanese relative clause construction types and a basic corpus of Japanese relative clause construction instances, to investigate the success of various parameter configurations in classifying relative clause constructions. The system of relative clause construction ("RCC") types was originally devised in Baldwin (1998b), in addition to proposing a set of lexical and semantic parameters to characterise RCCs according to the proposed typology; the proposed parameter set was implemented in the form of a declarative rule set to determine RCC type. One area in which we are particularly interested is the complementation of the original parameter set with new features, in order to attain gains over the performance ceiling hit with the original rule set.

The parametric characterisation of RCCs is dogged by analytical ambiguity, in particular for word sense, phrase boundary and phrase head/attachment ambiguity. The latter two of these concerns are not an issue in our case, as we assume that we have a syntactically-annotated corpus (the EDR corpus (EDR 1995), for the purposes of this chapter), which we seek to augment with semantic tags; this leaves the question of word sense ambiguity. We sidestep full-on verb sense disambiguation by associating a unique case frame with each verb stem type. Even here, however, we must have some means of dealing with verb homonymy and integrating analyses for cosubordinated relative clauses. We investigate various techniques to resolve such ambiguity and combine the analysis of multiple component clauses.

One feature of the proposed system is that it is designed for shallow, low-cost analysis, centring principally around a basic case frame and verb class description. The analysis is thus suggested to be readily applicable to MT from Japanese (e.g. Japanese–English MT), in determining the semantic type of the RCC for transferral across to the target language. This is particularly true for case-slot gapping RCCs, where it is a trivial process to produce an English translation of a Japanese RCC given that we have direct access to the position from which the head NP was gapped. For attributive RCCs also, each RCC class is associated with a translation template which the relative clause and head NP can be slotted directly into, given that we know the type of the L1 (Japanese) RCC.

3.2 Definitions

Relative clause modification occurs in three major semantic categories, indistinguishable lexically: case-slot gapping, attributive and idiomatic. With **case-slot gapping** RCCs (aka "internal"/"inner relation" relative clauses (Teramura 1975–78) or "clause host" RCCs (Matsumoto 1997)), the head NP can be considered to have been gapped from a case slot subcategorised by the main verb of the relative clause. Note here that, whereas the case slot from which gapping has occurred tends to have a distinctive case marking schema, that marking is not preserved either within the relative clause or on the head NP. **Attributive** RCCs (aka "external"/"outer relation" relative

clauses (Teramura 1975–78) or "noun host" RCCs (Matsumoto 1997)[1]) occur when the relative clause modifies or restricts the denotatum of the head NP. **Idiomatic** RCCs are produced when the overall relative clause construction produces an RCC-specific idiomatic reading. One feature of idiomatic RCCs is that they can be described by a largely lexicalised construction template, and are incompatible with conjugational alternation and peripheral case slots. Examples of the three RCC types are, respectively:

(3) DIRECT OBJECT CASE-SLOT GAPPING:
kinō *katta* *bōsi*
yesterday bought hat
"the hat (I) bought yesterday"

(4) CONTENT:
bōsi-o *katta* *riyū*
hat-ACC bought reason
"the reason (I) bought a hat"

(5) IDIOMATIC:
hito-o *miru* *me*
person-ACC see eye
"the ability to judge a person"

The inherent difficulty in determining the type of RCC construal comes from the fact that these 3 categories of RCC construal and the 19 RCC sub-types subsumed by them, are lexically indistinguishable. The semantic type of the RCC is therefore not readily accessible from a simple structural analysis of the RCC as contained within a standard treebank.

[1]Matsumoto (1997) also defines a hybrid third RCC type: clause/noun host RCCs. In Matsumoto's terms, this is the instance of the head NP being framed by the relative clause, at the same time as the relative clause being framed by the head NP. We recognise that interaction does take place between case-slot gapping and attribution for certain RCC types, namely temporal and local relative RCCs, but maintain that this is not a true instance of case-slot gapping. If it were the case that case-slot gapping was occurring with these RCC types, it should not be possible to instantiate the temporal and locative case slots, respectively, within the relative clause. This is possible, however, suggesting that a true gap does not exist:

a. *1963-neN-ni* *Kenedī daitōryō-ga* *aNsatu-sareta* *yokutosi*
1963-DAT Kennedy president-NOM was assassinated the next year
(lit.) "the year after President Kennedy was assassinated in 1963"

b. **1963-neN-ni* *Kenedī daitōryō-ga* *aNsatu-sareta* *tosi*
1963-DAT Kennedy president-NOM was assassinated year
(lit.) "the year President Kennedy was assassinated in 1963"

Naturally, the case frame must be compatible with a temporal or locative case slot in order for the realisation of temporal and local relative RCCs, respectively, to be possible. However, we draw the distinction between simple case slot compatibility and the head NP actually being linked to that case slot. For further details, see Baldwin (1998b).

3.2.1 Case-slot gapping

Sub-categories

For our purposes, **case-slot gapping** is considered to occur in nineteen sub-categories,[2] which can be further partitioned into eight **complement case slot** types (case slots described in the case frame and intrinsically linked to predicate) and eleven **peripheral case slot** types (case slots providing peripheral information about the action/state described by the relative clause); these are detailed in Tables 3.1 and 3.2, respectively. For full definitions of the different case slot types, the reader is referred to Baldwin (1998b).

Type	*Description*
Subject	Surface subject position case slot gap E.g. *watasita hoNniN* "the person who handed (it) over"
Direct object	Surface direct object position case slot gap E.g. *watasita mono* "the thing handed over"
Indirect object	Surface indirect object position case slot gap E.g. *watasita aite* "the person (I) handed (it to)"
Co-actor	Co-actor[3] position case slot gap E.g. *kaidaN-sita aite* "the person (I) talked with"
Co-patient	Co-patient[4] position case slot gap E.g. *eNziN-o kōkan-sita siNpiN* "(lit.) the new part replacing the engine"
Causee	Causee position case slot gap E.g. *Tarō-ga ikaseta hito* "the person Taro made go"
Passive agent	Passive agent position case slot gap E.g. *asi-o humareta aite* "the person who trod on (my) foot"
Stative topic	The topic of the state described by the relative clause E.g. *daiya-no haitta yubiwa* "a ring embedded with diamonds"

Table 3.1: The complement case-slot gapping types

Grammatical relations vs. case-roles

In the complement case slot set, syntactic markers such as subject, indirect object and passive agent override the more conventional case-role descriptors of agent and patient. For a passivised

[2] While there are nineteen sub-categories at the matrix relative clause level, case-slot gapping can also occur from a subordinate clause in the relative clause, in the form of "subordinate case-slot gapping" (Baldwin 1998b):

[[	*kaigi-ni*	*saNka-suru*]	*koto-ga*	*kakuniN-sareteiru*	]	*hito*
	meeting-DAT	attends	-NML-NOM	is confirmed		person

"people who are confirmed to attend the meeting"

Here, the case-slot gap appears in a subordinate clause, which is either marked with the quotative case marker (*to*) or nominalised (as observed above for (a)). In all other respects, subordinate case-slot gapping is identical to conventional case-slot gapping. Rather than treating these as distinct types of case-slot gapping, therefore, we mark subordinate case-slot gapping RCCs as such, and in analysis, treat the subordinate clause as if it were the matrix clause. The method described here is thus oblivious to the existence of subordinate case-slot gapping.

Type	*Description*
Instrument	The head NP corresponds to the instrument facilitating the action described in the relative clause E.g. *kiru mono* "(lit.) a thing with which to cut"
Degree	The head NP describes the degree or measure of the action described by the relative clause E.g. *hasiru sokudo* "speed at which (it) runs/drives"
Temporal	The head NP describes the point in time at which the event described by the relative clause occurred E.g. *okiru zikaN* "time at which (they) wake up"
Temporal durational	The head NP describes the interval over which the event described by the relative clause occurred E.g. *kaigi-ga okonawareru issyūkaN* "the week the conference will be held over"
Locative	The head NP describes the location at which the action described by the relative clause occurred E.g. *okononawareru basyo* "the place where (it) will be held"
Local ablative	The head NP describes the physical location from which the action described by the relative clause emanated E.g. *deta basyo* "the place where (it) came out from"
Local allative	The head NP describes the physical location towards which the action described by the relative clause is directed/headed E.g. *mukau saki* "the place (they) are headed for"
Local perlative	The head NP describes the physical path traversed by the action described by the relative clause E.g. *aruku miti* "the road (they) walk along"

Table 3.2: The peripheral case-slot gapping types

relative clause such as the following, therefore, we would classify the RCC as being of the subject case-slot gapping type rather than the direct object case-slot gapping type, due to the case slot linked to the head NP being a surface subject:

(6) *seNzitu* *okonawareta* *kaigi*
the other day was held conference
"the conference (that was) held the other day"

As a natural consequence, the characterisation of (6) differs from that for its active alternate (7) (subject case-slot gapping vs. direct object case-slot gapping):

(7) *seNzitu* *okonatta* *kaigi*
the other day carry out/run conference
"the conference (they) ran the other day"

Our motivation in this is that cosubordinated case-slot gapping relative clauses tend to concur in the type of the gapping case slot along syntactic rather than case-role semantic lines (Baldwin 1998b). That is, it does not occur that we have a subject case-slot gapping relative clause and instrument case-slot gapping relative clause, say, cosubordinated in a single RCC.

Clause "cosubordination" represents an in-between class between clause coordination and subordination, and is defined through the notions of dependence and embedding (Foley and Van Valin 1984; Van Valin 1984). A clause is said to be dependent if can exist only in the context of a surrounding clause, due to co-occurrence with particular operators or for distributional reasons. An embedded clause, on the other hand, forms/functions as part of the host clause, without which the host clause is incomplete. Coordinated clauses are thus neither dependent nor embedded, whereas subordinated clauses are both dependent and embedded. Cosubordination clauses, on the other hand, are dependent but not embedded. [5]

Clause cosubordination in Japanese is indicated by the use of a cosubordinating conjunction of the type *nagara*, *te*, *tutu* and *si*, or through *ren'yo* type inflection (aka. *continuative* (Kuno 1973)).

(8) [[*pasukaru-ga* t_i *kōan-si,*] t_i *seisaku-si-ta*] *keisan-kikai*$_i$
Pascal-NOM DO design-REN DO make-PAST computing device
'a computing device designed and produced by Pascal'

(9) [[t_i *arubaito-o* *si-nagara*] t_i *gakkō-ni* *kayo-u*] *gakusei*$_i$
SBJ part-time work-ACC to do-WHILE SBJ school-DAT attend-PRES student
'students who work part-time while at school'

(10) [[*20-seiki-ni* *hatumei-sare*] *ryūkō-sita*] *sinamono*
20th century-DAT were invented gained popularity items
"items which were invented and gained popularity in the 20th century"

Cosubordinated RCCs are distinguished from conventional coordinated RCCs along syntactic lines, in that each relative clause is marked individually for tense with coordinated RCCs, but not cosubordinated RCCs.

(11) [*ano yūmeikyoku-o* *sakkyoku-sita,*] [*daremo-ga*
that famous piece-ACC composed everyone-NOM
yoku *sitte-iru*] *Bētōben*
well knows Beethoven
"Beethoven, who you all know well, and wrote *that* famous piece of music"

[5] The fourth possibility of an independent, embedded clause is actualised by parentheticals.

With both coordinated and cosubordinated RCCs, the complex RCC (containing multiple relative clauses) is decomposable into unit RCCs formed by individually combining each component relative clause with the NP head; this is possible without any change to the semantic relation between each relative clause and the NP head. With a cosubordinated RCC, the order of the cosubordinated relative clauses can be altered (allowing for conjugational adjustments to the matrix verb of each transposed relative clause[6]) while maintaining the same basic RCC semantics.

Returning to our example cosubordinated RCCs above, the effects of inter-clausal interpretational consistency are immediately apparent. In (8) and (9), active relative clauses are cosubordinated to form a direct object and subject case-slot gapping RCC, respectively. With (10), on the other hand, the cosubordination is between an active and passive relative clause, to produce an overall (surface) subject case-slot gapping reading. Here, it would not be possible to maintain consistency of interpretation if case-slot gapping types were based on deep grammatical relations or case-roles.

Note that a common analysis is not guaranteed across relative clauses with a coordinated RCC. This is evident in (11), where the first relative clause displays subject case slot gapping and the second direct object case slot gapping. Indeed, it is possible to coordinate an attributive and case slot gapping relative clause within a single RCC.[7]

An additional advantage of surface grammatical relations is that it is possible to model the type of accessibility hierarchy as described by Keenan and Comrie (1977) and Silverstein (1976). Accessibility hierarchies constitute a partial ordering of grammatical relations, and were originally developed in order to predict the range of relativisable grammatical relations (i.e. grammatical relations which can participate in case-slot gapping) for an arbitrary language. They are associated with the language-universal prediction that if a given grammatical relation is relativisable, then all grammatical relations higher up in the hierarchy will also be relativisable. Inoue (1976) proposed an accessibility hierarchy along these same lines but covering both grammatical relations and surface case, as a description of the ease of relativisability of different grammatical relations for Japanese, and hence an indication of the plausibility that a given RCC will be of a particular case-slot gapping type. According to Inoue, Japanese (surface) subjects are most readily relativised, followed by direct objects, indirect objects and so on, as given by the following partial ordering of case slot types:

> subject $\geq$ direct object $\geq$ indirect object $\geq$ locative *ni* $\geq$ locative *o* $\geq$ goal *ni* or *e* $\geq$ locative *de* $\geq$ instrument *de* $\geq$ standard *de* $\geq$ ablative $\geq$ genitive $\geq$ source $\geq$ comitative $\geq$ reason $\geq$ object complement

Nariyama (2000) similarly postulated an accessibility hierarchy for use in Japanese zero anaphora resolution.

Bound RCCs

Case-slot gapping can also be realised by way of binding or possession of an instantiated case slot, over the full range of complement case slot types; we term such RCC modification as **binding**. An example of a bound RCC (with binding on the subject position) is:

(12) *10-gatu-kara* *siNgakki-ga* *hazimatta* *pekiN-daigaku*
October-ABL new term-NOM began Beijing University
"Beijing University, which began its new term in October"

[6] With *-nagara* clauses (see Martin (1975:412–7)), in the case that a subject case slot is contained in the final relative clause of the original RCC, the subject case slot must additionally be transposed to the final clause of the derived RCC.

[7] Although here, the constraint that any attributive relative clauses must precede case slot gapping relative clauses applies.

In the dataset used in evaluation, the only observed bound positions were the subject and direct object case slots, and in practice, these are the only bound case slot types considered.

3.2.2 Attribution

Type	*Description*
Content	The relative clause embellishes or otherwise supplements the semantics of the head NP E.g. *au kikai* "the chance to get together"
Quantity descriptive	The relative clause contextualises the degree or measure of the head NP quantitative expression E.g. *sakkā-ga dekiru hirosa* "large enough to play soccer (in/on)"
Resultative	The head NP is the result of the action described by the relative clause E.g. *gamu-o katta oturi* "the change from buying chewing gum"
Inclusive	The relative clauses overtly lists instances implied within the denotatum of the head NP E.g. *kyoziN-o hukumu zeN-yakkyū-tīmu* "all baseball teams, including the Giants"
Exclusive	The relative clause excludes specific instances from the denotatum of the head NP E.g. *Igirisu-o nozoku Yōroppa-syokoku* "all European countries except for the U.K."
Relative temporal	The head NP relative time expression is grounded by the relative clause E.g. *kekkoN-suru yokuzitu* "the day after getting married"
Relative local	The head NP relative local expression is grounded by the relative clause E.g. *tosyokaN-ga tatta (sono-)tonari* "adjacent to where the library was built"

Table 3.3: Attributive RCC types

Attributive RCCs come in seven varieties according to the nature of modification, as detailed in Table 3.3. In all cases, the relative clause acts to ground, exemplify, expound upon, etc. the denotatum of the head NP. Note that structurally, attributive RCCs are indistinguishable from case-slot gapping RCCs.

For a given RCC, we can identify it as being of the case-slot gapping type by way of the head NP being compatible with clefting *without the need for supplementary case marking* and *conditional to the maintenance of the original RCC interpretation.* This qualification on the need for case marking is made because the head NPs of certain head restrictive RCCs can be gapped with case marker (typically *de*) insertion, a device which is either optional or infelicitous with case-slot gapping RCCs. Note that clefting may bring about need for quantification of the head NP, typically by way of a determiner such as *sono* "the/that"; this occurs for restrictive RCCs with unbounded

head NPs. The requirement for the original interpretation of the RCC to be maintained applies particularly to relative temporal and local RCCs. While both of these RCC types are syntactically compatible with clefting, the reference point for grounding is modified in the process, leading to a modified interpretation. This can be observed with the following relative temporal RCC and structurally-equivalent cleft sentence:

(13) *oriNpikku-ga* *hirakareta* *yokutosi*
the Olympics were held following year
"the year after the Olympics were held"

(14) *oriNpikku-ga* *hirakareta-no-wa* *yokutosi* *da*
the Olympics were held-NML-TOP following year is
"it was the following year that the Olympics were held"

In (13), the Olympics mark the commencement of the year, whereas in (14), the Olympics are the end point, marking a shift in reference point and change in the interpretation of the situation described by the original RCC. Relative temporal RCCs are therefore incompatible with clefting, and conform with the attributive rather than the case-slot gapping category.

In applying the cleft test to RCCs (3) and (4), a grammatical sentence is generated in the first case but not the second, supporting the respective case-slot gapping and attributive judgements for the base RCCs:

(15) *kinō* *katta-no-wa* *bōsi* *da*
yesterday bought-NML-TOP hat is
"that which (they) bought yesterday is a hat"

(16) **bōsi-o* *katta-no-wa* *(sono-)riyū* *da*
hat-ACC bought-NML-TOP (the) reason is
"that (they) bought a hat is for that reason" (INTENDED)

3.2.3 Idiomatic RCCs

Idiomatic RCCs are treated as forming a singleton class separate from the case-role gapping and attributive RCC types. The main distinguishing feature of idiomatic RCCs over conventional RCCs incorporating fixed expression relative clauses, is that the interpretation is produced at the relative clause construction level, and generally non-compositional between the relative clause and head NP. We describe idiomatic RCCs by way of an exhaustive construction template, incompatible with adjunct case slot insertion or predicate conjugation other than that indicated in the template.

Due to the idiomatic unit nature of full clause-based idioms, it seems meaningless to attempt to analyse constructions of this type by way of the proposed case-slot gapping/attributive RCC dichotomy. They are thus excluded from the classification process, and simply marked as idioms on detection.

Examples of full clause-based idioms are:

(17) *mite* *minu* *huri*
to look to not look pretence
"close one eyes on (it)"

(18) *hito-o* *miru* *me*
thing see eye
"the ability to judge a person" (lit. "eyes which see a person")

For (18), it could certainly be argued that *me* has been case-slot gapped from the instrument case slot, but this seems to gain no benefit in terms of idiom comprehension. It also does not account for the highly restricted lexical context of (18) required in order to produce an idiomatic reading:

(19) **hito-o* *mi-ta* *me*
thing-ACC see-PAST eye

(20) **hito-wa* *miru* *me*
thing-TOP see eye

Here, the given RCC-level idiom is shown to be incompatible with either tense or case marking alternation, both of which are commonly observable for non-idiomatic RCCs.

3.3 The original rule-based system

As mentioned at the outset, this research builds on the successes of a rule-based Japanese RCC analysis system developed variously in Baldwin *et al.* (1997a, 1997c, 1997b) and Baldwin (1998a, 1998b). Here, we take a moment to describe the basic design of the original rule set and types of information drawn upon by it.

The input to the rule-based system is a RCC pre-annotated with basic syntactic structure, included explicit indication of the relative clause/head NP demarcation, identification of the head noun of the RCC head NP, partitioning of the relative clause into unit clauses, indication of the nature of clause linkage (i.e. cosubordination or subordination), and chunking of case slots for each unit clause into an NP and case marker (see Section 3.5 for an example input).

The rule set has access to a simple case frame dictionary, case frame transformation facility, noun thesaurus and verb class hierarchy. Each distinct verb type (distinguished by the canonical kanji form of the verb) has a unique case frame listing, which is complemented with a list of verb classes describing the alternation behaviour of the case frame. Conjugational analysis is carried out on the matrix verb of each component clause, and a case frame transformation module then transforms the case frame according to verb morpheme content, making reference to the verb class description in the process. The final transformed case frame is then compared to the input, and analysis made of which complement case slots (i.e. case slots listed in the case frame) are instantiated and which not. The semantics of the head of the head NP are then determined through simple thesaurus lookup, and semantic compatibility with particular case slots, both complement and adjunct, ascertained. Finally, the system proceeds through the rule set, essentially executing one rule cluster per verb class the matrix verb is a member of, until a rule is triggered and an output produced.

In the instance that the matrix verb links up with multiple case frames (which occurs primarily for verbs spelled out in hiragana, and also for occurrences of the passive morpheme, where there is often ambiguity between the passive, potential and subject honorific readings), an analysis is generated for each case frame, weighted according to a range of heuristics related to the analysis type and the frequency of the stem verb for each verb having been used in the realisation in the input. These scores are then summed up for each analysis type, and the analysis of highest score returned as the overall RCC interpretation type.

Main sources of error with the original rule-based system are RCCs with inanimate NP heads, where it has great difficulty in selecting between an attributive and case-slot gapping interpretation. Additionally, given the limited semantic resources of the system, it was unable to pick up on subtle interactions between the relative clause matrix verb and head NP, particularly for the less frequent RCC types such as bound and resultative RCCs.

3.4 Supervised learning

Having described the particular feature-based domain of interest, we next turn to the disambiguation mechanism, namely **supervised learning**. In supervised learning, a classifier is trained over a set of class-annotated training examples, and then run over a set of held-out test examples. The supervision comes in the classifier having access to the class membership of each training exemplar, and being able to use this in characterising the data space.

High-profile examples of supervised learning systems are C4.5 and TiMBL (v3.0), both of which we use in this research.

3.4.1 Supervised learning systems

An outline of C4.5

C4.5 (Quinlan 1993) is an eager learning system which induces a decision tree classifier for the training data based on information gain (see below). The decision tree classifier constitutes an abstraction of the training data, through which means the test data is classified. In addition to evaluating the test accuracy of the classifier on the held-out test data, therefore, it is also possible to run the generated decision tree back over the training data to evaluate training accuracy (or conversely error rate). Commonly, decision trees learned from small-sized datasets "over-learn", in the sense that they reflect the distributional idiosyncrasies of the training data too closely and do not accurately characterise the test set. The training accuracy over small datasets thus tends to be artificially high, and the test accuracy markedly low. As the dataset increases in size, the training and test accuracies tend to converge, providing an indication of the maximum classification accuracy attainable by C4.5 using the given paramaterisation (i.e. any further increase in the number of exemplars will not bring about a significant gain in classification accuracy).

C4.5 has become the default benchmark supervised learning system in machine learning circles, and has seen prominent applications within the natural language processing community, including in automatic verbal case frame acquisition (Almuallim *et al.* 1994; Tanaka 1996), ellipsis resolution (Yamamoto and Sumita 1998) and grapheme-phoneme alignment (Ling and Zhang 1998; Baldwin and Tanaka 2000a).

An outline of TiMBL v3.0

TiMBL (Daelemans *et al.* 2000) is a lazy learning system based on the k-nearest neighbour (k-NN) memory-based learning paradigm. In memory-based learning, training examples are stored in memory in their original form, and each input is compared to each individual training example to determine the k best matches. The most commonly-occurring class among these k exemplars is then used to classify the input. Unlike C4.5, therefore, no abstraction of the training data occurs, and there is no notion of training accuracy.

In the particular configuration of TiMBL we use throughout this research, the feature vector match mechanism is based on simple feature-wise overlap. Feature vector distance is then calculated as the sum of weights associated with each matching feature, where the weight of a given feature is calculated based on the distribution of the values it takes in the training data, and correspondence between the various values and class categorisations. Feature weights can be calculated by way of a number of metrics, and in evaluation, we test a number of the metrics implemented within TiMBL, as well as experimenting with methods of integrating the different weighting methods. Unless otherwise mentioned, feature weighting is performed using "gain ratio", as defined below (Section 3.8.3).

3.4.2 Supervised machine learning vs. hand-crafted rule systems

Two limitations of any hand-crafted rule system are scalability and adaptability to new domains. Features which were not available when the rule set was originally devised may become available at some future point. In order to integrate these into the rule set, the complete reworking of the rule structure is generally necessary. Additionally, rule sets are typically customised to particular domains and development datasets, and if any evaluation on unseen data (i.e. open evaluation) is to take place, it generally only ever occurs within that same domain. While this is perfectly acceptable for many applications, there may be instances where we wish to transfer a rule set across to a new domain. The only real option in this case is to have an experienced rule designer revise the original rule set based on close observation of its behaviour over a new development dataset.

Supervised machine learning offers a solution in this regard, in that classifier generation is scalable and adaptable, given a fixed feature format and corpus of class-annotated training exemplars. That is, the overhead for supervised learning is one-off in the development of the data, and different learning methods or system parameter settings can be tested automatically on training data without additional labour cost, such that instead of having to tweak the rule set manually in order to achieve optimal performance, different feature combinations, parameter settings and system types can be trialled automatically. This overhead for data annotation is generally justifiable when we consider the effort required to design a rule set, and the flexibility we will have in applying the data once it comes on-stream. The main reservation with supervised machine learning methods is that we must ensure that we have enough data to avoid "over-training". In a way, this issue applies equally to hand-crafted rule systems, except that automatically learned classifiers model the feature–class distribution of the training data to a much finer granularity than rule sets, making them more susceptible to the effects of over-training.

One advantage of supervised machine learning systems such as C4.5 is that they operate as glass boxes, in the sense that it is possible to view their inner workings through the medium of the decision tree or declarative rule set comprising a given classifier. This allows for direct feedback as to the strengths and weaknesses of the classifier, and also the types of features that have the greatest discriminatory potential. The same can not be said for memory-based systems such as TiMBL, as no abstraction of the data takes place during classification, and the training examples are simply retained in memory and individually compared to the input to determine the closest match.

Referring specifically to the rule-based RCC interpretation system which this research aims to expand upon, supervised learning offers a means of answering several outstanding issues regarding the implementation and extendibility of the rule set.

First and foremost is the optimality of the proposed rule set over the given parameter set. That is, would it be possible to produce better performance for different rule orderings or parameter combinations? With decision tree-based learning in particular, this is a pivotal concern in the construction of the classifier, and the final decision tree generally approximates the optimal "rule" ordering for the given test data set, assuming the quantity of data is commensurate with the complexity of the classification task.

Related to this is the question of the rule set implementation over-fitting the data on which the system was evaluated: would the system perform comparably on truly unseen data? Again, this ties in closely with the training/held-out evaluation dichotomy around which supervised learning is based. In particular, by cross validating results, we are able to get a realistic idea of the general applicability of generated classifiers.

Supervised learning also provides the means to determine a performance ceiling for the given parameter set (and hence the type of surface analysis we are targeting), investigate the possibilities

of expanding the parameter set, and examine different methods of resolving ambiguity within the given domain. We consider all of these issues in evaluation below.

3.5 Parameter description

As with the original rule set, we assume that the input is partitioned off into a relative clause and head NP, that the noun head of the RCC head NP is supplied, and that for each cosubordinated relative clause, the matrix verb is identified and each verb argument is demarcated into an NP (or S) and case marker. The following is the equivalent of (8) as formatted for input to the system:

{{(*pasukaru*)-[*ga*] (*kōaN-si*)} (*seisaku-sita*)}{*keisaN*(*kikai*)}

Here, each clause is sectioned off by curly brackets, each NP and head noun of the head NP by parentheses, and each case marker by square brackets.

As is evident from the above example, we assume the basic structure of the RCC to be provided and also the boundaries of the RCC to be pre-established. In doing so, we are discounting the interaction between RCC interpretation and internal structure, which ideally should be played off against each other in RCC disambiguation. Pointers in this direction include work done on relative clause attachment resolution, that is determination of the head NP of a relative clause. Fisher and Riloff (1992), for example, apply the t test in determining the statistical likelihood of different NP sites being the head NP, whereas Cardie (in press) describes an extensive machine learning-based method operating around cognitive biases, building on earlier work based on a flatter feature structure (Cardie 1992). For the part of Japanese, Yokoo and Hayashi (1987) describe a rule-based attachment resolution method for Japanese, which involves working through the nouns proceeding the relative clause in sequence, until a match is found with the selectional constraints on a case slot in the given case frame.[8]

Parameters employed in the basic parameterisation parallel those appearing in the original rule set and include a generalised case frame description, a verb class characterisation, verb conjugational analysis, head NP semantics and various trigger patterns, as detailed in Table 3.4. These combine to form the 49-feature parameterisation of each RCC.

Case frames are applied in determining which complement case slots are defined for the current matrix verb and instantiated—hence making them *un*available for case-slot gapping—and conversely which case slots are defined for the current matrix verb and *un*instantiated—making them available for case slot gapping. The range of complement case slots coincides exactly with the set of case-slot gapping types given in Table 3.1, contributing the first eight RCC features.

Complement case slot instantiation features are set by comparing a given case frame to the actual input, and aligning case slots between the two according to case marker correspondence. The case frame dictionary employed for this purpose is the same dictionary as was used with the rule-based system, where a single generalised case frame is given for each distinct kanji–reading verb pairing. Case frames were generated from the Goi-Taikei pattern-based valency dictionary (Ikehara *et al.* 1997) by conflating the major senses for each distinct verb stem. In essence, case frames are simply a list of the complement case slots for the verb in question in their canonical ordering (case frames include no peripheral case slots). Each case slot is marked for canonical case marking and case slot type (with case slot indices taking the form described above for case-slot gapping – see Section 3.2.1). The case frame for the verb *au* "to meet", for example, as listed in the dictionary is $\texttt{NP}_s\texttt{[}ga\texttt{]}\ \texttt{NP}_w\texttt{[}to\texttt{]}$, indicating that *au* takes two complement case slots, a nominatively-marked subject (s) and comitatively-marked co-actor (w), canonically occurring in that order.

[8]The method relies upon the case frame having been determined prior to relative clause analysis.

Feature no.	*Feature description*	*Value range*
1	Subject (s) position defined and uninstantiated	0, 1
2	Direct object (d) position defined and uninstantiated	0, 1
3	Indirect object (i) position defined and uninstantiated	0, 1
4	Co-actor position defined and uninstantiated	0, 1
5	Co-patient position defined and uninstantiated	0, 1
6	Causee position defined and uninstantiated	0, 1
7	Passive agent position defined and uninstantiated	0, 1
8	Stative topic position defined and uninstantiated	0, 1
9	Accusative-marked case slot present	0, 1
10	Idiom RCC compatibility	0, 1
11	Gapped fixed argument head NP	0, s, d, i
12	Nominalised adjective head NP (e.g. *ōkisa* "size")	0, 1
13	Agentive head NP (e.g. *hito* "person")	0, 1
14	First-person pronoun head NP (e.g. *watasi* "I/me")	0, 1
15	Goal agentive head NP (e.g. *aite* "counterpart/opponent")	0, 1
16	Locative head NP (e.g. *basyo* "place")	0, 1
17	Instrumental head NP (e.g. *dōgu* "instrument")	0, 1
18	Degree-type head NP (e.g. *sokudo* "speed")	0, 1
19	Abstract head NP (e.g. *zyōtai* "condition")	0, 1
20	Non-gapping head NP (e.g. *mokuteki* "objective")	0, 1
21	Temporal-type head NP (e.g. *zikaN* "time")	0, 1
22	Relative temporal-type head NP (e.g. *yokuzitu* "next day")	0, 1
23	Time durational-type head NP (e.g. *kikaN* "duration")	0, 1
24	Exclusive RCC compatibility	0, 1
25	Inclusive RCC compatibility	0, 1
26	Copular matrix verb	0, 1
27	Conjoining matrix verb (e.g. *atai-suru* "to equate to")	0, 1
28	Existential verb (e.g. *sumu* "to live/reside")	0, 1
29	Experiential verb (e.g. *iru* "to be/exist")	0, 1
30	Tool-aided action verb (e.g. *kaku* "to write")	0, 1
31	Partitive verb (e.g. *tidimu* "to shrink")	0, 1
32	Proximal verb (e.g. *ayumiyoru* "to approach")	0, 1
33	Distal verb (e.g. *syukkō-suru* "to leave port")	0, 1
34	Travelling verb (e.g. *zyuNkai-suru* "to patrol")	0, 1
35	Inter-personal relational verb (e.g. *au* "to meet")	0, 1
36	Generic relational verb (e.g. *arasou* "to battle/fight")	0, 1
37	Quantitative verb (e.g. *kakaru* "to take (time)")	0, 1
38	Tri-agentive verb (e.g. *syōkai-suru* "to introduce")	0, 1
39	Source empathy verb (e.g. *morau* "to receive")	0, 1
40	Target empathy verb (e.g. *kureru* "to give")	0, 1
41	Empathy verb (e.g. *wakareru* "to separate/break up with")	0, 1
42	Quotative verb (e.g. *iu* "to say")	0, 1
43	Conflated ergative verb (e.g. *kaisi-suru* "to start")	0, 1
44	Body-part action verb (e.g. *kaihuku-suru* "to recover")	0, 1
45	Unaccusative verb (e.g. *kaku* "to write")	0, 1
46	-Ize verb (e.g. *kokusaika-suru* "to internationalise")	0, 1
47	Physical movement verb (e.g. *hasiru* "to run")	0, 1
48	General action verb (e.g. *ugoku* "to move")	0, 1
49	Matrix verb potential morpheme collocation	0, 1

Table 3.4: The 49-feature basic RCC parameterisation

Fixed expressions co-exist with generalised case frames in the case frame dictionary, where a fixed expression is defined as a case frame with one or more lexicalised case slots which must be overtly realised for that case frame to be triggered (cf. idiomatic RCCs where the head NP is also specified, and stringent restrictions apply to the scope for verbal inflection, case marker alternation and co-occurrence with peripheral case slots/adverbials). Examples of fixed expressions are *ki-o tukeru* (mind-ACC fix/attach) "to be careful/keep an eye out for (something)" and *yume-o miru* (dream-ACC see) "to dream". Fixed expressions are governed by the constraint that all lexicalised case slots must be instantiated in the input for that case frame to come into play, including the possibility of fixed arguments being expressed as the head NP of the RCC.

Fixed expressions come into play in case frame selection, where in the case of analytical ambiguity between a fixed and generalised expression, the fixed expression is given *a priori* preference over the generalised case frame (see below). They have a more direct role to play in RCC interpretation in cases where a fixed argument has been gapped to the head NP position, such as with the RCC *kinō mita yume* "the dream I had last night". Here, the case slot type of the transposed fixed argument is returned as the overall RCC type, by way of the "gapped fixed argument head NP" feature. That is, the value for this feature is the case slot marker on the gapped case slot. Only a small proportion of fixed arguments can, in fact, be gapped to the head NP position, with potential for gapping being determined by factors such as the semantic transparency of the fixed argument. We model this variability by way of "gapability" judgements on each fixed argument, within the case frame dictionary.

The minimalistic case frame description is complemented by **verb classes**. Verb classes are used to describe such effects as peripheral case slot compatibility (recalling that no peripheral case slots are contained within case frames), case slot interaction, and potential for valency-modifying alternation. As an example of case slot compatibility, verbs in the "proximal" verb class are compatible with local allative case slots. With inter-personal relational verbs, on the other hand, two case slots in the case frame are indexed to indicate that they both take agentive fillers which interact in some way, with empathy on the first-occurring of the two. The conflated ergative verb class provides a means of deriving the (unaccusative) intransitive usage of verbs such as *kaisi-suru* "to start"; this has uses in conjugational analysis and in selecting between the transitive and intransitive usages in the case that the relative clause does not contain any complement case slots (i.e. allowing us to say that *kaisi-suru* occurs in the intransitive usage in the RCC *kaisi-suru hiniti* "the starting date").

In processing each unit relative clause, we carry out **conjugational analysis** of the matrix verb, returning a listing of conjugational features and verb morpheme collocates. This has applications in case frame transformation, as trigger conditions for various analysis types, and in the resolution of case frame ambiguity. Case frame transformation is carried out prior to matching case slots between the input and case frame, producing a description of the surface realisation of the case frame which reflects the voice, causality, etc. of the inflected matrix verb. For *awaseru* "to have meet"—the causative alternate of *au* from above—the transformed case frame would take the form $\texttt{NP}_s\texttt{[}ga\texttt{]}\ \texttt{NP}_d\texttt{[}o\texttt{]}\ \texttt{NP}_w\texttt{[}to\texttt{]}$, where the original subject case slot has been converted to a direct object and a new subject (causer) case slot inserted. Case frame transformation is based on the method of Baldwin *et al.* (1998), and can potentially produce fan-out in the number of clause analyses. One instance where this commonly occurs is with the *(r)are* verb morpheme, which has passive, potential/spontaneous and honorific readings, all associated with distinct forms of case frame transformation (Jacobsen 1992; Shibatani 1990; Tsujimura 1996). We produce all legal case frames in this case, and leave the selection of the correct verb interpretation for later processing (see Section 3.5.1). Note that the only conjugational feature to make an appearance as an independent feature is potentiality, although the various constructions also make access to the list of

conjugational types as part of determining construction compatibility.

Head NP semantics are used to (a) morpho-semantically classify the head NP, and (b) filter out locative and temporal case slots from the relative clause. Head NPs are classified according to the binary vector of ±`nominalised_adjective`, ±`agentive`, ±`1st_person_pronoun`, ±`goal_agentive`, ±`locative`, ±`instrumental`, ±`degree`, ±`abstract`, ±`non_gapping`, ±`temporal`, ±`relative_temporal` and ±`time_durational`, with each feature indicating the potential membership in that class of the head noun(s) of the RCC head NP. There is some interaction between these features, with `goal_agentive` head NPs, for example, being subsumed by `agentive` head NPs.

As we have no means of disambiguating noun sense, this characterisation corresponds to the union of features of all senses of the head noun, as defined by way of a noun database (`non_gapping`, `1st_person_pronoun` and `goal_agentive`),[9] adjective database and adjective conjugation module (`nominalised_adjective`), cascaded regular expression (`temporal`, `relative_temporal` and `time_durational`), or specific regions of the Goi-Taikei thesaurus (`agentive`, `locative`, `instrumental`, `degree` and `abstract`). For RCCs with coordinated head NPs, we stipulate that all coordinated noun heads must display a particular feature, for it to be triggered in the final feature vector. For example, all noun heads must be compatible with the `locative` class for this feature to be set on in the final feature vector. This combined representation of semantic features is achieved by logically `AND`'ing the head NP semantic feature vectors for the different noun heads together.

Locative and temporal case slots are filtered off from the input in order to avoid them erroneously matching with complement case slots in the case frame, giving the false impression of that case slot being instantiated. As part of this process, we confirm the verb class membership of the matrix verb to determine case marker type (e.g. for the locative case slot, *de* case marking for general action verbs but *ni* case marking for existential verbs) and case slot compatibility (e.g. unmarked temporal NPs constitute quantity complements with quantitative verbs, not temporal case slots).

In addition to simple features of the type described above, there are a number of construction compatibility features which represent multi-feature **trigger patterns**. Each construction compatibility feature corresponds directly to a particular RCC type, namely exclusive RCCs, inclusive RCCs and idiomatic RCCs. The trigger pattern for exclusive RCCs operates over the excluding verb class (containing *nozoku* "to exclude", for example), and combines simple past or non-past main verb inflection and the occurrence of only an accusatively-marked case slot within the relative clause:

```
IF (excluding-type verb AND simple main verb inflection AND unique accusatively
marked argument) RETURN exclusive
```

The satisfaction of these constraints results in the exclusive RCC compatibility feature being set, as occurs for RCC (21) below.

(21) *nitiyōbi-o* *nozo-ku* *mainiti*
Sunday-ACC exclude-PRES everyday
"every day except Sundays"

For details of the individual verb classes and their impact on RCC interpretation, the reader is referred to Baldwin (1998b). Potential verb conjugation is represented as a distinct feature due to it combining with nominalised adjectives to produce quantity descriptive RCCs.

[9]The full extent of the non-gapping noun, first person pronoun and goal agentive noun lists are given in Sections B.2.1, B.2.2 and B.2.3, respectively, of Appendix B.

3.5.1 Types of analytical multiplicity

Analytical ambiguity arises when multiple clause analyses exist, and can be either intra- or inter-clausal. Multiple intra-clausal analyses arise in the case of verb homophony/homography and for fixed expressions, whereas inter-clausal ambiguity occurs as a result of complex relative clauses.

Intra-clausal analytical ambiguity

For the purposes of our system, **verb homophony** refers to the state of multiple verb entries in the case frame dictionary sharing the same kana content (and hence pronunciation) but having distinct kanji spell-outs, whereas **verb homography** occurs when multiple verb entries coincide in kanji content but not pronunciation (Baldwin 1998a). Both verb homophony and homography can be either full or partial, that is all forms of a given verb pair can be homophonous/homographic, or there can be partial overlap for particular types of verb inflection. For example, the verbs 変わる [*kawaru*] "to change/alter" and 代わる [*kawaru*] "to replace/take the place of" are fully homophonous and coincide in pronunciation under identical verbal conjugation, whereas 着る [*kiru*] "to wear" and 切る [*kiru*] "to cut" are partially homophonous (e.g., in the simple non-past they are differentiated between *kita* and *kitta*, respectively). For verb homography, 止める [*tomeru*] "to stop" and 止める [*yameru*] "to quit" are fully homographic, whereas 行う [*okonau*] "to carry out" and 行く [*iku*] "to go" are partially homographic (with overlap produce for the simple past tense, for example, in the form of 行った [*okonatta*] vs. 行った [*itta*]). Such overlap in lexical form leads to the situation of multiple verb entries being triggered, producing independent analyses for the RCC input.

Fixed expressions are verb-governed verbal expressions with one or more lexically determined case slots, of non-compositional sense. Examples of fixed Japanese expressions are *asi-o arau* (foot-ACC wash) "to quit (involvement in an illicit concern)" and *ki-o tukeru* (mind-ACC fix/attach) "to take care". One key difference setting fixed expressions apart from idiomatic RCCs is that they can occur in matrix contexts and do not place lexical restrictions on the types of head NPs they can combine with. Additionally, fixed expressions invariably combine fixed and generalised case slots in their case frames, whereas idiomatic RCCs are either fully lexicalised to the degree that no peripheral (e.g. adjunct) case slots can be inserted in the body of the relative clause.

Most verbs participating in fixed expressions can also occur in generalised usages, as is the case for *arau* and *tukeru* from the above fixed expression examples, meaning "to wash" and "to attach", respectively, in their standard usage. Analytical ambiguity comes about when selecting between the generalised usage and fixed expression instances of a given verb.

Inter-clausal analytical ambiguity

Inter-clausal ambiguity occurs when the relative clause is composed of two or more cosubordinated unit clauses. Here, it is possible to take two approaches. We can treat each cosubordinated unit relative clause as an independent RCC and generate a separate analysis for each. Alternatively, we can actively apply the constraint on interpretational consistency described in Section 3.2.1, in combining the linguistic evidence from all unit relative clauses into one, and enforcing a unique overall analysis for the RCC.

It is important to realise that we do not consider adjunct clauses in forming an interpretation for the RCC.

3.5.2 Methods of resolving analytical ambiguity

Here, we discuss basic means of resolving intra- and inter-clausal ambiguity by way of either selecting between multiple feature vector analyses, or integrating feature vectors together, to produce a unique feature vector characterisation. The reason that we insist on producing a unique feature vector for each matrix relative clause, and preferably each RCC, is that we want to keep our dataset as well distributed as possible, in the sense that we would prefer to have a constant number of feature vectors per RCC so as to ensure that each RCC has equal weighting. We would also like to ensure that the annotational quality of each feature vector in the dataset is as high as possible to avoid clouding feature relevance judgements.

Intra-clausal ambiguity resolution

We select between multiple analyses for a given relative clause in the first by preferring analyses stemming from fixed expressions, over those deriving from verb class-based trigger patterns, in turn over those generated through generalised techniques. We define each such stratum as comprising a distinct **expressional type**. In implementation terms, this equates to exhaustively searching the case frame dictionary for all hits, and taking only the fixed expressions from amongst these to generate feature vectors. In the case that no fixed expression analysis is possible, we generate the feature vector for all case frames. Out of these, we take only those feature vectors which are compatible with idiom, exclusive or inclusive RCC readings, assuming that such feature vectors exist. Failing this, all feature vectors are allowed through to the next stage of intra-clausal ambiguity resolution.

Expressional type is on the whole a powerful and reliable disambiguation mechanism, but is not infallible. The main area in which it comes unstuck is in giving fixed expressions absolute priority over other case frame analyses. Often, with non-compositional fixed expressions and fixed expressions governed by functional verbs such as *kakeru* or *tukeru*, the lexicalised arguments in a fixed expression are incompatible with generalised usages. With more transparent fixed expressions such as *asi-o arau* (foot-ACC wash) "to quit (involvement with an illicit concern)", on the other hand, *asi* can occur as the direct object of *arau* in non-specialised usage, meaning simply "to wash (one's) feet". Under our strict hierarchy of expressional type, the generalised sense would not be acknowledged, and the feature vector for the fixed sense returned as the unique output. Without expressional type preferences, however, non-compositional fixed expressions incompatible with non-lexicalised senses of that matrix verb, would be analysed as conforming with the generalised case frame, producing analytic ambiguity where it does not exist. In this sense, expressional type seems a justifiable means of pruning the feature vector space for a given relative clause.

In the case that ambiguity is not resolved through such *a priori* expressional type preferences, we apply a succession of heuristics of decreasing reliability, to produce a single feature vector interpretation. These heuristics are, in order of application: minimum verb morpheme content, best case frame match and representational preference. Each of these is described below.

The heuristic of **minimum verb morpheme content** involves determining the morphemic content of the matrix verb of the current relative clause, based on the conjugational class and verb stem of each dictionary entry it matches with, and selecting those feature vectors associated with the least morphemes and verb auxiliaries. Verb morpheme content is output as a list of morphemes and auxiliaries collocating with the verb stem in the matrix verb complex, listed in their order of lexical occurrence. We simply count the number of such morphemes, and select the feature vectors with the minimum such count.

Essentially this methodology offers a means of picking up on more specialised or more highly stem-lexicalised verb entries in the case frame dictionary. While it is slightly idiosyncratic, it is

highly effective at pruning errant analyses. Its disambiguating potential is perhaps best illustrated by way of a series of examples. Consider the matrix verb 見える [*mieru*], analysable as either *mi-e-ru* "to see-POT-PRES" or *mie-ru* "to be visible-PRES" within the constraints of the case frame dictionary. Of these, the second analysis is correct due to it containing a single "morpheme" as compared to two morphemes in the first case; this result is successfully predicted by the minimum verb morpheme content heuristic. While it may appear that the second analysis can be decomposed into the first and that the two analyses are essentially equivalent, the first analysis is in fact blocked by the existence of the second, and the second analysis associated with a case frame unrealisable under case frame transformation from the first. Our preference for minimum verb morpheme content thus provides a valuable result here. As a second example, consider the input 割り込む [*warikomu*]. Here, the analytic ambiguity is between *wari-ko-mu* "to split/smash-in-PRES" and *wariko-mu* "to push in-PRES", of which the analysis of lesser conjugational complexity (i.e. the second analysis) is once again correct, as predicted correctly by our heuristic.

Despite its high accuracy, the coverage of the minimum verb morpheme content heuristic is limited, and it does little to resolve full and partial verb homophony/homography. This leads us on to the second heuristic of **best case frame match**. Here, we look beyond the matrix verb to analyse the degree of correspondence between the case frame listed in each dictionary entry, and the actual verb argument content of the input; this is carried out after having filtered off temporal and locative case slots from the input.[10] In following with the low processing cost dogma associated with feature-based disambiguation, we simply calculate the number of verb arguments in the input which align with case slots in the current case frame (based on case marker overlap), and divide this by the sum of the total number of case slots contained in the case frame and the total number of (non-local or temporal) verb arguments in the input. We additionally add one to the numerator to give preference to case frames of lower valency (i.e. fewer case slots) in the case that there is no overlap with the input. This can be formalised as:

$$CFM(IN, CFRAME) \quad = \quad \frac{1 + |IN \hat{\cap} CFRAME|}{|IN| + |CFRAME|} \tag{3.1}$$

where *IN* is the set of verb arguments in the input, *CFRAME* the set of case slots described in the current case frame, and $\hat{\cap}$ the case slot overlap operator. Note that the ordering of the case slots plays no part in calculations, in an attempt to capture the relative freedom of case slot order in Japanese.

To give an indication of how this works, consider the relative clause input *Nakata-seNsyu-to kawatta* "Nakata-player-COM replaced" meaning "(who) replaced Nakata". Assuming that *kawatta* is lexicalised in hiragana, ambiguity is produced between our example full verb homophone examples from above, 代わった [*kawatta*] "replaced" and 変わった [*kawatta*] "changed/altered", of which the first is the correct analysis (as evident from the gloss). The case frame in the first instance is $\texttt{NP}_s\texttt{[}ga\texttt{]}$ $\texttt{NP}_w\texttt{[}ni/to\texttt{]}$, and that in the second instance is $\texttt{NP}_s\texttt{[}ga\texttt{]}$. Given that we have a single comitatively-marked verb argument in the input, the match ratio for first analysis is $\frac{1+1}{2+1} = \frac{2}{3}$, as compared to $\frac{0+1}{1+1} = \frac{1}{2}$ in the second instance. The method thus correctly selects the first analysis ("replaced") over the second.

While best case frame match has greater coverage than minimum verb morpheme content, there will inevitably be analyses which are equivalently plausible under both these heuristics. We thus employ a third, less reliable heuristic of 100% coverage to resolve any remaining ambiguities. This

[10] As mentioned above, the reason we must filter off temporal and locative case slots from the input before carrying out the case frame match, is to avoid having them incorrectly match up with case slots in the case frame. This is particularly pertinent for the best case frame match heuristic in order to calculate the match ratio with maximum accuracy.

is based on the **representational preference** for the current verb to take different lexical forms. The representational preference (RP) of lexical form a of verb entry f (i.e. a_f) is defined as the likelihood of f being realised as a, with a median score of 1.

$$RP(a_f) = \frac{1 + freq(a_f)}{1 + \sum_{i \neq a} freq(i_f)} \tag{3.2}$$

This is normalised over the representational preference for all source entries a_i, to produce $VS(a_f)$.

$$VS(a_f) = \frac{RP(a_f)}{\sum_i RP(a_i)} \tag{3.3}$$

All frequencies are calculated based on the EDR corpus (EDR 1995).

In the case of a tie in representational preference, we select one of the tied analyses randomly.

Below, we consider a number of methods for applying the various VS scores for entries producing a common analysis.

Inter-clausal ambiguity resolution

The above methods are applicable to **intra-clausal disambiguation**, which is performed prior to **inter-clausal cross-indexing**. That is, we first generate a unique feature vector for each unit clause before combining the clause vectors for each unit relative clause into a single integrated exemplar for the overall RCC. For the case of the cosubordinated RCC `[[`$\texttt{S}_1$ $\texttt{S}_2$`] NP]`, therefore, we individually disambiguate $\texttt{S}_1$ and $\texttt{S}_2$, and apply the final clausal interpretations in forming a single inter-clausal representation for the overall RCC. We consider a number of methods for this purpose, namely logically `AND`'ing or `OR`'ing the unit clause feature vectors together, or alternatively ignoring all other than the final relative clause. An additional linguistic constraint that we apply in inter-clausal ambiguity resolution where one of the cosubordinated clauses occurs with the *-nagara* verb morpheme, is that if there are uninstantiated non-subject complement case slots in the *-nagara* clause,[11] then the RCC cannot be subject case-slot gapping. In this case, we thus set the subject case slot feature in the final feature vector to zero, irrespective of its value in other cosubordinated relative clauses.

3.5.3 Verb semantic attributes

A potential source of disambiguation not tapped into in the original rule-based implementation, is the verb semantic attribute ("VSA") annotation from the Goi-Taikei pattern-based valency dictionary (Nakaiwa *et al.* 1994; Nakaiwa and Ikehara 1997; Nakaiwa 2000). Within the ALT-J/E machine translation system for which they were originally developed, VSAs have been applied successfully in zero pronoun resolution (Nakaiwa and Ikehara 1994; Nakaiwa and Ikehara 1995; Nakaiwa *et al.* 1995). Given the obvious parallels between this task and RCC categorisation (for case-slot gapping RCCs at least), we were keen to test their applicability in the overall formulation.

For the purposes of this chapter, VSAs can be considered to be orthogonal to our verb classes, as they target the overall type of activity or state described by the verb, whereas our verb classes relate to case slot interaction and compatibility with highly specialised trigger patterns. VSAs were retained in the case frame dictionary simply by taking the union of all VSAs for those verb entries used to form the case frame dictionary entry, producing an average of around 1.35 VSAs per non-fixed sense dictionary entry, out of a total of 36 attribute types.

[11] The subject case slot of the *-nagara* must necessarily be uninstantiated, as *-nagara* clauses cannot occur with overt subject case slots (Perlmutter and Postal 1984; Baldwin 1998a; Baldwin 1998b).

Class	***Absolute frequency***	***Relative frequency***
Subject case-slot gapping	3245	64.00%
Content RCC	685	13.51%
Direct object case-slot gapping	377	7.44%
Idiom RCC	119	2.35%
Exclusive RCC	114	2.25%
Locative case-slot gapping	110	2.17%
Temporal case-slot gapping	106	2.09%
Co-actor case-slot gapping	61	1.20%
Stative topic case-slot gapping	45	0.98%
Time durational case-slot gapping	47	0.93%
Bound subject case-slot gapping	44	0.87%
Inclusive RCC	28	0.55%
Relative temporal RCC	20	0.39%
Indirect object case-slot gapping	20	0.39%
Degree case-slot gapping	18	0.36%
Quantity descriptive RCC	8	0.16%
Local ablative case-slot gapping	7	0.14%
Resultative RCC	6	0.12%
Local perlative case-slot gapping	4	0.08%
Bound direct object case-slot gapping	3	0.06%
Passive agent case-slot gapping	2	0.04%
Instrument case-slot gapping	1	0.02%

Table 3.5: The distribution of RCC types in the dataset

3.6 Parameter composition and evaluation

In evaluation, we variously compare: (a) different intra-clausal interpretation selection techniques; (b) clause-integrated vs. unit clause feature vectors for cosubordinated relative clauses; and (c) the applicability of VSAs to the resultant system configurations. We further go on to investigate the efficacy of different parameter partitions on disambiguation, and generate a learning curve.

Evaluation was carried out by way of stratified 10-fold cross validation throughout, for both C4.5 and TiMBL. N-fold cross validation involves partitioning the dataset into N equal-sized portions, and using each portion as the test data and the remaining $N-1$ portions as the training data, over N iterations. In this way, each portion of the data is evaluated as test data once, and is combined with other data to make up the training data on $N-1$ iterations. The final classification accuracy is measured by way of the average accuracy over the N iterations. Stratified cross validation represents a slight extension to this method, whereby the partitioning of the data is carried out so as to maintain a uniform class distribution across all portions. Our interest in semi-stratification and choice of 10 folds for cross validation, stem from the findings of Kohavi (1995) that 10-fold stratified cross validation is associated with both low bias and statistical variance, and is hence more representative of the data than other cross validation and bootstrapping evaluation methods.

The C4.5 pruning constraint was set to 10%, and TiMBL run under the default settings.

The data used in evaluation is a set of 5143 RCC instances from the EDR corpus (EDR 1995); these 5143 RCCs comprise a total of 5408 unit relative clauses (through cosubordination). We thus have 5143 clause-integrated feature vectors and 5408 unit clause feature vectors.

An absolute benchmark on accuracy is obtained through allotting a subject case-slot gapping analysis to every RCC input, based on the frequencies of the RCC types in the 5143 clause-integrated feature vectors presented in Table 3.5. We attain a baseline accuracy of 64.0% in this way.

When run over the 5143 RCC instances targeted in evaluation, the original RCC analysis system performed at a creditable accuracy of around 87.6% (cf. the 88.6% accuracy quoted in Baldwin (1998b) over a slightly smaller data set).

3.6.1 Intra-clausal disambiguation

Intra-clausal disambiguation refers to the selection of the most plausible interpretation for each unit clause (*UC*), after partitioning off each cosubordinated relative clause and forming an independent RCC from it. Here, we compare: (a) a random selection baseline method (*UC+best_rand*); (b) a method where all feature vectors for the current clause are logically `AND`'ed together (*UC+and*); (c) a method where all feature vectors for the current clause are logically OR'ed together (*UC+or*); and (d) the VS maximisation method from above (*UC+vs*). The results for the various methods within a basic unit clause feature vector framework (with no interaction between clause analyses) are presented to the left of Figures 3.1 (C4.5) and 3.2 (TiMBL), based on the full data set. Note that 28.8% of clauses occurring in the data are associated with analytical ambiguity.

UC+vs outperforms the *UC+best_rand* baseline to a level of statistical significance,[12] in both training and testing for C4.5 and testing with TiMBL (recall that no training accuracy is produced by TiMBL). *UC+or* lags behind *UC+vs* in testing in particular, but holds a statistically significant edge over *UC+best_rand* in training. *UC+and* is inferior to *UC+best_rand* in both training and testing, due to the `AND`'ing together of multiple analyses fatally reducing the scope of positive evidence.

[12] All statistical significance judgements are based on the one-tailed paired t test ($\alpha \leq 0.05$).

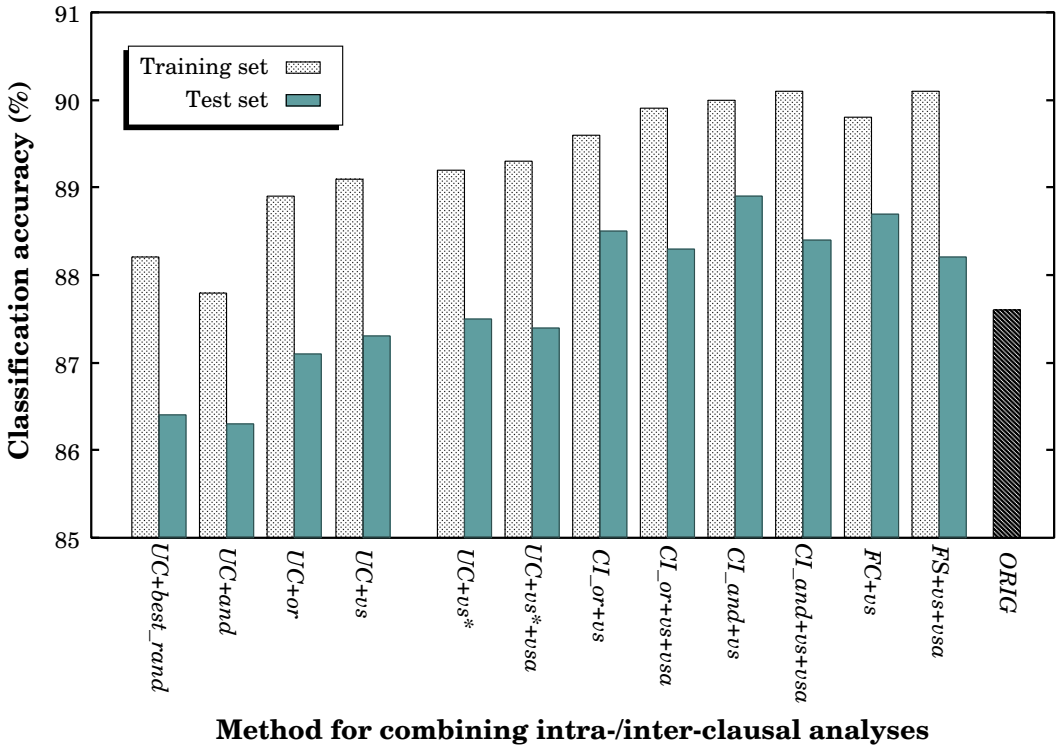

Figure 3.1: C4.5-based evaluation of the basic parameter set under different clause processing configurations

In an attempt to establish a ceiling on the performance of the intra-clausal disambiguation method, we ran C4.5 over the 3784 unit clause feature vectors not involving any intra-clausal ambiguity using the VS method. This returned a training accuracy of 87.8% and test accuracy of 86.2% for C4.5 and test accuracy of 84.9% for TiMBL, all of where are significantly down on the respective accuracies for *UC+vs* over the full data set (at 89.1%, 87.3% and 87.5%, respectively). Clearly, it should not be possible for *UC+vs* to outperform an omniscient method which is able to disambiguate correctly in all instances, suggesting that unit clause instances crucial to training were included in the ambiguous data.

Based on the above results, we choose *UC+vs* as our intra-clausal disambiguation technique for all subsequent evaluation.

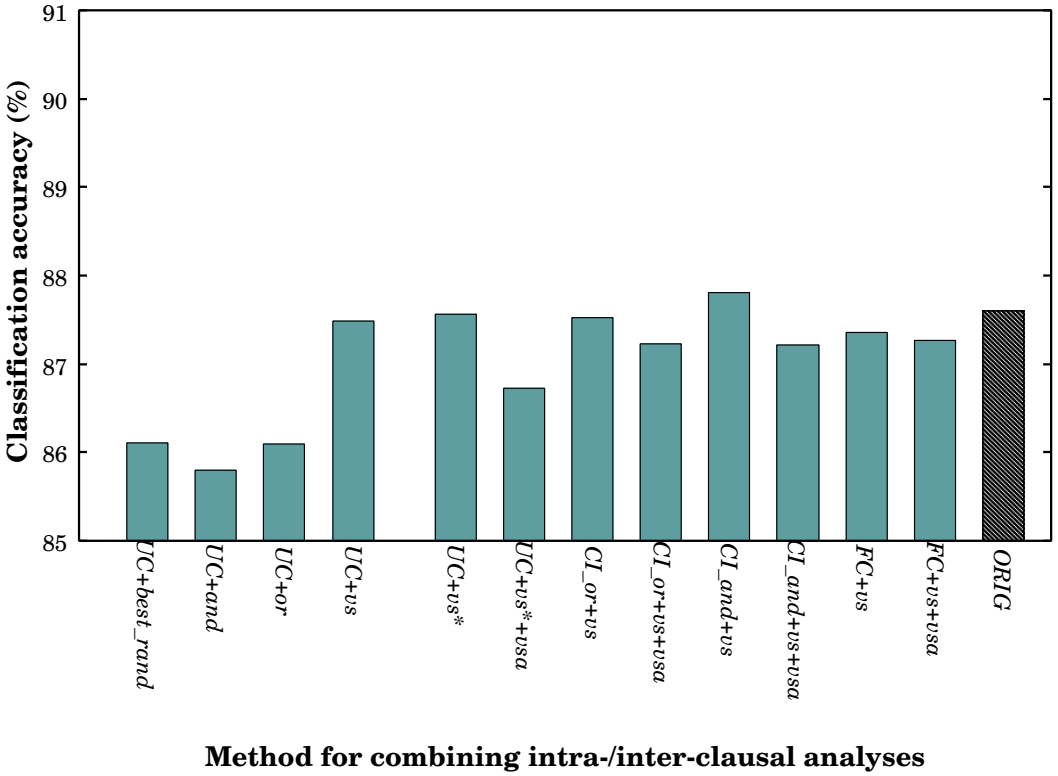

Figure 3.2: TiMBL-based evaluation of the basic parameter set under different clause processing configurations

3.6.2 Inter-clausal cross-indexing

Next, we look at the different inter-clausal analysis methods. The two core paradigms we consider are unit clause (UC) and clause-integrated (CI) analysis. Note that relative clause cosubordination was observed for 4.7% of the RCCs targeted in evaluation.

For unit clause analysis, we extend the basic *UC+vs* methodology from above, by logically `AND`'ing together the case slot compatibility flags between unit clause feature vectors to maintain a consistently applicable case-slot gapping analysis (*UC+vs**); note that VS is applied as is for intra-clausal disambiguation.

For clause-integrated analysis, we again apply VS in intra-clausal analysis, then either logically

`OR` or `AND` the component unit clause feature vectors together, producing methods *CI_or+vs* and *CI_and+vs*, respectively.

To benchmark these integrated approaches, we test the accuracy of simple final clause analysis for cosubordinated relative clauses (*FC*). Here, we simply ignore all other than the final clause in both the training and testing phases, and perform intra-clause disambiguation with VS as for the other methods.

We go on to produce an additional variant of methods *UC+vs**, *CI_or+vs* and, *CI_and+vs* by adding in to the feature vector the VSA content for each selected unit clause interpretation, leading to *UC+vs*+vsa*, *CI_or+vs+vsa* and *CI_and+vs+vsa*, respectively.

The training and test accuracies for the described methods are given in Figures 3.1 and 3.2, juxtaposed against the 87.6% accuracy attained for the original system (labelled as *ORIG*). As for intra-clausal disambiguation, the presented evaluation is over the entire data set, despite relative clause cosubordination occurring for only 4.7% of inputs. It is not expected, therefore, that the addition of inter-clausal disambiguation will affect performance to a large degree.

In both training and testing, the two clause-integrated analysis methods outstrip the unit clause analysis methods to varying degrees, with the superior method being *CI_and+vs* at a test accuracy of 88.9% for C4.5 and 87.8% for TiMBL. In training with C4.5, all visible disparities in accuracy between VSA and non-VSA methods other than that between *CI_and+vs+vsa* and *FC+vsa*, are statistically significant. For test accuracies, on the other hand, with C4.5 a statistically significant performance improvement was seen only for *CI_and+vs±vsa* over *UC+vs*±vsa* and *FC* over *UC+vs**; with TiMBL, no significant difference was found between any of the methods. As such, the absolute superiority of *CI_and+vs* is somewhat doubtful for the given data size, but it can be expected to produce genuine performance gains given greater data. These figures are particularly promising given the relative scarceness of cosubordinated RCCs in the input data.

The slight disparity in C4.5 training accuracies of system configurations with VSAs over those without,[13] and drop in C4.5 test accuracy over non-VSA data sets, would tend to suggest that the given data set is too small to bring out the full disambiguating power of VSAs, and that they have potential to tweak the system performance marginally, assuming sufficient data.

It is difficult to gauge the significance of the results given that cosubordinating RCCs account for only 4.7% of the total data. One way in which we can establish a cap on the optimal expected accuracy is to test C4.5 on only simple RCCs, and assume that the system should not be able to improve on this performance for cosubordinated RCCs. This gives a training accuracy of 90.6% and test accuracy of 89.3% for C4.5, and test accuracy of 88.8% for TiMBL, each of which is above the corresponding accuracy for the best-performing *CI_and+vs* configuration. Interestingly, however, the disparity in C4.5 test accuracies is not statistically significant, whereas it is with the TiMBL test accuracies, such that *CI_and+vs* would appear to be approaching the best we can hope to achieve in inter-clause cross-indexing.

We see minor performance improvements for the best-performing C4.5 and TiMBL system configurations over the original system formulation. Looking to the actual rule set inferred by C4.5 from *CI_and+vs* in closed evaluation (with the C4.5 module `c4.5rules`), for example, we see striking similarities with the original system rule set. There are a number of occurrences of low-applicability, high-specificity rules in the C4.5 rule set which were not contained in the original rule set, generally representing over-training on the input data. At the same time, no generalised rule not picked up on in the original system was induced. One interesting effect was that C4.5 was able to apply negative evidence more effectively than the original system formulation in enhancing

[13] Note that, according to the one-tailed paired t test, the training accuracies for *CI_or+vsa* and *FC+vsa* are superior to those for their respective non-VSA counterparts, with confidence $\alpha = 0.001$, and for both C4.5 and TiMBL, there is no significant difference between the test accuracies for the four basic configurations with and without VSAs.

the precision of various rule instances, and at the same time maintain recall by positing multiple rules founded around the same positive evidence. A fragment of the rule set version of the decision tree induced from the entire dataset, is presented in Figure 3.3 (with the training accuracy of each rule indicated in square brackets).

```
                        ⋮
Rule 163:
        Exclusive RCC compatible = 1
        ->  class exclusive  [95.8%]

Rule 153:
        Passive agent position defined and uninstantiated = 0
        Gapped fixed argument head NP = 0
        Agentive head NP = 0
        Time durational-type head NP = 1
        ->  class temporal durational case-slot gapping  [90.1%]

Rule 142:
        Subject (s) position defined and uninstantiated = 1
        Co-actor position defined and uninstantiated = 1
        Non-gapping head NP = 0
        Time durational-type head NP = 0
        Inter-personal relational verb = 1
        Empathy verb = 1
        ->  class co-actor case-slot gapping  [91.9%]

Rule 38:
        Subject (s) position defined and uninstantiated = 0
        Direct object (d) position defined and uninstantiated = 0
        Non-gapping NP = 0
        Temporal-type head NP = 0
        Empathy verb = 1
        Physical movement verb = 0
        ->  class co-actor case-slot gapping  [75.8%]
                        ⋮
```

Figure 3.3: A fragment of the induced C4.5 rule set

One predictable correlation to come from the inferred data, is the high degree of correspondence between agentive head nouns and subject and co-actor case-slot gapping analyses, and locative head nouns and the various local case-role gapping analysis types. C4.5 was also able to hone in on the true sense of the head noun through complex combinations of head noun semantic parameters.

RCC types the system seemed to have most trouble classifying were bound case-slot gapping and content RCCs, two clause types which also proved problematic for the original system. In the case of bound case-slot gapping RCCs, for example, seven separate rules were posited to cover only 55 RCC instances, at an average of 7.9 attributes per rule. Even here, C4.5 is only able to achieve a precision and recall of 74.6%, although this does compare favourably against the over-generalised approach adopted in the original research, performing at a precision of 45.9% and recall of 61.8% (under the standard definitions for precision and recall). The heavy-handed methodology adopted

in the original research to recognise content RCCs, was essentially to have a lexicon of head nouns which commonly produce this analysis type (i.e. non-gapping head NPs). On detection of such "non-gapping" head nouns and non-triggering of any other attributive[14] RCC type, we assume the RCC to be of the content type. This coarse technique was carried over to the parameterisation of the data set, and C4.5 appeared unable to fashion any more reliable generalised technique to produce this analysis type.

3.6.3 Additional evaluation

We further partitioned off the parameter space and ran C4.5 and TiMBL over the different combinations thereof, using *CI_and+vs*. The particular parameter partitions we target are: case slot compatibility flags (C — 11 attributes), head noun semantics (N — 14 attributes) and verb classes (V — 27 attributes). We additionally apply VSAs (VSA — 36 attributes) in isolation to gauge their potential in RCC analysis.

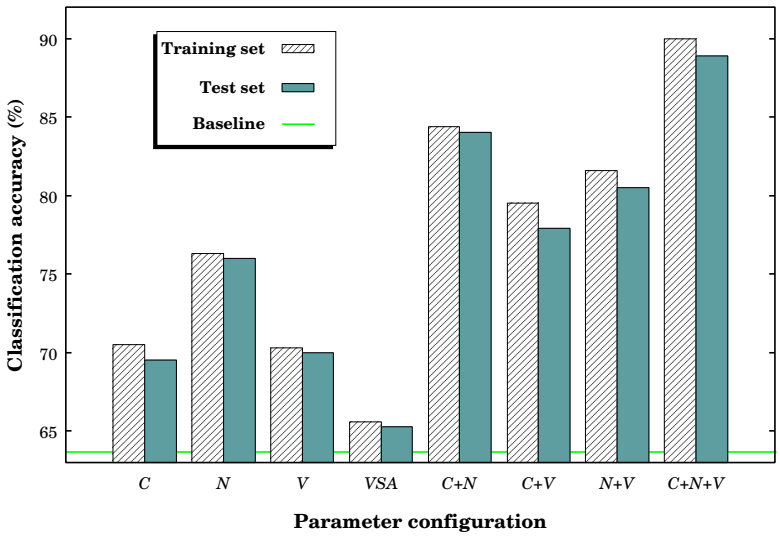

Figure 3.4: C4.5-based evaluation of different parameter combinations

The system results over the individual parameter partitions, and the various combinations of case slot compatibility, head noun semantics and verb classes (e.g. $N+V$ = head noun semantics and verb classes), are presented in Figures 3.4 and 3.5.[15] The line at $y = 63.0$ at the base of both graphs represents the absolute baseline, that is the classification accuracy of classifying every RCC as being of the subject case-slot gapping type.

The value of head noun semantics is borne out by the high test accuracy for N of 76.0% and 76.1% for C4.5 and TiMBL, respectively. We can additionally see that case slot instantiation and verb class member attributes provide approximately equivalent discriminatory power, both well above the absolute baseline of 63.0%. This is despite case slot instantiation flags being less than half the number of verb classes, largely due to the direct correlation between case slot instantiation judgements and case-role analyses, which account for around 80% of all RCCs. The accuracy for VSAs is well down, just above the absolute baseline accuracy in testing at 65.3% and 65.2% for

[14]Termed "head restrictive" in the original research.

[15]Note that $C+N+V$ corresponds to the full parameter space, and is identical to *CI_and+vs* in Figures 3.1 and 3.2.

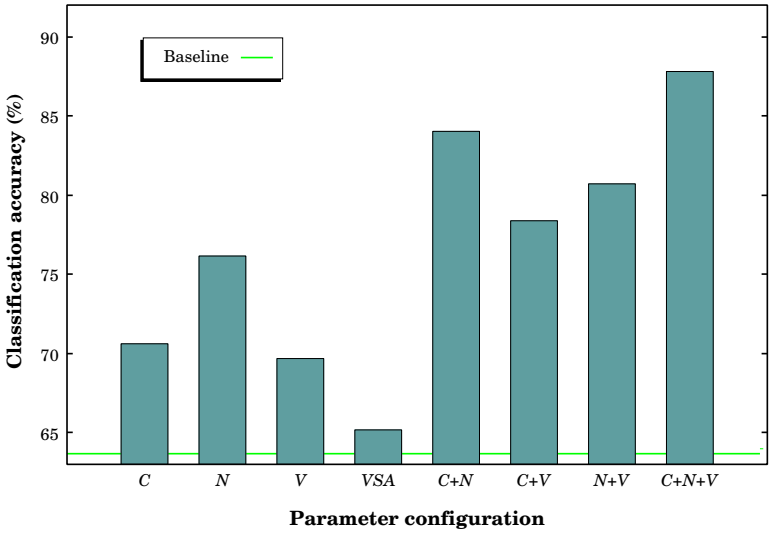

Figure 3.5: TiMBL-based evaluation of different parameter combinations

C4.5 and TiMBL, respectively. Verb classes thus provide a clear advantage over VSAs in RCC analysis.

The affinity between case slot instantiation judgements and the semantics of the head noun is evidenced in the strong performance of *C+N*, although even here, verb classes gain us an additional 5% of performance (both C4.5 and TiMBL). Essentially what is occurring here is that associational preferences between particular head noun semantics and certain case-roles/analysis types are incrementally enhanced as we add in the extra dimensions of case slot instantiation and verb classes. The crude set of selectional preferences produced for each analysis type by head noun semantics is enhanced by case slot instantiation values, due to the filtering off of case-role gapping analyses where the associated case slot is instantiated. Subsequently adding in verb classes produces better localisation of the selectional preferences to the different verb types, and at the same time allows for more regulated interaction between particular case slot positions. The orthogonality of the three dimensions is demonstrated by the incremental performance improvement as we add in extra parameter partitions.

Another item worthy of interest is the learning curve for the C4.5 and TiMBL systems. Here, we target the *CI_and+vs* system and run it over data sets of 100, 250, 500, 1000, 2000, 3000, 4000, 5000 and finally 5143 RCC instances, with each dataset comprising a proper subset of those larger than it. The training and test accuracies over these data sets are given in Figure 3.6.

Other than the characteristic knoll at the lower reaches of the training curve, caused by overtraining, the training accuracy appears to be levelling out to a figure somewhere between 90 and 91% for C4.5, and slightly less for TiMBL. The lack of significant diversion beyond about 3000 entries would tend to suggest that our training accuracy is not going to increase much given extra data, and that the 91% accuracy is a ceiling on test performance for the given parameterisation.

As a final point of evaluation, we randomly extracted a set of 100 fresh RCCs from the EDR corpus to test the robustness of the original system as compared to C4.5 and TiMBL. The particular RCCs extracted proved difficult for all three systems, with the original system producing an accuracy of 67.0%, as compared to a proportionally deflated 69.0% for the *CI_and+vs* configuration of C4.5 and 72.3% for TiMBL under the same system configuration. The fact that all systems should have performed equally badly suggests that they are able to handle unseen data equivalently well.

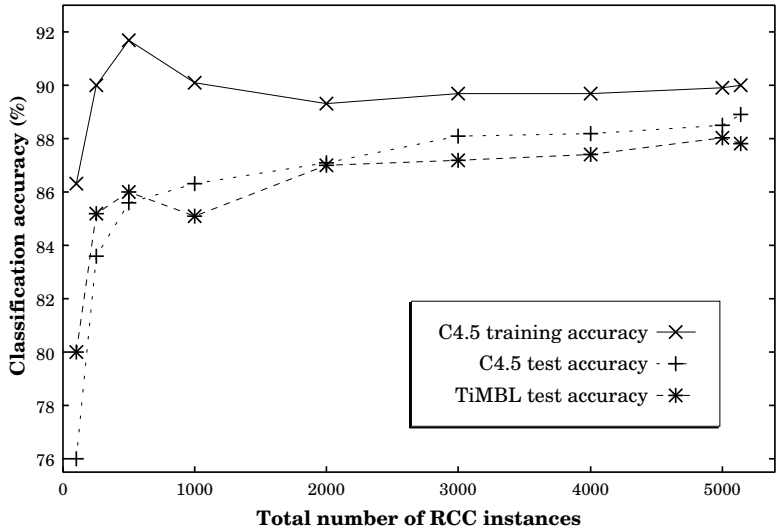

Figure 3.6: Learning curve for C4.5 and TiMBL

From this, we can make the statement that the original system is as robust to new data as could be expected given the composition of the original data, and by extension, the original system has been trained near-optimally on the given data.

3.6.4 Summary and discussion of results to this point

To summarise the key features of evaluation of the basic dataset, VS was found to be the most successful for the intra-clausal disambiguation methods and logical `AND`'ing the most successful of the inter-clausal cross-indexing methods. In combination, these produce a slightly appreciated test accuracy over that for the original rule set, for both C4.5 and TiMBL. A ceiling on test accuracy of between 90 and 91% was shown to exist, which the best system configurations were approaching. As such, we have verified that the original rule set formulation was near optimal for the given sample of RCCs and feature space, but also that the same results can be attained with supervised learning methods, requiring no external input other than the actual data.

To the authors' knowledge, the only published research relating to Japanese RCC resolution which includes broad-coverage empirical evaluation is that of Baldwin *et al.* (1997b), who claim an 89.5% *non-deterministic* accuracy. The accuracy is non-deterministic in the sense that the proposed system outputs multiple analyses, and is adjudged to be correct if one of those is the correct analysis. This makes direct comparison difficult, other than to say that we have been able to achieve a similar accuracy *deterministically*.

Alternatively, it is possible to construe RCC resolution as a special case of zero pronoun resolution and observe that accuracies for Japanese anaphora resolution systems tend to fall around the 80% mark (Nakaiwa and Ikehara 1994; Murata and Nagao 1998), suggesting that our results just below the 90% mark are good.

A more direct point of reference is found in the work of Li *et al.* (1998) on Korean RCCs, which display the same structural ambiguities as Japanese RCCs. Li *et al.* attain an accuracy of 90.4% through statistical analysis of the distribution of verb-case filler collocates, except that they classify relative clauses according to only 5 categories and consider only case-slot gapping RCCs. With our method, restricting analysis to only gapping RCCs (still retaining a total of nineteen RCC types)

produces an accuracy of 94.1% for the *CI_and+vs* system with C4.5.

3.7 Complementing the original feature space

Having established that we have squeezed just about all we can out of the given parameterisation, we next look to maximally expand the feature space to enhance classification accuracy. This is performed over the same basic set of RCCs as was used above.

3.7.1 Kitchen sink learning and a description of underlying concepts

The approach we adopt in complementing the original feature space is to add in whatever information we have at our fingertips, and rely on "feature selection" and "feature construction" methods to selectively estimate which features are strongly and weakly relevant, and which are irrelevant or redundant, in a process we nickname "kitchen sink learning". In this, we throw all the features we can at the problem, before sifting through the features to determine what is useful for the task at hand.

It is appropriate to define a number of machine learning concepts integral to kitchen sink learning at this point.

First, the **relevance** of a feature in the context of a given task, is a judgement on whether that feature enhances class discrimination. Formally, a feature is relevant if there is a direct correlation between the values of that feature and class characterisation. Conversely, a feature is **irrelevant** if there is no correspondence between it and the class membership of exemplars. Relevance is further broken down into the two categories of strong and weak relevance. A **strongly relevant** feature is such that there is some minimal pair of examples in the sample space, which differ only in their value of that feature, but have different class labels (John *et al.* 1994; Blum and Langley 1997). In other words, strongly relevant features provide the sole means to differentiate certain exemplars, and without them, those exemplars would be indistinguishable. A **weakly relevant** feature, on the other hand, contributes to prediction accuracy in combination with other weakly relevant features. It is possible for relevant features to be **redundant** in the case that alternative features offer identical discriminatory potential.

In an ideal world, we would like to be able to ignore all weakly relevant, irrelevant and redundant features, and operate based only on strongly relevant features, but there is of course no guarantee that strongly relevant features will be able to discriminate between all exemplars. The focus thus shifts to retaining all strongly relevant features, filtering off all irrelevant and redundant features, and retaining the minimum component of weakly relevant features which facilitates the correct classification of all exemplars. This process is termed feature selection.

Feature selection is defined as the process of selectively working through the overall feature space and either discarding or retaining each individual feature based on evaluation of its impact on classification (Langley 1994; Blum and Langley 1997). Feature selection offers a range of benefits, including increased comprehensibility of classifiers (e.g. smaller decision trees with less features), faster running times, smaller data storage overheads, and steeper rates of learning (i.e. the learning curve peaks for smaller data sizes).

In **feature construction**, rather than simply pruning off individual features, we dynamically combine features together into new features, in the hopes of generating strongly relevant features from weakly relevant features (Matheus and Rendell 1989; Pagallo and Haussler 1990). Feature construction has the potential to make base features redundant in the process of constructing new features, such that it is generally interleaved with feature selection in incrementally honing in on a set of strongly relevant features of high overall coverage.

Both feature selection and feature construction can be either **greedy** or **exhaustive**. With greedy feature selection, we search only a proper subset of the full range of feature combinations, usually taking each feature in turn, and in the context of the feature space at that point, make a decision once and for all as to whether to retain or discard that feature. With exhaustive feature selection, on the other hand, we search across all possible feature combinations, and use some form of evaluation to determine the optimal such combination. Clearly, greedy feature selection has advantages in terms of speed, whereas exhaustive methods have a better chance of determining the optimal feature combination. Having said this, it is of course quite possible for greedy approaches to achieve the same levels of accuracy as for exhaustive approaches, assuming that the search is well directed (i.e. the optimal feature combination is contained within the range of our search).

Feature selection methods can be grouped into three classes: "embedded", "filter" and "wrapper". **Embedded** feature selection methods incorporate feature selection as part of the induction algorithm. C4.5 provides a prime example of this, in its decision tree pruning mechanism. Unlike memory-based learning, the final decision tree induced by C4.5 often does not span the entire feature space, preferring to concentrate on features with high classification reliability and coverage. In this sense, we already make use of feature selection in evaluation above. With **filter** feature selection methods, feature selection is carried out independent to the induction algorithm as a preprocessing stage. Prominent examples of this strategy include Relief (Kira and Rendell 1992) and FOCUS (Almuallim and Dietterich 1991; Kononenko *et al.* 1996). Filter feature selection methods tend to rely on information-theoretic metrics or stochastic methods (e.g., in the case of Koller and Sahami (1996), Markov blankets) to evaluate feature relevance. Finally, **wrapper** feature selection methods employ the induction algorithm directly in feature selection (Caruana and Freitag (1994), John *et al.* (1994), Langley and Sage (1994), *inter alia*). Thus, for a C4.5-based classification task, we would first use C4.5 to select the features to use, and then train C4.5 over those features, allowing scope for C4.5 to ignore certain features in the decision tree induction process.

Wrapper and filter feature selection methods are certainly compatible with embedded feature selection, and indeed much work on wrapper methods is based on the feature pruning-capable C4.5. Further, it has been suggested that the potential gains availed by feature selection are greater for memory-based learning methods (e.g. nearest neighbour algorithms) than for decision tree-based systems such as C4.5 (Blum and Langley 1997; Langley and Sage 1997). The main reason for this is that memory-based learning methods are vulnerable to the effects of irrelevant and redundant features, while systems such as C4.5 have some defence against such phenomena in the form of embedded feature selection.

Wrapper and filter methods tend to be in direct competition and mutually exclusive. Much work has been done to compare the relative performance of filter and wrapper methods (Aha and Bankert (1994), John *et al.* (1994), Langley (1994), Mlanenič and Grobelnik (submitted), *inter alia*), and with few exceptions, the finding has been that wrapper methods are empirically superior to filter methods, largely due to them being able to customise the feature set to the idiosyncrasies of the induction algorithm. It is for this reason that we opt for a wrapper method. The flip side of the coin, however, is that wrapper methods tend to be expensive due to them being based on "nested cross-validation".

Nested cross-validation (Schaffer 1993) provides the foundation for wrapper-style feature selection. In conventional cross-validation, the dataset is sectioned off into N partitions of equal size, and over N iterations, each partition is in turn taken as the test set and the remaining sets as the combined training data. In nested cross-validation, we take this methodology one step further in, on each top-level iteration, further partitioning the training data into M partitions, and validating some informational concept through secondary cross-validation, before applying the results to the top-level held-out test data. Nested cross-validation maintains the integrity

of the original test data for each top-level iteration (i.e. is truly open evaluation), while cross-validating the performance of a particular data configuration over the training data. We maximise the effectiveness of nested cross-validation by facilitating stratification at both the primary and secondary levels of cross-validation. That is, on each application of cross-validation, we divide the data so as to maximise the uniformity of class spread between partitions.

3.8 Automatic feature selection & construction

1. Divide the overall dataset D into N equally-sized partitions of maximally uniform class distribution. For each partition D_i:
   (a) Set $Test_i = D_i$, $Train_i = D - D_i$ and F to the overall feature set
   (b) **Apply the feature relevance estimation measure over $Train_i$ in determining the ordered list of all features $F_{RelRank} = f_1, f_2, ..., f_{|F|}$, in increasing order of feature relevance**
   (c) Divide $Train_i$ into M equally-sized partitions of maximally uniform class distribution.
   (d) For each partition $Train_{i,j}$:
      i. Set $NestTest_{i,j} = Train_{i,j}$ and $NestTrain_{i,j} = Train_i - Train_{i,j}$
      ii. Using $NestTrain_{i,j}$ as the training data and $NestTest_{i,j}$ as the test data, calculate the test accuracy $NestAcc_j$ over F
   (e) Set $NestAcc_{prev} = \frac{\sum_{j=1}^{M} NestAcc_j}{M}$
   (f) For each f_d $(d = 1, 2, .., |F|)$:
      i. For each partition $Train_{i,j}$:
         A. Set $NestTest_{i,j} = Train_{i,j}$ and $NestTrain_{i,j} = Train_i - Train_{i,j}$
         B. Using $NestTrain_{i,j}$ as the training data and $NestTest_{i,j}$ as the test data, calculate the test accuracy $NestAcc_j$ over $F - f_d$
      ii. Set $NestAcc_{curr} = \frac{\sum_{j=1}^{M} NestAcc_j}{M}$
      iii. If $NestAcc_{curr} \geq_{del} NestAcc_{prev}$, then set $NestAcc_{prev} = NestAcc_{curr}$ and $F = F - f_d$
   (g) Using $Train_i$ as the training data and $Test_i$ as the test data, calculate the test accuracy Acc_i over the final F
2. Set $FinalAcc = \frac{\sum_{i=1}^{N} Acc_i}{N}$

Figure 3.7: The basic feature selection method

3.8.1 Basic feature selection method

The components that go to make up a feature selection method are: determination of a starting point, organisation of the feature space search, a method to evaluate attribute subsets, and a halting condition (Langley 1994; Blum and Langley 1997).

Having established that we are committed to using a wrapper approach to feature selection, the next issue is what method of feature selection we will combine it with. We adopt the general

search strategy of backward sequential search (BSS), which has been used successfully in much research relating to feature selection (Aha and Bankert (1994), Choubey *et al.* (1996), Choubey *et al.* (1998), *inter alia*). In BSS, the starting point for the search is the full feature set. The search proceeds by taking one feature at a time, generally according to a pre-determined feature ordering, and determining whether to retain or discard it, based on fluctuation in classification accuracy in the presence and absence of that feature. The search then halts when we have made our way through the entire feature set. BSS thus constitutes a greedy approach, in that we evaluate only a small portion of the full range of feature combinations (n feature combinations, for an original feature set made up of n elements).

The reason that we opt for BSS rather than a forward sequential search, for example, is that by taking all features together to begin with, we capture all feature interactions. For weakly relevant features, therefore, we evaluate them in the context of other interacting weakly relevant features, giving them maximal potential to demonstrate their classificational worth. Clearly, as the feature selection process progresses, we are going to being pruning off particular features and end up evaluating features further downstream in absence of those pruned features. However, if one is to assume that there is a distinct set of weakly relevant feature clusters that produce strong relevance as a whole, then when the first member of such a cluster is considered for pruning, we are essentially evaluating the import of all feature clusters in which the feature plays a part. In this sense, the relevance of every feature cluster is evaluated at least once during feature selection, if in the context of other relevance/redundancy judgements relating to the feature under consideration at that particular time.

The method used to evaluate attribute subsets with our implementation of BSS is nested cross-validation, and features are processed according to increasing order of estimated feature relevance. If secondary cross-validation results in a drop in classification accuracy, then the feature is retained, whereas if classification accuracy benefits from the feature being removed, it is pruned off. This process is outlined in Figure 3.7.

3.8.2 The dividing line between feature deletion and retention

One area where we must exercise caution with wrapper-style feature selection is in comparing the mean test accuracies with and without a given feature, to determine its relevance. While it is certainly possible to take the line that any loss in mean test accuracy through deleting a feature points to relevance, we can alternatively interpret a marginal drop in accuracy as being insignificant, making it safe to delete that feature. This latter interpretation does not, of course, immediately remedy the problem of where to draw the line between feature deletion and retention, and we turn to the one-tailed paired t test of statistical difference to resolve this issue.

The one-tailed paired t test (Snedcor and Cochran 1989; Manning and Schütze 1999) takes two samples of homogeneous numerical values (e.g. system outputs), and determines whether there is a statistical difference between them through analysis of the mean and pooled variance of the two datasets. The test returns a t value, which, in combination with a standard statistical table, provides a judgement on the superiority of one dataset over the other, within a fixed corridor of confidence. t is calculated by way of:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{2s^2}{n}}} \tag{3.4}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the sample means of datasets 1 and 2, respectively, n is the sample size and

s^2 is the pooled sample variance, defined by:

$$s^2 = \frac{\sum_{i=1}^{n}(x_{1,i} - \bar{x}_1)^2 + \sum_{i=1}^{n}(x_{2,i} - \bar{x}_2)^2}{2(n-1)} \tag{3.5}$$

Here, each $x_{1,i}$ and $x_{2,i}$ is a data instance from dataset 1 and 2, respectively.

We take dataset 1 to be the test accuracies across each pass of cross validation for the feature set *without* the feature in question, and dataset 2 to be the test accuracies *with* that feature. The null hypothesis is thus that the accuracy in the absence of the feature, is superior to that with it, in which case we wish to prune the feature.

Rather than setting a threshold for the t value from a statistical table, we select an arbitrary value for use throughout nested cross-validation. As described above, in the instance that the deletion of the feature leads to a marginal drop in accuracy, identical accuracy or any gain, then we are prepared to proceed with pruning that feature. The threshold is thus set to a low-valued negative value, and any t value above this interpreted as a justification for deleting the feature. In evaluation, we test the effects of different threshold values on the final accuracy and feature composition.

This qualified method of determining whether the disparity between the test accuracies with and without a feature is significant, is indicated as the "$\geq_{del}$" operator in Figure 3.7.

3.8.3 Feature relevance estimation methods

This leaves the question of how we model feature relevance. We experimented with a range of relevance estimation methods for this purpose, namely information gain, gain ratio, χ^2 ('chi-square') and shared variance.[16] We also propose a fifth method which integrates all four basic feature relevance estimation methods. The scores returned by these relevance methods are used to rank the features in increasing order of estimated relevance, and this is the order features are processed in.

The different feature relevance estimation methods are slotted into the basic procedure given in Figure 3.7 at the step indicated in boldface.

Below, we detail the basic mechanism employed in the four basic relevance estimation methods and briefly discuss their relative strengths, before going on to describe the integrated relevance estimation method. Note that all methods operate over individual features, and are unable to capture feature interaction. For the four basic methods, higher scores indicate a higher level of feature relevance, whereas smaller values indicate higher relevance for the combined method.

Information gain

The **information gain** (Quinlan 1986; Quinlan 1993) of a given feature is an indication of the informativeness of that feature in classifying the dataset. It is measured by way of the difference between the class entropy with and without that feature:

$$InfoGain(i) = H(C) - \sum_{v \in V_i} P(v)H(C|v) \tag{3.6}$$

where:

$$H(C) = -\sum_{c \in C} P(c) \log_2 P(c) \tag{3.7}$$

$$H(C|v) = -\sum_{c \in C} P(c|v) \log_2 P(c|v) \tag{3.8}$$

[16] This combination happens to correspond to the four feature weighting methods implemented within the TiMBL system.

Here, i represents the feature in question, C is the set of class labels, and V_i is the set of values that feature i takes. The value of each of the probabilities is estimated from the relative frequencies of the values in the training data.

One well-documented failing of information gain is that it tends to overestimate the relevance of features with high numbers of values (Quinlan 1993; Caruana and Freitag 1994; Mitchell 1997). This does not present a problem for the original parameterisation, where all of the features are binary (except for the "gapped fixed argument head NP" feature which takes three values). In the expanded feature set, however, there is a greater range of value cardinalities, including a number of lexical-valued classes which take hundreds of distinct values. We must therefore proceed with caution in using information gain.

Gain ratio

Gain ratio was developed by Quinlan (1986) in direct response to the bias towards multi-valued features of information gain. The way it resolves this bias is to divide the information gain by "split information", which is an indication of how broadly and uniformly a given feature splits the data. Split information is calculated as given in equation (3.10), the value of which is used to normalise the information gain value as in equation (3.9):

$$GainRatio(i) = \frac{InfoGain(i)}{SplitInfo(i)} \tag{3.9}$$

$$SplitInfo(i) = -\sum_{v \in V_i} P(v) \log_2 P(v) \tag{3.10}$$

where each variable is as defined for information gain. By taking the ratio of information gain to split information, the relevance of features which have large numbers of uniformly distributed values is reduced.

While gain ratio is less susceptible than information gain to over-rating the relevance of multi-valued features, it is not completely immune to this effect, as demonstrated by White and Liu (1994). This leads us to consider other relevance evaluation measures.

Chi square

The χ^2 metric is statistics- rather than information theory-founded, and based on comparison of the expected and observed frequencies of each value for a given feature occurring with each class. The difference between the observed and expected frequencies is scaled according to the magnitude of the expected frequency.

$$\chi^2(i) = \sum_{j,k} \frac{(E_{jk} - O_{jk})^2}{E_{jk}} \tag{3.11}$$

where:

$$E_{jk} = \frac{n_{.k} n_{j.}}{n_{..}} \tag{3.12}$$

Here, for feature i, O_{jk} is the observed number of instances of value v_j which co-occur with class c_k and E_{jk} is the expected number of instances of value v_j occurring with class c_k. E_{jk} is calculated as given in equation (3.12), where $n_{.k}$ is the total number of instances of class c_k occurring with the given feature, $n_{j.}$ is the total number of instances of value v_j, and $n_{..}$ is the total number of occurrence of the given feature (i.e. the number of training examples).

One shortcoming of the χ^2 measure is that it generally requires a large data sample, and also performs badly when a significant proportion of expected frequencies are small (Snedcor and Cochran 1989).

Shared variance

Shared variance is a slight variant over χ^2 corrected for the degrees of freedom by way of:

$$SharedVar(i) \quad = \quad \frac{\chi_i^2}{N \times \min(|C|, |V|) - 1} \tag{3.13}$$

where N is the number of training examples, $|C|$ is the number of distinct classes and $|V|$ is the number of distinct values that feature i takes.

Due to its direct reliance on χ^2, shared variance is fragile for small data sizes and skewed distributions. Also, features occurring with a single value throughout the training data are evaluated as being maximally relevant (with a shared variance of 1). In practice, this does not have any impact on feature selection when shared variance is used in isolation, as evaluation of the general relevance of a single-valued feature will readily reveal that it can be omitted without affecting classification accuracy.

Averaged relevance ranking

As noted above for each individual method, particular data compositions and distributions produce questionable feature relevance judgements for all four methods. In order to develop a method which is robust over all data types, we propose a simple method which integrates the four basic relevance estimation methods into one, which we name "averaged relevance ranking".

Averaged relevance ranking consists simply of determining the score for each feature under each of the information gain, gain ratio, χ^2 and shared variance measures, and generating independent feature rankings based on the results of each method. As part of this process, we apply a filter to the results of each of the χ^2 and shared variance methods, in setting the estimated relevance to zero in the case that the information gain and gain ratio are zero.[17] We then compute a final score for each feature from the sum of the ranks of that feature under the four basic methods, and re-rank the features in decreasing order of summed rank, from least to most relevant.

3.8.4 Combined feature selection and construction

We are now in a position to interleave feature construction with the feature selection process. Feature construction takes place within the same framework as outlined above for feature selection, and makes use of the same ordering of features in ascending order of estimated relevance. At the same time as determining whether to delete each feature, we consider constructing new features based on the current feature. This is carried out through the medium of a single construction operator: binary conjunction over the values of the two features in question. We exhaustively combine together the current feature with all other features higher than the current feature in the feature relevance ranking, and estimate the relevance of the resultant constructed features in the same manner as for feature relevance. In the case of feature construction, however, we first normalise the various relevance scores by dividing them by the higher of the original relevance scores for the two base features, and rank constructed features in *descending* order of relative relevance. We then work through the list of features and, by way of nested cross-validation, determine whether

[17] In practice, gain ratio must necessarily be zero when information gain is zero, due to the numerator of gain ratio being information gain.

1. Divide the overall dataset D into N equally-sized partitions of maximally uniform class distribution. For each partition D_i:
   (a) Set $Test_i = D_i$, $Train_i = D - D_i$ and F to the overall feature set
   (b) **Apply the feature relevance estimation measure over $Train_i$ in determining the ordered list of all features $F_{RelRank} = f_1, f_2, ..., f_{|F|}$, in increasing order of feature relevance**
   (c) Divide $Train_i$ into M equally-sized partitions of maximally uniform class distribution.
   (d) For each partition $Train_{i,j}$:
      i. Set $NestTest_{i,j} = Train_{i,j}$ and $NestTrain_{i,j} = Train_i - Train_{i,j}$
      ii. Using $NestTrain_{i,j}$ as the training data and $NestTest_{i,j}$ as the test data, calculate the test accuracy $NestAcc_j$ over F
   (e) Set $NestAcc_{prev} = \frac{\sum_{j=1}^{M} NestAcc_j}{M}$
   (f) Set $OrigFeatNo = |F|$. For each f_d ($d = 1, 2, .., OrigFeatNo$):
      i. Set $g = 0$ and $CurrFeatDeleted = 0$
      ii. Construct the set of all new conjunctive features $F_{const} = f_d \cap f_e$ (where $e = d + 1, d + 2, ..., |F|$), for all values of f_d and f_e
      iii. **Apply the feature relevance estimation measure over $Train_i$ in determining the ordered list of all features $F_{ConstRelRank} = fc_1, fc_2, ..., fc_{|F_{const}|}$, in decreasing order of feature relevance**

ConstLoop1:
      iv. For each partition $Train_{i,j}$:
         A. Set $NestTest_{i,j} = Train_{i,j}$ and $NestTrain_{i,j} = Train_i - Train_{i,j}$
         B. Using $NestTrain_{i,j}$ as the training data and $NestTest_{i,j}$ as the test data, calculate the test accuracy $mathitNestAcc_j$ over $F - f_d$
      v. Set $NestAcc_{curr} = \frac{\sum_{j=1}^{M} NestAcc_j}{M}$
      vi. If $NestAcc_{curr} \geq_{del} NestAcc_{prev}$, then set $NestAcc_{prev} = NestAcc_{curr}$, $F = F - f_d$ and $CurrFeatDeleted = 1$

ConstLoop2:
      vii. Set $g = g + 1$
      viii. For each partition $Train_{i,j}$:
         A. Set $NestTest_{i,j} = Train_{i,j}$ and $NestTrain_{i,j} = Train_i - Train_{i,j}$
         B. Using $NestTrain_{i,j}$ as the training data and $NestTest_{i,j}$ as the test data, calculate the test accuracy $NestAcc_j$ over $F + fc_g$
      ix. Set $NestAcc_{curr} = \frac{\sum_{j=1}^{M} NestAcc_j}{M}$
      x. If $NestAcc_{curr} \geq_{add} NestAcc_{prev}$, then set $NestAcc_{prev} = NestAcc_{curr}$ and $F = F + fc_g$, and add fc_g into each $Train_{i,j}$. If $CurrFeatDeleted$ then GOTO *ConstLoop2*, else GOTO *ConstLoop1*
   (g) Using $Train_i$ as the training data and $Test_i$ as the test data, calculate the test accuracy Acc_i over the final F

2. Set $FinalAcc = \frac{\sum_{i=1}^{N} Acc_i}{N}$

Figure 3.8: The combined feature selection & construction method

the addition of that feature to the dataset enhances classification accuracy. In the case that the addition of that feature is of benefit to overall accuracy, we update the feature set, add that feature into all exemplars in the primary test data, and continue on to consider the next constructed feature. Once we reach a feature that is not worthy of inclusion in the dataset, we terminate evaluation of the constructed features.

In order to restrict the scope of feature construction to feature values with a wide range of application, only values which occur in at least 5% of exemplars are considered for feature construction. This stipulation is largely targeted at low-frequency lexical values, which may well produce localised strong relevance for that feature. The effect of such relevance on overall classification accuracy, however, is highly limited.

On each iteration, feature construction takes place over all features which have not yet been processed, that is all features which have a higher estimated relevance than the current feature. We do not construct features in combination with previously-processed features, irrespective of whether they have been retained or not, as their relevance would have been estimated during the processing of the second feature. For complex constructed features, on the other hand, incremental build-up in feature complexity must take place in increasing order of relevance of the base features. In other words, only one developmental path is allowed for a given complex feature.

Our choice of the binary conjunction operator as our sole form of constructing features, stems from the observation that any disjunction of values can be formed with the conjunction and negation operators (where the negation operator is modelled by a value of 0 for a constructed feature). Note that while we construct only binary features involving a given feature on a given iteration, complex conjunctions can arise from construction over features constructed on previous iterations. That is, feature construction on each iteration takes place over all features in the current feature set, whether they be base features or features constructed on previous iterations.

As for the basic feature selection method, we require that the gain in test accuracy facilitated by the addition of a constructed feature, be of a certain magnitude for that feature to be included in the feature set. Recall that our driving motivation is to restrict the feature space as much as possible, such that we want to be reasonably confident that the addition of any constructed feature is going to genuinely enhance accuracy. The method used to evaluate whether the gain in accuracy is significant, is essentially the same as for feature selection and hinges around the t test. Once again, we base determination of whether to add in a constructed feature on a user-defined threshold on the t value returned by the test (not necessarily the same absolute value as for feature selection). Unlike feature selection, however, this threshold must be positive to ensure that the disparity between the classification accuracy with the constructed feature is indeed significantly superior to that without it. The accuracy gain operator is this redefined slightly from $\geq_{del}$, to produce $\geq_{add}$.

At that same time as evaluating whether it is worthwhile expanding the feature set with a constructed feature, on addition of each new feature, we recheck whether this makes the base feature under current consideration redundant. This is carried out until either the base feature has been pruned off (in which case we still continue adding constructed features until no more gain in classification accuracy is forthcoming), or no further feature can be added. Theoretically, this allows for maximal compaction of features and explicit modelling of feature interaction, in that we are replacing the base features by feature clusters with higher relevance.

3.9 Extra features in Japanese RCC interpretation

Particular features we were keen on incorporating into the feature set are the case-slot type and surface case marker of the overall RCC in the context of the superordinate clause, a flag for RCCs

Feature no.	*Feature description*	*Value range*
1 ⋮ 49	Features #1–49 from the basic parameterisation	0, 1 or 1, s, d
50 ⋮ 85	RCC matrix verb VSAs	0, 1
86	Noun head of RCC head NP	*noun*×3038
87	RCC matrix verb (normalised)	*verb stem*×987
88 ⋮ 116	RCC matrix verb inflection/auxiliary verb collocation	0, 1
117	Superordinate case slot type	*case slot type*×19, unknown
118	Superordinate case marker	*case marker*×53, unknown
119	Appositively coordinated RCC	0, 1
120	Superordinate matrix verb	*verb stem*×1074, unknown
121 ⋮ 142	Superordinate matrix verb class membership	0, 1, unknown
143 ⋮ 178	Superordinate matrix verb VSAs	0, 1, unknown

Table 3.6: The 178-feature basic RCC parameterisation

collocating with appositives, and lexical items in the form of the RCC matrix verb, the matrix verb in the superordinate clause, and the head noun of the RCC head NP. We hypothesise that there is a high level of regularity between the RCC interpretation type and the type of case-slot the RCC occurs in in the superordinate clause, and that such information should aid RCC analysis; in order to fully capture this type of correspondence, we describe case slot type by way of the same set of case slot types as is used to represent case-slot gapping. While the correspondence between RCC interpretation and the type of case marker of the RCC in the superordinate clause is more tenuous, we provisionally include this in the expanded feature set also. The appositive collocation flag is based around the observation that RCCs involved in such structural relations tend to be case-slot gapping. The inclusion of the RCC matrix verb and superordinate clause matrix verb features is aimed at facilitating the automatic construction of verb classes not picked up on in the original research; similarly, by providing the noun head of the RCC head NP as a distinct feature, we open the door for the automatic postulation of new noun classes which aid RCC interpretation.

In terms of the actual implementation of these features, we run into difficulties with cosubordinated RCCs, RCCs with coordinated head NPs, and also with superordinate clause contexts where the RCC is embedded within an NP or otherwise not immediately case marked. With cosubordinated RCCs, we simply take the matrix verb of the final (encapsulating) relative clause, and with coordinated head NPs we arbitrarily set the noun head lexicalisation to that noun head which occurs first in the head NP. Matrix verbs are described in regular form, in absence of any verb morpheme or auxiliary verb collocation. The situation with RCCs embedded within NPs presents a more difficult problem, and forces us into permitting an "unknown" value for the superordinate case slot type and case marker features. With the superordinate clause matrix verb, also, there is the possibility that the verb is not contained in the case frame dictionary, in which case we have no way of normalising it to canonical form and simply return "unknown".

Other feature types which we evaluate more because they are available than due to us sensing that they may be valuable, are verb morphemes and auxiliaries collocating with the matrix verb of the RCC (29 binary features,[18] again taking such features from the matrix verb of the final relative clause in the case of cosubordination), the full set of VSAs for the RCC (despite the damning results from above, in the hopes that feature selection will be able to pick out any useful VSAs and discard the rest), and a full description of the verb class and VSA content of the superordinate verb governing the RCC. Due to coverage problems with the case frame dictionary, we make allowance for an "unknown" value for each of the superordinate clause verb class and VSA features.

Intra-clause disambiguation and inter-clause indexing are carried out according to the VS and `AND`'ing methods, respectively, based on the findings of the original evaluation. In `AND`'ing an unknown value with a positive value, we return "unknown", and for a negative value, we return a negative value.

In total, we end up with an impressive 178 features (up from 49) of varying relevance and with varying levels of cardinality (from 2 values up to 1075), leaving the question of just how to perform feature selection and construction.

3.10 Evaluation of the feature selection and construction methods

The feature selection and construction methods were tested over both the expanded 178-feature and original 49-feature datasets, the results for which are presented in Tables 3.7 and 3.8, respectively. All results are based exclusively around TiMBL, under 10×10 nested cross-validation

[18] Selected by taking those verb morphemes and auxiliaries which occurred with a frequency of more than 50 in the dataset.

Method	*Selection threshold*	*Construction threshold*	*Mean test accuracy (e.r. reduction)*	*Pruning rate*	*Ave. constructed features*
Basic TiMBL [-S,-C]	—	—	85.50%	—	—
Information gain [+S,-C]	0	—	85.86 (2.52%)	9.33%	—
Gain ratio [+S,-C]	0	—	85.88 (2.66%)	9.11%	—
χ^2[+S,-C]	0	—	85.78 (1.98%)	10.39%	—
Shared variance [+S,-C]	0	—	85.80 (2.12%)	9.27%	—
Ave. rank [+S,-C]	0	—	85.88 (2.66%)	10.22%	—
Random ranking [+S,-C]	0	—	85.61 (0.78%)	8.21%	—
Ave. rank [+S,+C]	0	0	**86.64** (7.88%)	52.23%	6.3
Ave. rank [+S,+C]	−0.1	0.1	**86.89** (9.63%)	53.07%	3.3
Ave. rank [+S,+C]	−0.2	0.2	**87.79** (15.80%)	81.56%	1.0
Ave. rank [+S,+C]	−0.3	0.2	**87.55** (14.19%)	80.61%	0.2
Ave. rank [+S,+C]	−0.3	0.3	**87.81** (15.93%)	89.78%	0.1
Ave. rank [+S,+C]	−0.5	0.5	**87.38** (12.98%)	93.24%	0

Table 3.7: Results for feature selection (±S) and construction (±C) over the 178-feature expanded dataset

Method	*Selection threshold*	*Construction threshold*	*Mean test accuracy (e.r. reduction)*	*Pruning rate*	*Ave. constructed features*
Basic TiMBL	—	—	87.82%	—	—
Ave. rank [+S,+C]	0	0	88.33 (4.23%)	16.35%	6.7
Ave. rank [+S,+C]	−0.1	0.1	88.00 (1.52%)	49.62%	1.5
Ave. rank [+S,+C]	−0.2	0.2	88.08 (2.16%)	57.31%	0.5
Ave. rank [+S,+C]	−0.3	0.2	87.89 (0.56%)	62.31%	1.4
Ave. rank [+S,+C]	−0.3	0.3	87.89 (0.56%)	61.15%	0.2
Ave. rank [+S,+C]	−0.5	0.5	87.77 (−0.40%)	66.92%	0.1

Table 3.8: Results for feature selection (±S) and construction (±C) over the original 49-feature dataset

(10-fold primary and secondary cross-validation). The reason we chose TiMBL and not C4.5 to test the proposed methods is principally because k-NN methods are more sensitive to the effects of redundant and irrelevant features (Blum and Langley 1997; Langley and Sage 1997). For each system configuration, we present both the mean test accuracy and (in brackets) the relative error reduction over the baseline method.

In evaluation over the 178-feature dataset, we first compare the various feature relevance estimation methods under simple feature selection, before moving on run the best such configuration over combined feature selection/construction, testing a variety of threshold settings. The first statistic of note in Table 3.7 is the baseline accuracy of 85.50% for TiMBL, falling well short of the 87.82% accuracy for the basic feature set. This result is perhaps unsurprising, as nearest neighbour methods such as TiMBL are highly prone to the effects of feature redundancy and irrelevance, as noted at various points in this chapter. It is a little disappointing, however, to see that our effort in expanding the feature set has simply led to a depreciation in classification accuracy.

Looking next to the comparison of the feature relevance estimation methods for simple feature selection with the selection threshold set to zero (i.e. the naive policy of pruning a feature only if it brings about a drop in classification accuracy), we see that there is very little to separate the five methods. The best-performing methods are gain ratio and our averaged relevance ranking method, which perform at a mean test accuracy of 85.88%, representing an error reduction of 2.66% over the full-feature baseline system. To gain a more direct insight into the effectiveness of our feature selection method, we also generated a random feature relevance ranking for testing. The results here are slightly down on those for the feature relevance estimation methods, but still up on the full dataset. This demonstrates both that an explicit modelling of feature relevance is more successful at capturing feature interaction, and at the same time that the proposed feature selection method (i.e. nested cross-validation) is highly robust to noisy data.

Based on these preliminary results and also our intuition that it should be more robust than gain ratio, we select the averaged relevance ranking method to model feature relevance for combined feature selection/construction.

Feature selection/construction is performed under a range of both feature selection and feature construction thresholds. With both thresholds set to zero, we see an immediate gain over the simple feature selection method, up to a mean test accuracy of 86.64%, almost tripling the relative size of reduction in error. As we then expand the scope of pruning by reducing the selection threshold, and reduce the scope of constructed feature addition by increasing the construction threshold, we see further appreciable gains, up to a peak accuracy of 87.81% with the feature selection thresholds set to -0.3 and the feature construction threshold to 0.3. The relative error gain over the baseline method here is an impressive 15.93%, and we have finally reached the level of accuracy of the basic feature set. Threshold values closer to zero produced lower test accuracies, suggesting that we are adding features where they are not required and/or we are not pruning extensively enough. Threshold values further away from zero also led to a degradation in performance, due to over-pruning (at over 93% of features).

The average pruning rate for the $-0.3/0.3$ configuration was close to 90%, although disappointingly, only one constructed feature was added out of the 10 passes of primary cross-validation. A total of 10 features were retained over all 10 passes of primary cross-validation, as detailed in Table 3.9, all of which, interestingly, were from the original feature set. The permanence of these features in the final feature set would suggest that they are strongly relevant. Out of curiosity, we ran TiMBL over these 10 features only, producing a mean test accuracy of 76.41%. This underlines the importance of weakly relevant features to complement strongly relevant features, and achieve optimal coverage of all classes. Taking the opposite tack, a total of 138 features were found to be

either redundant or irrelevant, and were pruned over all 10 passes of primary cross-validation.[19] Once again, we ran TiMBL over all remaining features, producing an accuracy of 85.47%, right around the level of the baseline. What this tells us is that significant numbers of the retained features are also irrelevant, redundant or very weakly relevant, and that proper selection must be made between these to attain the performance level we saw for feature selection/construction.

Turning next to the results over the original 49-feature dataset, we see a departure from the trend as was observed for the expanded dataset, in that the best performance is achieved with both thresholds set to 0, at a test accuracy of 88.33% (pruning rate: 16%, average constructed features per iteration: 6.7). In this case, however, the reduction in error over the baseline is slight, at a little over 4%. The main reason for this is that there is little redundancy in the basic parameterisation and few irrelevant features. Two positive facts to come from this evaluation, however, were that (a) our method did not depreciate accuracy, making it usable in cases where there is uncertainty as to the level of irrelevant and redundant features; and (b) feature construction came into its own, and contributed to the appreciation in performance.

Feature no.	*Feature description*
1	Subject case slot defined and instantiated
2	Direct object case slot defined and instantiated
12	Nominalised adjective head NP
13	Agentive head NP
15	Goal agentive head NP
17	Instrumental head NP
24	Exclusive RCC compatibility
25	Inclusive RCC compatibility
26	Copular matrix verb
30	Tool-aided action verb

Table 3.9: Features retained over all passes of primary cross-validation under combined feature selection/construction

3.11 Final discussion and wrap-up

The proposed feature selection method proved both effective and robust, particularly when paired with the feature construction method. The best mean test accuracy achieved was 87.81% for combined feature selection/construction, representing an impressive error reduction of 15.93% over the baseline system using all features. Relating this back to the results for the original parameterisation, however, that we have simply regained the ground we lost in expanding out the feature set in the first place. One interpretation of this result is that the top-ranking accuracies achieved with C4.5 and TiMBL over the original feature set of 89–90%, and then TiMBL again with the expanded feature set, represent not only a cap on the optimal interpretational accuracy achievable for the original parameterisation, but the best we can hope to achieve for the RCC interpretation task under the feature-based disambiguation paradigm.

Before we draw this fatalistic conclusion, it would be worth looking at other forms of supervised learning and feature selection over both datasets. One learning paradigm that may well produce

[19] We do not list all these features here, but simply observe that the bulk of the pruned features were those added in expansion of the feature set.

slight performance gains is hierarchical classification, where individual classifiers are trained and tested over independent portions of the feature space, and the output from each combined by an additional classifier. This could be performed over linguistically-motivated feature subsets (e.g. head noun semantics, verb classes, case slot flags), or alternatively through more formal means such as bagging and boosting. Bagging (Breiman 1994) attempts to find a set of classifiers that is consistent with the training data, different from each other and distributed such that the aggregate sample distribution approaches the distribution of samples in the training set. Boosting (Freund and Schapire 1996; Schapire 1999), on the other hand, repeatedly runs a learning algorithm over differing distributions of the training data, and combines the resultant classifiers into a single composite classifier. Both have been applied successfully within the natural language processing fraternity (e.g. Abney *et al.* (1999); Henderson and Brill (2000); Escudero *et al.* (2000)). We leave their implementation in the RCC domain as an issue for future research.

While this research has focused on feature selection, research has also been done on instance selection (Skalak 1994; Blum and Langley 1997) and parameter selection (Cherkauer and Shavlik 1996; Kohavi and John 1995; Veenstra *et al.* 2000). In the first case, the composition of instances in the training set is adjusted to maximise test accuracy, and in the latter case, the parameters under which the system is run (e.g. the C4.5 pruning rate) are dynamically adjusted to maximise performance. Both of these approaches have promise for the RCC task.

To reiterate the contributions and findings of this chapter, the Japanese relative clause construction interpretation task was singled out as an instance of inter-language feature-based disambiguation. First, we proposed a classification of RCC types and outlined a basic set of parameters called upon in an existing rule-base formulation of RCC interpretation. We then detailed a number of ambiguities that exist in mapping an RCC onto a feature vector representation, and proposed methods of resolving each type of ambiguity. Direct comparison was then made between a supervised learning approach to the problem and the rule-based formulation. Supervised learning was exemplified by the C4.5 and TiMBL systems, representative decision tree induction and memory-based learning systems, respectively. In full configuration evaluation, the supervised methods were found to be marginally superior to the rule set, but evidence was found for a test accuracy cap of around 90% on feature-based approaches to the task. We experimented unsuccessfully with the complementation of the original feature set with verb semantic attributes, and also documented the independent roles of noun semantics, verb classes and case slot instantiation flags, in RCC disambiguation.

In the second half of this chapter, we then went on to extend the feature set extensively, including the addition of features pertaining to the superordinate clause of the RCC. This was accompanied with description of wrapper-type feature selection and construction techniques based on nested cross-validation and a multi-modal feature relevance estimation method. Based on evaluation with TiMBL, the feature selection and construction methods were found to be effective, although the resulting performance gain was equivalent only to the drop-off in test accuracy resulting from the expansion of the feature space.

Chapter 4

Argument Status in Japanese Verb Sense Disambiguation

4.1 Introduction

The purpose of this chapter is to devise a robust verb sense disambiguation (VSD) technique for Japanese–English transfer-based MT, based around selectional constraints. For a given clausal input, the VSD method both determines the correct sense of the main verb and aligns *input case slots* from the input, with *target case slots* within the valency frame described for the chosen verb sense. Determination of sense in the model of transfer-based MT we target (i.e. the transfer model of the ALT-J/E system (Ikehara *et al.* 1991) as reflected in Goi-Taikei) equates to selecting the L2 translation for that verb, as "senses" are in fact L2 verb usages with distinct L2 translations (Baldwin *et al.* 1999). The unique valency frame associated with each verb sense is similarly paired with an L2 clause-level translation skeleton, and individual L1 case slots indexed to arguments in the translation skeleton. For this reason, case slot alignment is a crucial component of VSD.

The driving mechanism behind the proposed system is **argument status**, that is an expanded model of complementhood/adjuncthood as laid out by Somers (1984) incorporating 'integral complements', 'complements', 'middles' and 'adjuncts'. Our particular interest in argument status stems from the fact that the properties of each category of argument status can be generalised to provide a surprisingly accurate model of such effects as surface case alternation, scrambling, and propensity for semantic backing-off. Although we will refer to the VSD task exclusively within the bounds of Japanese–English MT, the system mechanism is transferable to other languages for which the notion of argument status is equally well defined.

In performing any research in VSD, there is clearly a requirement for some means of sense distinction/demarkation, and in our case this is based around the Goi-Taikei pattern-based valency dictionary (Ikehara *et al.* 1997; Shirai *et al.* 1997), as developed by NTT for their ALT-J/E machine translation system (Ikehara *et al.* 1991). A valency frame entry in the Goi-Taikei valency dictionary is represented as a list of case slots, each of which is provided with a set of class-based selectional constraints and/or lexical filler candidates.

By nature, selectional constraints commonly overlap between individual case slots for a given valency frame, and also between valency frames; the co-existence of lexical filler candidates for many verb senses further complicates things. This brings about the need for an interface to the dictionary which is able to select between the potentially sense-compatible candidates, and at the same time have recourse to relax the selectional constraints appropriately in the case of over-restriction.

Overlap can also occur between class-based selectional constraints and lexical filler candidates,

	Integral Complement	**Complement**	**Middle**	**Adjunct**
Obligatoriness	Lexically oblig.	Lexically/argument oblig.	Argument opt.	Argument opt.
Semantic demarkation	High (absolute)	Low	Medium	High
Scope for Case Alternation	Almost none	High	Medium	Low
Case Slot Order	Highly restricted	Free, but produces markedness	Relatively free	Free

Table 4.1: Argument status matrix

suggesting the need for some metric capable of evaluating these two forms of case filler restriction homogeneously, and with the ability to differentiate between lexical filler instances designed to complement selectional constraints (i.e. lexical candidates outside the scope of the basic selectional constraints) and fixed arguments. Again, this is achieved principally through the precept of argument status.

A robust VSD system must be able to cope with non-canonical input. This correlates to being able to model semantic effects such as metonymy (see Mahesh *et al.* (1997b)), sense extension (see Mahesh *et al.* (1997a)) and occurrences of unknown words, and lexical effects such as (in the case of Japanese) surface case alternation, zero anaphora, and scrambling. While we make no attempt to individually analyse the various types of semantic effects, there is a clear need to be able to back-off from the level of selectional constraint described in Goi-Taikei (**semantic backing-off**). This need is met through the application of argument status in balancing the trade-off between over-relaxation of selectional constraints bringing incompatible senses into play, and over-restriction discounting the correct analysis.

To summarise, the main intent of this research is to apply argument status in scoring the various semantic and lexical idiosyncrasies faced in robustly disambiguating verb sense. Section 4.2 defines argument status and sets the scene for its later system application; Section 4.3 then goes on to describe the overall system architecture and various scoring schema utilised therein, followed by a basic evaluation of the system on Japanese inputs in Section 4.4.

4.2 Argument status

The argument status of a case slot is defined as the degree of boundedness of that case slot within the valency frame, or a measure of the intrinsic association between an argument and the predicate. The system of argument types utilised herein is a simplified version of the sixfold scale proposed by Somers (1984, 1987), and consists of integral complements, complements, middles and adjuncts. **Integral complements** (i.e. fixed arguments) are highly restricted as to scope for surface case alternation and case slot order alternation, are fixed as to lexical content, and are lexically obligatory (must have a lexical spell-out); **complements**, as major verb constituents, generally display high scope for surface case alternation, produce marked semantics upon case slot permutation, are generally not heavily demarcated semantically, and are argument obligatory (zero instances must be co-indexed to discourse entities or otherwise instantiated through deixis – see (Kameyama 1985)); **middles** form an 'in-between' category between complements and adjuncts, and are generally reasonably restricted in terms of surface case alternation, relatively free in case slot positioning, generally well-demarcated semantically, and argument optional (zero instances indicate the non-

Argument status	*arg_weight*
Integral complement	1
Complement	1
Middle	0.5
Adjunct	0

Table 4.2: Weights for the various argument statuses

instantiation of that case-role); and **adjuncts** are the most peripheral argument type, being highly restricted in terms of surface case alternation, freely insertable into almost any position within the case frame, strongly demarcated semantically, and argument optional. A summary of the characteristics of each argument type is given in Table 4.1.[1]

The following sentence exemplifies the four classifications of argument status:

(22) *Tarō-ga* *sigoto-de* 5-*fuN-sika* *kao-ga* *dasenaku-natta*
Taro-NOM work-LOC 5-minutes-only face-NOM not show-became
"Taro will be able to come for only 5 minutes, due to work commitments"

Here, *kao* is an integral complement, *Tarō* a complement (the subject), *sigoto* a middle (reason case slot), and 5-*fuN-sika* an adjunct (temporal durational case slot).

Argument status is not indicated within the basic Goi-Taikei framework, and was thus predicted from the case-role mark-up of each case slot and post-checked manually.

Close inspection of the properties described in Table 4.1 reveals, perhaps unsurprisingly, that there is a linear transformation in degree of compatibility with the various effects from adjuncts to middles to complements, and extending to integral complements in the case of obligatoriness and case slot ordering, but with integral complements allied most closely with adjuncts in the case of semantic delineation and case marker alternation. This duality for integral complements is attributable to their being closely bound to the predicate both lexically and semantically, producing severe limitations on scope for positioning away from the predicate analogous to the markedness of scrambling of complements, but at the same time imposing over-bearing restrictions on the case slot in terms of both lexical filler variance (semantic delineation) and surface case marking. In the case of other categories of argument status, restrictions on certain lexical effects free up other aspects of argument behaviour. Thus, heavy-handed semantic delineation and surface case constraints for adjuncts, leads to case-role recoverability irrespective of relative positioning within the clause (case slot ordering).

The compatibility with different lexical effects for complements, middles and adjuncts is translated into the *arg_weight* function, with integral complements conditionally equivalent to complements in weight.

4.3 Basic system framework

The basic means of analysis is to produce a bipartite graph of all possible mappings between input and valency frame case slots (see Figure 4.1). Each edge is then scored through evaluation of the relative satisfaction by the input case filler of the selectional constraints for the target case slot. These preliminary mappings are subsequently verified for surface case marker compatibility, the

[1] See Meyers *et al.* (1996) for a discussion of additional semantic factors affecting the basic complement/adjunct distinction.

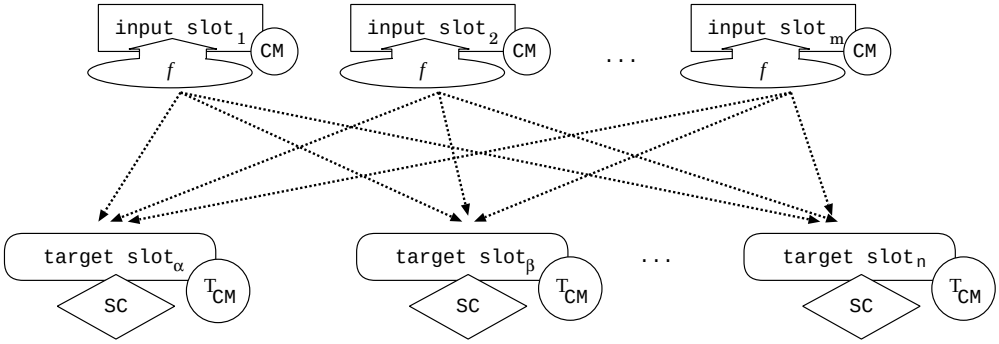

Figure 4.1: Input/valency frame case slot bipartite graph

full set of 1-to-1 mappings from input to target case slots is extracted for each verb sense (including null mappings where permissible), and each candidate mapping is scored accordingly.

4.3.1 Case slot restrictiveness

All non-fixed case slots are encoded with a set of selectional constraints indexed to the Goi-Taikei thesaurus (Ikehara *et al.* 1997) and/or a list of lexical fillers. The Goi-Taikei thesaurus is set up in a conventional tree structure, of non-uniform depth and incorporating lexical items at all levels (average depth from the root ≈ 8.26). So as to establish a measure of restrictiveness of each subtree (node) in the thesaurus structure, the naive notion of **case slot restrictiveness** is introduced, which draws on the intuition that, in cases of selectional overlap between verb senses such as between the "read" and "look at" senses of *miru* with *siNbuN-o miru* "newspaper-ACC read/look at", the more highly specialised (restrictive) sense of "read" is inherently preferred in the absence of other distinguishing information.

The degree of case slot restrictiveness (CSR) of a given node x in the thesaurus is estimated as the inverse of the mean subtree depth of all leaf nodes $l_{1..n}$ subsumed by x, with the $tree\_depth$ function defined as the number of nodes between the subtree root x and leaf l_i, inclusive.

$$CSR(x) = \frac{n}{\sum_{i=1}^{n} tree\_depth(x, l_i)} \tag{4.1}$$

Hence, CSR for a leaf is one, while CSR for a case slot of unrestricted selectional scope (i.e. with selectional constraint set to the root node) is $\frac{1}{8.26} \approx 0.12$. Lexical fillers (e.g. fixed arguments) are treated as leaf nodes, and hence have CSR of one.

We are now in a position to posit a metric for the degree of **satisfaction of selectional constraint** (SS) of target case slot t with selectional constraint c, by sense f_s of input case filler f. In the formulation presented in equation (4.2), $\succeq$ is the subsumption operator ($a \succeq b \Rightarrow$ 'a subsumes b'), $sub(a, b)$ returns the least common hypernym node subsuming both a and b, and $rdepth(a)$ is a statement of the inclusive path length from the thesaurus root to a. SS essentially allows for semantic back-tracking up the thesaurus structure until c has been relaxed sufficiently to subsume f_s. As we ascend the thesaurus structure, the degree of SS is diminished through a deflated CSR value, as well as the $rdepth$ ratio of the relaxed selectional constraint to the original selectional constraint being accentuated.[2] Clearly in the case that c subsumes f_s, the $rdepth$ ratio becomes

[2]Parallels can be drawn between this general method and the conceptual similarity metric of Palmer and Wu (1995).

Argument type (a)	**Canonical case marker** (i_{CM})	**Input case marker**				
		wa	ga	no	o	...
Integral complement	wa	1	0	0	0	
	ga	0	1	1	0	
⋮						
Complement	wa	1	0	0	0	
	ga	1	1	1	0	
⋮						

Table 4.3: A fragment of the case marker alternation matrix

1, and the CSR remains unchanged. On occurrence of semantic back-tracking, SS is additionally weighted by way of the *arg_weight* of target case slot t, in line with the observations on semantic demarkation presented above. This acts so as to uphold adjunct constraints (*arg_weight* = 0) and penalise the relaxation of middle constraints (*arg_weight* = 0.5), but permit the relaxation of complement constraints without additional penalisation (*arg_weight* = 1). Note that integral complements are never represented by way of selectional constraints.

$$SS(f_s, c) = \begin{cases} CSR(c) & \textbf{if } c \succeq f_s \\ \frac{rdepth\Big(sub(f_s,c)\Big)}{rdepth(c)} \cdot CSR\Big(sub(f_s, c)\Big) \cdot arg\_weight(t) & \textbf{otherwise} \end{cases} \tag{4.2}$$

Due to the potential for multiple senses of the input case filler, and multiple sets of selectional constraints/candidate fillers for the target case slot, we score each edge in the alignment graph as the maximum value of SS (i.e. SS_{max}) for the given combination of filler f, with senses $f_{1..m}$, and target case slot t, with selectional constraints $c_{1..n}$:

$$SS_{max}(f, t) = \max_{i,j} SS(f_i, c_j) \tag{4.3}$$

Note that in the current formulation, we give consideration only to verb sense disambiguation and choose not to independently disambiguate case fillers. Clearly, there is scope to augment this unidirectional verb-driven approach and potentially improve the precision of VSD through intra-case slot *local* and intra-clausal *topical* disambiguation techniques (Yarowsky 1994; Ng and Lee 1996; Leacock *et al.* 1998), as well as through consideration of domain (Wilks and Stevenson 1998), which are left as topics for future research.

Case marker alternation

Surface case alternates derive from the canonical case marker type(s), and argument status of the target case slot. In the current implementation, this is represented simply as a matrix of possible case marker alternations for each canonical case marker, for a given argument type. This matrix has been developed based around the relative degrees of freedom of case marker alternation outlined in Table 4.1, in that whereas a complement-type nominative case marker (ga) can commonly alternate with any of a range of case markers including the topic (wa), and genitive (no) markers, an integral complement type nominative case marker can generally alternate only with the genitive marker.

A fragment of the connection matrix which documents such potential case marker alternations is provided in Table 4.3.

From the case marker alternation matrix, we define the binary function *calt* for input case marker i_{CM} and the canonical case marker set T_{CM} for target case slot t, such that $calt(i_{CM}, T_{CM})$ is 1 if $connect(i_{CM}, j) = 1$ for some case marker $j \in T_{CM}$, and 0 otherwise.

$$calt(i_{CM}, T_{CM}) = \begin{cases} 1 & \textbf{if } \exists j \in T_{CM}, (connect(i_{CM}, j)) \\ 0 & \textbf{otherwise} \end{cases} \tag{4.4}$$

Scoring individual case slot alignments

The score *align* for each case slot alignment $\langle i, t \rangle$, incorporating input case slot i (filler i_f and case marker i_{CM}) and target case slot t (selectional constraint set t_c and case marker set T_{CM}), is determined via the product of SS_{max} and *calt*.

$$align(i, t) = SS_{max}(i_f, t_c) \cdot calt(i_{CM}, T_{CM}) \tag{4.5}$$

4.3.2 Penalising non-alignment

Equally important as scoring case slot alignments is the enforcement of penalties on unaligned case slots. In this, we treat input and target case slots distinctly.

Potential adjuncthood of unaligned input case slots

One concern which inevitably arises when attempting to capture the valency content of a verb sense, is the handling of adjuncts. The methodology employed in the development of the Goi-Taikei valency dictionary has been to describe complement and middle case slots, but largely avoid overt description of the broad range of adjunct types. As a result, we are invariably left with a residue of adjunct case slots when mapping case slots onto the valency frame. We thus devised a classification of nuclear adjunct types (*temporal*=$TEMP$, *locative*=LOC and *adverbial*=ADV[3]) and, under the assumption that all adjuncts of same basic type coincide in semantic demarkation, we developed a fixed set of lexical and semantic filters for each. These filters and the associated set of canonical case markers for each adjunct type, are employed as meta-case slots to match non-aligning input case slots against, so as to distinguish between 'dangling' input case slots not aligning at any level of processing, and adjunct case slots which are simply not described within the valency frame proper.

The scoring of adjunct case slot alignment is performed identically to that for other case slots, with each adjunct type being described by a pre-defined region within the Goi-Taikei thesaurus. The final score for the **potential adjuncthood** *PA* of a given input case slot i is given as the maximum score for the three given adjunct types.

$$PA(i) = \max_i \Big(align(i, t_{TEMP}), align(i, t_{LOC}), align(i, t_{ADV}) \Big) \tag{4.6}$$

[3]Strictly speaking, all three types can also occur as 'extra-peripherals' within the original Somers nomenclature, and what we define as adjuncts corresponds to an amalgam of Somers' adjuncts and extra-peripherals.

Non-alignment of target case slots

Non-alignment of target case slots is penalised by way of the mean CSR measure for the case slot in question. In this, however, we have to be careful not to over-penalise unaligned adjunct and middle case slots, which we would expect to be readily omissible. This is achieved by multiplying the mean CSR value for each unaligned target case slot t with the *arg_weight* value corresponding to the argument status of t, and taking the maximum such value as the combined penalty for non-alignment of target case slots (PEN_{targ}).

$$PEN_{targ} = \max_{t} arg\_weight(t) \cdot \sum_{i=1}^{n} \frac{CSR(t_i)}{n} \tag{4.7}$$

Our motivation for taking the maximum here is that we want to identify that unaligned target case slot which is most characteristic (i.e. highly constrained) of the given verb sense, while respecting the potential for that case slot to be genuinely uninstantiated in the case of middles and adjuncts (hence the *arg_weight* factoring), and avoiding over-penalisation of valency frames with higher numbers of case slots.

4.3.3 Scoring and ranking valency frame mappings

The scores and penalties detailed above are added to return a single combine *score* for each mapping M.

$$score(M) = \sum_{\langle i,x\rangle \in M} \Big(align(i,x)\Big) + \sum_{\langle i,\_\rangle \notin M} \Big(PA(i)\Big) - PEN_{targ} \tag{4.8}$$

The various mapping candidates are then ranked in descending order according to their respective scores.

4.4 Evaluation

We evaluated our system on all verbs found in the Goi-Taikei valency dictionary which derive from the verb *miru* "to see". By this is meant that, in addition to all dictionary entries for the base verb *miru*, all verbs containing the kanji '見' prefix in their stem were considered. Examples of verbs included in this set are *mieru* "can see/to be visible" and *minaosu* "to reconsider/re-evaluate". The motivation for this seemingly arbitrary choice of verbs is the high degree of lexical ambiguity that exists between them, in the form of full and partial verb homophony (Baldwin 1998a). In the case of the verb set in question, full and partial verb homophony occur when verbs with distinct stem content coincide in lexical form due to combination with auxiliary verb suffixes. This occurs between *mi-a(u)* "(to) see-MUTUAL", with stem *mi*, and *mia(u)* "(to) correspond/(be) commensurate", with stem *mia*.

The dictionary composition is as follows:

Total number verb senses in dictionary: 148
Number distinct verb stem types: 54
Average entries per verb stem: 2.72

The 148 verb senses were used in a verb sense evaluation task on a set of 289 simplex clauses extracted from the EDR corpus (EDR 1995), with each extracted clause having a main verb lexically matching one or more verbs in the dictionary. Unfortunately, the verb sense indices used in the EDR corpus do not align well with those designated in the Goi-Taikei valency dictionary, such that all 289 clauses had to be manually annotated for both verb sense and case slot alignment. Because the Goi-Taikei verb senses are linked to unique English verb translations, appropriateness of the English translation was used as the sole criterion in sense annotation. Hence, verb sense annotation was construed as an interlingual lexical selection (or L2 translation disambiguation) task. The labour overhead involved in this annotation severely restricted the size of the test corpus, and results given below should be interpreted in light of the limited scope of the evaluation task.

The reader is additionally cautioned that, in this evaluation task, all inputs were both automatically clustered into case-marked phrase units (i.e. input case slots) and segmented (for automatic head determination purposes) according to the original EDR mark-up.

Total number clauses in corpus: 289
Total number fixed sense clauses: 23
Coverage of distinct verb senses: 58
Average number case slots per clause: 1.46

In terms of determining the semantic head of each phrase for calculation of SS, we identified the maximum segment-preserving suffix of the overall case filler which matches with a thesaurus entry; for unknown words, SS was set to 0.1.

In evaluation, solutions were ranked based upon the overall *score* for that mapping, with solutions of equivalent score down-ranked. An output was adjudged to be correct if and only if it coincided both in verb sense and case slot alignment, with those annotated for the input in question.

Evaluation of the test corpus in the manner described above, with PA and PEN_{targ} variously activated and deactivated ($\pm PA$ and $\pm TARG$, respectively), produced the results given in Figure 4.4. Note that the *arg_weight* schema given in Table 4.2 was applied consistently for all given system configurations. BEST-n in Figure 4.4 indicates the percentage of clauses for which the correct solution was found in the top n ranked outputs.

As a point of reference, the native ALT-J/E parser was run over the same corpus and *sense-level* accuracy (ignoring correctness of case slot alignment) calculated on the same BEST-n scale. The ALT-J/E verb sense scoring mechanism can be likened to a coarse-grained version of our CSR, weighted according to the part-morphological, part-semantic "case-role" of the target case slot (Bond and Shirai 1997). Unfortunately, we were unable to get ALT-J/E to work on the pre-parsed input format required by our system, such that inputs had to be given as unformatted raw text. This tainted performance in that any internal errors in morphological and syntactic analysis led to noise during verb sense scoring. Having said this, the restricted length of the simplex clause inputs is thought to have minimised the scope for parsing errors.

The correct solution (over both verb sense and case slot alignment) was located at a mean rank of 1.26 out of an average of 6.84 outputs, with the correct verb sense (irrespective of case slot alignment) identified at a mean rank of 1.22. The results show that we are able to identify the correct case slot alignment for the correct verb sense in over 83% of cases, in the instance that both PA and PEN_{targ} are activated. Further, in nearly 98% of cases, the correct alignment and verb sense were contained in the top three ranking system solutions. Close inspection of the relative accuracy with component scoring mechanisms deactivated, reveals that the use of PEN_{targ} avails a performance gain around 10% for $n \leq 5$. Similarly, PA produces significant gains, particularly when employed in tandem with PEN_{targ}.

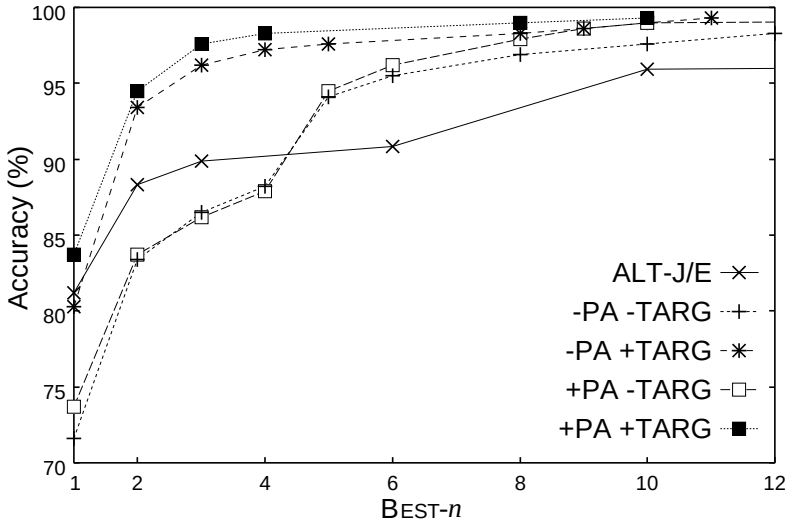

Figure 4.2: BEST-n accuracy over different system configurations

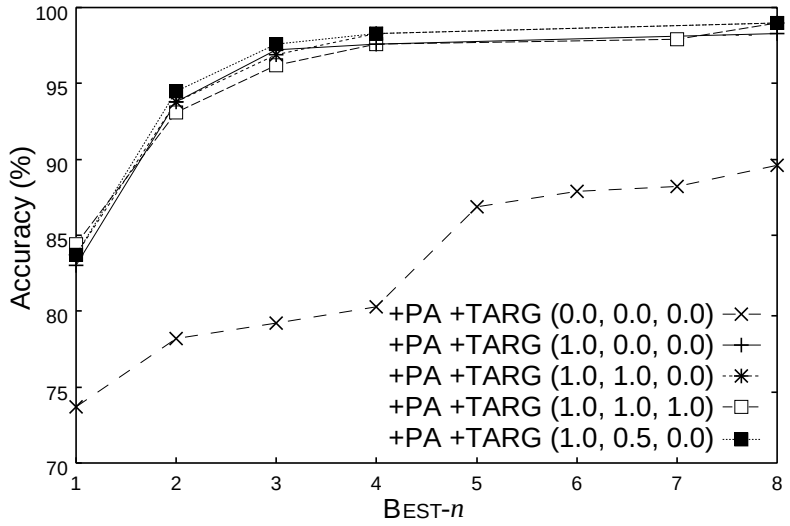

Figure 4.3: System accuracy for differing *arg_weight* values

No. alignment candidates	Ave. rank of solution	No. clause instances
1	1.00	21
2	1.04	113
3	1.17	6
4	1.28	29
5	1.33	3
6	2.00	4
7	1.00	17
8	1.00	1
9	1.75	4
10+	1.71	89

Table 4.4: Mean rank vs. analytical ambiguity

Despite the overall high accuracy rates attained, there were two clauses for which the correct analysis could not be produced due to lexical obligatoriness constraints on target case slots not being met. This represents an over-constraint on the part of the Goi-Taikei dictionary, and is not perceived as a threat to the integrity of our system.

Looking again to Figure 4.4, we notice that our system outperformed the ALT-J/E parser despite the more stringent criterion of both alignment and sense accuracy imposed on it (vs. simple sense accuracy for ALT-J/E). Indeed, the relative performance of the two systems is perhaps better reflected in the figures of 86.51% vs. 81.22% — the simple sense BEST-1 accuracy for our system against that for ALT-J/E. As an additional measure of true performance gain, we additionally calculated the accuracy of a naive "first-sense" technique on the test corpus. Here, we simply return the most common/prominent verb sense lexically matching with the input, as defined within Goi-Taikei. This produces a baseline accuracy of 81.66%, underlining the significance of the 86.51% sense accuracy for our system.

While these figures are promising, they do not directly verify the validity of our claim that argument status aids VSD. We thus additionally tested the system ($+PA + TARG$) with various *arg_weight* parameter settings, indicated in Figure 4.4 as the triple ($complement\_weight, middle\_weight, adjunct\_weight$), with the inequality

$$1 \geq complement\_weight \geq middle\_weight \geq adjunct\_weight \geq 0$$

upheld in all cases. The most striking result in Figure 4.4 is the degradation in performance for the parameter triple $(0.0, 0.0, 0.0)$, that is where all case slots are treated as adjuncts. The principal reason for this fall-off in performance is the blocking of semantic backing-off for complements and middles. There is little separating the remaining parameter combinations, although the proposed $(1.0, 0.5, 0.0)$ setting did marginally outperform other settings at all positions other than BEST-1. This suggests that we get better performance when we treat middles as an in-between category than when conflating them with either complements or adjuncts, and provisionally backs up Somers' claim as to a need for this extra category. Part of the reason for the minimal differentiation between the different *middle_weight* scores is that the particular verb types targeted in evaluation tended not to collocate with middle case slots, suggesting the need for further evaluation on 'middle-heavy' verbs.

One effect not apparent above is the relation between the number of input case slots and accuracy in analysis. We thus further evaluated the influence the number of input case slots had on the average rank, with results given below. Interestingly enough, we get a 100% success

rate for input case frames devoid of case slots, pointing to an ability to cope admirably with underspecification. There is then a drop in accuracy for a single case slot in the input, with gradually recovers with increasing numbers of input case slots.

No. input case slots	No. clause instances	Ave. rank
0	14	1.00
1	172	1.33
2	87	1.20
3	14	1.21
4	2	1.00

An additional item worthy of verification is the ability of the system to correctly discriminate between fixed and general sense. Analysis of the 23 clauses containing a verb of fixed sense revealed that the system returned the correct analysis with the highest rank in all cases, and that there were no instances of a fixed sense solution being returned for a general-sense verb. While this certainly bodes well, these results should perhaps be played down, as all clauses of fixed sense were very clearly so. The performance of the system on more borderline examples of fixed expressions thus remains to be determined.

4.5 Discussion

In this research, we distance ourselves from much of conventional VSD research in that we specifically set out to handle cases of underspecification and non-canonical input. This is particularly salient in the case of Japanese due to the high levels of zero anaphora and commonality of case marker alternation, as are not found in English. It is thus difficult to identify comparable results in the VSD literature, although Kurohashi and Nagao (1994) and Fujii (1998) cite average sense-level accuracies of 76.5% and 82.3%, respectively, the former on underspecified and the latter on fully instantiated inputs. In this respect, our performance levels would appear to improve on previous research.

One other important distinction between our work and much VSD research is that we do not explicitly employ the case-role, case marking or grammatical relation of a case slot to weight it. While we do utilise argument status to weight occurrences of semantic backing-off and penalise non-alignment of target case slots, we do not differentiate between aligned case slots other than implicitly through the CSR score. In this, we distance ourselves from the heuristical, static formulation of ALT-J/E and also the observations of Yarowsky (1993) that certain syntactic relations derive more disambiguating evidence than others (e.g. objects are better disambiguators of verbs than subjects). We do claim, however, that CSR is able to dynamically capture the types of phenomena targeted in these handlings.

To sum up, the verb sense disambiguation method presented here was devised around the Goi-Taikei valency dictionary, as a means of overcoming problems related to underspecification, case marker alternation, word order, and fixed expressions, primarily through the application of argument status-based weighting and analysis of the selectional constraints encoded within the original dictionary. Argument status was successfully applied in semantic backing-off techniques, penalisation of input and target case slot non-alignment, and prediction of case marker alternation. The system returned an accuracy of over 83% on a limited test set of 289 simplex clauses.

Areas of further research include the need to expand evaluation of the system, possible integration with disambiguation of case fillers, expansion of the handling of fixed expressions, and consideration/handling of the effects of scrambling.

Chapter 5

Conclusion

In this thesis, we have taken a broad view of inter-language disambiguation, in studying lexical disambiguation, feature-based disambiguation and fully-fledged semantic disambiguation. Lexical disambiguation was defined as operating over word forms, feature-based disambiguation over a well-defined feature characterisation of the data, and fully-fledged semantic disambiguation over semantic taxonomies or other complex structure types. Each disambiguation type was analysed through a particular embodiment, namely Japanese–English translation retrieval, Japanese relative clause construction analysis, and selectional restriction-based verb sense disambiguation, respectively.

In Chapter 2, lexical disambiguation was discussed through the medium of Japanese–English translation retrieval. Particular emphasis was placed on the orthogonal parameters of segmentation, segment order and segment contiguity, and the impact of each on retrieval performance. The effects of segment order were modelled by way of a number of representative bag-of-words and segment order-sensitive string comparison methods, which were then run over both segmented and unsegmented text, and through a range of N-gram models of local segment contiguity. Evaluation took place via a novel fully-automated translation retrieval evaluation methodology, which used a given string comparison method to retrieve the translation records with the highest level of correspondence to the model translation, and then calculated the degree of coincidence of each system output with the set of "optimal" translations. We also gauged the effects of different segmentation systems on translation retrieval, tested a number of static and dynamic smoothing methods, and verified the role that kanji play in Japanese translation retrieval, before reversing the retrieval direction and further verifying the results for Japanese over English. Key findings can be summarised as:

- Character-based indexing is superior to word-based indexing for Japanese–English translation retrieval (i.e. segmentation is detrimental to retrieval performance).

- The particular bag-of-words methods tested (the vector space model and "token intersection") are at least as accurate in translation retrieval as the segment order-sensitive methods tested (3-operation edit distance & similarity, and weighted sequential correspondence). At the same time, they are faster, more scalable to larger-sized translation memories and easier to implement, making them our string comparison paradigm of choice. This finding was backed up by results for English–Japanese translation retrieval.

- Of all the string comparison methods tested, 3-operation edit similarity was the most consistently accurate, but suffered from significant slowdown over larger data sizes.

- N-gram models of segment contiguity produce benefits for both retrieval accuracy and speed, particularly for character-based indexing. Of the N-gram methods tested, bigrams were found to perform best for character-based indexing, and mixed unigrams/bigrams for word-based indexing.

- If segmentation is to be used, segmentation precision and recall do affect retrieval performance, and analytical robustness over technical terms and other unknown words is highly desirable for technical domains. Lexical normalisation can produce slight gains in translation accuracy.

- Static segment weighting based on character type enhances retrieval performance marginally, and modest gains are also possible with the IDF dynamic weighting schema.

- When using a character bigram model or mixed word bigram model for Japanese–English retrieval over fully alphabetised data equivalent results to retrieval over fully lexically-differentiated data can be obtained. From this, we can conclude that kanji play only a minor role in directing the retrieval process, and also that N-gram models are highly robust.

In Chapter 3, we moved on to the analysis of Japanese relative clause constructions, through feature-based disambiguation techniques. We first defined a classification of RCC types and described the linguistic properties of each, before presenting the basic set of 49 features used to characterise each RCC exemplar. This basic feature set traces back to a rule-based system, which we outlined briefly and established as a touchstone for evaluation. Discussion was next made of intra- and inter-clausal ambiguity and means for resolving such ambiguity, to facilitate the generation of a unique feature vector for a given RCC. The C4.5 and TiMBL induction systems were then run over the data through a number of system and data configurations, and the results related back to those for the rule set. In the second half of the chapter, we went on to expand the original feature set to 178 items, largely drawn from analysis of the superordinate clause and matrix verb governing the RCC. We also presented a wrapper-style feature selection method based on nested cross-validation and a backward sequential search through the feature space, from features of least to most estimated relevance. This was then expanded upon in proposing a combined method of feature selection and construction. In combination with TiMBL, the combined feature selection and selection method generated a reduction in error rate upwards of 15% for the expanded feature set, and a lesser 4% for the basic feature set. In summary, the contributions of this chapter were:

- We verified the possibility of using induction algorithms to produce the same classification accuracy as is possible with expert system-style rule sets for the same feature space.

- A number of surface intra- and inter-clausal disambiguation techniques of high applicability were described. Intra-clausal disambiguation was based around the inflectional content of the matrix verb of the RCC, degree of case slot overlap between the input RCC and case frame, and finally representational preference for different verbs to occur a given realisation. Inter-clausal indexing was suggested as a means of integrating multiple component relative clause feature vectors and enforcing consistency of interpretation, and achieved through logic operators on the constituent feature vectors.

- An accuracy cap of 90–91% was established for feature-based Japanese RCC interpretation, at least using the types of features described herein.

- The effectiveness of both backward sequential search and nested cross-validation for feature selection and construction was validated. As part of this, we proposed an integrated method of feature relevance estimation, drawing on information gain, gain ratio, χ^2 and shared variance.

Evaluation of the worth of adding a constructed feature or pruning an existing feature, was performed using the paired one-tailed t test, and thresholding over the t value.

Finally, in Chapter 4, description was given of a selectional restriction-based verb sense disambiguation (VSD) method, to illustrate the basic nature of fully-fledged semantic disambiguation. The verb sense disambiguation method was based heavily on argument status, that is the complementhood/adjuncthood of verbal arguments. Argument status was suggested as a powerful means of describing propensity for case marker alternation and case slot gapping, and compatibility with semantic backing-off. An argument status weighting schema was proposed, which was then interwoven into a system for scoring each individual alignment for each sense/analysis of the verb in question, where the sense set was defined as the Goi-Taikei pattern-based valency dictionary. The overall scoring mechanism hinged around the concept of "case slot restrictiveness", which constitutes a description of the semantic density of the selectional constraints annotating a given case slot. Selectional constraints are indexed to nodes in the tree-structured Goi-Taikei thesaurus, and the mean leaf depth of that node is taken as an indication of semantic density. Rather than making a binary judgement as to the satisfaction of a given set of selectional constraints by a case filler, we support semantic backing-off, that is incremental relaxation of the selection constraints to a point where a sense of the case filler is subsumed. Penalisation of semantic backing-off was integrated with case slot restrictiveness via argument status, such that the selectional constraints on complement case slots are less stringent than those on adjunct case slots, for example. A battery of adjunct case slots was provided to supplement case frames from the dictionary. The final score for a given alignment between the input and case frame for a particular sense, was determined based on the quality of match/level of restrictiveness of the selectional constraints for each aligned case slot, from which a case slot restrictiveness score was deducted for the most highly specified non-aligned case slot. In limited evaluation, the method achieved an aligned sense accuracy of over 83% and was shown to be superior to the native ALT-J/E verb sense disambiguation method. It was also found to be highly robust to underspecification and successful at predicting fixed expression sense. These features of our verb sense disambiguation method can be summarised as:

- Our VSD method makes heavy use of argument status and also the degree of semantic restrictiveness of the selectional constraints associated with each case slot. Argument status is suggested as a powerful tool in modelling scope for case marker alternation, semantic backing-off and argument omission.

- Case slot restrictiveness was calculated from the topological characteristics of the region on the tree-based thesaurus, described by selectional constraints.

- Semantic backing-off is supported, linked in to the argument status of the case slot in question and penalised so as to reflect the degree of violation of the original selectional constraints.

- The method is robust to underspecification, by virtue of penalising non-alignment of case slots by way of the maximum score of case slot restrictiveness for a non-instantiated case slot, once again weighted according to argument status to capture the ready omissibility of non-complement case slots.

While the particular disambiguation contexts we have covered are relatively specialised, the methods proposed for resolving the various types of ambiguity encountered are transferrable to a wide range of applications, both intra- and inter-language. At the very least, they are illustrative of the diverse range of ambiguity types that exist in MT, and more generally, natural language

processing, and the proposed disambiguation methods indicative of the ways of tackling such ambiguity.

Appendix A

Publication list

Japanese relative clauses

- Baldwin, T., H. Tanaka, and T. Tokunaga. 1997a. Analysis of head gapping in Japanese relative clauses. In *Information Processing Society of Japan SIG Notes*, volume 97, no. 4, 1–8.
- Baldwin, T., T. Tokunaga, and H. Tanaka. 1997b. Semantic verb classes in the analysis of head gapping in Japanese relative clauses. In *Proc. of the 4th Natural Language Processing Pacific Rim Symposium 1997 (NLPRS'97)*, 409–14.
- Baldwin, T., T. Tokunaga, and H. Tanaka. 1997c. Syntactic and semantic constraints on head gapping in Japanese relative clauses. In *Proc. of the Third Annual Meeting of the Japanese Association for Natural Language Processing*, 277–80.
- Baldwin, T. 1998a. Relative clause coordination and subordination in Japanese. In *Proc. of the Australian Natural Language Processing Postgraduate Workshop*, 1–10.
- Baldwin, T. 1998b. *The Analysis of Japanese Relative Clauses*. Master's thesis, Tokyo Institute of Technology.

Grapheme-phoneme alignment

- Bilac, S., T. Baldwin, and H. Tanaka. 1999. Incremental Japanese grapheme-phoneme alignment. In *Information Processing Society of Japan SIG Notes*, volume 99-NL-209, 47–54.
- Baldwin, T., and H. Tanaka. 1999a. The applications of unsupervised learning to Japanese grapheme-phoneme alignment. In *Proc. of the ACL Workshop on Unsupervised Learning in Natural Language Processing*, 9–16.
- Baldwin, T., and H. Tanaka. 1999b. Automated Japanese grapheme-phoneme alignment. In *Proc. of the International Conference on Cognitive Science*, 349–54.
- Baldwin, T., and H. Tanaka. 2000a. A comparative study of unsupervised grapheme-phoneme alignment methods. In *Proc. of the 22nd Annual Meeting of the Cognitive Science Society (CogSci 2000)*, 597–602.
- Baldwin, T., and H. Tanaka. in press. A comparative study of unsupervised grapheme-phoneme alignment methods. *Journal of Natural Language Processing*.

Translation retrieval

- Baldwin, T., and H. Tanaka. 2000b. The effects of word order and segmentation on translation retrieval performance. In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000)*, 35–41.

- Baldwin, T., and H. Tanaka. 2000c. Balancing up efficiency and accuracy in translation retrieval. In *IPSJ SIG Notes*, volume 2000, 109–16.

- Baldwin, T., and H. Tanaka. in press. Balancing up efficiency and accuracy in translation retrieval. *Journal of Natural Language Processing* .

Verb alternations

- Baldwin, T., F. Bond, and B. Hutchinson. 1999a. An alternation-based Japanese valency dictionary architecture. In *Proc. of the Fifth Annual Meeting of the Association of Natural Language Processing*, 193–6.

- Baldwin, T., and H. Tanaka. 2000d. Verb alternations and Japanese — how, what and where? In *Proc. of the 14th Pacific Asia Conference on Language, Information and Computation (PACLIC 14)*, 3–14.

- Baldwin, T., T. Tokunaga, and H. Tanaka. 1999a. Preliminary analysis of the range and frequency of Japanese verb alternations. In *Information Processing Society of Japan SIG Notes*, volume 99-NL-134, 139–46.

- Baldwin, T., F. Bond, and K. Ogura. to appear Dictionary-driven analysis of Japanese verbal alternations In *Proc. of the Seventh Annual Meeting of the Association of Natural Language Processing*.

Word sense disambiguation

- Fujii, A., K. Inui, T. Baldwin, T. Tokunaga, and H. Tanaka. 1997. Towards the application of word sense disambiguation to information extraction. In *Proceedings of the International Workshop on Lexically Driven Information Extraction*, 39–49.

- Baldwin, T., T. Tokunaga, and H. Tanaka. 1998c. Lexical effects in verb sense disambiguation. In *Information Processing Society of Japan SIG Notes*, volume 98-NL-209, 47–54.

- Baldwin, T., and H. Tanaka. 1999c. Argument status in Japanese verb sense disambiguation. In *Proc. of the 8th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*, 207–217.

Miscellaneous

- Baldwin, T., T. Tokunaga, and H. Tanaka. 1998d. A computational account of modality-based case frame transformation. In *Proc. of the Fourth Annual Meeting of the Japanese Association for Natural Language Processing*, 58–61.

• Baldwin, T., B. Hutchinson, and F. Bond. 1999b. A valency dictionary architecture for machine translation. In *Proc. of the 8th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*, 207–217.

Appendix B

Miscellaneous Word Lists

B.1 SMART word list

The following is a listing of all words contained in the SMART (Salton 1971) stop word list, normalised to lower case. This was used to score down stop words over other English words ("content words") in the translation retrieval task.

> a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, asking, associated, at, available, away, awfully, b, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by, c, c'mon, c's, came, can, can't, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, couldn't, course, currently, d, definitely, described, despite, did, didn't, different, do, does, doesn't, doing, don't, done, down, downwards, during, e, each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except, f, far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore, g, get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings, h, had, hadn't, happens, hardly, has, hasn't, have, haven't, having, he, he's, hello, help, hence, her, here, here's, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however, i, i'd, i'll, i'm, i've, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, isn't, it, it'd, it'll, it's, its, itself, j, just, k, keep, keeps, kept, know, knows, known, l, last, lately, later, latter, latterly, least, less, lest, let, let's, like, liked, likely, little, look, looking, looks, ltd, m, mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself, n, name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere, o, obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own, p, particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides, q, que, quite, qv, r, rather, rd, re, really, reasonably,

regarding, regardless, regards, relatively, respectively, right, s, said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, shouldn't, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure, t, t's, take, taken, tell, tends, th, than, thank, thanks, thanx, that, that's, thats, the, their, theirs, them, themselves, then, thence, there, there's, thereafter, thereby, therefore, therein, theres, thereupon, these, they, they'd, they'll, they're, they've, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two, u, un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually, uucp, v, value, various, very, via, viz, vs, w, want, wants, was, wasn't, way, we, we'd, we'll, we're, we've, welcome, well, went, were, weren't, what, what's, whatever, when, whence, whenever, where, where's, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, who's, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, won't, wonder, would, would, wouldn't, x, y, yes, yet, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves, z, zero

B.2 Noun groups used in the RCC interpretation task

B.2.1 Non-gapping nouns

"Non-gapping nouns" typically occur as the head NP of attributive RCCs, and are taken from Baldwin (1998b).

daNkai "stage", *dōki* "motivation", *dōtei* "course, lead-up", *geN-daNkai* "present stage", *geNsyō* "phenomenon", *hima* "spare time", *hōhō* "method", *hōsiki* "formula, method", *hōsiN* "direction", *huri* "pretence", *hūsyū* "custom", *ikeN* "opinion", *ikō* "intention", *iNsyō* "impression", *issiN* "devotion", *itoguti* "beginning", *kaeri-miti* "way home", *kakkō* "form, appearance", *kaN* "feeling", *kaNgae* "thinking, idea", *kanōsei* "possibility", *kaNzi* "feeling", *katati* "form, appearance", *katei* "process", *keikaku* "plan", *keikō* "trend", *kekka* "result", *kēsu* "case", *kikai* "opportunity", *kimoti* "feelings", *kioku* "memory", *kōka* "effect", *koNkyo* "underlying purpose/motivation", *kotu* "knack", *kuse* "habit", *maebure* "forewarning, advance notice", *magiwa* "just before", *mitōsi* "outlook", *mokuteki* "objective", *moyō* "pattern, situation", *muki* "heading, aspect", *nerai* "objective", *nōryoku* "capacity", *omoi* "thought, feeling", *osore* "sign, danger", *oto* "sound", *rei* "example", *reNsyū* "practice", *riyū* "reason", *sainō* "talent", *sikake* "trap, device", *sikumi* "structure", *siNpai* "fear, worry", *sisei* "position", *sutairu* "style", *syokku* "shock", *syōko* "proof", *tati* "disposition, inclination", *tatiba* "stance", *tūneN* "common idea", *ugoki* "trend", *wake* "reason", *wariai* "proportion", *yaku* "role", *yakume* "role", *yakusoku* "promise", *yōi* "preparation, readiness", *yōiN* "cause", *yotei* "plan", *yoyū* "room to manoeuvre, composure", *yueN* "cause, source", *zikkaN* "realisation", *zittai* "substance", *zizitu* "fact", *zyōtai* "state"

B.2.2 First person pronouns

Japanese contains a range of first person pronouns, marked according to formality and meter.

boku, *ore*, *ware*, *wareware*, *wasi*, *watakusi*, *watasi*

We make the (over-)generalisation that any of these can optionally co-occur with the plural suffix markers *tati* and *ra*.

B.2.3 Goal agentive nouns

The goal agentive noun set has two members:

aite "opponent, opposite", *saki* "opposing party, destination"

Bibliography

Abney, S., R.E. Schapire, and Y. Singer. 1999. Boosting applied to tagging and pp attachment. In *Proc. of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Aha, D.W., and R.L. Bankert. 1994. Feature selection for case-based classification of cloud types: An empirical comparison. In *Proc. of the AAAI-94 Workshop on Case-Based Reasoning*.

Almuallim, H., Y. Akiba, and T. Yamazaki. 1994. Two methods for learning ALT-J/E rules from examples and a semantic hierarchy. In *Proc. of the 15th International Conference on Computational Linguistics (COLING '94)*, 57–63.

——, and T.G. Dietterich. 1991. Learning with many irrelevant features. In *Proc. of the 9th Annual Conference on Artificial Intelligence (AAAI-91)*, 547–52.

Azzam, S., K. Humphreys, and R. Gaizauskas. 1998. Evaluating a focus-based approach to anaphora resolution. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 74–8.

Baldwin, T. 1998a. Relative clause coordination and subordination in Japanese. In *Proc. of the Australian Natural Language Processing Postgraduate Workshop*, 1–10.

——, 1998b. *The Analysis of Japanese Relative Clauses*. Master's thesis, Tokyo Institute of Technology.

——, B. Hutchinson, and F. Bond. 1999. A valency dictionary architecture for machine translation. In *Proc. of the 8th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*, 207–217.

——, and H. Tanaka. 1999. The applications of unsupervised learning to Japanese grapheme-phoneme alignment. In *Proc. of the ACL Workshop on Unsupervised Learning in Natural Language Processing*, 9–16.

——, and H. Tanaka. 2000a. A comparative study of unsupervised grapheme-phoneme alignment methods. In *Proc. of the 22nd Annual Meeting of the Cognitive Science Society (CogSci 2000)*, 597–602.

——, and H. Tanaka. 2000b. The effects of word order and segmentation on translation retrieval performance. In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000)*, 35–41.

——, H. Tanaka, and T. Tokunaga. 1997a. Analysis of head gapping in Japanese relative clauses. In *Information Processing Society of Japan SIG Notes*, volume 97, no. 4, 1–8.

——, T. Tokunaga, and H. Tanaka. 1997b. Semantic verb classes in the analysis of head gapping in Japanese relative clauses. In *Proc. of the 4th Natural Language Processing Pacific Rim Symposium 1997 (NLPRS'97)*, 409–14.

——, T. Tokunaga, and H. Tanaka. 1997c. Syntactic and semantic constraints on head gapping in Japanese relative clauses. In *Proc. of the Third Annual Meeting of the Japanese Association for Natural Language Processing*, 277–80.

——, T. Tokunaga, and H. Tanaka. 1998. A computational account of modality-based case frame transformation. In *Proc. of the Fourth Annual Meeting of the Japanese Association for Natural Language Processing*, 58–61.

Blum, A.L., and P. Langley. 1997. Selection of relevant features and examples in machine learning. *Artificial Intelligence: Special Issue on Relevance* 97.245–71.

Bond, F., and S. Shirai, 1997. *Practical and Efficient Organization of a Large Valency Dictionary.* Handout at the *Workshop on Multilingual Information Processing, held in conjunction with NLPRS'97.*

Breiman, L. 1994. Bagging predictors. Technical Report 421, University of California, Berkeley.

Brill, E. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics* 21.543–65.

Brown, P.F., V.J. Della Pietra, P.V. deSouza, J.C. Lai, and R.L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics* 18.467–79.

Cardie, C. 1992. Corpus-based acquisition of relative pronoun disambiguation heuristics. In *Proc. of the 30th Annual Meeting of the ACL*, 216–23.

—— in press. A cognitive bias approach to feature selection and weighting for case-based learners. *Machine Learning* .

Caruana, R., and D. Freitag. 1994. Greedy attribute selection. In *Proc. of the 11th International Conference on Machine Learning*, 28–36.

Charniak, E. 2000. A maximum entropy-based parser. In *Proc. of the 1st Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL2000).*

Cherkauer, K.J., and J.W. Shavlik. 1996. Growing simpler decision trees to facilitate knowledge discovery. In *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, 315–8.

Choubey, S.K., J.S. Deogun, V.V. Raghavan, and H. Sever. 1996. A comparison of feature selection algorithms in the context of rough classifiers. In *Proc. of the International Conference on Fuzzy Systems (FUZZ-IEEE '96)*, volume 2, 1122–8.

——, ——, ——, and ——. 1998. On feature selection and effective classifiers. *Journal of the American Society for Information Science* 49.423–34.

Collins, M.J. 1996. A new statistical parser based on lexical dependencies. In *Proc. of the 34th Annual Meeting of the ACL*, 184–91.

—— 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of the 33rd Annual Meeting of the ACL*, 16–23.

Daelemans, W., J. Zavrel, K. van der Sloot, and A. van den Bosch, 2000. Timbl: Tilburg memory based learner, version 3.0, reference guide. ilk technical report 00-01. Avalable from `http://ilk.kub.nl/~ilk/papers/ilk0001.ps.gz`.

Dagan, I., and A. Itai. 1994. Word sense disambiguation using a second language monolingual corpus. *Computational Linguistics* 20.563–96.

Dorr, B.J., and M.B. Olsen. 1996. Multingual generation: The role of telicity in lexical choice and syntactic realization. *Machine Translation* 11.37–74.

EDR, 1995. *EDR Electronic Dictionary Technical Guide*. Japan Electronic Dictionary Research Institute, Ltd. (In Japanese).

Elworthy, D. 1994. Does Baum-Welch re-estimation help taggers? In *Proc. of the 4th Conference on Applied Natural Language Processing (ANLP)*, 53–8.

Escudero, G., L. Màrquez, and G. Rigau. 2000. Boosting applied to word sense disambiguation. In *Proc. of the 11th European Conference on Machine Learning, ECML-2000*, 129–141.

Ferrández, A., M. Palomar, and L. Moreno. 1998. Anaphor resolution in unrestricted texts with partial parsing. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 385–91.

Fisher, D., and E. Riloff. 1992. Applying statistical methods to small corpora: Benefitting from a limited domain. In *Working Notes of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, 47–53.

Foley, W., and R. Van Valin. 1984. *Functional Syntax and Universal Grammar*. CUP.

Freund, Y., and R.E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proc. of the 13th International Conference on Machine Learning*.

Fujii, A., 1998. *Corpus-Based Word Sense Disambiguation*. Tokyo Institute of Technology dissertation.

Fujii, H., and W.B. Croft. 1993. A comparison of indexing techniques for Japanese text retrieval. In *Proc. of 16th International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'93)*, 237–46.

Fung, P., and K.W. Church. 1994. K-vec: A new approach for aligning parallel texts. In *Proc. of the 15th International Conference on Computational Linguistics (COLING '94)*, 1096–102.

Gale, W., K. Church, and D. Yarowsky. 1992. A method for disambiguating word senses in a large corpus. *Computers and the Humanities* 26.415–39.

Gale, W.A., and K.W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics* 19.75–102.

Henderson, J.C., and E. Brill. 2000. Bagging and boosting a treebank parser. In *Proc. of the 1st Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL2000)*, 34–41.

Ikehara, S., M. Miyazaki, A. Yokoo, S. Shirai, H. Nakaiwa, K. Ogura, Y. Ooyama, and Y. Hayashi. 1997. *Nihongo Goi Taikei – A Japanese Lexicon*. Iwanami Shoten. 5 volumes. (In Japanese).

Ikehara, Satoru, Satoshi Shirai, Akio Yokoo, and Hiromi Nakaiwa. 1991. Toward an MT system without pre-editing – effects of new methods in **ALT-J/E**–. In *Proc. of the Third Machine Translation Summit (MT Summit III)*, 101–106, Washington DC. (http://xxx.lanl.gov/abs/cmp-lg/9510008).

Inoue, K. 1976. *Henkei Bunpo to Nihongo*. Tokyo: Taishukan.

Jacobsen, W.M. 1992. *The Transitive Structure of Events in Japanese*. Kurosio Publishers.

John, G.H., R. Kohavi, and K. Pfleger. 1994. Irrelevant features and the subset selection problem. In *Proc. of the 11th International Conference on Machine Learning*, 121–9.

Kameyama, M., 1985. *Zero anaphora: The case of Japanese*. Stanford University dissertation.

Keenan, E. L., and B. Comrie. 1977. Noun phrase accessibility and universal grammar. *Linguistic Inquiry* 8.63–99.

Kira, K., and L.A. Rendell. 1992. A practical approach to feature selection. In *Proc. of the 9th International Machine Learning Conference*, 249–56.

Kitamura, E., and H. Yamamoto. 1996. Translation retrieval system using alignment data from parallel texts. In *Proc. of the 53rd Annual Meeting of the IPSJ*, volume 2, 385–6. (In Japanese).

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, volume 2, 1137–43.

——, and G.H. John. 1995. Automatic parameter selection by minimizing estimated error. In *Proc. of the 12th International Conference on Machine Learning*, 304–12.

Koller, D., and M. Sahami. 1996. Toward optimal feature selection. In *Proc. of the 13th International Conference on Machine Learning*.

Kononenko, I., M. Robnil-Šikonja, and U. Pompe. 1996. ReliefF for estimation and discretization of attributes in classification, regression and ILP problems. In *Proceedings of AIMSA'96*, 31–40.

Kukich, K. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys* 24.377–439.

Kuno, S. 1973. *The Structure of the Japanese Language*. MIT Press, Cambridge MA.

Kurohashi, S., and M. Nagao. 1994. A method of case structure analysis for Japanese sentences based on examples in case frame dictionary. *IEICE Transactions on Information and Systems* E77-D.227–39.

——, and ——. 1998. *Nihongo keitai-kaiseki sisutemu JUMAN* [Japanese morphological analysis system JUMAN] version 3.5. Technical report, Kyoto University. (In Japanese).

Langley, P. 1994. Selection of relevant features in machine learning. In *Proc. of the AAAI Fall Symposium on Relevance*.

——, and S. Sage. 1994. Oblivious decision trees and abstract cases. In *Working Notes of the AAAI-94 Workshop on Case-Base Reasoning*, 113–7.

——, and ——. 1997. Scaling to domains with irrelevant features. In *Computational Learning Theory and Natural Learning Systems*, ed. by R. Greiner, volume 4, 17–29. MIT Press.

Leacock, C., M. Chodrow, and G.A. Miller. 1998. Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics* 24.147–65.

Li, H.F., J.H. Lee, and G. Lee. 1998. Identifying syntactic role of antecedent in Korean relative clause using corpus and thesaurus information. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 756–62.

Ling, C., and B. Zhang. 1998. Grapheme generation in learning to read English words. In *Proc. of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI'98*, ed. by Mercer and Neufield, 184–95. Springer.

Mahesh, K., S. Nirenburg, and S. Beale. 1997a. If you have it, flaunt it: Using full ontological knowledge for word sense disambiguation. In *Proc. of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-97)*, 1–9.

——, ——, ——, E. Viegas, V. Raskin, and B. Onyshkevych. 1997b. Word sense disambiguation: Why statistics when we have these numbers? In *Proc. of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-97)*, 151–9.

Mandala, R., T. Tokunaga, and H. Tanaka. 1999. Combining multiple evidence from different types of thesaurus for query expansion. In *Proc. of the 22th International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, 191–97.

Manning, C., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Martin, S. E. 1975. *A Reference Grammar of Japanese*. Tuttle.

Masek, W.J., and M.S. Paterson. 1980. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences* 20.18–31. 1980.

Matheus, C.J., and L.A. Rendell. 1989. Constructive induction on decision trees. In *Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 1, 645–50.

Matsumoto, Y., A. Kitauchi, T. Yamashita, and Y. Hirano. 1999. *Japanese Morphological Analysis System ChaSen Version 2.0 Manual*. Technical Report NAIST-IS-TR99009, NAIST.

Matsumoto, Yoshiko. 1997. *Noun Modifying Constructions in Japanese*. John Benjamins.

Meyers, A., C. Macleod, and R. Grisham. 1996. Standardization of the complement/adjunct distinction. In *Proc. of the Seventh Euralex International Congress*. (`ftp://cs.nyu.edu/pub/html/comlex.html/adj-compl.ps`).

Mitchell, T.M. 1997. *Machine Learning*. McGraw-Hill.

Mlaneniċ, D., and M. Grobelnik. submitted. Feature selection for learning on large text data. *Machine Learning* .

Murata, M., and M. Nagao. 1998. An estimate of referent of noun phrases in Japanese sentences. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 912–6.

Nagao, M., and S. Mori. 1994. A new method of N-gram statistics for large number of N and automatic extraction of words and phrases from large text data of Japanese. In *Proc. of the 15th International Conference on Computational Linguistics (COLING '94)*, 611–5.

Nakaiwa, H. 2000. An environment for extracting resolution rules of zero pronouns from corpora. In *Proc. of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, 44–52.

——, and S. Ikehara. 1994. Japanese zero pronoun resolution in a machine translation system using verbal semantic attributes. In *Proc. of SPICIS '94*, B341–6.

——, and ——. 1995. Intrasentential resolution of Japanese zero pronouns in a machine translation system using semantic and pragmatic constraints. In *Proc. of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-95)*, 95–105.

——, and ——. 1997. A system of verbal semantic attributes in Japanese focused on syntactic correspondence between Japanese and English. *Journal of the Information Processing Society of Japan* 38.215–25. (In Japanese).

——, S. Shirai, S. Ikehara, and T. Kawaoka. 1995. Extrasentential resolution of Japanese zero pronouns using semantic and pragmatic constraints. In *Proc. of the AAAI Spring Symposium Series, Empirical Methods in Discourse Interpretation and Generation*, 99–105.

——, A. Yokoo, and S. Ikehara. 1994. A system of verbal semantic attributes focused on the syntactic correspondence between Japanese and English. In *Proc. of the 15th International Conference on Computational Linguistics (COLING '94)*, 672–8.

Nakamura, N. 1989. Translation support by retrieving bilingual texts. In *Proc. of the 38th Annual Meeting of the IPSJ*, volume 1, 357–8. (In Japanese).

Nariyama, S., 2000. *Referent Identification for Ellipted Arguments in Japanese*. The University of Melbourne dissertation.

Ng, H. T., and H. B. Lee. 1996. Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In *Proc. of the 34th Annual Meeting of the ACL*, 40–7.

Nirenburg, S., C. Domashnev, and D.J. Grannes. 1993. Two approaches to matching in example-based machine translation. In *Proc. of the 5th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-93)*, 47–57.

Pagallo, G., and D. Haussler. 1990. Boolean feature discovery in empirical learning. *Machine Learning* 5.71–99.

Palmer, M., and Z. Wu. 1995. Verb semantics for English-Chinese translation. *Machine Translation* 10.59–92.

PAPAGEORGIOU, C.P. 1994. Japanese word segmentation by Hidden Markov Model. In *Proc. of the ARPA Human Language Technology Workshop*, 283–8.

PERLMUTTER, D. M., and P. M. POSTAL. 1984. The 1-advancement exclusiveness law. In *Studies in Relational Grammar 2*, ed. by D. M. Perlmutter and C. G. Rosen, chapter 3, 81–125. The University of Chicago Press.

PLANAS, E., 1998. *A Case Study on Memory Based Machine Translation Tools.* PhD Fellow Working Paper, United Nations University.

——, and O. FURUSE. 1999. Formalizing translation memories. In *Proc. of Machine Translation Summit VII*, 331–9.

——, and ——. 2000. Multi-level similar segment matching algorithm for translation memories and example-based machine translation. In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000)*, 621–7.

PRAGER, J., D. RADEV, E. BROWN, and A. CODEN. 2000. The use of predictive annotation for question answering in trec8. In *Proc. of the 8th Text REtrieval Conference (TREC-8)*, 399–411.

QUINLAN, J.R. 1986. Induction of decision trees. *Machine Learning* 1.81–206.

—— 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

ROBERTSON, A.M., and P. WILLETT. 1998. Applications of n-grams in textual information systems. *Journal of Documentation* 54.48–69.

SALTON, G. 1971. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall.

——, and M.J. MCGILL. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill.

SATO, R., 1989. *Nihongo no Rentai-shūshoku-setsu no Imi-kaiseki ni-kansuru Kenkyū.* Master's thesis, Tokyo Institute of Technology. (In Japanese).

SATO, S. 1992. CTM: An example-based translation aid system. In *Proc. of the 14th International Conference on Computational Linguistics (COLING '92)*, 1259–63.

—— 1994. Example-based translation and its MIMD implementation. In *Massively Parallel Artificial Intelligence*, ed. by H. Kitano and J. Hendler, 171–201. AAAI/MIT Press.

——, and T. KAWASE. 1994. *A High-Speed Best Match Retrieval Method for Japanese Text.* Technical Report IS-RR-94-9I, JAIST.

——, and M. NAGAO. 1990. Toward memory-based translation. In *Proc. of the 13th International Conference on Computational Linguistics (COLING '90)*, 247–52.

SCHAFFER, C. 1993. Overfitting avoidance as bias. *Machine Learning* 10.153–78.

SCHAPIRE, R.E. 1999. A brief introduction to boosting. In *+IJCAI99*.

SHIBATANI, M. 1990. *The Languages of Japan*. CUP.

Shirai, S., S. Yokoo, H. Inoue, H. Nakaiwa, S. Ikehara, and A. Yagi. 1997. Nichi-ei kikai-hon'yaku ni-okeru imi-kaiseki no tame no kōbun jisho [A structural dictionary for semantic analysis in Japanese-English machine translation]. In *Proc. of the Third Annual Meeting of the Japanese Association for Natural Language Processing*, 153–6. (In Japanese).

Silverstein, M. 1976. Hierarchy of features and ergativity. In *Grammatical Categories in Australian Languages*, ed. by R.M.W. Dixon, 112–71. Humanities Press.

Singhal, A., S. Abney, M. Bacchiani, M. Collins, D. Hindle, and F. Pereira. 2000. AT&T at TREC-8. In *Proc. of the 8th Text REtrieval Conference (TREC-8)*, 317–30.

——, C. Buckley, and M. Mitra. 1996. Pivoted document length normalization. In *Proc. of 19th International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'96)*, 21–9.

Sirai, H., and T. Gunji. 1998. Relative clauses and adnominal clauses. In *Topics in Constraint-Based Grammar of Japanese*, ed. by T. Gunji and K. Hasida, chapter 2, 17–38. Kluwer Academic.

Skalak, D.B. 1994. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proc. of the 11th International Conference on Machine Learning*, 293–301.

Snedcor, G.W., and W.C Cochran. 1989. *Statistical methods*. Iowa State University Press, 8th edition edition.

Somers, H. L. 1984. On the validity of the complement-adjunct distinction in Valency grammar. *Linguistics* 22.507–30.

—— 1987. *Valency and Case in Computational Lingustics*. Edinburgh University Press.

Stede, M. 1996. Lexical paraphrases in multilingual sentence generation. *Machine Translation* 11.75–107.

Sumita, E., and Y. Tsutsumi. 1991. A practical method of retrieving similar examples for translation aid. *Transactions of the IEICE* J74-D-II.1437–47. (In Japanese).

Tanaka, H. 1996. Decision tree learning algorithm with structured attributes: Application to verbal case frame acquisition. In *Proc. of the 16th International Conference on Computational Linguistics (COLING '96)*, 943–8.

—— 1997. An efficient way of gauging similarity between long Japanese expressions. In *Information Processing Society of Japan SIG Notes*, volume 97, no. 85, 69–74. (In Japanese).

Teramura, H. 1975–78. Rentai-shushoku no shintakusu to imi Nos. 1–4. In *Nihongo Nihonbunka 4–7*, 71–119, 29–78, 1–35, 1–24. Osaka: Osaka Gaikokugo Daigaku. (In Japanese).

Trujillo, A. 1999. *Translation Engines: Techniques for Machine Translation*. Springer Verlag.

Tsujimura, N. 1996. *An Introduction to Japanese Linguistics*. Blackwell.

Van Valin, R. 1984. A typology of syntactic relations in clause linkage. In *Proc. of the Tenth Annual Meeting of the Berkeley Linguistics Society*, 542–58.

Veale, T., and A. Way, 1997. *Gaijin:* a bootstrapping, template-driven approach to example-based MT. `http://www.compapp.dcu.ie/~tonvy/papers/ gaijin.html`.

Veenstra, J., Antal van den Bosch, S. Buchholz, W. Daelemans, and J. Zavrel. 2000. Memory-based word sense disambiguation. *Computers and the Humanities, Special Issue on* senseval 34.

Voorhees, E.M. 2000. The TREC-8 question answering track report. In *Proc. of the 8th Text REtrieval Conference (TREC-8)*, 77–82.

Wagner, A., and M. Fisher. 1974. The string-to-string correction problem. *Journal of the ACM* 21.168–73.

White, A.P., and W.Z. Liu. 1994. Bias in information-based measures in decision tree induction. *Machine Learning* 15.321–9.

Wilks, Y., and M. Stevenson. 1998. Word sense disambiguation using optimised combinations of knowledge sources. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 1398–1402.

Wu, D. 1994. Aligning parallel English–Chinese text statistically with lexical criteria. In *Proc. of the 32nd Annual Meeting of the ACL*, 80–7.

Xu, D., and C.L. Tan. 1999. Alignment and matching of bilingual English–Chinese news texts. *Machine Translation* 14.1–33.

Yamamoto, K., and E. Sumita. 1998. Feasibility study for ellipsis resolution in dialogues by machine-learning technique. In *Proc. of the 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics (COLING/ACL-98)*, 1428–34.

Yarowsky, D. 1993. One sense per collocation. In *Proc. of ARPA Human Language Technology Workshop*, 266–71.

—— 1994. Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proc. of the 32nd Annual Meeting of the ACL*, 88–95.

Yokoo, A., and Y. Hayashi. 1987. Analysis of Japanese embedded-noun-phrase structures. In *Proc. of the First Annual Meeting of the Japanese Society for Artifical Intelligence*, 369–72. (In Japanese).